Proceedings of the Spatial Computing Workshop (SCW 2013) colocated with AAMAS (W09)

Jean-Louis Giavitto, Stefan O. Dulman, Antoine Spicher, Mirko Viroli

AAMAS 2013
Saint Paul
Minnesota
USA
6th - 10th May 2013
twelfth international conference on autonomous agents and multiagent systems

W9 – The 6th International Workshop on Spatial Computing (SCW 2013)

J.-L. Giavitto (CNRS & IRCAM, UPMC, INRIA)
S. Dulman (Delft University)
A. Spicher (Paris-Est University)
M. Viroli (University of Bologna)

May 6

Spatial Computing 2013

colocated with AAMAS

Saint-Paul, 6 May 2013

Dr. Jean-Louis Giavitto (CNRS & IRCAM)
Dr. Stefan Dulman (Delft University)
Prof. Antoine Spicher (LACL - Univ. Paris-Est Créteil)
Prof. Mirko Viroli (University of Bologna)

Contents

Foreword

Many multiagent systems are *spatial computers* – collections of local computational devices distributed through a physical space, in which:

- the interaction between localized agents is strongly dependent on the distance between them, and
- the "functional goals" of the system are generally defined in terms of the system's spatial structure.

For example, spatial relationships are often used to organize the interactions between agents, at least in applications in which the problem and the space are intertwined. Furthermore, multiagent-based systems and their behaviors can be specified and analyzed relying on spatial notions like: location, region, frontier, neighborhood, obstruction, field, basin, communication, diffusion, propagation, etc.

Systems that can be viewed as spatial computers are abundant, both natural and man-made. For example, in wireless sensor networks and animal or robot swarms, inter-agent communication network topologies are determined by the distance between devices, while the agent collectives as a whole solve spatially-defined problems like "analyze and react to spatial temperature variance" or "surround and destroy an enemy."

On the other hand, not all spatially distributed systems are spatial computers. The Internet and peer-to-peer overlay networks may not in general best be considered as spatial computers, both because their communication graphs have little relation to the Euclidean geometry in which the participating devices are embedded, and because most applications for them are explicitly defined independent of the network structure. Spatial computers, in contrast, tend to have more structure, with specific constraints and capabilities that can be used in the design, analysis and optimization of algorithms.

The goal of **the 6th Spatial Computing International Workshop** is to serve as an inclusive forum for the discussion of ongoing or completed work focusing on the theoretical and practical issues of *explicitly using space* in the design process of multiagent or multiactor systems.

Indeed, the handling of space often remains implicit and elementary in general, reflected also in its limited adoption in the myriad of domain-specific programming languages. We believe that progress towards identifying common principles, techniques, and research

directions—and consolidating the substantial progress that is already being made—will benefit all of the fields in which spatial computing takes place. And, as the impact of spatial computing is recognized in many areas, we hope to set up frameworks to ensure portability and cross-fertilization between solutions in the various domains.

The Spatial Computing Workshop provides a premier forum for sharing both research and engineering results, as well as potential challenges and prospects.

Workshop History

The domain of spatial computing is young but the last 6 years have shown a constant interest of researchers from various communities: parallelism, self-organization, complex systems, systems biology, swarm robotics, autonomic systems, amorphous computing, and at last but not least, multiagent systems.

The Spatial Computing Workshop (SCW) series has been established in 2008 after an initial Dagsthul seminar *Computing Media and Languages for Space-Oriented Computation*[1] in 2006 and a followup in Paris in 2008 *From Amorphous Computing to Spatial Computing* where the decision was made to begin the workshop. Since 2008 and until 20111, SCW has been a satellite workshop of the SASO conference[2]. From 2012, SCW is colocated with AAMAS. The past SCW have been held in:

- Venice[3] (2008)
- San Francisco[4] (2009)
- Budapest[5] (2010)
- Ann Arbor[6] (2011)
- Valencia[7] (2012)

In 2009, a special issue of the ACM Transactions on Autonomous and Adaptive Systems (TAAS) were organized, where SCW participants and other researchers were invited to contribute.

In 2011, a special issue of The Computer Journal has been launched following the same open scheme.

[1] http://www.dagstuhl.de/06361/
[2] http://www.saso-conference.org/
[3] http://projects.csail.mit.edu/scw08/
[4] http://radlab.cs.berkeley.edu/saso2009/workshops.html and http://scw09.spatial-computing.org/
[5] http://www.inf.u-szeged.hu/saso10/index.php?menu=workshop and http://scw10.spatial-computing.org/
[6] http://www.cscs.umich.edu/SASO2011/index.php?menu=workshop and http://scw11.spatial-computing.org/
[7] http://aamas2012.webs.upv.es/index.php?option=com_content&view=article&id=6&Itemid=6 and http://scw12.spatial-computing.org/

Current Trends in Spatial Computing

The papers presented at this workshop give a sampling of the current work in Spatial Computing. In the past editions of SCW, the following topics were discussed:

- Relationships between agent interaction and spatial organizations, self-organization, self-assemblies, collective motions;
- Characterization of spatial self-organization phenomena as algorithmic building blocks;
- Control theory approaches for designing dynamic spatial computing applications;
- Theoretical and practical limitations arising from spatial properties, understanding and characterization of spatial computing specific errors, analysis of tradeoffs between system parameters;
- Studies of the relationship between space and time - propagation of information through the spatial computer, and computational complexity;
- Languages for programming spatial computers and describing spatial tasks and space/time patterns;
- Methods for compiling global programs to local rules for specific platforms (so called global-to-local compilers);
- Suitable design methodologies and tools, such as novel domain-specific languages, for implementing, validating and evaluating spatial applications;
- Application of spatial computing principles to novel areas, or generalization of area-specific techniques;
- Device motion and control in spatial computing algorithms (e.g. relationship between robot speed and gradient accuracy in robotic swarms);
- Novel spatial applications, emphasizing parallel, mobile, pervasive, P2P, amorphous and autonomic systems;
- Testbeds and use-studies of spatial applications;
- ...

This year also, the papers selected for SCW reflect the diversity of spatial computing. They investigate a wide range of spatial programming features and applications, from distributed computing to ambient computing, from robotic and artificial life to design and music.

Without the help of the program committee, the workshop would not have come about and we would like to thank them for their involvement in SCW. Grateful acknowledgments are also due to the efficient logistic of the AAMAS organizers and the AAMAS workshop chair, and to our supporting institutions: CNRS, Delft University, Inria MuSync team, Ircam and the RepMus team, University of Bologna, UPMC, University of Paris-Est Créteil and the LACL lab.

We hope you will have as much fun and interest as us, in the reading of these proceedings.

March 2013

Dr. Stefan Dulman (Delft University)
Dr. Jean-Louis Giavitto (CNRS & IRCAM)
Prof. Antoine Spicher (LACL - Univ. Paris-Est Créteil)
Prof. Mirko Viroli (University of Bologna)

Program Committee

- Michel Banatre (Inria, France)
- Jacob Beal (BBN Technologies)
- Sven Brueckner (Jacobs Technology Inc.)
- Nikolaus Correll, Dpt of Computer Science, University of Colorado at Boulder
- Ferruccio Damiani, Dipartimento di Informatica, UniversitÃ di Torino
- Rene Doursat, Complex Systems Institute, Paris Ile-de-France
- Alexis Drogoul, UMI UMMISCO 209, IRD & UPMC and Can Tho University
- Matt Duckham, University of Melbourne
- Jérôme Durand-Lose, LIFO - University of OrlÃ©ans
- Nazim Fates, LORIA - INRIA Nancy
- Fred Gruau, LRI U. of Paris South
- Guillaume Hutzler, IBISC Evry University
- Taras Kowaliw, ISC-PIF, CNRS
- Luidnel Maignan, LACL University Paris Est
- Olivier Michel, LACL University Paris Est
- Ulrik Schultz, University of Southern Denmark
- Susan Stepney, Dept. of Computer Science, University of York
- Christof Teuscher, Portland State University
- Kyle Usbeck, BBN Technologies
- Danny Weyns, Linnaeus University
- Dr. Eiko Yoneki (University of Cambridge, UK)
- Franco Zambonelli, University of Modena and Reggio Emilia

Organizing Committee

- Dr. Stefan Dulman (Delft University)
- Dr. Jean-Louis Giavitto (CNRS & IRCAM)
- Prof. Antoine Spicher (LACL - Univ. Paris-Est, Créteil)
- Prof. Mirko Viroli (University of Bologna)

Schedule

09:00 – 10.30 Models I

- *Engineering Confluent Computational Fields: from Functions to Rewrite Rules*
  Mirko Viroli
- *Declarative Multidimensional Spatial Programming*
  John Plaice, Jarryd P. Beck and Blanca Mancilla
- *Promoting Space-Aware Coordination: ReSpecT as a Spatial Computing Virtual Machine* Stefano Mariani and Andrea Omicini

10:30 – 11:00 coffee break

11:00 – 12:30 Algorithms I

- *Modeling Inertia in an Amorphous Computing Medium*
  Alyssa S. Morgan and Daniel N. Coore
- *Computing Activity in Space*
  Martin Potier, Antoine Spicher and Olivier Michel
- *Accelerating Approximate Consensus with Self-Organizing Overlays*
  Jacob Beal

12:30 – 1:30 lunch break

1:30 – 2:00 Algorithms II

- *AREA: an Automatic Runtime Evolutionary Adaptation mechanism for Creating Self-Adaptation Algorithms in Wireless Networks*
  Qingzhi Liu, Stefan Dulman and Martijn Warnier

2:00 – 3:30 Applications

- *Application of Force-Directed Graphs on Character Positioning*
  Christine Talbot and G. Michael Youngblood
- *Spatial Programming for Musical Transformations and Harmonization*
  Louis Bigo, Jean-Louis Giavitto and Antoine Spicher
- *Spatial Computing in an Orbital Environment: An Extrapolation of the Unique Constraints of this Special Case to other Spatial Computing Environments*
  Jeremy Straub

3:30 – 4:00 coffee break

4:00 – 5:00 Models II

- *Population Protocols on Graphs: A Hierarchy*
  Olivier Bournez and Jonas Lefèvre
- *Spatial Computation and Algorithmic Information content in Non-DNA based Molecular Self-Assembly*
  Germán Terrazas, Leong Ting Lui and Natalio Krasnogor

5:00 – 6:00 Coordination

- *Teams to exploit spatial locality among agents*
  James Parker and Maria Gini
- *Composing gradients for a context-aware navigation of users in a smart-city*
  Sara Montagna and Mirko Viroli

6:00 – 6:30 Demonstration, Discussion, Farewell

Regular Presentations

Accelerating Approximate Consensus with Self-Organizing Overlays

Jacob Beal
Raytheon BBN Technologies
Cambridge, MA 02138
Email: jakebeal@bbn.com

***Abstract*—Laplacian-based approximate consensus algorithms are an important and frequently used building block in many distributed applications, including formation control, sensor fusion, and synchronization. These algorithms, however, converge extremely slowly on networks that are more than a few hops in diameter and where values are spatially correlated. This paper presents a new algorithm, Power Law Driven Consensus, which uses a self-organizing virtual overlay network to accelerate convergence, at a cost of decreasing the predictability of the final converged value. Experimental comparison with the Laplacian approach confirms that PLD-consensus allows for drastically faster convergence in spatial networks.**

I. Introduction

Approximate consensus algorithms are an important building block for many distributed algorithms, including robot formation control [1], flocking and swarming [2], sensor fusion [3], modular robotics [4] and synchronization [5]. The dominant algorithmic approach to distributed approximate consensus is a Laplacian-based approach in which each device finds a weighted local average of its own current value with the values held by its neighbors. In effect, this approach is operating like particle diffusion, such that as the differences between devices eventually equalize, the network is brought into consensus. Although this approach supports a number of elegant mathematical results [6], [7], including an exponential rate of convergence derived from the graph Laplacian, the bounds on the rate of convergence are extremely loose, and may actually indicate a very slow rate of convergence indeed.

Spatial computers, which tend to have mesh-like network structure many hops across, are an example of where Laplacian-based consensus performs poorly. As demonstrated in [8], on spatial computers, the expected convergence time is actually $O(diameter^2)$ with a high constant factor, such that Laplacian-based approximate consensus is expected to converge extremely slowly whenever there is both significant diameter and also a spatial correlation in the initial values of devices. Since many applications of approximate consensus, including those mentioned above, are commonly executed on spatial computers with spatially correlated values, this can severely limit the efficacy and applicability of consensus-based applications.

This paper introduces a new algorithm, Power Law Driven Consensus, which uses a self-organizing virtual overlay network to accelerate convergence, at a cost of decreasing the predictability of the final converged value. Following a brief review and specification of problem context in Section II, I present the new PLD-consensus algorithm in Section III. Section IV then compares the new algorithm with the Laplacian approach in simulation, confirming that PLD-consensus allows for drastically faster convergence in spatial networks.

II. Approximate Consensus on Spatial Computers

A spatial computer is generally defined as any collection of devices in which the difficulty of moving information between any two devices is strongly dependent on the distance between them, and where the functional goals of the system are linked to its spatial structure. Examples include ad-hoc communication networks, swarms of unmanned aerial vehicles (UAVs), sensor networks, and colonies of engineered biological cells.

For purposes of this paper, we will consider a more restricted class of spatial computer, in which a set of n devices are arranged in a graph $G = \{V, E\}$ and also embedded by a function $p : V \to M$ into a Riemannian manifold M with distance function d. Edges are assumed to be bidirectional, to have equal weight, and not to exist between two vertexes i and j if $d(p(i), p(j))$ is greater than some threshold.

Laplacian-based average consensus algorithms [7] solve a specific class of distributed consensus problem: given a real-number initial local value $l_i(0)$ for each device i, Laplacian-based consensus computes the approximate mean of $l_i(0)$ by iterative applying the transformation:

$$l_i(t+1) = l_i(t) + \epsilon \sum_{j \in \mathcal{N}(i,t)} l_j(t) - l_i(t) \qquad (1)$$

where $\mathcal{N}(i,t)$ is the set of graph neighbors of device i at time t and the constant $\epsilon > 0$ is the step size of the algorithm. For simplicity, we give only the synchronous specification; only minor modification is required for a more general non-synchronized algorithm.

Laplacian-based consensus has been proven [7] to converge exponentially toward the mean value of $l_i(0)$, but the rate of convergence is set by the second eigenvalue of the graph Laplacian, which is very small for high-diameter mesh networks. In fact, as shown in [8], Laplacian-based consensus performs badly on spatial computers, with a convergence time that is $O(\textit{diameter}^2)$ and a high constant factor due to the generally large potential number of neighbors.

When there is no correlation in the distribution of values on devices, this does not matter as much, since a rough estimate of

global values can be made from locally sampled information. When values are correlated by their location in the network, however, no reasonable estimate of consensus can be made without moving information over long distances. Since spatial computers often have a large diameter and values are often highly correlated with location, a faster algorithm is clearly needed.

III. Power Law Driven Consensus

If the problems with Laplacian-based consensus come from the high diameter of the network, then one clear approach to accelerating consensus would be to reduce the effective network diameter through construction of an overlay network containing long-distance links.

Power Law Driven Consensus is a simple implementation of such an approach, where the overlay network self-organizes using a $1/f$ distribution to break symmetry, selecting certain devices to have their values spread over a longer distance. Under PLD-consensus, devices compete to become "dominant" over one another, drawing their bids for dominance from the scale-free $1/f$ distribution. The value of using a scale-free distribution of this sort is that, no matter the size or arrangement of the network, the expected distribution contains one device that receives a value large enough to dominate the network. Note that this is an initial version, and the process can likely be optimized significantly through modulation of this distribution.

Dominance decays over both time and space, so this competition results in a partition of the network into "dominance regions" that shift over time. Values then spread outward from dominant devices through the regions that they dominate, and each device blends the local dominant value with its own value.

Since this distribution is scale-free, the regions of dominance begin small, allowing local blending of values, much like Laplacian-based consensus. The regions then expand rapidly until the entire network is dominated by a single device, effectively reducing diameter and allowing for rapid convergence. Finally, because dominance decays over time, any dominant device that fails will eventually be replaced by other dominant devices.

A. Formal Algorithm Specification

Having giving some intuitions for how PLD-consensus will function, we will now provide a formal specification of the algorithm. For simplicity, this specification will be stated in terms of synchronous rounds. In fact, however, there is no requirement for synchrony, both the overlay construction and blending operations are relaxation methods, meaning that any local updates (within the stable range) is expected to move the system as a whole closer to a converged state. The algorithm is thus expected to operate well under a wide range of non-synchronous conditions as well (indeed, the simulations in Section IV are non-synchronous).

Let us begin the specification by considering the computation of the dominance overlay. This sub-algorithm serves two functions: first, determining the relative dominance level of each device and second, flowing values down the dominance gradient from more dominant to less dominant devices.

The dominant value state for each device i at each round t is a tuple $(d_i(t), u_i(t), v_i(t))$ of the current dominance level $d_i(t)$, a tie-breaker unique identifier $u_i(t)$, and a dominant value $v_i(t)$. In addition, each device i has a local value $l_i(t)$, which is its current candidate for a consensus value.

At each round, the algorithm considers three sources for the new dominant value state. The driven state S_d is:

$$S_d = (\lfloor \frac{1}{u(0,1)} \rfloor, u_i(t), l_i(t)) \tag{2}$$

where $u(0,1)$ is a uniform random distribution over the interval $(0,1)$. Taking the inverse of $u(0,1)$ produces $1/f$-noise, giving a power-law distribution of candidate new dominance levels being injected into the network at each round.

For each neighbor $n \in \mathcal{N}(i,t)$, where $\mathcal{N}(i,t)$ is the set of neighbors of device i at time t, the neighbor-derived state S_n is:

$$S_n = (d_n(t) - 1, u_n(t), v_n(t)) \tag{3}$$

which takes the neighbor's value at a decremented dominance level.

Finally, if no neighbor has a higher dominance level, the leader state S_l is:

$$S_l = (L_i, u_i(t), l_i(t)) \tag{4}$$

where the leader dominance L_i is

$$L_i = \begin{cases} d_i(t) - 1 & \text{if } \forall n \in \mathcal{N}(i,t),\ d_i(t) > d_n(t) \\ 0 & \text{else} \end{cases} \tag{5}$$

which decrements the old dominance level but inserts the new local value.

The new state is then set to whichever of these three sources has the highest dominance level:

$$(d_i(t+1), u_i(t+1), v_i(t+1)) = \\ \text{Lmax}(S_d \cup S_l \cup \{S_n | n \in \mathcal{N}(i,t)\}) \tag{6}$$

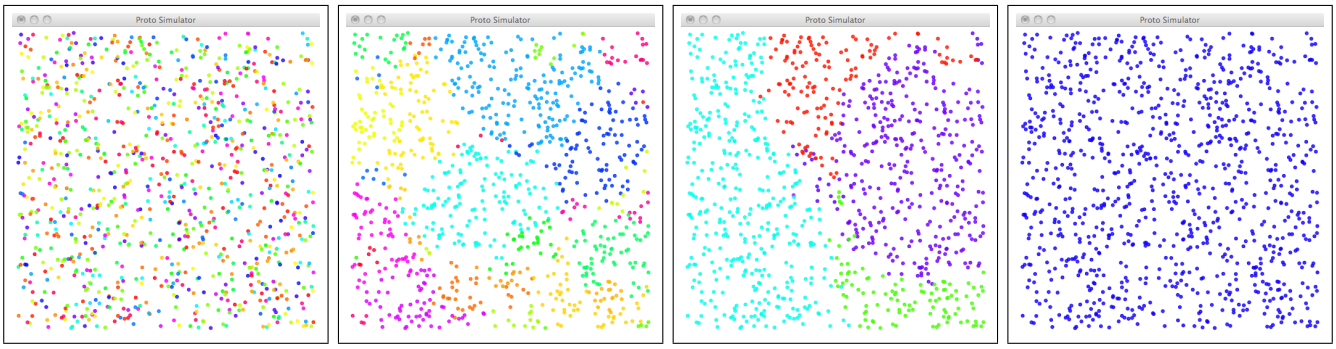
where Lmax is a lexicographic maximum, such that the first elements of a tuple are compared, then if they are equal the second elements are compared, etc.

Figure 1 shows an example of the regions of dominance produced by this computation. Initially, no devices are dominant over their neighbors, but as dominance levels are injected via the $1/f$-noise, regions of dominance grow rapidly until eventually some device is able to dominate the entire network.

Once the dominance overlay has been established, computation of consensus is relatively straightforward. Given an initial local value of $l_i(0)$ at each device i, the value at round t may be computed as a simple proportional blend:

$$l_i(t) = \alpha \cdot v_i(t) + (1 - \alpha) \cdot l_i(t-1) \tag{7}$$

where $v_i(t)$ is the dominant value as provided from the overlay, and α is the proportional blending constant. Note,



(a) Initial (b) 10 rounds (c) 30 rounds (d) 100 rounds

Fig. 1. Visualization of dominance competition driven by $1/f$-noise, by having each device spread a unique RGB color computed from its UID. From an initial state where no devices are dominant (a), regions of dominance are expected to grow rapidly (b,c) until eventually some device is able to dominate the entire network.

```
(def dominant-value (value)
  (3rd
   (rep ;; Declare three state variables; only third will be returned
    (tup dominance id spread-value)
    ;; Initialize to no dominance, local ID and value
    (tup 0 (mid) value)
    (let* ;; Three options for finding dominant value:
          ;; 1: Drive new dominance with 1/f noise
          ((local (floor (/ 1 (rnd 0 1))))
           ;; 2: Take a decremented dominance from a neighbor
           (hoodmax (max-hood
                     (nbr (tup (- dominance 1)
                               id spread-value))))
           ;; 3: Decrement old dominance, but use new value
           (leader (= (2nd hoodmax) (mid))))
      ;; Use whichever option has greater dominance
      (if (or leader (> local (1st hoodmax)))
        (tup (max local (1st hoodmax)) (mid) value)
        hoodmax)))))
```

Fig. 2. Proto code for computing dominant value. Comments are in blue, state variables in green.

however, that updating in this way uses only overlay values and no neighbor values at all.

To soften this non-locality, PLD-consensus mixes overlay blending and the local blending of Laplacian-based consensus by the simple expedient of adding the standard Laplacian differential term:

$$l_i(t) = \alpha \cdot v_i(t) \; + \; (1-\alpha) \cdot l_i(t-1) \; + \\ \beta \sum_{n \in \mathcal{N}(i,t)} w(e_{i,n}) \cdot (l_n(t-1) - l_i(t-1)) \quad (8)$$

where β is the step size for Laplacian-based consensus. PLD-consensus thus makes use of both neighbor values and values delivered through the overlay, and reduces to pure overlay blending when $\beta = 0$ and to Laplacian-based consensus when $\alpha = 0$.

IV. Experimental Validation

In this section, I present experimental validation of the PLD-consensus algorithm in simulation. These experiments have two aims: first, to quantitatively compare the performance

```
(def PLD-consensus (initial alpha beta)
  (rep
   value    ;; Declare one state variable: current consensus value
   initial ;; Initialize consensus with local initial value
   ;; Find regionally-dominant value ...
   (let ((dominant (dominant-value value)))
     ;; ... and blend incrementally with local value ...
     (+ (+ (* alpha dominant) (* (- 1 alpha) value))
        ;; ... and add Laplacian differential
        (* beta (sum-hood (- (nbr value) value)))))))
```

Fig. 3. Proto code for PLD-consensus algorithm. Comments are in blue, state variables in green.

of PLD-consensus against Laplacian-based consensus, and second, to gain an initial understanding of the behavior of PLD-consensus under different configurations and conditions of execution.

For these experiments, I implemented the PLD-consensus algorithm in Proto [9], [10]. The code for creating and propagating values through the dominance overlay is listed in Figure 2, and the code for the full PLD-consensus algorithm is listed in Figure 3. Proto was a desirable language for implementation and experimental validation for two reasons: first, Proto's programming model allows a concise and direct implementation of the mathematical specification given in the previous section, and, second, the network simulator distributed with MIT Proto made it simple to run and analyze experiments on large spatial networks.

Except where otherwise noted, all experiments are run with the following parameters: the network consists of 1000 devices distributed uniformly randomly in a 100 meter by 100 meter rectangle. Devices use a unit disk model of communication, communicating with all other devices within r meters, where r is computed to give an expected 10 neighbors per device. The algorithms are run for 1000 rounds of partially synchronous execution (equal frequency, random phase), and PLD-consensus is run with $\alpha = 0.02$ and $\beta = 0$, while Laplacian-based consensus is run with a step size of $\epsilon = 0.02$. For each experimental condition, 10 trials are run. When analyzing the convergence statistics of a network, the devices

with the top and bottom 2.5% of values are ignored, to ensure that any devices that may be disconnected due to the random distribution are excluded. A network is then considered to have converged if the difference between the minimum and maximum value within the median 95% of devices is less than 1% of the initial difference (e.g., less than 0.01 if all devices start with values between 0 and 1).

A. Illustrative Comparison of Algorithms

Let us begin with an illustrative comparison of Laplacian-based consensus and PLD-consensus, to provide an initial intuition of the difference between these two approaches. For this experiment, we use a set of 1000 devices distributed uniformly randomly in a 100 meter by 100 meter rectangle. Devices can communicate via broadcast to all other devices within 7 meters, giving approximately 15 expected neighbors per device. Devices are initialized to one of two values based on their spatial location, with devices in the left hand side of the plane given an initial value $l_i(0) = 10$, and on the right hand side $l_i(0) = 30$, producing a highly spatially correlated distribution. To have the purest comparison of PLD-consensus and Laplacian-based consensus, we consider the case of $\alpha = 0.01$ and $\beta = 0$ for PLD-consensus, meaning that only dominance overlay values are used, and compare against a step size of $\epsilon = 0.01$ for Laplacian-based consensus.

Figure 4 shows the evolution of values held by the various devices over time during a single trial run. Although both cases begin the same, PLD-consensus converges rapidly, within a few hundred rounds, while even after 5000 rounds of computation Laplacian-based consensus has not converged, with a difference of 1.87 between the minimum and maximum of the median 95%—nearly 10% of the initial value difference.

The trade-off for the faster convergence of PLD-consensus, however, is a decreased accuracy in finding the overall mean value. In this case, the Laplacian consensus values at $t = 5000$ have a mean of 19.56, which is quite close to the true mean of 20. The converged value of PLD-consensus, on the other hand, is somewhat farther off at 14.35.

B. Convergence Rate

For a more thorough comparison of convergence rate, let us compare the evolution of value distributions in PLD-consensus and Laplacian-based consensus. For this experiment, trials were run for two initial distribution conditions: a homogeneous condition in which each device's initial value $l_i(t)$ is drawn uniformly randomly from the interval $[0, 1]$, and a spatially-correlated condition in which devices in the left half of the distribution start with $l_i(t) = 0$ and devices in the right half start with value $l_i(t) = 1$. Values were then recorded every 10 rounds for 1000 rounds.

Figure 5 plots the difference between highest and lowest value in the median 95% against time. In both cases, PLD-consensus clearly greatly outperforms Laplacian-based consensus. Under the homogeneous condition, Laplacian-based consensus initially converges more quickly than PLD-consensus, but greatly slows when further convergence requires equalization of values over many hops. PLD-consensus converges rapidly in both the homogeneous and spatially-correlated conditions. With spatially-correlated initial values, however, Laplacian-based consensus has barely even begun to converge by $t = 1000$, and will not complete its convergence for many thousands of rounds more, as seen in the prior illustrative comparison.

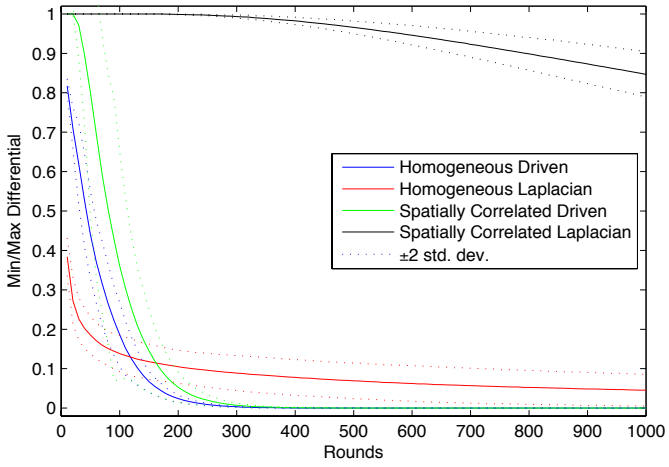


Fig. 5. PLD-consensus progresses toward convergence much faster than Laplacian-based consensus, though Laplacian-based consensus initially progresses more quickly when initial values are homogeneously distributed.

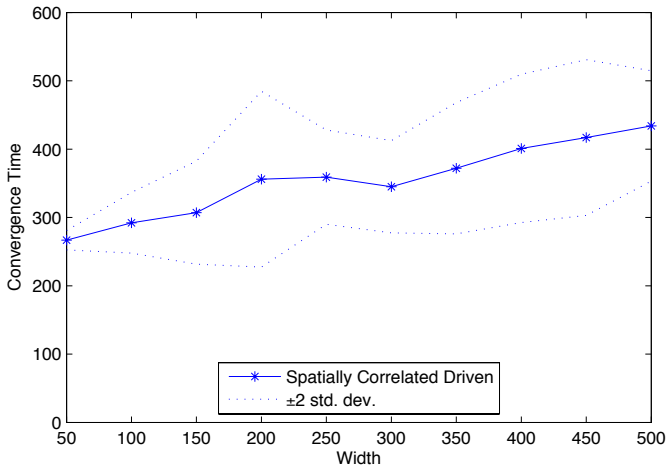
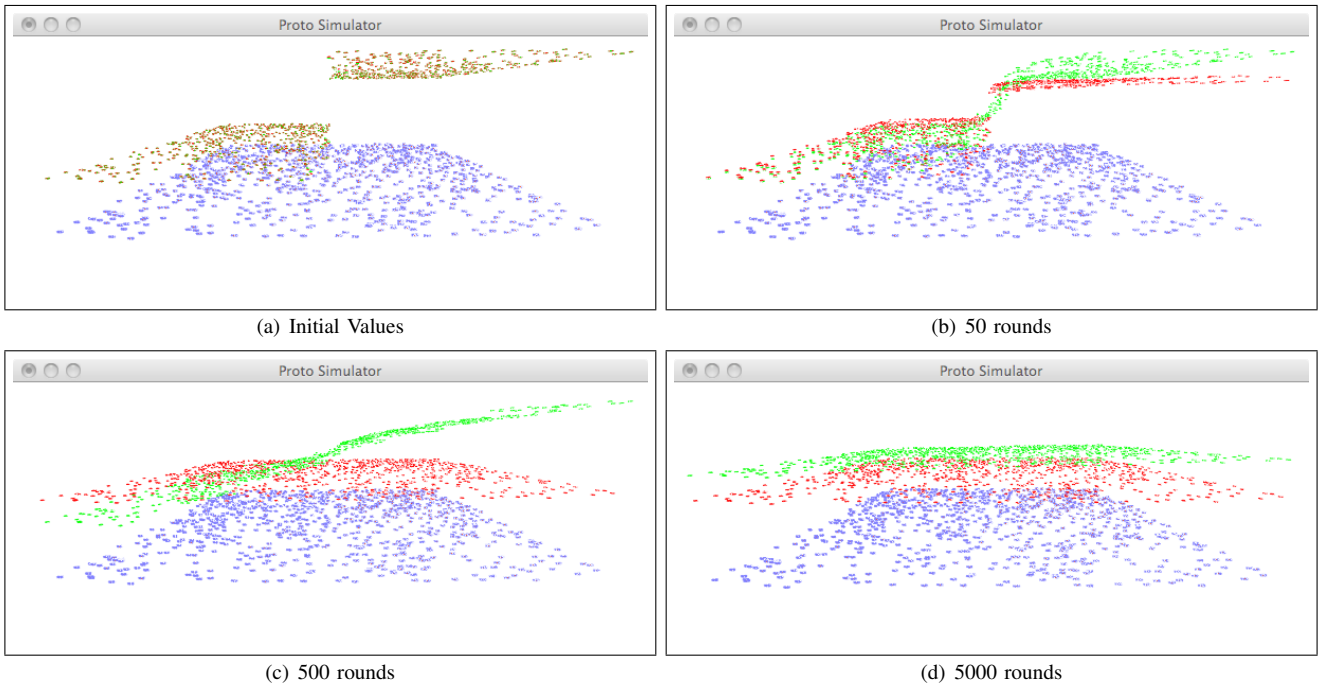


Fig. 6. Convergence time for PLD-consensus increases linearly with the width of the network.

C. Scaling with Diameter and Mobility

The poor scaling of Laplacian-based consensus as network diameter increases has already been demonstrated in [8]. Since the $1/f$-noise driving PLD-consensus is scale-free, however, the limiting factor should instead be communication time, and thus the algorithm should scale in $O(diameter)$. To test this, I ran the PLD-consensus algorithm with spatially-correlated initial values on networks with dimensions of X meters width by 50 meters, where the width X ranged from 50 to 500 meters



(a) Initial Values (b) 50 rounds

(c) 500 rounds (d) 5000 rounds

Fig. 4. PLD-consensus vs. Laplacian-based consensus on a mesh network of 1000 devices with approximately 15 neighbors each. The devices are shown as blue dots distributed in a plane, viewed at an angle, with their current values for consensus showed as the height of red (PLD-consensus) and green (Laplacian-based) dots above the plane. From an initial spatially-correlated distribution (a), PLD-consensus begins to converge rapidly (b), arriving at an approximation of the mean value within a few hundred rounds (c), while Laplacian-based consensus is eventually closer to the true mean, but even after 5000 rounds still retains nearly 10% of the initial value difference (d).

in steps of 50, and with a constant density of devices (thus ranging in number from 250 to 2500).

For this experiment, convergence is nearly universal, the only exceptions being one trial for $X = 50$ and one for $X = 100$, in which the random distribution contained too many disconnected devices. Figure 6 shows the scaling of convergence time width respect to network width X: as expected the convergence time increases approximately linearly with the diameter, atop an approximately constant base convergence time determined by the blending rate.

In many consensus applications, the participating devices are not stationary. This could both be helpful, in lowering the effective diameter of the network, and problematic, in scrambling the structure of the overlay. To investigate the effect of mobility, I ran the PLD-consensus algorithm with spatially-correlated initial values with devices moving at a velocity v varying geometrically from 0.01 to 1.0. Each device moves towards a randomly chosen point in space at velocity v, and upon reaching it (within quantization error) chooses a new point to move towards.

Figure 7 shows the results of this experiment: PLD-consensus converges for every trial, which is unsurprising since mobility means that no devices will remain disconnected. Perhaps more surprisingly, device movement does not appear to have any significant effect on convergence rate. It is likely that mobility does have some effect, but it is small enough to not be observable under these experimental conditions. Laplacian-based consensus, on the other hand, only converges

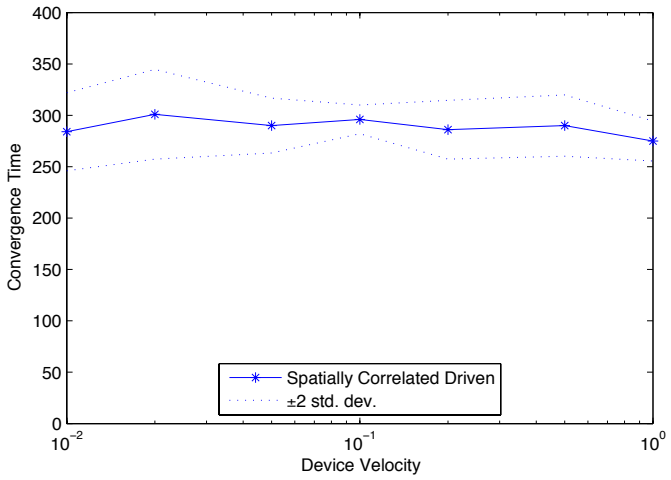
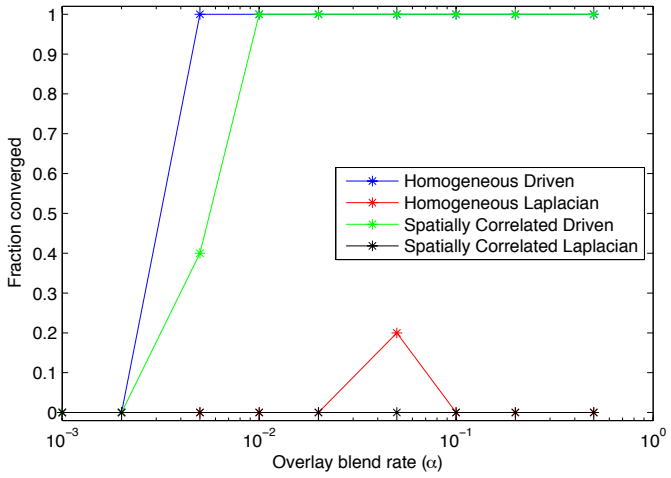


Fig. 7. Device movement does have any significant effect on the rate at which PLD-consensus converges within the range of parameters studied.

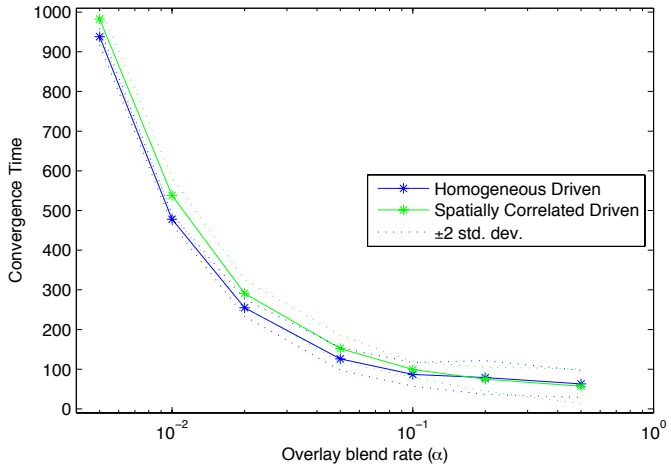
at all for $v = 1$, when 3 of the 10 trials converge just barely before $t = 1000$.

D. Effect of α and β Parameters

Finally, we consider the effects of the blending parameters α and β. For both of these parameters, there is a tension between speed and stability: the higher they are, the faster the network moves towards convergence, but if they are too high then it may become unstable.



(a) Percentage Converging



(b) Time of Convergence

Fig. 8. PLD-consensus converges reliably for all $\alpha \geq 0.01$, while Laplacian-based consensus nearly never converges (a). When PLD-consensus converges, the convergence time appears to be dominated by blend rate for low α and by overlay construction for high α (b).

To study the effect of varying α, I ran the PLD-consensus algorithm with spatially-correlated initial values, holding $\beta = 0$ while ranging α geometrically from 0.001 to 0.5. The experiment for β is the same, except that $\alpha = 0.02$, and β ranges from 0.001 to 0.5. For comparison with α variation, Laplacian-based consensus is run with a step size ϵ that is varied identically.

Figure 8 shows the results of varying α. PLD-consensus converges reliably for all $\alpha \geq 0.01$. For those trials that converge, the convergence time for low α is close to inversely proportional to α, indicating that convergence is likely dominated by blend rate. For high α, convergence time is nearly flat, indicating that convergence convergence is likely dominated by overlay construction. Laplacian-based consensus, on the other hand only converges at all for two moderate-alpha trials: with low α it converges too slowly and with high α it becomes unstable and values diverge rapidly.

Variation of β, on the other hand, produces no significant effect on either the convergence time or on the converged value, until $\beta = 0.1$, where the Laplacian component becomes unstable and values diverge. These results indicate that the Laplacian component is likely to be operating so slowly on spatially-correlated distributions as to have no measurable effect at the diameter studied by this experiment; only at lower α or diameter is there likely to be a significant effect.

V. Contributions and Future Work

This paper has presented a new approximate consensus algorithm, PLD-consensus, which uses a self-organizing overlay network to accelerate consensus. This allows much faster convergence, at a cost of higher expected deviation from the mean of the network's initial values.

This new algorithm has the potential for a major improvements across a wide class of consensus-based applications. Further study is required, however, before such applications can be made. For example, some applications may depend more on obtaining an accurate mean of initial values, while others may be able to tolerate more inaccuracy in exchange for speed. Likewise, this paper considered only the problem of one-shot approximate consensus, while applications such as flocking depend on tracking of a changing consensus over time. Similarly, PLD-consensus only considers the simplest overlay structure and means of combining Laplacian- and overlay-based consensus approaches; more sophisticated approaches may be able to produce greatly improved performance.

References

[1] M. Egerstedt and X. Hu, "Formation constrained multi-agent control," *IEEE Trans. on Robotics and Automation*, vol. 17, no. 6, pp. 947–951, 2001.

[2] R. Olfati-Saber, "Flocking for multi-agent dynamic systems: Algorithms and theory," *IEEE Trans. on Automatic Control*, vol. 51, no. 3, March 2006.

[3] L. Xiao, S. Boyd, and S. Lall, "A scheme for asynchronuous distributed sensor fusion based on average consensus," in *Fourth International Symposium on Information Processing in Sensor Networks*, 2005.

[4] C.-H. Yu and R. Nagpal, "Self-adapting modular robotics: A generalized distributed consensus framework," in *International Conference on Robotics and Automation (ICRA)*, 2009.

[5] J.-J. Slotine and W. Wang, "A study of synchronization and group cooperation using partial contraction theory," *Cooperative Control*, pp. 443–446, 2005.

[6] R. Olfati-Saber and R. Murray, "Consensus problems in networks of agents with switching topology and time-delays," *Automatic Control, IEEE Transactions on*, vol. 49, no. 9, pp. 1520–1533, Sept. 2004.

[7] R. Olfati-Saber, J. Fax, and R. Murray, "Consensus and cooperation in networked multi-agent systems," *Proceedings of the IEEE*, vol. 95, no. 1, pp. 215–233, Jan. 2007.

[8] N. Elhage and J. Beal, "Laplacian-based consensus on spatial computers," in *AAMAS 2010*, 2010.

[9] J. Beal and J. Bachrach, "Infrastructure for engineered emergence in sensor/actuator networks," *IEEE Intelligent Systems*, vol. 21, pp. 10–19, March/April 2006.

[10] "MIT Proto," software available at `http://proto.bbn.com/`, Retrieved June 22nd, 2012.

Spatial Programming for Musical Transformations and Harmonization

Louis Bigo*†, Jean-Louis Giavitto†, Antoine Spicher*,
*LACL/Université Paris-Est Créteil, 94010 France
†UMR CNRS STMS 9912/IRCAM Paris, 75004 France

***Abstract*—This paper presents a spatial approach to build spaces of musical chords as simplicial complexes. The approach developed here enables the representation of a musical piece as an object evolving over time in this underlying space, leading to a *trajectory*. Well known spatial transformations can be applied to this trajectory as well as to the underlying space. These spatial transformations induce a corresponding musical transformation on the piece. Spaces and transformations are computed thanks to MGS, an experimental programming language dedicated to spatial computing.**

***Index Terms*—MGS; musical transformation ; harmonization; counterpoint; self-assembly; *Tonnetz*.**

I. Introduction

Musical objects and processes are frequently formalized with algebraic methods [1]. Such formalizations can sometimes be usefully represented by spatial structures. A well-known example is the *Tonnetz* (figure 1), a spatial organization of pitches illustrating the algebraic nature of triads (*i.e.*, *minor* and *major* 3-note chords) [2]. In [3] we have introduced an original method that extends and generalizes the approach of [4] for the building of pitch spaces using simplicial complexes [5]. This combinatorial structure is used to make explicit algebraic relations between notes and chords, as in *Tonnetze*, or more general relationships like co-occurrences.

In such spaces, a musical sequence is represented by a sub-complex evolving over time: a *trajectory*. It is then compelling to look at the musical effect of well known spatial transformations on a trajectory. In section IV we investigate geometrical transformations, as discrete translations and discrete central symmetries, leading to the well known operations of musical transpositions and inversions. Such discrete geometrical transformations can be generalized, leading to a new family of transformations with less known musical interpretation. Some audio examples are available online[1]. In section V, the problem of counterpoint is investigated from a spatial perspective. We propose to generate the additional voice such that the distance with other played notes in a particular underlying space is minimized. This generalizes some spatial approaches to generate musical voice-leading. The underlying space is a parameter of the algorithm and by changing spaces, alternate (families of) solutions are generated.

[1]see the web page: http://www.lacl.fr/~lbigo/scw13

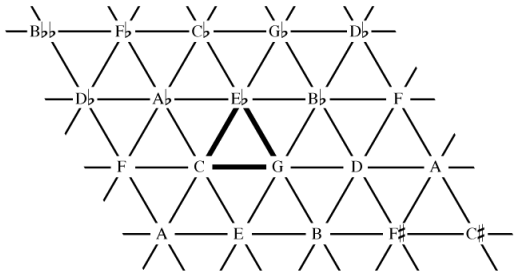


Figure 1. The original *Tonnetz*. Pitches are organized following the interval of fifth (horizontal axis), and the intervals of minor and major thirds (diagonal axis). Triangles represent minor and major triads.

II. A Short Introduction to MGS

MGS is an experimental domain specific language dedicated to spatial computing [6], [7]. MGS concepts are based on well established notions in algebraic topology [5] and uses of rules to compute declaratively spatial data structures. In MGS, all data structures are unified under the notion of *topological collection*. Simplicial complexes defined below are an example of topological collections. *Transformations* of topological collections are defined by rewriting rules [8] specifying replacement of sub-collections that can be recursively performed to build new spaces.

A *simplicial complex* is a space built by gluing together more elementary spaces called simplices. A *p-simplex* is the abstraction of an elementray space of dimension p and has exactly $p+1$ faces in its border. A 0-simplex corresponds to a point, a 1-simplex corresponds to an edge, a 2-simplex is a triangle, *etc*. The geometric realization of a p-simplex is the convex hull of its $p+1$ vertices as shown in Figure 3 for p-simplices with $p \in \{0,1,2,3\}$.

For any natural integer n, the *n-skeleton* of a simplicial complex is defined by the set of faces of dimension n or less.

A simplicial complex can be built from a set of simplices by self-assembly, applying an accretive growing process [9]. The growth process is based on the identification of the simplices in the boundaries. This topological operation is not elementary and holds in all dimensions. Figure 2 illustrates the process. First, nodes E and G are merged. Then, the resulting edges $\{E,G\}$ are merged.

A simple way to compute the identification is to iteratively apply, until a fixed point is reached, the merge of topological cells that exactly have the same faces. The corresponding topological surgery can be expressed as a simple MGS transformation as follows:

```
transformation identification = {
  s1 s2 / (s1==s2 & faces(s1)==faces(s2))
  =>
  let c = new_cell (dim s1)
                   (faces s1)
                   (union (cofaces s1)
                          (cofaces s2))
  in  s1*c
}
```

The expression `new_cell` p f cf returns a new p-cell with faces f and cofaces cf. The rule specifies that two elements `s1` and `s2`, having the same label and the same faces in their boundaries, merge into a new element `c` (whose cofaces are the union of the cofaces of `s1` and `s2`) labeled by `s1` (which is also the label of `s2`).

In Fig. 2, the transformation `identification` is called twice. At the first application (from the left complex to the middle), vertices are identified. The two topological operations are made in parallel. At the second application (from the complex in the middle to the right), the two edges from E to G that share the same boundary, are merged. The cofaces of the resulting edge are the 2-simplices I_C and III_C corresponding to the union of the cofaces of the merged edges. Finally (on the right), no more merge operation can take place and the fixed point is reached.

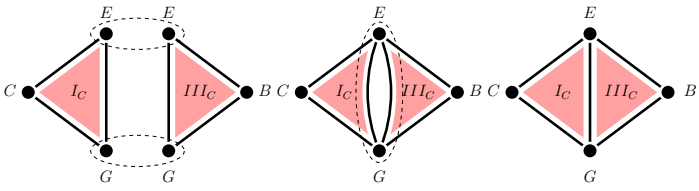


Figure 2. Identification of boundaries.

III. Chord Spaces

A. Building Chord Spaces

A chord space is an organization in space of a collection of musical chords. Such organizations are typically represented by graphs [10], or more recently by orbifolds [11]. Chords are generally represented in these spaces by vertices. A sequence of chords, which are included in the space, can thus be represented by a path. A *trajectory* generalizes the notion of path to higher dimensional simplices and a trajectory is not necessarily connected.

We use a method presented in [12] to represent chords by simplices. A n-note chord, viewed as a set of n notes, is represented by a $(n-1)$-simplex. To simplify the presentation, we consider pitch classes instead of notes. This abstraction is customary in musical analysis and gathers all notes equivalent up to an octave under the same class. For example, the notes

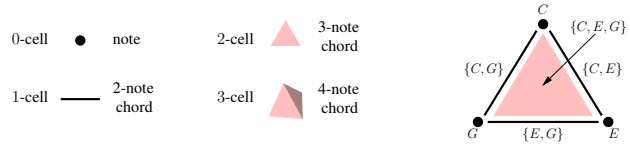


Figure 3. A chord represented as a simplex. The complex on the right corresponds to the 3-note chord C, E, G and all 2-note chords and notes included in it.

$C1$, $C2$, $C3$ …, all played by distinct keys on the piano, are grouped under the pitch class C.

In the simplicial representation of chord, a 0-simplex represents a single pitch class. More generally, a $n-1$-simplex represents a n-note chord, as illustrated on figure 3.

We build a *chord space simplicial complex* by representing each chord of a chord collection by a simplex, then by applying the self-assembly process described in section II. The application of this process to a collection of n-note chords gives rise to a $n-1$-dimensional simplicial complex. For example, the self-assembly process applied to the 24 major and minor triads (3-note chords) builds a toroidal simplicial complex. This complex extends the notion of *Tonnetz* developped in musical theory and illustrated on figure 1.

B. Chord Class Spaces

In this subsection, we present two particular types of chord spaces that will be used in next sections.

a) Chromatic Chord Class Spaces: Musical chords can be classified according different methods. One of the most popular classification spreads chords in 351 pitch class sets, called the *Forte Classes* [13]. Two pitch class sets belong to the same class if they are equivalent up to a transposition. We merge further chords equivalent up to transposition *and* inversion. The resulting classes can be obtained by listing orbits of the action of the dihedral group D_{12} on subsets of the cyclic group $\mathbb{Z}_{12}$ [14]. There are 224 such classes, we call chromatic chord classes. Chords belonging to a chromatic chord class share the same interval structure X: a sequence of intervals defined up to circular permutation and retrograde inversion. We note $\mathcal{C}(X)$ the simplicial complex representing the chromatic chord class associated with the interval structure X. In the chromatic system, the elements of X are elements of $\mathbb{Z}_{12}$.

b) Tonal Chord Class Spaces: A tonal chord class space is obtained by assembling chords sharing the same diatonic interval structure and including pitches of a particular tonality. If the scale of the tonality is heptatonic, (*i.e.*, the tonality includes seven pitch classes), the 16 spaces associated with the tonality can be obtained by enumerating the orbits of the action of the dihedral group D_7 on subsets of $\mathbb{Z}_7$. Such a space is noted $\mathcal{C}(X)$ where the elements of X belongs to $\mathbb{Z}_7$. For more details on these spaces, see [3].

C. Unfolding Chord Class Spaces

Chord class spaces of a same dimension can have different topologies. For example, $\mathcal{C}(3,4,5)$ and $\mathcal{C}(2,5,5)$ are both

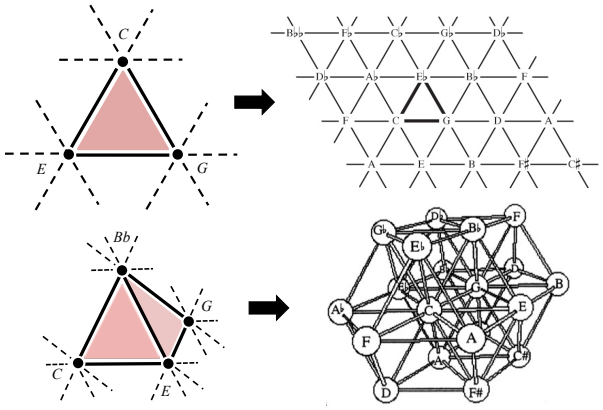


Figure 4. On the top, the unfolding process is applied to $\mathcal{C}(3,4,5)$ by extending C Major 1-simplices to infinite lines on the plane. At the bottom, unfolding process is applied to $\mathcal{C}(2,4,3,3)$ in the 3D space.

two-dimensional simplicial complexes but the first one has the shape of a torus and the second one has the shape of a strip [15]. However, chord class spaces of a given dimension can be *unfolded* in topologically equivalent infinite spaces. The unfolded representation is built as follows: an arbitrary chord of the class is represented by the geometric realization of its simplex. Then, 1-simplices (*i.e.*, edges) are extended as infinite lines. The interval labelling an edge is assigned to the corresponding line and all its parallels. Pitch classes and chords are organized and repeated infinitely following the lines according their assigned intervals.

The major difference between a simplicial complex and its unfolded representation is that in the former, notes are represented once, and in the latter, by an infinite number of occurrences. Moreover, the associated 1-skeleton can be embedded in the euclidean space such that parallel 1-simplices (representing 2-note chords) relate to the same interval class. By considering 1-skeletons of the unfolded complexes representing major and minor triads (Figure 4), one gets the neo-Riemannian Tonnetz [2]. The 1-skeleton of the unfolded complex representing seventh and half-diminished seventh chords is equivalent to Gollin 3D Tonnetz [16]. These two complexes are chromatic chord class spaces.

Chord class spaces resulting from the assembly of n-note chords are unfolded as $(n-1)$-dimensional infinite spaces. From 2-note chords one gets an infinite line, from 3-note chords an infinite triangular tessellation. Note that n-simplices don't systematically tessellate the n-dimensional Euclidean space. For example, 2-simplices (triangles) tessellate the 2D plan but 3-simplices (tetrahedra) do not tessellate the 3D space. For this reason, the 3D unfolded representation of complexes as the one at the bottom right of the figure 4 contains some holes.

IV. Spatial Transformations and their Musical Interpretation

We focus on unfolded representations of chord class spaces resulting from the self assembly of 3-note chords. These

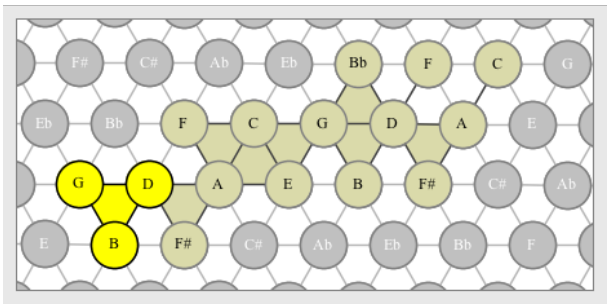


Figure 5. Path representing the first measures of J-S. Bach choral BWV 255. The chord class space used for the representation is obtained by unfolding $\mathcal{C}(3,4,5)$ which is the assembly of the 24 minor and major triads.

unfoldings are infinite triangular tessellations which have the property to preserve local neighborhoods between elements. If two elements are neighbor in a folded space then they are neighbor its unfolded representation, and vice versa. Note that this property does not systematically hold at higher dimensions. The advantage to consider unfolded representations of 3-note chords is that it preserves the neighborhood of each simplex while enabling the specification of discrete counterparts of euclidean transformation. This is not easily achieved in the initial finite space.

A. Representation of a Musical Sequence in a Chord Space

As previously said, a note corresponds to an infinite number of possible locations in an unfolded chord space. To represent a musical sequence as a moving object in such a space, only one of those locations has to be chosen for each played note over time. The precise location of a note played at some date is chosen in order to minimize the distance with both previously and simultaneously played notes. These two criteria enable the representation of the sequence by a trajectory "as connected as possible". Figure 5 illustrates such a path in $\mathcal{C}(3,4,5)$.

B. Spatial Transformations

Now we have a spatial representation of a musical sequence, we can apply some spatial transformations to it and listen to the musical result. We consider two kinds of transformations:

- the first one applies a geometrical transformation *on the trajectory*, (*i.e.*, on the spatial object representing the sequence in a predefined space) as illustrated on figure 7;
- the second one applies transformations on *the underlying space* of the piece, that is, the triangular tessellation (figure 8). This is possible because all such transformations amount to change the labels of the underlying space.

Musical examples of different pieces, before and after transformations, are available in MIDI format at http://www.lacl.fr/~lbigo/scw13 .

1) Geometrical Transformations: The regularity of the triangular tessellation enables to specify a discrete counterpart of usual geometrical operations like translations or some rotations.

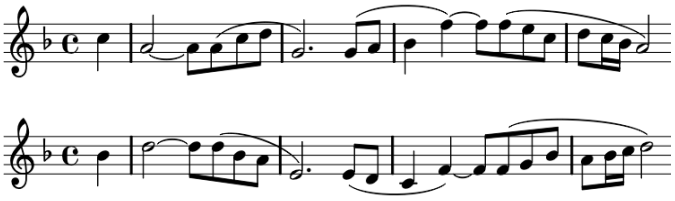


Figure 6. On the top, the first measures of the melody of the song *Hey Jude*. On the bottom, the same measures after three rotations in the complex $\mathcal{C}(1,2,4)$ related to F major tonality

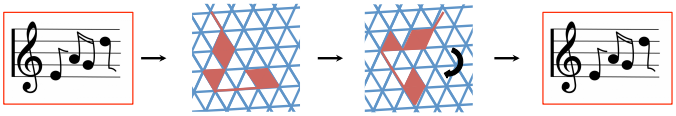


Figure 7. Rotation of a path in a triangular tessellation.

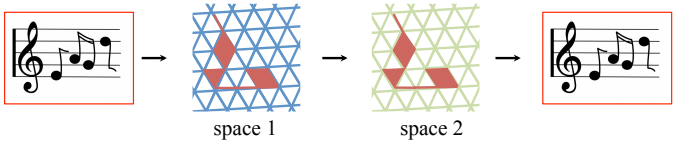


Figure 8. Transformation of the support space.

a) Translations: As previously mentioned, a direction in an unfolded space is associated with a constant interval. Then, a n-step translation of a path in a direction associated with the interval i reaches to a transposition of $n \times i$ on each note of the sequence. This translation is thus interpreted as a transposition (if the chord space is chromatic) or as a modal transposition (if the chord space is tonal). Audio example 1 is the result of a one-step translation of the path representing Beethoven's piece *Für Elise* in $\mathcal{C}(3,4,5)$. The direction of the translation is associated with the interval of fourth (the left direction on figure 5). The result is the transposition of the whole piece a fourth higher. Example 2 is the beginning of Mozart's 16^{th} sonata after a translation in $\mathcal{C}(1,2,4)$ related to C major tonality. Example 3 illustrates the same transformation on the song *Hey Jude* written by Paul McCartney, in $\mathcal{C}(1,2,4)$ related to F major tonality. This transformation corresponds to a modal transposition. The result is that the two original pieces switched from major mode to minor mode.

b) Rotations: Figure 7 illustrates a discrete $\pi/3$ rotation. Around a given vertex, five different rotations are possible in a triangular tessellation. This property is easily understandable by seeing that a note has six neighbors into six different directions. Thus, the motion to a note to one of his neighbors can be rotated five times around the starting note. Six rotations reach to an entire rotation around the center and is equivalent to identity. Three rotations are equivalent to a central symmetry.

This last operation is particularly interesting since it produces a trajectory having exactly the opposite direction from the original one. Each interval i being mapped to his opposite interval $-i$, this rotation is musically interpreted as an inversion (if the chord space is chromatic) or as an operation we could call a *modal inversion* (if the space is tonal).

Other rotations act as interval mappings depending on the properties of the chord space. Audio example 4 is the beginning of Mozart's 16^{th} sonata after 3 rotations (*i.e.* central symmetry) in $\mathcal{C}(3,4,5)$. Examples 5 and 6 are the same sequence after respectively 2 and 3 rotations in $\mathcal{C}(1,2,4)$ related to C major tonality.

Examples 7, 8 and 9 result from the same operations on the song *Hey Jude*. Figure 6 compares the first measures of the melody of the song before and after the central symmetry in $\mathcal{C}(1,2,4)$ related to F major tonality.

2) Change of Space: This operation consists in changing the labels of the underlying space, which is a triangular tessellation, for the labels of another unfolded two-dimensional chord class space. Thanks to topological equivalence of the two unfolded representations, the label mapping between the two spaces is straightforward. In this operation, the trajectory representing the musical sequence stays "unchanged". Example 10 is the beginning of J.-S.Bach's choral BWV 256 after the initial support space $\mathcal{C}(3,4,5)$ is transformed into $\mathcal{C}(2,3,7)$. The transformation achieves a surprising use of the pentatonic scale, giving a particular color to the transformed sequence. Transforming a chromatic space into a tonal space will lead to a musical sequence including notes of a unique tonality. An atonal piece thus becomes tonal. Example 11 illustrates this phenomena with the atonal piece *Semi-Simple Variations for piano* of Milton Babbit: The piece is represented in $\mathcal{C}(1,4,7)$. Then, this complex is transformed in $\mathcal{C}(1,2,4)$ related to the D minor tonality. The transformation, maps each note of the piece to a note in the D minor tonality.

3) Musical Interpretation: Some of these transformations have a natural interpretation in music. For example, the translation in a chromatic scale corresponds to a transposition. Our spatial approach highlights many other transformations that are not systematically studied in music theory, like for instance the n-rotations (with $1 \leq n \leq 5$ and $n \neq 3$).

These transformations can be combined to generate new musical results. For example, one can apply successively a rotation, a translation and a change of space, enabling a huge set of recombinations to generate new material from an initial musical sequence. Notice that some of these operations are equivalent and produce the same musical result. For example, the central symmetry operation corresponds to the same musical inversion in all chromatic chord spaces. Note also that these transformations impact pitches only. However, the representation of a musical sequence in a space that does not include all the pitches (this is the case for tonal spaces),

will induce a loss of some notes, thus impacting the rhythm of the sequence.

V. Spatial Counterpoint

Counterpoint consists in the writing of musical lines that are independent from each other but sound harmonious when played simultaneously. Numerous rules have been proposed to determine a note on a line according to previously played notes on the same line and simultaneously played notes of other lines. The way these notes are chosen determine to a large extent the musical style of the piece. Different sets of counterpoint rules have been proposed over time by music theorists. One of the most popular is probably the *Gradus Ad Parnassum* from Joseph Fux [17], used for composition by, among others, Haydn, Mozart, Beethoven and Schubert. This set of rules, published in 1725, still fascinates music theorists, and has been formalized relying on various frameworks, agebraic [1] or spatial [18].

We propose a method to translate some counterpoint rules, as the ones defined in Fux's *Gradus Ad Parnassum*, in chord spaces. The goal of this study is not to propose yet another more efficient and exhaustive method for counterpoint composition, but to show how the spatial approach can be used to express existing rules and can suggest some new rules for composition.

A. Segmentation

We focus on the generation of a melodic voice, which will be added to a pre-existing musical sequence.

First we divide the sequence in successive temporal segments. For each segment s_t, a pitch p_t or a silence has to be chosen and concatenated to the generated voice. If a same pitch is generated for two successive segments, the note can be hold or repeated. We use a simple segmentation process in this preliminary study: Each time a note is played or stopped in the pre-existing sequence, the previous segment stops and a new one starts. Figure 9 illustrates this process for the generation of a voice, in parallel with three others. The set P_t includes other voice's pitches sounding during the segment s_t. In this example, 8 pitches have to be determined to complete the fourth voice of this measure.

Note that this process only allows the generated voice to move simultaneously with an other pre-existing one. More sophisticated systems would typically allow new notes to be generated between pre-existing notes. The approach described here is constrained but sufficient for this preliminary study.

B. Translation of the Rules

Counterpoint rules can generally be classified in three categories:

- Vertical (or harmonic) rules: How p_t fits with pitches in P_t;
- Horizontal (or melodic) rules: How p_t fits with p_{t-1} (and sometimes with p_{t-2} or earlier);
- Transverse rules: How $\{P_t, p_t\}$ fits with $\{P_{t-1}, p_{t-1}\}$.

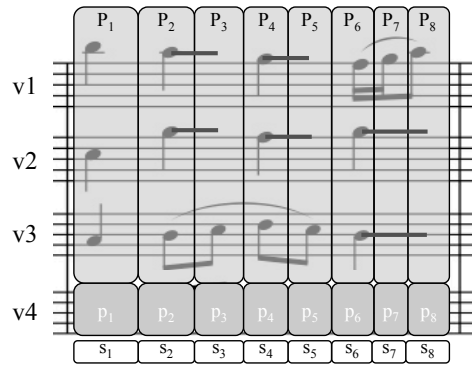


Figure 9. The generation of the voice $v4$ consists in adding to the pitch set P_t a pitch p_t (or a silence) for each segment s_t.

1) Vertical Rules: Vertical rules typically consist in promoting, avoiding or forbidding the formation of particular intervals or chords in $\{P_t, p_t\}$. We build a chord complex V corresponding to these rules by assembling simplices specifying the permitted intervals and chords. For example, a rule that allows the formation of minor and major chords is typically translated by the choice of the chord class complex $\mathcal{C}(3, 4, 5)$. If the rule forbids a particular interval, V contains no edge corresponding to this interval.

Once the complex V is assembled, we use a method presented in [3] to measure how the set of pitches defined by $\{P_t, p_t\}$ fits within this space. This method consists in measuring the compactness of the sub-complex made by the pitches of $\{P_t, p_t\}$ in V. For a given V, p_t is chosen in order to maximize this compactness. Informally, for a connected set S of simplices in a complex V, the compactness depends on the length of the paths in V between two arbitrary simplices of S.

An entire set of vertical rules rarely matches the structure of a particular complex, and a compromise needs generally to be done.

2) Horizontal Rules: Horizontal rules mostly specify allowed or forbidden intervals between p_{t-1} and p_t. Note that some complex rules can forbid some longer pitch sequences, for example p_t may depend also on p_{t-2}. In this preliminary study, we focus on rules concerning only the previous generated pitch p_{t-1}.

We build the complex H by assembling all the edges corresponding to allowed intervals. The resulting space is a one-dimensional complex, which is an undirected graph. The pitches p_t are successively determined by constructing a trajectory as connected as possible in H. Notice that a pitch transition in H is not oriented: for instance, if the notes F and G are neighbor in H, both transitions $F \to G$ and $G \to F$ are allowed.

3) Transverse Rules: A transverse rule consists in allowing or forbidding particular *n-pitch transitions*. A n-pitch transition consists in two consecutive sets of n pitches. Here is an example of a rule on 2-pitch transitions: If the pitches of two voices are separated by an interval of fifth during the segment

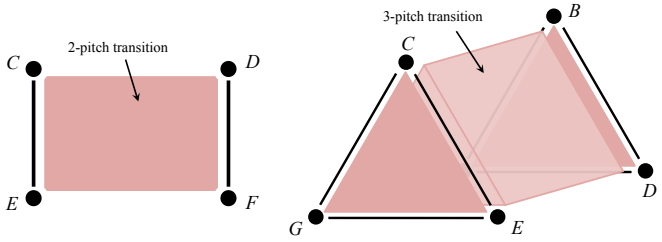


Figure 10. On the left, an allowed 2-pitch transition, between $\{C, E\}$ and $\{D, F\}$, represented by a square-shaped 2-cell. This 2-cell is not a simplex: it has four edges in its border while a 2-simplex has only three edges in its border. On the right, the 3-pitch transition between $\{C, E, G\}$ and $\{B, D, F\}$.

s_{t-1}, they cannot be separated by this same interval during s_t. This rule is related to the *parallel fifth* rule, widely used during the baroque period.

We represent an allowed n-pitch transition by a n-cell built as the *extrusion* of a $(n-1)$-simplex. Here, the extrusion is the product of an arbitrary simplex with a 1-simplex. The two sides of the extrusion correspond to the $(n-1)$-simplices respectively representing the pitch set $\{P_{t-1}, p_{t-1}\}$ and $\{P_t, p_t\}$. For example, a 2-transition is represented by a square-shaped cell (the extrusion of an edge). A 3-transition is represented by the extrusion of a triangle (figure 10). Notice that the resulting cell is not a simplicial cell, it is a a *simploid i.e.*, the product of two simplices.

As for horizontal spaces, this representation does not specify the direction of the transition. For example, the square cell on the left of figure 10 represents both transitions $\{C, E\} \to \{D, F\}$ and $\{D, F\} \to \{C, E\}$. To specify rules on directed transitions, n-cells representing n-transitions have to be oriented, in the same way that an edge (which is a 1-cell) can be oriented. An alternative approach would consist in updating the structure of H at each segment according to the played pitch set.

We build the space H by assembling the allowed n-pitch transitions. The resulting space is a *simploidal set*, a slight generalization of a simplicial complex [19].

All the notions we have presented on simplicial complexes lift immediately on simploidal sets. Thus, the pitch p_t is chosen so that the simploid spanned by $\{P_{t-1}, p_{t-1}\}$ and $\{P_t, p_t\}$ exists and maximizes the compactness in H.

4) Application: The respect of the rules by a potential pitch p_n is evaluated in each of the three spaces V, H and T. If no pitch is found, rules can be weakened by relaxing some constraint in one of the spaces, for instance by including some additional cells. An other possibility is to put a silence.

Some traditional rules cannot be easily represented solely by the structure of these complexes. However, we believe that the spatial approach can be an inspiration to propose new sets of rules for various kinds of music.

Moreover, the analyse of a set of musical pieces in a particular style can provide elements to design customized spaces to realize counterpoint in a similar style. For example, one can look for the complex in which a piece (or a set of pieces) is represented as compact as possible [3]. Using this complex for V is a good starting point to harmonize another piece in a similar style. Similar processes can be done to determine H and T.

VI. Conclusion

The starting point of this work is the abstract spatial representations of various musical objects defined in [4], [12], [3]. These representations have been unified using a simple self-assembly process, and further defined in MGS. They have already shown their usefulness, for instance for the computation of all-interval series [12].

In this paper we take a step further and we propose two spatial formulations of some non trivial compositional processes: the definition of a class of musical transformations that includes the transposition from a major to a minor mode and the spatial formulation of counterpoint rules, leading to a new algorithm to generate an additional voice.

The spatial framework works here as a powerful heuristic. In section IV we show that some straightforward spatial transformations have a well defined musical interpretation. The others, that is the straightforward spatial transformations that do not correspond to a well known chord or melodic transformation, suggest alternative musical transformations that are not easily expressed in the usual algebraic setting used in musicology. In section V, we demonstrate how counterpoint rules can be encoded on three cellular complexes that respectively represent the constraints on the notes that are played simultaneously, the possible successions of notes in a line, and the possible succession of chords in the sequence. Again, the spatial framework suggests some alternative rules, or new rule parametrization.

All the mechanisms described here have been implemented and the audio examples illustrating this work are accessible at the url http://www.lacl.fr/~lbigo/scw13 . The first results are very encouraging and open various perspectives. We mention two of them. In another direction, the research of an adapted space with a musical piece rarely accommodates with a unique complex. The comparison of how complexes fit with a piece over the time gives elements for an harmonic segmentation of the piece. A study of the successive most adapted complexes during a piece can be represented by another complex and gives interesting elements on composers practices.

The building and processing of abstract spaces appears to be a key issue for musical analysis and composition. We believe that the path taken in this paper can help to improve and to develop new tools.

Acknowledgment

The authors are very grateful to M. Andreatta, C. Agon and G. Assayag from the REPMUS team at IRCAM and to O. Michel from the LACL Lab. at University of Paris Est for endless fruitful discussions. This research is supported in part by the IRCAM and the University Paris Est-Créteil Val de Marne.

REFERENCES

[1] G. Mazzola *et al.*, *The topos of music: geometric logic of concepts, theory, and performance*. Birkhäuser, 2002.

[2] R. Cohn, "Neo-riemannian operations, parsimonious trichords, and their "tonnetz" representations," *Journal of Music Theory*, vol. 41, no. 1, pp. 1–66, 1997.

[3] L. Bigo, J. Giavitto, and A. Spicher, "Computation and visualization of musical structures in chord-based simplicial complexes," *submited to Mathematics and Computation in Music*, 2013.

[4] L. Bigo, A. Spicher, and O. Michel, "Spatial programming for music representation and analysis," in *Spatial Computing Workshop 2010*, Budapest, Sep. 2010.

[5] M. Henle, *A combinatorial introduction to topology*. New-York: Dover publications, 1994.

[6] J.-L. Giavitto and O. Michel, "MGS: a rule-based programming language for complex objects and collections," in *Electronic Notes in Theoretical Computer Science*, M. van den Brand and R. Verma, Eds., vol. 59. Elsevier Science Publishers, 2001.

[7] J.-L. Giavitto, "Topological collections, transformations and their application to the modeling and the simulation of dynamical systems," in *Rewriting Technics and Applications (RTA'03)*, ser. LNCS, vol. LNCS 2706. Valencia: Springer, Jun. 2003, pp. 208 – 233.

[8] A. Spicher, O. Michel, and J.-L. Giavitto, "Declarative mesh subdivision using topological rewriting in mgs," in *Int. Conf. on Graph Transformations (ICGT) 2010*, ser. LNCS, vol. 6372, Sep. 2010, pp. 298–313.

[9] J.-L. Giavitto and A. Spicher, *Systems Self-Assembly: multidisciplinary snapshots*. Elsevier, 2008, ch. Simulation of self-assembly processes using abstract reduction systems, pp. 199–223, doi:10.1016/S1571-0831(07)00009-3.

[10] G. Albini and S. Antonini, "Hamiltonian cycles in the topological dual of the tonnetz," in *Mathematics and Computation in Music*, ser. Communications in Computer and Information Science, E. Chew, A. Childs, and C.-H. Chuan, Eds. Springer Berlin Heidelberg, 2009, vol. 38, pp. 1–10.

[11] C. Callender, I. Quinn, and D. Tymoczko, "Generalized voice-leading spaces," *Science*, vol. 320, no. 5874, p. 346, 2008.

[12] L. Bigo, J. Giavitto, and A. Spicher, "Building topological spaces for musical objects," *Mathematics and Computation in Music*, pp. 13–28, 2011.

[13] A. Forte, *The structure of atonal music*. Yale University Press, 1977.

[14] M. Andreatta and C. Agon, "Implementing algebraic methods in openmusic." in *Proceedings of the International Computer Music Conference, Singaphore*, 2003.

[15] M. Catanzaro, "Generalized tonnetze," *Journal of Mathematics and Music*, vol. 5, no. 2, pp. 117–139, 2011.

[16] E. Gollin, "Some aspects of three-dimensional" tonnetze"," *Journal of Music Theory*, pp. 195–206, 1998.

[17] J. J. Fux and A. Mann, *The study of counterpoint from Johann Joseph Fux's Gradus ad Parnassum*. WW Norton & Company, 1965, vol. 277.

[18] D. Tymoczko, "Mazzola's model of fuxian counterpoint," *Mathematics and Computation in Music*, pp. 297–310, 2011.

[19] S. Peltier, L. Fuchs, and P. Lienhardt, "Simploidals sets: Definitions, operations and comparison with simplicial sets," *Discrete Applied Mathematics*, vol. 157, no. 3, pp. 542–557, 2009.

Population Protocols on Graphs: A Hierarchy

Olivier Bournez
LIX, Ecole Polytechnique
91128 Palaiseau Cedex, France
Email: Olivier. Bournez@lix. polytechnique. fr

Jonas Lefèvre
LIX, Ecole Polytechnique
91128 Palaiseau Cedex, France
Email: jlefevre@lix. polytechnique. fr

***Abstract*—Population protocols have been introduced as a model in which anonymous finite-state agents stably compute a predicate of the multiset of their inputs via interactions by pairs. In this paper, we consider population protocols acting on families of graphs, that is to say on particular topologies. Stably computable predicates on strings of size n correspond exactly to symmetric languages of $NSPACE(n)$, that is to say to non-deterministic space of Turing machines. Stably computable predicates on cliques correspond to semi-linear predicates, namely exactly those definable in Presburger's arithmetic. Furthermore, we exhibit a strict hierarchy in-between when considering graphs between strings and clique.**

Keywords: population protocols, computability, hierarchy, space complexity

I. Introduction

The model of population protocol has been introduced in [1] as a model of anonymous agents, with finitely many states, that interact in pairs according to some rules. Agents are assumed to be passively mobile, in the sense that there is no control over the way the interactions happen: interactions can be seen as resulting from a fair scheduler. The model has been designed to decide predicates. Given some input configuration, the agents have to decide whether this input satisfies the predicate, in which case the population of agents has to eventually stabilize to a configuration in which every agent is in accepting state. A protocol must work for any size of population. Predicates computable by classical population protocols have been characterized as being precisely the semi-linear predicates, that is those predicates on counts of input agents definable in first-order Presburger arithmetic. Semi-linearity was shown to be sufficient in [1], and necessary in [2].

So far, most of the works on population protocols has concentrated on characterizing what predicates on the input configurations can be stably computed in different variants of the models and under various assumptions. Variants of the original model considered so far include restriction to one-way communications [3], restriction to particular interaction graphs [4], and random interactions [1]. Various kinds of fault tolerance has been considered for population protocols [8], including the search for self-stabilizing solutions [5]. We refer to [6] for a comprehensive 2007 survey.

From a computability point of view, the equivalence between predicates computable by population protocols and Presburger's arithmetic definable sets (semi-linear sets) is rather intriguing. The upper bound obtained in [2] is non-trivial and rather different from classical arguments in computability or complexity theory when dealing with computational models. The work in this present paper originates from an attempt to better understand the relations between the population protocol model and classical models like Turing machines.

In that perspective we consider population protocols over various interaction graphs in the spirit of [1], [4]. This can be also seen as considering population protocols over particular spatial models.

Clearly classical population protocols corresponding to semi-linear sets correspond to the case where the interaction graph is a clique (i.e. complete graph). It has already been stated that one can simulate a Turing machine when working over a path (or more generally a bounded degree graph) in [4].

We review these constructions and we somehow extend them to more general classes of graphs/topologies.

In particular, we discuss intermediate classes, by considering interaction graphs in between bounded degree graphs and clique, namely *separable graphs.* We prove that there is a whole hierarchy between paths (non-deterministic linear time) and cliques (semi-linear sets), that can be proved to be strict for graphs verifying some conditions.

A. Related work.

The following works, not covered by 2007 survey [6] can also be mentioned: in [9] a variant of population protocols where agents have unique identifier and can store a constant number of bits and a constant number of other agents' identifiers is considered. The model has been proved to be able to compute any predicate computable by a non-deterministic Turing machine in space $n \log n$. Moreover the construction has been proved to tolerate Byzantine failures.

The idea of restricting to an underlying graph for interactions is already present in the initial definition of the model in [1]. The paper [4] is devoted to understand which properties of graphs can be computed by population protocols. It is proved for example that one can detect whether a given graph is a path or a tree using population protocols, or whether a graph has a bounded degree.

A variant of the classical model is studied in the paper [7]. This variant considers that each agent is a Turing machine with a space $f(n)$ for a population of size n. For $f = \Omega(\log)$, the computational power of the model is characterized to be precisely the class of all symmetric predicates recognizable by a non-deterministic Turing machine in space $nf(n)$. A hierarchy for population protocols is inherited from Turing machines. Furthermore it is proved that if agents have $o(\log \log)$ space then the model is no more powerful than the classical model. Our hierarchy is using both the restrictions of the communication graph and the computational power of the agents.

II. Definitions

We restate the population protocol model introduced by Angluin, Aspnes, Diamadi, Fisher and Peralta in [1] in the rest of this section.

Informally, in the basic population protocol model, we have a collection of anonymous agents. Each agent is given an input

value, then agents interact pairwise in an order fixed by some scheduler that is assumed to satisfy some fairness guarantee. Each agent is assumed to have finitely many states, and the "program" for the system describes how the states of the two agents evolve when two agents interact. The agents' output values change over time and must eventually converge to the correct output [6]. More formally:

Definition 1 (Population protocol): A *population protocol* $(Q, \Sigma, in, out, \delta)$ is formally specified by:

- Q, a finite set of *states*, that correspond to possible states for an agent;
- Σ, a finite input alphabet;
- in, an input function from Σ to Q, where $in(\sigma)$ represents the initial state of an agent whose input is σ;
- out, an output function from Q to $\{0, 1\}$, where $out(q)$ represents the output value of an agent in state q;
- $\delta \subset Q^2 \times Q^2$, a transition relation that describes how pairs of agents can interact. We will mostly write $(s_1, s_2) \to (r_1, r_2)$ for $((s_1, s_2), (r_1, r_2)) \in \delta$.

A computation of such a protocol takes place over a fixed set A of n agents, where $n \geq 2$. A population configuration (over Q) is a mapping $C : A \to Q$ specifying the state of each agent. We will say that a configuration C contains a state q if $C(a) = q$ for some agent $a \in A$. Agents with the same state are assumed in the model indistinguishable. Hence we will always reason modulo permutations of agents. In other words, configurations can also be considered as ordered multisets of states, and we assume that for all permutation π of A, configurations C and C' with $C'(a) = C(\pi(a))$ are the same.

To a word $\omega = \omega_1\omega_2 \dots \omega_n \in \Sigma^*$ of length n over input alphabet Σ, corresponds any initial configuration $C_0[\omega]$ where agents have corresponding initial state: fixing any numbering of the set A of agents, agent $1 \leq i \leq n$ is in state $in(\omega_i)$. Given some configuration C, we write $C \to C'$ if C' can be obtained from C by a single interaction of two agents: this means that C contains two states q_1 and q_2, and C' is obtained from C by replacing q_1 and q_2 by q_1' and q_2', where $((q_1, q_2), (q_1', q_2')) \in \delta$. We denote $\to^*$ for the transitive and reflexive closure of relation $\to$.

An *execution* over word $\omega = \omega_1\omega_2 \dots \omega_n \in \Sigma^*$ is a finite or infinite sequence of configurations $C_0C_1 \dots C_iC_{i+1} \dots$ such that $C_0 = C_0[\omega]$ and, for all i, $C_i \to C_{i+1}$.

The order in which pairs of agents interact is assumed to be unpredictable, and can be seen as being chosen by an adversary. In order for only meaningful computations to take place, some restriction on the adversarial scheduler must be made (otherwise, nothing would forbid for example that all interactions happen always between two same agents). As often in the distributed computing context, this is model by some fairness hypotheses: a *fair execution* is an execution such that if a configuration C appears infinitely often in the execution and $C \to C'$ for some configuration C'. Therefore C' must appears infinitely often in the execution. This fairness property states that a configuration which is always reachable will eventually be reached. There is absolutely no control over any finite execution, but any possibility that can not be evade with a finite execution will be met. Observe that this is equivalent to say that if a configuration C' is reachable infinitely often, then it is infinitely reached.

At any step during an execution, each agent' state determines its output at that time: whenever an agent is in state q, its output is $out(q)$. A configuration C is said to output 1 (respectively 0) iff all agents outputs 1 (resp. 0): in other words $out(C(a)) = 1$ for all agent $a \in A$ (resp. 0). If in a configuration there are agents outputting 1 and other outputting 0, the output of the configuration is undefined. A configuration C is said to be *output-stable* with output $b \in \{0, 1\}$ if $\forall C', C \to^* C', \forall s \in C', out(s) = b$. Namely a configuration is output-stable if its output is defined and any reachable configuration has the same output.

A fair execution $C_0C_1 \dots C_iC_{i+1} \dots$ stabilizes with output b if there exists some m such that for all $j \geq m$, the output of configuration C_j is defined and given by b. In particular, a fair execution converges with output b iff it reaches an output-stable configuration with output b.

A protocol is well-specified if for any initial configuration $C_0[w]$, every fair execution starting from $C_0[w]$ converges with an output, that is determined by that input configuration. A well specified protocol induces a subset $L \subset \Sigma^*$ of configurations yielding to output 1: this subset will be said to be the language decided by the protocol.

We will also need to consider population protocols over graphs. Observe that the previous model is just the case where the graph is a clique.

Definition 2 (Population protocol on a graph): Given a graph G with n vertices, a population protocol over G is a population protocol whose set of agents A is the set of vertices of G, and where in definition above two agents can interact by a single interaction only if they are adjacent in graph G: in definitions above, this only changes the definition of $C \to C'$. Formally: $C \to C'$ if C contains two states q_1 and q_2 in respective vertices v_1 and v_2, and C' is obtained from C by replacing q_1 and q_2 by q_1' and q_2', where $((q_1, q_2), (q_1', q_2')) \in \delta$, and (v_1, v_2) is an edge of graph G.

We insist on the fact that we do not change anything else. In particular the input is still distributed on the population without any control.

Given a family of graphs $(G(n) = (V(n), E(n)))_{n\in\mathbb{N}}$, where $G(n)$ has $n = |V(n)|$ vertices, a well-specified protocol also induces a subset $L \subset \Sigma^*$: this subset will also be said to be the language decided by the protocol over this family of graphs.

Equivalently a language $L \subset \Sigma^*$ is decided by population protocol working over this family if there is a well-specified protocol verifying the following property: the $w \in L$ correspond exactly to initial configurations $C_0[w]$ for which every fair execution of the protocol on $G(n) = (V(n), E(n))$ with $|V(n)| = n$, where n is the length of w, eventually stabilize to output 1. From our definitions, languages corresponding to population protocols are necessarily invariant by permutations of letters, as initial configurations are assumed to be indistinguishable when agents are permuted.

Population protocols on bounded degree graphs can be related to Turing machines. The languages decided by population protocols on bounded degree graphs correspond exactly to symmetric languages of $NSPACE(n)$, that is to say to language recognized in non-deterministic space n by Turing machines, invariant by permutation of letters.

In the spirit of [7], we will also consider variants where the agents are no longer just finite automata but deterministic Turing machines with memory space constraints: this model will be called the *passively mobile machines model*, following the terminology of [7]: the agents are now Turing machines.

Formally, they are Turing machines with four tapes: working tape, output tape, incoming message tape and outgoing message tape. They have internal transitions, corresponding to local computations, and external transitions, corresponding to interactions, where the two agents copy in their own incoming message tape the contents of the outgoing message tape of the other agent.

The authors in [7] show that if the n agents of the population have a memory of size at least $\log(n)$ then they can be organized to model a Turing machine. Basically, the idea is that agents can use interactions to compute eventually unique identifiers, and then use these identifiers so that each agent then simulate a cell (or few cells) of the tape of a (global) Turing machine.

Here will be discussed smaller cases. Let us first come back to classical population protocols where agents are finite state.

We will often use variations of the following routine electing a leader in a uniform population: consider set of states is $Q_{leader} = \{L, x\}$, where state L means that this agent has a leader token and rules $\delta_{leader} = \{(L, L) \to (L, x), (x, L) \to (L, x)\}$. Thanks to the first rule, if two agents with leader token meet, one token is deleted. With this rule, the quantity of leader token will decrease to one, but the last token can never be deleted. The second rule allows any leader token to move on the graph. The computation starts with every agent with state L: in any fair execution, eventually there will remain exactly one L state in the population. Indeed, since leaders can move they can always meet, and hence the fairness hypothesis guarantees that leaders will eventually meet and disappear until only one remains.

III. Modeling a Turing Machine on graphs

The paper [4] explains how to model a Turing machine with a population protocol working on a graphs family with degree bounded by d. We will explain the main steps of the construction.

The algorithm of [4] can be decomposed into three subalgorithms: construct a distance-2-coloring of the graph ; use this labeling to build a spanning tree on the graph ; and use this spanning tree to model the tape of a Turing machine.

A distance-2-coloring of a graph G is a coloring of the vertices of G such that two vertices that are adjacent or have a common neighbor receive distinct colors.

A graph family will be said to be distance-2-colorable if there exists a population protocol such that starting from any initial configuration the system eventually evolves (restricting as usual to fair computations only) to a configuration that corresponds to a distance-2-coloring.

Proposition 1: A graph family is distance-2-colorable iff the graph family has a bounded degree: there exists some constant d that bounds the degree of each graph of the family.

Proof: The idea of the proof of the if part of the proposition is to use a moving leader token. After a move the token stops and interacts successively with two of its neighbors. If they have the same color, one changes, and the token moves away. Thanks to the fairness property, the population will eventually reach a configuration where no two neighbors of any agent have the same color. If two moving leader tokens meet, one (and only one) is destroyed. A more detailed proof of this can be found in [4].

Conversely, since agents are finite state, the degree is directly bounded by the number of colors (i.e. states) minus one in a distance-2-coloring. ■

Notice that some more general families of graphs could also be distance-2-colorable, if we consider that agents would not be finite state, that is to say passively mobile machines instead of population protocols.

Proposition 2 ([4]): Consider a distance-2-colorable graph family. One can build a population protocol that builds a spanning tree over this graph family.

Proof: The idea of the proof is to observe that a distance-2-coloring of the graph allows one to emulate pointers. Indeed, each agent can store in his state a color c_0 and by definition of the coloring, it can have at most one neighbor with color c_0; in other words the color of an agent is like a unique pointer address for its neighbors. This allows one to build a forest where the parent links in the trees are the pointers and the forest can eventually be transformed into a spanning tree using leaders and the fairness of the scheduler.

The construction itself uses leader tokens. The leader token starts its construction on a vertex that will be the root. Then it explores the graph in depth-first fashion. While it finds a neighbor not yet in the tree, the token moves to this agent, and makes this agent point to the one it comes from. If no new son can be added to the tree, the leader token backtracks until it comes to an agent having neighbor not yet in the tree. If a leader encounters an other leader, one is erased and restart markers are produced to erase the trees already built. The construction then starts again with one less leader. It will eventually succeed when the number of leaders will reach one (by the fairness assumption). The coloring and the spanning tree construction being simultaneous, if an agent part of a tree is recolored, a *restart* marker is produced.

Eventually, the coloring will be stable, only one leader token will make the depth-first search, and a spanning tree will be built on the graph. ■

The class $ZPSPACE(s)$ corresponds to language accepted by (so-called) Zero-error Probabilistic Turing machines using a memory space bounded by s.

Equivalently, (and we will use this definition), it can be shown to the class of languages accepted by a non-deterministic Turing machine, with three special internal states *Yes*, *No* and $Don'tKnow$ verifying the following property: if $\omega \in L$ (resp. $\omega \notin L$) then a possible execution of the machine on ω can only output *Yes* or $Don'tKnow$ (resp. *No* or $Don'tKnow$). And for every input, there are executions outputting a firm answer (*Yes* or *No*). The construction of this machine is described in [10]. The class $ZPSPACE_{sym}(s)$ is the restriction of the symmetric languages, that is to say the language L such that for all permutation π, $\omega \in L$ iff $\omega' \in L$ where $\omega'[i] = \omega[\pi(i)]$.

Proposition 3: Let (G_n) be a graph family. If there is a population protocol on (G_n) that organizes the agents into a spanning tree, then there is a population protocol that model a Turing machine tape with a tape of size n on G_n.

Proof: Once a spanning tree is built, the spanning tree can be used as the tape of the Turing machine: we can order the vertices of the graph using the spanning tree (using a breadth-first search for instance). The numeration gives the location of the agent in the tape.

Let $L \in ZPSPACE_{sym}(n)$. Using the tools from above, we can model the behavior of a Turing machine computing L. For any input x, the computation outputs either a firm answer, either $Don'tKnow$. Whenever the protocol ends a sim-

ulation, it starts a new one. The first simulations of the Turing machine may be made when the routine that organize the population into a tape is not finished. Those simulations gives answers with no warranty. The fairness makes certain though that the population will eventually be organized, after what exact simulation can occur. The looping simulation ensures that configurations corresponding to exact simulation of the Turing machine are always reachable. Then fairness guarantees every correct simulation will occur during any execution of the population protocol. By fairness, some execution giving the firm answer will occur. And from this instant the simulation will only answer either the correct and firm answer, either $Don't Know$. By outputting the last firm answers computed by a simulation, the protocol can decide "$x \in L$?". ■

IV. Separable graph family

Definition 3: A graph family $\big(G(n) = (V(n), E(n))\big)$ is s, d-*separable* if for all n, the set of vertices $V(n)$ can be partitioned into U_1 and U_2 such that:

- $(U_1, E(n)_{|U_1})$ is a connected graph with $s(n)$ vertices
- for all vertex $u \in U_1$, its degree (in $G(n)$) is bounded by $d(n)$
- for all vertex $v \in U_2$, v belongs to a clique of size at least $2^{d(n)+1}$.

U_2 may be empty.

Notice that there are such separable graph families only if $d(n) \leq \log n$ and $s(n) \leq n$.

For instance with $d(n) = 2$, a path of size $s(n)$ connected (in one or two vertices) to a clique of size at least 8 provides a $s, 2$-separable family of graphs. The figure 1 shows a possible graph of a $s, 2$-separable graph family. In the figure 2 a graph of a $s, 4$-separable graph family is shown.

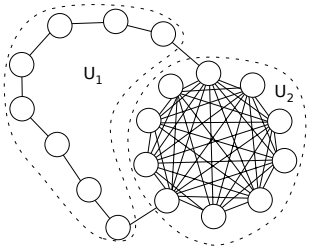


Fig. 1. A graph of a $s, 2$-separable family

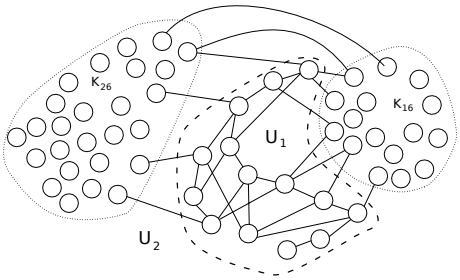


Fig. 2. A graph of a $s, 4$-separable family

We will construct a simulation of Turing machine on separable graphs. The first step is to recognize the clique part of the graph.

Lemma 4.1: Let be $G = (V, E)$ a s, d-separable graph with $d = O(1)$. There is a population protocol *Clique* that distinguishes U_1 from U_2.

That means its set of states can be partitioned in Q_{clique} and $Q_{bounded}$, and for any fair execution (C_t) of the protocol *Clique*, there is t_0 such that in configurations C_t with $t \geq t_0$, an agent has a state in Q_{clique} iff it belongs to U_2.

Proof: The construction of this protocol uses two routines.

Color, the first routine, colors the agent with $d+1$ colors in such a way that an agent has every colors in its neighborhood if it belongs to a clique of size at least 2^{d+1}: At the beginning of this routine, every agent is candidate for the color 1. If two candidate for the color i interact, one is definitely colored with color i while the other is candidate for the color $i+1$. The color $d+1$ is an exception since the candidates are automatically colored. This way, half the agents of a clique take the color 1, half the remaining agents take the color 2, and so on. At the end a proportion $\frac{1}{2^d}$ of the agents of a clique will have the color d, and the same quantity will have the color $d+1$.

Count, the second routine, counts (up to $d+1$) how many different colors there are in the neighborhood of any agent, and therefore decides if the agent is in the bounded degree sub-graph (where an agent can not have $d+1$ different colors in its neighborhood) or in the clique (where the agents have more than $d+1$ colors in their neighborhood). ■

V. Modeling a Turing machine on separable graph family

A. Counting protocol

To achieve the simulation we need a protocol counting the input letters, the objective is to have the input into a format the Turing machine can use. The idea is to use the bounded-degree sub-graph as a tape to store the entry in a more efficient form. Indeed, the protocol input is symmetric and can be describe by the number of symbol of each type without losing any information. With a tape, we can write those numbers in binary because the agents are implicitly ordered. Of course, this is possible only if this "tape" is big-enough with respect to the total number of agents.

Lemma 5.1: Let be G a s, d-separable graph with $s = \Omega(\log)$. There is a population protocol *WriteInput* which organizes the bounded degree part as a Turing machine tape and which writes (using binary encoding) the initial input multiset on this tape.

Proof: The protocol *WriteInput* uses different routines.

First, *Clique* (from lemma 4.1) recognizes the bounded degree sub-graph. Then the routines described in the propositions 1 and 2 organize the bounded degree sub-graph.

The next routine has two phases:

1) Writing on the tape, it counts the input letter and mark them (so they will not be counted twice),
2) It saves the value, if greater than the last saved value, and send *Erase* token in the population to erase the marks. After the tokens are back, it starts again the counting phase.

This routine will never count too many of any kind of letter, but it can count every one of a kind. Eventually they will count every states. And the multiset corresponding of the input will be written (in binary notation) on the tape. This can be done only if the tape has enough cells, that is to say if there is at least $\Omega(\log(n))$ cells. ■

B. Main result of this section

Let $L(s)$ be the class of languages decided by population protocols on s, d-separable graph family with $d = O(1)$.

If $s = \Omega(\log)$, using the routine *WriteInput* to prepare the population, we can simulate a Turing machine on the bounded degree part. The Turing machine we model uses at most $s(n)$ cells on its working tape and the result of any computation does not change if we apply any permutation on the input. In fact, we prove that we model any Turing machine that computes a language of the class $NSPACE_{sym}(s)$ using same principles.

Theorem 5.1: $\forall s = \Omega(\log), L(s) = NSPACE_{sym}(s)$

Proof: Fix some $s(n) = \Omega(\log n)$ and $d = O(1)$.

First we prove $L(s) \subseteq NSPACE_{sym}(s)$.

A configuration of a population of size n on a s, d-separable graph needs $O(s(n) + \log(n - s(n))$ cells on a tape to be stored. Since $s(n) = \Omega(\log(n))$, we only need $O(s(n))$ cells. To compute the result of a computation by a population protocol we need to find a reachable configuration such that we cannot find any other reachable configuration where the output is different or undefined. Then it is computable by a Turing machine of $NSPACE(s)$, since $NSPACE(s) = coNSPACE(s)$ for $s = \Omega(\log)$ (see e.g. [10]). And the languages of $L(s)$ are symmetric then $L(s) \subseteq NSPACE_{sym}(s)$.

We prove now $ZPSPACE_{sym}(s) \subseteq L(s)$.

Let M be a machine corresponding to a language of $ZPSPACE_{sym}(s)$. We can construct a population protocol of $L(s)$ computing the same language. As direct application of the lemma 5.1, we are able to model a Turing machine using a tape with $s(n)$ cells. The proof of the proposition 3 can be used to prove that the simulated Turing machines can compute languages of the class $ZPSPACE_{sym}(s)$. Therefore $ZPSPACE_{sym}(s) \subseteq L(s)$

As $s = \Omega(\log)$, it is known (see e.g. [10]) that we have $NSPACE_{sym}(s) = ZPSPACE_{sym}(s)$.

To conclude:

$$L(s) = NSPACE_{sym}(s) = ZPSPACE_{sym}(s)$$

■

C. Passively mobile machines model

Let s, s' functions such that $s(n) = O(n)$. We will consider the passively mobile machines model (i.e. population protocol where the agents are Turing machines) with memory of the agents bounded by s'.

In [7], a detailed proof of the following result can be found.

Proposition 4: Let $s' = \Omega(\log)$. The set of languages decided by the passively mobile machines model with memory of the agents bounded by s' is exactly $NSPACE_{sym}(n.s'(n))$.

In other words, if the agents have at least a logarithmic memory space, then whatever the interaction graph is, the model is equivalent to a Turing machine with a memory space equal to all the possible available space. If the agents have less memory space, considering the interaction graph may permit to gain some computational power. So we will now only consider cases were $s' = o(\log)$.

Let $L_{s'}(s)$ be the class of languages decided by the passively mobile machines model with memory of the agents bounded by s' on s, d-separable graph family with $d(n) = 2^{O(s'(n))}$. We remind that $d(n) \leq \log n$ in any case.

The results of the previous sections can be extended to the population protocol model where the agents are not finite states, but Turing machines with memory bounded by s'. Then the agent can have up to $2^{O(s'(n))}$ different "states" (in fact they are configurations). So we can use those extra states to construct distance-2-coloring with more colors, and hence we may be able to construct a spanning tree on s, d-separable graphs for non constant d. More precisely we have the following result that extends Lemma 4.1.

Lemma 5.2: Let be $G = (V, E)$ a s, d-separable graph. There is a passively mobile machines model with memory bounded by $\log(d(n))$ that distinguish U_1 from U_2.

It gives a way to model a Turing machine on the bounded degree sub-graph. The modeled Turing machine has a tape modeled on $s(n)$ agents, each one having $s'(n)$ cells ; then the total space the Turing machine have is $s(n).s'(n)$ (each of the $s(n)$ agents contribute with $s'(n)$ cells). And then the generalisation of the proposition 5.1 is the following.

Theorem 5.2: Let d, s and s' such that $d = 2^{O(s'(n))}$, $d(n) = O(\log n)$, $s' = o(\log)$ and $s(n) \leq n$. The class of the language computable by passively mobile machines of $DSPACE(s')$ on s, d-separable graph family is $NSPACE_{sym}(s.s')$.

$$L_{s'}(s) = NSPACE_{sym}(s.s')$$

D. A Hierarchy

With Theorems 5.1 and 5.2, we can transpose the hierarchy of non-deterministic Turing machines above the logarithmic space.

Theorem 5.3: Let s_1, s_2, s'_1, s'_2 such that $s_i(n) = O(n)$ and $s'_i = O(\log\log)$ for $i = 1, 2$. If $s'_1.s_1 = o(s'_2.s_2)$ then $L_{s'_1}(s_1) \subseteq L_{s'_2}(s_2)$. And the inequality is strict if $s'_2.s_2 = \Omega(\log)$.

For instance, let be $s' = o(\log)$ and consider the sequence of function $\log^k n/s'(n)$. If $k \geq 1$, we have $L_{s'}(\log^k n/s'(n)) = NSPACE_{sym}(\log^k)$. Therefore, we have the following hierarchy with:

$$\begin{aligned}
L_1(1) &\subsetneq L_{s'}(1) \\
&\subsetneq L_{s'}(\log n/s'(n)) \\
&\subsetneq L_{s'}(\log^2 n/s'(n)) \\
&\vdots \\
&\subsetneq L_{s'}(\log^k n/s'(n)) \\
&\vdots \\
&\subsetneq L_{s'}(n) = NSPACE_{sym}(n.s'(n))
\end{aligned}$$

$L_1(1)$ corresponds to what is decided by the classical model with finite automaton agents and the complete interaction graph: $L_1(1)$ is the class of the semi-linear languages. $L_{s'}(1)$ corresponds to the passively mobile machines model where agents use $O(s')$ memory and the complete interaction graph. $L_{s'}(n)$ corresponds to the passively mobile machines model where agents use $O(s')$ memory and the bounded degree interaction graph. The proof of the strict inequality $L_1(1) \subsetneq L_{s'}(1)$ is detailed in [7].

Note that in [7], a hierarchy is built by using only the memory available on the agents. Their hierarchy can not be exactly characterized if $s' = o(\log)$ and collapsed if $s' = o(\log\log)$. Using restrictions on the interaction graph, we are able to construct a more accurate hierarchy.

VI. Conclusion

In this paper we determine the computational power of a model of population protocol working on different kind of interaction graph families and with different kind of restrictions on the computational power of the agent. The classical population protocols are the case where the interaction graph is complete and where the agents are finite automata. The case where the interaction graph is a uniformly bounded degree graph and the agents are finite automata has the same computational power as linear space non deterministic Turing machines. With intermediate interaction graphs and the agents having more memory allowed, the model is equivalent to non-deterministic Turing machine working with some bounded memory. We have constructed a Turing machine simulation with as much memory as the sum of all the memory of the agents of a "good" sub-graph. By varying the "good" sub-graph and the bound over the memory of the agents, we disserted a hierarchy. This hierarchy is inherited from the one over the non-deterministic Turing machines.

References

[1] D. Angluin, J. Aspnes, Z. Diamadi, M.J. Fischer, and R. Peralta. Computation in networks of passively mobile finite-state sensors. In *Proceedings of the twenty-third annual ACM symposium on Principles of distributed computing*, pages 290–299. ACM, 2004.

[2] D. Angluin, J. Aspnes, Z. Diamadi, M.J. Fischer, and R. Peralta. Computation in networks of passively mobile finite-state sensors. *Distributed Computing*, 18(4):235–253, 2006.

[3] D. Angluin, J. Aspnes, D. Eisenstat, and E. Ruppert. The computational power of population protocols. *Distributed Computing*, 20(4):279–304, 2007.

[4] Dana Angluin, James Aspnes, Melody Chan, Michael J. Fischer, Hong Jiang, and René Peralta. Stably computable properties of network graphs. In Viktor K. Prasanna, Sitharama Iyengar, Paul Spirakis, and Matt Welsh, editors, *Distributed Computing in Sensor Systems: First IEEE International Conference, DCOSS 2005, Marina del Rey, CA, USE, June/July, 2005, Proceedings*, volume 3560 of *Lecture Notes in Computer Science*, pages 63–74. Springer-Verlag, June 2005.

[5] Dana Angluin, James Aspnes, Michael J. Fischer, and Hong Jiang. Self-stabilizing population protocols. volume 3, page 13, November 2008.

[6] James Aspnes and Eric Ruppert. An introduction to population protocols. In *Bulletin of the EATCS*, volume 93, pages 106–125, 2007.

[7] I. Chatzigiannakis, O. Michail, S. Nikolaou, A. Pavlogiannis, and P.G. Spirakis. Passively mobile communicating machines that use restricted space. *Theoretical Computer Science*, 2011.

[8] Carole Delporte-Gallet, Hugues Fauconnier, Rachid Guerraoui, and Eric Ruppert. When birds die: Making population protocols fault-tolerant. In Phillip B. Gibbons, Tarek F. Abdelzaher, James Aspnes, and Ramesh Rao, editors, *Distributed Computing in Sensor Systems, Second IEEE International Conference, DCOSS 2006, San Francisco, CA, USA, June 18-20, 2006, Proceedings*, volume 4026 of *Lecture Notes in Computer Science*, pages 51–66. Springer, 2006.

[9] Rachid Guerraoui and Eric Ruppert. Names trump malice: Tiny mobile agents can tolerate byzantine failures. pages 484–495, 2009.

[10] M. Saks. Randomization and derandomization in space-bounded computation. In *Computational Complexity, 1996. Proceedings., Eleventh Annual IEEE Conference on*, pages 128–149. IEEE, 1996.

AREA: an Automatic Runtime Evolutionary Adaptation mechanism for Creating Self-Adaptation Algorithms in Wireless Networks

Qingzhi Liu[1,2], *Stefan Dulman*[1], *Martijn Warnier*[2]
1. Embedded Software, EEMCS Faculty, Delft University of Technology, The Netherlands
2. System Engineering, TPM Faculty, Delft University of Technology, The Netherlands
{q.liu-1, s.o.dulman, m.e.warnier}@tudelft.nl

***Abstract*—The application requirements and the spatial environments of wireless networks continue to become more and more complex and changeable. Most existing algorithms for wireless sensor networks are designed with a specific type of environment in mind. While such algorithms work well in the environment they have been designed for, once the environment changes beyond the domain in which the design can adapt, the algorithms can hardly work properly. This paper proposes a novel design mechanism called the *automatic runtime evolutionary adaptation (AREA)* mechanism. It has been designed to automatically adapt, during runtime, to the variation of environments in wireless networks. This adaption is realized by self-creating and self-evolving algorithms: *AREA* allows the created algorithm not only to be adaptive but also to evolve to other adaptive abilities according to the variation of the application requirement and the spatial environment. The *AREA* mechanism is validated by applying it to a data aggregation example in wireless networks. This shows that the mechanism can adapt to the changing environments and outperform the other strategies.**

I. Introduction

Nowadays an increasing amount of research is focused on various wireless network applications, such as environment monitoring [1], traffic control [2] or navigation localization [3]. As the application complexity increases, more new research requirements are involved. However, most existing algorithms are designed with a specific type of (spatial) environment in mind, such as assumed bandwidth, node density, etc. Once the environment changes beyond the presumed domain, the algorithm will no longer be able to adapt: it will not function properly anymore. Therefore, it is necessary to design a mechanism that allows wireless networks to automatically self-create and self-evolve algorithms according to the changes in the (spatial) environment.

Based on this motivation, we propose a novel algorithm design mechanism called an *Automatic Runtime Evolutionary Adaptation (AREA)* mechanism. The *AREA* mechanism has three main properties, including automatic computing, runtime processing, and evolutionary adaptation.

The self-adaptation property has been widely recognized as an important performance metric for wireless networks. However, existing self-adaptive algorithms, based on design mechanisms such as swarm intelligence [4], stigmergy [5], and autopoiesis [6] only maintain the adaptation properties for specific environments. Once the deployment environment changes beyond the scope of the originally envisioned domain, the algorithm performance will decrease. The *AREA* mechanism allows the created algorithm not only to be adaptive but also to evolve to other adaptive abilities based on the variation of the application requirements and the spatial environment. In addition, *AREA* is totally distributed. Each agent only spatially coordinates with neighbors, uses runtime local information, and the whole processing flow is automatically executed in each agent during the runtime.

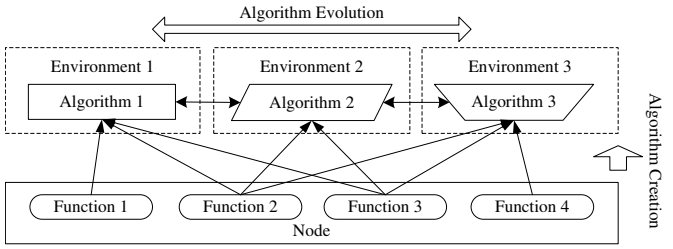


Fig. 1. *AREA* lets each agent (node) self-create algorithms that adapt to different application requirements. The created algorithms self-evolve to other function combinations, self-adapting to their (changing) environment.

The *AREA* mechanism assumes that each agent has some basic functions, such as routing, forwarding messages, etc. Each agent mutates local function combinations and learns from neighbors. Finally, the function combination that meets the environment requirements emerges and spreads throughout the network. If the environment or the agent function parameters change, and the selected function combination no longer meets the application requirements, agents evolve the algorithm and converge to new function combinations. In this way, agents always use the function combinations that suit the application requirements and the spatial environment. We also present a stabilization algorithm that reduces the churning phenomenon in different function combinations while maintains fast convergence of function selection.

We validate the *AREA* mechanism by applying it to the simulation of a data aggregation example. In the simulation example, each agent is supposed to have four basic functions: forwarding messages, routing messages, joining into clusters, and increasing the transmission range. The application requirement for every agent is to maintain the message arriving rate above a predefined threshold. By changing the agent density and transmission bandwidth parameters, agents in the network self-create and self-evolve various function combinations according to the spatial distribution. For the implementation of each component in *AREA*, we in detail illustrate how to select the parameters of fitness functions, etc.

based on the requirement. We use three other algorithms with fixed strategies for comparison. According to the test results, the *AREA* mechanism always maintains the best performance even when the environment and agent parameters change.

The remainder of the paper is organized as follows. Section II overviews the *AREA* mechanism and the application example. Section III illustrates *AREA*'s components in detail and the example implementation. We present the simulation results and evaluations in Section IV, related work follows in Section V and the paper ends with conclusions.

II. The *AREA* Mechanism Overview

In this section, we present the design framework of the *AREA* mechanism, the working components and the processing flow. And we demonstrate the general implementation of the data aggregation example based on *AREA*.

A. Mechanism Framework

It is supposed that each agent has some basic functions in the function set, such as forwarding messages, routing, etc. The agents are given an application requirement, such as "the message arriving rate should be larger than a predefined value". The agents self-create algorithms by selecting suitable combinations from predefined basic functions. When the environment changes, the agents self-evolve to select a different function combination as the new algorithm. In the system, each agent is independent and distributed, and only accesses the information of neighboring agents. The working process is automatic and during runtime. Figure 1 demonstrates an example of the *AREA* design framework. In the environment 1, the agent (node) converges to select the combination of function 1, 2 and 3 as the algorithm to fulfill the requirement. As the environment changes from environment 1 to 2, the originally created algorithm cannot meet the application requirement any longer. The agent evolves the algorithm and converge to select the combination of function 2 and 3. No matter how the environment changes, the agent always evolves to the function combination that will fulfill the predefined application requirements. After these requirements are fulfilled, the algorithm selected by the agents converges to a spatially stable state.

The *AREA* mechanism has four working components as shown in Figure 2(a). The first component is the definition of the *basic functions* of the agents in the function set. All predefined basic functions should work independently of each other. The second component is *function mutation*. Each basic function has a mutation probability that can change between being used and unused. So each agent can use different function combinations. The third component is *environment selection*. The agents in the network need to meet application requirements, such as maximizing the message arriving rate, minimizing the power consumption, etc. Each agent calculates a fitness credit for every function in the environment, and learns the function usage rule (using or unusing) of the agent with the largest function fitness credit in the neighboring

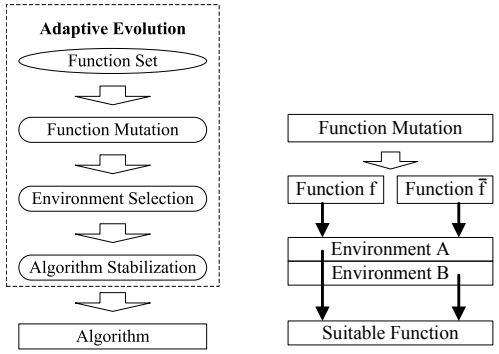


Fig. 2. (a).The working flow components of the *AREA* mechanism combine the most suitable functions into an algorithm. (b).The function mutation component creates different function usage rules (f: use the function; $\overline{f}$: unuse the function), and the rule that best suits the environment (A or B) is selected as the suitable function.

agents. This component allows the most suitable function combinations to be survived and diffused in the spatial network. To make the algorithm usable in practice, the algorithm needs to converge once the most suitable function combination is found, For this, the *algorithm stabilization* component is used. This component decreases the churn of selecting different function combinations in the network by each agent. The four components outlined above, make it possible for the agents in the network to select a suitable and stabilized function combination as the new algorithm. If the environment changes and the created algorithm can no longer fulfill the application requirements, the *AREA* mechanism will make the existing algorithm evolve to another function combination in order to meet the new application requirements.

B. Application Example

The *AREA* mechanism is validated by implementing it in an application example: data aggregation. Data aggregation is a basic building block for spatial network application. Most of the existing data aggregation algorithms are designed for specific deployment environments. In deployment scenarios where the environment may change sometimes, these algorithms can no longer work properly. We use the *AREA* mechanism to implement data aggregation in changing environments.

Suppose agents are randomly scattered in an area. Each agent can only communicate with neighbors. A sink agent is predefined to aggregate messages from other agents. If new agents come into the network, the agent density increases. If existing agents leave the network, the agent density decreases. We define the bandwidth as the number of the messages that can be forwarded by a agent at one tick. The agent density and bandwidth are changeable in the environment. We suppose that each agent knows the number of messages that are sent out and arrive at the destination. The data aggregation implementation based on *AREA* allows the agents in the network to automatically and during runtime find suitable function combination that meet its application requirements. The created function combination for data aggregation maintains a high message arriving rate and low power consumption values.

In the simulation, the environment is changed by adapting the agent density and bandwidth. The different simulation scenarios are outlined in Figure 3. At the start of the experiment the agents are randomly deployed. Under standard agent density and bandwidth, agents self-create function combinations that use forwarding and routing functions, as shown in Figure 3(a). Then we change the environment by increasing the agent density. Because the bandwidth is limited, some messages are now dropped. So the agents start to evolve the existing data aggregation approach, and spatially self-organize to clusters to increase the message arriving rate in the network as shown in Figure 3(b). In case the agent density decreases, and the network becomes disconnected the message arriving rates of the agents that are not connected to the destination agent become 0. When this happens, some of the agents evolve their algorithm and increase the transmission range to connect the network as shown in Figure 3(c). The increased transmission range of the agents will increase their message arriving rates.

III. The *AREA* Components

Based on the overall design presented in Section II, in the next subsections, we first define the content and structure of each component from *AREA*, and then present the detailed component implementation for the data aggregation example.

A. The Function Set

Each agent has some basic functions in a function set. These basic functions, such as joining into a cluster, are predefined, For each agent, all the possible function combinations form the search space of the available algorithms. For the data aggregation application, we use four basic functions in the function set for each agent: forwarding messages, routing the messages via the shortest path, joining to a cluster with other neighbors and increasing the transmission range.

Firstly, each agent has the basic function that it can forward messages to a neighboring agent. The bandwidth of forwarding messages is limited. It is supposed that the agent can only forward a limited number of messages during one time slot. If the number of messages to be forwarded is larger than the bandwidth, the agent drops the excess messages.

For the second function it is assumed that each agent can calculate the next hop on the shortest path based on the gradient [7]. If the agent knows the next hop on the shortest path, the agent forwards the messages to the next hop. Otherwise, the agent forwards the messages to a random neighboring agent. We assume that if the message is forwarded to a hop that is further away from the destination agent, then next hop agent drops the message.

Agents also have the function to join in a cluster. In a cluster, the cluster leader represents all the members to send out messages. Because the bandwidth is limited, we suppose that the forwarding priority of the message from a cluster is higher than the message from a single agent. And it is assumed that the forwarding priority of the message from a cluster with larger number of member agents is higher than the message from a cluster with smaller number of member agents. Agents forward messages from higher to lower priority.

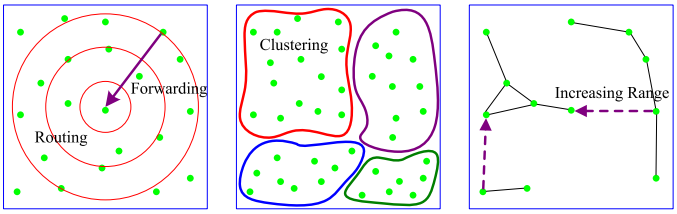


Fig. 3. The application example of *AREA* for data aggregation. (a). Agents evolve to use forwarding and routing functions. (b). Agents form clusters to send out messages. (c). Agents unconnected to the destination agent increase their transmission range.

Finally, agents have the basic function to increase their transmission range. Each agent has a default and an increased transmission range. If the agent cannot connect to the network with the destination agent using the default transmission range, then it can reconfigure (mutate) to use the increased transmission range.

B. Function Mutation

In this section, we present how the function combinations are created for each agent. We define the function combination series $\{f_1,f_2,...,f_i,...,f_n\}$. Where n is the total number of functions. f_i represents the basic function i explained in Section III-A. The value of f_i can be 0 or 1. $f_i = 0$ means that the function i is not used in the function combination, and $f_i = 1$ means that the function i is used. All the basic functions together form a function combination series. In the initial state, agents select a random value (0 or 1) for f_i $(i = 1, 2, ..., n)$ of each basic function. In the function mutation component, *AREA* makes each agent change the function value f_i by setting $f_i = \overline{f_i}$ with a mutation probability.

In the application example, agent x has a $Switch_{xi}$ value 0 or 1 for each function i representing unused or used. We predefine a mutation probability P_m for all the basic functions. If a random value is smaller than P_m, then $Switch_{xi}$ does not change. Otherwise, $Switch_{xi}$ is changed to $\overline{Switch_{xi}}$.

C. Environment Learning

In this section, we demonstrate how agents learn from each other, and make the most suitable function combination spread throughout the network. We define the fitness credit of the function for each agent by $C_i = w_{i1}c_{i1} + w_{i2}c_{i2} + \cdots + w_{ik}c_{ik} + \cdots + w_{im}c_{im}$. Where i is the function number as defined in Section III-B. k $(k = 1, 2, ..., m)$ is the number of the evaluation criteria for the function, and m is the total number of evaluation criteria. For example, the performance of the network can be determined by the arriving rate, power consumption, etc. c_{ik} is the fitness credit of the function i evaluated by the criteria k. w_{ik} is the weight of the fitness credit with evaluation criteria k. In the learning component, each agent records the fitness credit of each function. At the same time, each agent detects the fitness values of the functions from neighboring agents and learns the function

usage strategy (used or unused) from the agent with the largest fitness credit. Finally, the function combination that has the largest fitness credit in the network survives and spreads. The mutation and learning process is shown in Figure 2(b).

For the application example, we define the fitness credit for every function as follows. The fitness credit of forwarding the messages is defined as $AR{\cdot}W_{AR}{+}FR{\cdot}W_{FR}{+}FC{\cdot}W_{FC}$. The AR value equals the percentage of the number of messages arriving at their destination. If the agent forwards messages of other agents, then $FR = 1$; otherwise, $FR = 0$. FC is the power consumption credit for forwarding messages. If the agent forwards messages, FC is 0; otherwise, FC is 1. W_{AR}, W_{FR} and W_{FC} are the weight values of AR, FR and FC. The value domain of AR is $[0, 1]$. FR and FC can be 0 or 1. For different application requirements, AR, FR and FC can be set to different values. In the simulation, each agent sends out 10 messages for each communication round, so the precision accuracy is 0.1. The message arriving rate is assumed to be the most important evaluation criteria. Therefore, the weight value of the arriving rate is set as $W_{AR} = 1$. $W_{FR} = 0.02$ and $W_{FC} = 0.01$. Their sum is smaller than the arriving rate precision accuracy 0.1, so their weight values do not affect the distinction among different message arriving rate values. In the network, we suppose that forwarding messages is more important than saving energy. We set $W_{FR} = 0.02$, which is larger than $W_{FC} = 0.01$. The fitness credit of routing messages is defined as $AR{\cdot}W_{AR}{+}FR{\cdot}W_{FR}{+}RC{\cdot}W_{RC}$. RC is the power consumption credit for routing. If the agent calculates the next hop on the shortest path, $RC = 0$; otherwise $RC = 1$. The weight value $W_{RC} = 0.01$. The fitness credit of constructing a cluster is defined as $AR \cdot W_{AR} + CC \cdot W_{CC}$. CC is the power consumption credit for constructing and maintaining a cluster. If the agent joins in a cluster, $CC = 0$; otherwise $CC = 1$. The weight value $W_{CC} = 0.01$. The fitness credit of adjusting the transmission range is defined as $AR \cdot W_{AR} + RN \cdot RU \cdot W_{RU} + RC \cdot W_{RC}$. Where RN is the parameter that shows whether the agent has a connection route to the destination agent. If the agent cannot find a connection route to the destination agent, then RN is set to 1, which means that the agent has increased its transmission range to the maximum allowed value. RU is the usage value of increased transmission range. If the agent increases the transmission range, and the increased range is used to forward messages of other agents, then $RU = 1$; otherwise $RU = 0$. The weight value W_{RU} is set to 0.02. RC is the power consumption credit for using different transmission ranges. If the agent uses the default transmission range, $RC = 1$; if it uses the increased transmission range, $RC = 0$. The weight value $W_{RC} = 0.01$.

D. Stabilization

To make the selected function combination feasible, it must be stable. In this section, we introduce an algorithm that can stabilize the system. The mutation and learning probability are the main factors that affect churning in the selection and spread of the function combination among agents. First, we define a fitness credit threshold value T_i. This value equals the minimum acceptable fitness credit value by using or unusing the function i. Then we define a stabilization rate D, which equals the rate between the normal mutation or learning probability and the minimum acceptable mutation or learning probability. If the fitness credit of the function i is smaller than T_i, then the agent selects learning probability P_L and mutation probability P_M. Otherwise, the agent selects learning probability $\frac{P_L}{D}$ and mutation probability $\frac{P_M}{D}$. So each agent uses low mutation and learning probabilities in the state with acceptable fitness credit to decrease churning of different function combinations. The detail processing flow is shown in Algorithm 1. Each agent x is given a $Switch_{xi}$ value for each function i. Where the switch values change between 0 and 1 representing unusing and using the function, R is the random probability from 0 to 1 and Nb is the neighbor agents.

Algorithm 1 : Stabilization for Mutation and Learning

Mutation:
for all Agent x **do**
  if $(C_i \leq T_i \& R < P_M) || (C_i > T_i \& R < (P_M/D))$ **then**
    $Switch_{xi} = 1 - Switch_{xi}$
  end if
end for

Learning:
for all Agent x **do**
  $maxC = max([C_i]\ of\ Nb)$
  if $maxC > C_i$ **then**
    if $(C_i \leq T_i \& R < P_L) || (C_i > T_i \& R < (P_L/D))$ **then**
      $Switch_{xi} = [Switch_{xi}]\ of\ Nb\ with\ [maxC]$
    end if
  end if
end for

To encourage the use of the forwarding and routing functions, in the data aggregation application, the threshold values of forwarding and routing are set to 1.01. So only agents with fitness credit 1.02 and higher can use $\frac{P_L}{D}$ and $\frac{P_M}{D}$ to learn and mutate. Whether to actually use the clustering and increasing the transmission range functions is based on the environment condition, such as agent density, bandwidth, etc., so the threshold value of clustering and ranging are set to 0.99 and 1.0 respectively.

IV. Test and Evaluation

We use NetLogo [8] for the simulation experiment. The deployment area is a 200×200 square region. Agents are static and randomly scattered across the area. Each agent sends out one message per one tick. After sending 10 messages, each agent waits 10 ticks for calculating the message arriving rate. We call the 20 ticks as one communication round. The default transmission range of each agent is 40 units. All the agents send messages to one specified destination agent. We run the experiment 10 times for each testing point.

A. Adaptation to Environments

In the experiments, the environment changes by varying the agent density and the bandwidth. The agent density ranges from 5 to 40, with 5 offset. The bandwidth is 5×2 units. m is from 0 to 7, with 1 offset. Three evaluation parameters are

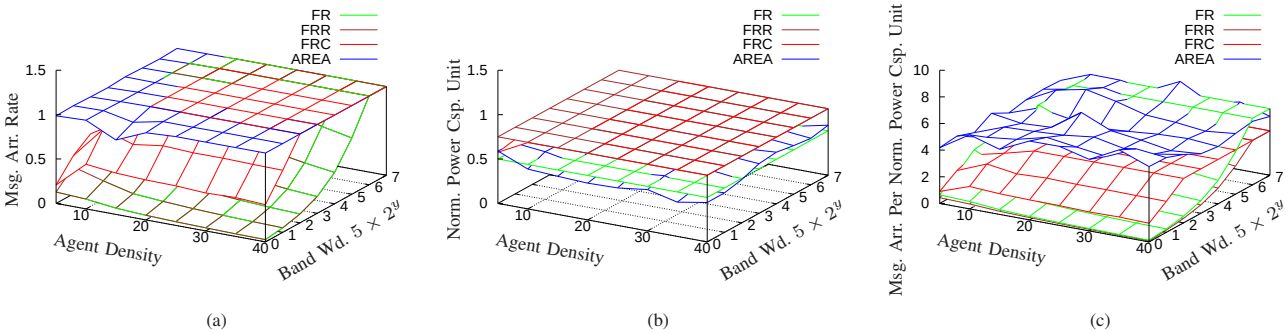


Fig. 4. (a).Average message arriving rate for various agent densities and bandwidths. (b).Normalized average power consumption unit for various agent densities and bandwidths. (c).Average message arriving count per normalized power consumption unit for various agent densities and bandwidths.

used: average message arriving rate, average power consumption unit and average arriving rate per power consumption unit. Three other data aggregation methods are used for comparison. The first method, named as *FR*, makes all agents forward and route the messages. The second method, named as *FRC*, makes all agents construct a fixed number of clusters. All agents can forward and route the messages, but only the cluster leader represents the cluster to send out messages. The cluster number is set to 20. The third method, named as *FRR*, makes all agents use the increased transmission range to forward and route the messages. The increased transmission range is 80 units. To evaluate the power consumption performance, we suppose that the running of each basic function consumes 1 unit power per round.

Figure 4(a) shows the average message arriving rate for various agent densities and bandwidths. The *AREA* mechanism has the best arriving rate in the low density area 5. This is because the network is generally unconnected with agent density 5. If the agents that are not connected to the destination agent do not increase their transmission range, they cannot send messages to the destination. Furthermore, the *AREA* mechanism makes some agents increase their transmission range to link the connected network. In the low bandwidth area from 0 to 2, the *AREA* mechanism constructs clusters to meet the bandwidth restrictions. The method *FRC* uses clusters to increase the message arriving rate. When the bandwidth is equal or larger than 20, which is equal to the fixed cluster number of *FRC*, the *FRC* method has the best arriving rate. But because it is construct with a fixed number of clusters, the method is not adaptive to the variation of the environment. If the environment has unconnected subnetwork or very low bandwidth, agents using the *FRC* method will drop messages. Agents that use the *FRR* method also use the increased transmission range all the time. In the connected network with low bandwidth, increasing the transmission range cannot always increase the message arriving rate. Although *AREA* does not have the best arriving rate, the difference to the best arriving rate is very small. This is because agents need to keep a mutation rate to adapt to the new environments in *AREA*. When the bandwidth is larger than 6, the *FR* and *FRR* methods can also have near optimal message arriving rate. This is because that the bandwidth is larger than the total number of agents in the network. So no matter how the network topology is constructed, all the messages will finally arrive at their destination.

Figure 4(b) illustrates the power consumption for various agent density and bandwidth. Because there are four types of basic functions, the average power consumption unit is normalized to 4. It can be found that the *FRC* and *FRR* methods consume the same amount of energy in all the testing points. This can be explained because all agents always use clustering in the *FRC* method and increase their transmission range when using the *FRR* method. This is because it only uses two basic functions: forwarding and routing. The power consumption of *AREA* is larger than *FR*, because *AREA* sometimes increases its transmission range and constructs a cluster to increase the message arriving rate. But *AREA* makes all the agents in the network automatically adapt to the different environments.

In order to evaluate the efficiency of the *AREA* algorithm, we calculate the rate between the message arriving count and the power consumption unit for various agent densities and bandwidth values, as shown in Figure 4(c). It can be seen that the *AREA* algorithm is the most efficient method for almost all the testing points. Only in the very high bandwidth area, the *FR* method is slightly more efficient. This is again because the total bandwidth is larger than the total number of agents. Since the *AREA* algorithm always will mutate to other function combinations, its efficiency is slightly less at this point.

B. Stabilization Efficiency

Agents in environments with a different spatial distribution could churn in different function combinations. Stabilization of the function selection is an important evaluation parameter. We test the stabilization algorithm presented in Section III-D.

The testing results are shown in Figure 5. We initialize the environment with an agent density of 20 and bandwidth of 40. The y coordinates of Figure 5(a) and Figure 5(b) are the message arriving rate and the normalized power consumption unit respectively. The x coordinates in both figures are calculated by $x = \lg D$, in which D is the stabilization rate with value 1, 5, 10, 50, 100, 500, 1000, 5000, 10000. The other testing

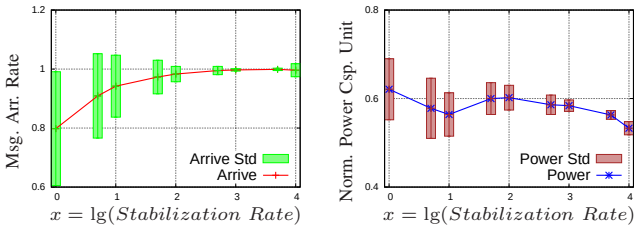


Fig. 5. (a).The average message arriving rate and standard deviation with various stabilization rates D. (b).The normalized power consumption unit and standard deviation with various stabilization rates D.

parameters are the same as the previous experiments. According to the results, as the stabilization rate D increases from 1 to 10000, the standard deviations of the message arriving rate and power consumption value decreases significantly. When the stabilization rate becomes 10000, which is 4 ($= \lg 10000$) in the figures, the standard deviations of the arriving rate and power consumption are 0.022 and 0.015. As the stabilization rate increases, the message arriving rate increases from 0.87 to 1. This is because the low stabilization rate D make the system unstable, and further decreases the message arriving rate. Therefore the stabilization Algorithm 1 can effectively stabilize the function mutation and learning.

V. Related Work

Traditional spatial computing algorithms [9] [10] primarily focus on function implementation in a fixed environment. The spatial and temporal distribution of the agents in the network significantly affect the relations among agents. In this paper, we make advantage of the coordination between agent interaction in the spatial network to promote the evolution of algorithms adapting to various changing environments.

Evolutionary dynamics [11] researches the power of advancing the system to evolves from one state to another on a global population level. Evolution dynamics theory forms the theoretical basis on which the *AREA* mechanism is founded. Evolutionary computing is widely researched for producing optimized systems [12]. Traditional evolutionary computation [13] selects members via centralized methods, which is not very efficient in distributed environments. Nakano and Suda [14] and Lee et al. [15] improve the adaptation mechanisms using evolutionary computing. They propose design structures that build adaptive network services using bio-inspired distributed agents. By evolutionary adaptation, agents can evolve and adapt their behavior to the changing environments. And Champrasert et al. [16] present a structure to self-optimize and self-stabilize cloud applications. It extends the biological evolutionary adaptation from agents to related platforms. But the above papers do not explain the influence of the parameters to the performance of the system. Moreover, these works focus on the construction of services using interacting agents, and the simulation is on a grid network, which does not map naturally (nor easily) to wireless networks. Mirko et al. [17] extends the tuple spaces by chemical-inspired model to coordinate spatially pervasive services. Although the mechanism can effectively make services diffuse and interact in a spatial network, it can not coordinate multiple different services together to cope with various problems.

VI. Conclusions

We propose a new mechanism: *automatic runtime evolutionary adaptation (AREA)* for spatial algorithm design. It can effectively create and evolve the algorithms in wireless networks to meet application requirements and adapt to the changing of the environment. The working process of *AREA* is automatically updated in every agent during runtime and the created algorithm works stably in the network. The mechanism is validated by the simulation experiment involving data aggregation. For the future work, we plan to use the *AREA* mechanism for the creation of other algorithms, such as self-created gradient, synchronization, etc. Further more, the *AREA* mechanism has the potential to be used in other domains, such as Internet services, swarm robotics control, etc., which will be further researched.

References

[1] G. Werner-Allen, J. Johnson, M. Ruiz, J. Lees, and M. Welsh, "Monitoring volcanic eruptions with a wireless sensor network," in *Wireless Sensor Networks*. IEEE, 2005, pp. 108–120.

[2] U. Lee, B. Zhou, M. Gerla, E. Magistretti, P. Bellavista, and A. Corradi, "Mobeyes: smart mobs for urban monitoring with a vehicular sensor network," *Wireless Comm., IEEE*, vol. 13, no. 5, pp. 52–57, 2006.

[3] G. Mao, B. Fidan, and B. Anderson, "Wireless sensor network localization techniques," *Comp. Net.*, vol. 51, no. 10, pp. 2529–2553, 2007.

[4] J. Kennedy, "Swarm intelligence," *Handbook of nature-inspired and innovative computing*, pp. 187–219, 2006.

[5] M. Dorigo, E. Bonabeau, and G. Theraulaz, "Ant algorithms and stigmergy," *Future Generation Computer Systems*, vol. 16, no. 8, pp. 851–871, 2000.

[6] N. Nanas and A. De Roeck, "Autopoiesis, the immune system, and adaptive information filtering," *Natural Computing*, vol. 8, no. 2, pp. 387–427, 2009.

[7] Q. Liu, A. Pruteanu, and S. Dulman, "Gde: a distributed gradient-based algorithm for distance estimation in large-scale networks," in *MSWiM, ACM*, 2011, pp. 151–158.

[8] U. Wilensky, "Netlogo, center for connected learning and computer-based modeling," *Northwestern University*, 1999.

[9] J. Beal and R. Schantz, "A spatial computing approach to distributed algorithms," in *45th Asilomar Conference on Signals, Systems, and Computers*, 2010.

[10] M. Duckham, "Decentralized spatial algorithm design," *Spatial Computing 2012 colocated with AAMAS*, pp. 13–18.

[11] M. Nowak, *Evolutionary Dynamics: exploring the equations of life*. Belknap Press, 2006.

[12] Y. Jin and J. Branke, "Evolutionary optimization in uncertain environments : A survey," *Evolutionary Computation, IEEE Transactions on*, vol. 9, no. 3, pp. 303–317, 2005.

[13] M. Melanie, "An introduction to genetic algorithms," *Cambridge, Massachusetts London, England, Fifth printing*, 1999.

[14] T. Nakano and T. Suda, "Self-organizing network services with evolutionary adaptation," *Neural Networks, IEEE Transactions on*, vol. 16, no. 5, pp. 1269–1278, 2005.

[15] C. Lee, J. Suzuki, and A. Vasilakos, "An evolutionary game theoretic framework for adaptive, cooperative and stable network applications," *BIONETICS*, pp. 189–204, 2012.

[16] P. Champrasert, J. Suzuki, and C. Lee, "Exploring self-optimization and self-stabilization properties in bio-inspired autonomic cloud applications," *Concurrency and Computation: Practice and Experience*, 2012.

[17] M. Viroli, M. Casadei, S. Montagna, and F. Zambonelli, "Spatial coordination of pervasive services through chemical-inspired tuple spaces," *ACM Trans. Auton. Adapt. Syst.*, vol. 6, no. 2, pp. 14:1–14:24, 2011.

Promoting Space-Aware Coordination: ReSpecT as a Spatial Computing Virtual Machine

Stefano Mariani Andrea Omicini
ALMA MATER STUDIORUM–Università di Bologna
Email: {s.mariani, andrea.omicini}@unibo.it

***Abstract*—Situatedness is a fundamental requirement for today's complex software systems—as well as for the computation models and programming languages used to build them. *Spatial situatedness*, in particular, is an essential feature for *coordination models and languages*, as they represent the most effective approach to face the critical issues of interaction. Following some seminal works [1], [2], [3], in this paper we try to bring some novel results from the Coordination field into the Spatial Computing perspective, by identifying a minimal set of primitives that could be used to build a virtual machine for a space-aware coordination language, using ReSpecT as our reference example.**

I. The Spatial Computing View

According to the definition in [1], a Spatial Computer is

> *"a collection of local computational devices distributed through a physical space, in which the difficulty of moving information between any two devices is strongly dependent on the distance between them, and the functional goals of the system are generally defined in terms of the system's spatial structure."*

So, Spatial Computing Languages (SCL) are basically concerned with linking the aggregate behaviour of a number of spatially-situated independent devices to the way in which they are individually programmed. Also, SCL should provide Spatial Computers programmers with the most suitable topological constructs to program the ensemble of devices—whereas the task of translating such constructs into locally executable ones should be left to a proper compiler/interpreter running on each device. Along this line, a layered architecture for devices running Spatial Computing programs is described in [1], which helps bridging the gap between the hardware and SCL:

- *Physical Platform* — The lowest level in the hierarchy, identifying the medium upon which the computation actually executes—e.g. a smartphone, a drone with a whole set of sensors and actuators, even a virtual device in the case of a simulation.
- *System Management* — Typically the OS layer, abstracting away from physical details, (hopefully) providing all the low-level drivers needed by spatial applications—e.g. for a GPS module, or a motion engine.
- *Abstract Device* — The top abstraction level exposing the basic API for SCL—e.g. a clock service, GPS coordinates tracking, and the like.

Focussing on the last layer, Subsection I-A describes the Abstract Device Model, allowing different devices to be distinguished by analysing their relationships with spatial aspects.

A. The Abstract Device Model

According to [1], the Abstract Device Model (ADM) relates computing devices to the space they occupy, and specifies the ways in which they communicate with one another. Although both aspects are essential in the Spatial Computing context, here we mainly focus on the latter, assuming for the former a discrete model in which any device occupies a null amount of space—as typically done in the context of sensor networks.

As far as communication between devices is concerned, three key properties are identified:

- *Communication Region* — The relationship between the communication actions of a given device and their spatial "coverage"—e.g., distance-limited (*aka* neighborhood) *vs.* global communication.
- *Transmission Granularity* — The relationship between the communication actions and the number of receivers—e.g., unicast / multicast / broadcast communication.
- *Mobility of Code* — The relationship between a given device's runnable program and other's—e.g. do they run the same program (possibly executing differently on each), or, are they heterogeneous? Can they share code?

This model is used in Subsection IV-B to describe our Spatial Computing virtual machine.

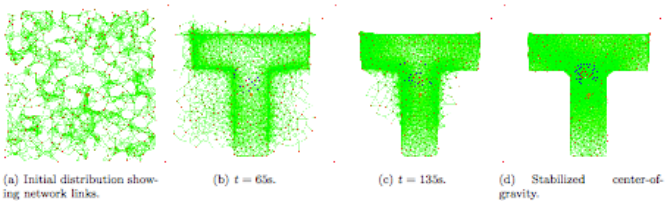
B. The "T-Program" Requirements

Independently of the layer of abstraction at which a given SCL can be placed, as well as of the kind of ADM it implements, three classes of operators are required to reach maximal expressiveness and computational power—the sort of *"Spatial-Turing equivalence"* discussed in [2]:

- *Measure Space* — Transforming spatial properties into computable information—e.g., distances, angles, areas.
- *Manipulate Space* — Translating information into some modification of the spatial properties of the device—e.g., turning wheels to face a given direction, slowing down the motion engine.
- *Compute* — Beyond Turing computation, any kind of "spatial-pointwise" operation as an interaction, or a non-spatial sensor or actuator—e.g., a light recogniser.

A fourth class (*Physical Evolution*) looks more like a sort of assumption over the (possibly autonomous) dynamics a given program/device can rely upon—e.g., the existence of actuators responding to the program/device commands, or the independent motion of a colony of cells.

As a reference benchmark to test the expressive power and the computational completeness of SCL, the "T-Program" is proposed in [1], consisting of the following three stages, depicted in Fig. 1 for Proto [3]: *(i)* cooperatively creating a local coordinate system; *(ii)* moving devices to create a "T-shaped" structure; *(iii)* computing such structure centre of gravity and draw a ring around it.



(a) Initial distribution showing network links. (b) $t = 65$s. (c) $t = 135$s. (d) Stabilized center-of-gravity.

Fig. 1. T-Program "run" from [1], implemented using the Proto language [3].

Stage *(i)* requires the capability to measure the spatial context where the program/device lives; stage *(ii)* requires the ability to manipulate the spatial properties of each device (thus relying also on the fourth category); stage *(iii)* requires both computational capabilities and, again, measuring capabilities.

Subsection IV-C provides both infrastructural and linguistic support to all the three classes of operators by means of our Spatial Computing virtual machine.

II. The Space-Aware Coordination View

The need for coordination models and languages to support adaptiveness and reactiveness to the contingencies that may occur in the spatio-temporal fabric was recognised by several proposals in the literature, accounting for spatial issues and/or enforcing spatial properties.

$\sigma\tau$LINDA [4] extends the LINDA model in three ways to enable the emergence of spatio-temporal patterns for tuples configuration in a distributed setting: *(i)* tuples are replaced by "space-time activities", that is, processes manipulating the space-time configuration of tuples in the network; *(ii)* space operators `neigh`, `$distance` and `$orientation` are added to allow respectively to send broadcast messages to the neighborhood spaces, measure the distance toward them, devise the relative orientation of a target space w.r.t. the current one; *(iii)* time operators `next`, `$delay` and `$this` are added allowing (respectively) to delay an activity execution, measure the flow of time, access current coordination space identifier.

GEOLINDA [5], another LINDA extension, deals with spatial aspects by attempting to reflect physical spatial properties in a mobile tuple space setting: *(i)* each tuple and each reading operation is associated to a geometrical volume (*addressing shape*); *(ii)* the semantics of reading operations is changed so as to unblock only when the shape of a matching tuple intersects with the addressing shape of the operation; *(iii)* each coordination space is associated to a communication range to allow detection of *incoming* and *outgoing* tuples/volumes.

The TOTA middleware [6] considers a distributed tuple space setting in which each tuple is equipped by two additional fields other than its content, so as to enable emergence of *computational fields*—such as gradients: *(i)* a *propagation rule*, determining how the tuple should propagate and distribute across the network of linked tuple spaces; *(ii)* a *maintenance rule*, dictating how the tuple should react to the flow of time and/or events occurring in the space.

In [7], the authors adopt a biochemically-inspired metaphor and apply it to tuple-based coordination, namely extending standard tuple spaces with the ability of evolving tuples so as to mimick chemical systems—in terms of reaction and diffusion rules that apply to tuples modulo semantic match.

Although the above models deal with some spatial-related issues, they are basically tailored to a specific problem or application domain. Thus, since they were conceived with a given goal in mind, they mostly lack of an exhaustive analysis of which are the basic mechanisms required to enable and promote "general-purpose" Space-Aware Coordination. To this very end, in the following we take the ReSpecT coordination language & virtual machine [8] – already augmented to account for time-related aspects [9] and environment situatedness [10] – and extend it with few fundamental constructs and mechanisms required to deal with spatial issues.

III. ReSpecT in Time & Enviroment

In short, a ReSpecT *tuple centre* is a tuple space that can be *programmed* so as to react to *events* occurring in the space—basically allowing its coordination policies to be tailored to the specific application needs [11], [8]. ReSpecT is a Prolog-like language where events – such as coordination operations issued by either agents or tuple centres – can be declaratively associated to computations carried out *atomically* and *transactionally* by the ReSpecT tuple centre [12].

Such a declarative specification is called *reaction*, and takes the form `reaction(Event, Guards, ReactionGoals)`:

- `Event` — Any event involving the tuple centre, such as the execution of a LINDA-like operation.

⟨*TCSpecification*⟩	::=	{ ⟨*SpecificationTuple*⟩ . }
⟨*SpecificationTuple*⟩	::=	`reaction(` ⟨*TCEvent*⟩ `,` [⟨*Guard*⟩ `,`] ⟨*Reaction*⟩ `)`
⟨*TCEvent*⟩	::=	⟨*TCPredicate*⟩ `(` ⟨*Tuple*⟩ `)` | `time(` ⟨*Time*⟩ `)` | ⟨*EnvPredicate*⟩
⟨*Reaction*⟩	::=	⟨*ReactionGoal*⟩ | `(` ⟨*ReactionGoal*⟩ { `,` ⟨*ReactionGoal*⟩ } `)`
⟨*ReactionGoal*⟩	::=	⟨*TCPredicate*⟩ `(` ⟨*Tuple*⟩ `)` | ⟨*ObsPredicate*⟩ `(` ⟨*Tuple*⟩ `)` | ⟨*Computation*⟩ | `(` ⟨*ReactionGoal*⟩ `;` ⟨*ReactionGoal*⟩ `)` | ⟨*TCLinkPredicate*⟩ | ⟨*TCEnvPredicate*⟩
⟨*TCPredicate*⟩	::=	`out` | `in` | `inp` | `rd` | `rdp` | `no` | `nop` | `get` | `set`
⟨*TCLinkPredicate*⟩	::=	⟨*TCId*⟩ `?` ⟨*TCPredicate*⟩
⟨*TCEnvPredicate*⟩	::=	⟨*EnvResId*⟩ (`<-|->`) ⟨*EnvPredicate*⟩
⟨*EnvPredicate*⟩	::=	`env(` ⟨*Key*⟩ `,` ⟨*Value*⟩ `)`
⟨*ObsPredicate*⟩	::=	⟨*EventView*⟩`_`⟨*EventInfo*⟩ `(` ⟨*Tuple*⟩ `)` | ⟨*EnvPredicate*⟩
⟨*EventView*⟩	::=	`current` | `event` | `start`
⟨*EventInfo*⟩	::=	`predicate` | `tuple` | `source` | `target` | `time`
⟨*Guard*⟩	::=	⟨*GuardPredicate*⟩ | `(` ⟨*GuardPredicate*⟩ { `,` ⟨*GuardPredicate*⟩ } `)`
⟨*GuardPredicate*⟩	::=	`request` | `response` | `success` | `failure` | `endo` | `exo` | `intra` | `inter` | `from_agent` | `to_agent` | `from_tc` | `to_tc` | `before(` ⟨*Time*⟩ `)` | `after(` ⟨*Time*⟩ `)` | `from_env` | `to_env`
⟨*Computation*⟩	is	a Prolog-like goal performing arithmetic / logic computations
...	...	...

TABLE I
CORE SYNTAX OF ReSpecT

- `Guards` — Predicates providing fine-grained control over reaction *triggering* based on observable event properties.
- `ReactionGoals` — The ReSpecT operations – including Prolog computations – to be issued in response to the triggering event.

Whereas the full formal syntax can be found in [8], here we focus solely on the grammar rules in TABLE I.

A. ReSpecT *with Time*

The extension of ReSpecT to deal with time-related aspects was discussed in [9]. According to TABLE I, ReSpecT deals with time through: the event `time(`⟨*Time*⟩`)`; the observation predicates ⟨*EventView*⟩ `_time`; and the guards `before(`⟨*Time*⟩`)`, `after(`⟨*Time*⟩`)`:

- `time(`⟨*Time*⟩`)` — Triggers a reaction at a given ⟨*Time*⟩.
- ⟨*EventView*⟩`_time` — Access timing information about the triggering event.
- `before(`⟨*Time*⟩`)`, `after(`⟨*Time*⟩`)` — Execute reaction goals if a certain timing condition is met.

Altogether, they make ReSpecT a *time-aware* language by extending ReSpecT tuple centres to time-aware coordination abstractions.

B. ReSpecT *with Situatedness*

The extension of the ReSpecT language towards a more comprehensive notion of *situatedness* was discussed in [10]. According to TABLE I, ReSpecT deals with general environment events through:

- ⟨*EnvResId*⟩ — Introduced as a new identifier – beyond ⟨*AgentId*⟩ and ⟨*TCId*⟩ [8] – to univocally represent environmental resources.
- `-> env(`⟨*Key*⟩`,`⟨*Value*⟩`)`, `<- env(`⟨*Key*⟩`,`⟨*Value*⟩`)` — Provide the ability to *observe* and *control* any environmental resource – typically sensors and actuators – directly handling environment events.
- `env(`⟨*Key*⟩`,`⟨*Value*⟩`)` — As an event descriptor, triggers reactions to environment events.
- `env(`⟨*Key*⟩`,`⟨*Value*⟩`)` — As an observation predicate, accesses the properties of an environment event.
- `from_env`, `to_env` — Guards to filter triggering of environment-driven reactions.

The above constructs allow coordination policies to be situated within the computational environment, which is an essential requirement for complex systems such as pervasive and adaptive ones [13].

C. ReSpecT *with Space: What is Missing?*

Although the environmental interpretation of the spatio-temporal fabric is undoubtedly useful, we believe a crucial point regarding the situatedness of coordination abstractions is missing: both time and space properties should be first-class entities directly embedded within the coordination medium, which lives immersed in the environment as much as agents do. Nevertheless, the simple yet expressive interface toward sensors and actuators – or, more generally, any kind of environmental resource – provided by the situated extension of ReSpecT remains essential, and is then used in our extension toward spatial coordination (Subsection IV-C). Thus, following the approach of [9] for time awareness, in Subsection IV-A we propose a novel extension along the spatial dimension, promoting *space-awareness* at the language level.

IV. Bridging The Gap

We believe the main concern of SCL – that is, *"the ability to link the aggregate behaviour of a number of independent devices to the way in which they are individually programmed"* (Section I) – contains in the very end a coordination issue. In fact, in spite of the (possibly higher) desired level of abstraction for a Spatial Computing programmer, "under the hood" its program should be somehow compiled/interpreted so as to meaningfully coordinate a set of independent devices—with the aim of achieving the application-specific goal. Furthermore, based on the Spatial Computing requirements, the distributed coordination process taking place between the programmed devices should clearly be space-driven, hence *space-aware coordination* is what is actually needed.

In the following, after presenting the extended ReSpecT model – here called *st*ReSpecT –, we describe why it could be taken as a reference Abstract Device Model, upon which to interpret/compile SCL, by highlighting its most appealing features. Then, in Subsection IV-C, we focus on meeting the requirements pointed out by the T-Program benchmark introduced in Subsection I-B.

A. *st*ReSpecT *Model*

In order to both express and enforce space-dependent coordination laws, a number of issues should be addressed: *(i)* first of all, the coordination language ontology should include all the necessary concepts so as to deal with spatial aspects; *(ii)* then, the language should provide a set of statements to enable coordination laws to "talk about space"; *(i)* finally, the coordination medium executing the language should "naturally" account for space-related aspects during its lifecycle.

Issue *(i)* requires some reasoning upon the different criteria according to which spatial properties can be described: *(locality)* space could be defined w.r.t. either a global coordinate system, known by any coordination artefact, or a local one; *(relativity)* such coordinate system could be either absolute, hence exploiting some spatial conventions – e.g. GPS data –, or relative, thus describing spatial relationships between different coordination artefacts solely; *(topology)* spatial properties may be described considering either the physical world – e.g. "Via Zamboni 33, Bologna" –, our *organisational view* of the world – e.g. "the Dean's office at DISI" – or even a logical interpretation of the world—e.g. "Stefano's workspace"; *(granularity)* spatial measures could be expressed either on a continuous scale – again, as GPS data – or on a discrete scale—e.g. hops in a network.

Since we are interested in supporting virtually any kind of space-aware coordination policy, our assumption here is that a global, absolute and "physical" description of the world is

...	...	...
⟨*TCEvent*⟩	::=	⟨*TCPredicate*⟩ `(` ⟨*Tuple*⟩ `)` | `time(` ⟨*Time*⟩ `)` | ⟨*EnvPredicate*⟩ | `from(` ⟨*Place*⟩ `)` | `to(` ⟨*Place*⟩ `)`
...	...	...
⟨*EventInfo*⟩	::=	`predicate` | `tuple` | `source` | `target` | `time` | `place`
...	...	...
⟨*GuardPredicate*⟩	::=	`request` | `response` | `success` | `failure` | `endo` | `exo` | `intra` | `inter` | `from_agent` | `to_agent` | `from_tc` | `to_tc` | `before(` ⟨*Time*⟩ `)` | `after(` ⟨*Time*⟩ `)` | `from_env` | `to_env` | `at(` ⟨*Place*⟩ `)` | `near(` ⟨*Place*⟩ `,` ⟨*Radius*⟩ `)`
...	...	...
⟨*Place*⟩	is	a ground term representing a point in an absolute space
⟨*Radius*⟩	is	a non-negative integer representing distance

TABLE II
*st*ReSpecT SYNTAX EXTENSIONS TO ReSpecT

available. Unlike time aspects, in fact, where a relative, local notion of time is often preferred in distributed environments, most of today hardware/software systems have an easy and effective access to GPS data, making a global, absolute and physical description of any spatial property easily available—even in distributed systems. Furthermore, given such a description is available at the infrastructural level, it is easy to build higher-level descriptions either at the application level or through a suitable middleware—e.g., designed using *st*ReSpecT itself.

Issue *(ii)* is possibly the most challenging one, since it requires first to (theoretically) foresee any kind of space-driven coordination law an application may need, then to provide a set of "spatial statements" to allow for it. Our proposal for *Space-Time-Aware* ReSpecT (*st*ReSpecT henceforth) is reported by difference in TABLE II—amending TABLE I.

By focussing on the new things only, *st*ReSpecT features:

- `from(`⟨*Place*⟩`)`, `to(`⟨*Place*⟩`)` — Triggering reactions to *spatial events*, such as leaving from and arriving at a given location.
- ⟨*EventView*⟩`_place` — Providing access to spatial information for the triggering event.
- `at(`⟨*Place*⟩`)`, `near(`⟨*Place*⟩`,`⟨*Radius*⟩`)` — Allowing for evaluation of the reaction goals when the given spatial conditions are met.

Complementarily to the timed extension of ReSpecT [9], the above constructs enhance the programmed coordination media to be space-aware—that is, able to recognise events, access information, and test conditions belonging to the spatial dimension in a *situated* coordinated system.

In particular, the extended notion of ⟨*TCEvent*⟩ in TABLE II accounts for issue *(iii)*, since it represents the ability by the coordination abstraction to pro-actively generate *space events*, in response to which some kind of space-related (reactive) computation can be programmed. Obviously, the *st*ReSpecT virtual machine relies on the capability of the hosting platform software/hardware to provide spatial information, e.g., availability of a GPS module. Looking at the HW/SW equipment of today mobile devices, this is apparently a reasonable assumption met by the vast majority of them.

In the remainder of the paper, we discuss the suitability and the benefits of using *st*ReSpecT as a Spatial Computing virtual machine upon which to build & run domain-specific SCL, by better framing it in the ADM, and showing how the T-Program requirements could be successfully met.

B. The *st*ReSpecT Device Model

Being *st*ReSpecT a distributed tuple centres programming model featuring a communication language and a coordination language both based on first-order logic tuples, it naturally provides a number of appealing features from the Spatial Computing standpoint, which we list here by recalling the ADM—from Subsection I-A:

- *Communication Region* — The *st*ReSpecT language fully complies with the *linkability model* introduced in [14], hence any tuple centre can communicate with any other network-reachable tuple centre through *linking operations* (⟨*TCLinkPredicate*⟩ in TABLE I). The basic communication model of *st*ReSpecT is thus global communication.
- *Transmission Granularity* — No first-class coordination operations exist in *st*ReSpecT allowing communication with a set of target tuple centres at once—unlike [15]. Thus, *st*ReSpecT supports point-to-point communication at the language level.
- *Mobility of Code* — Since *st*ReSpecT features meta-coordination operations dealing with specification tuples – pretty much as coordination operations do with communication tuples –, it requires no effort moving *st*ReSpecT "code" from space to space.[1] Therefore, *st*ReSpecT tuple centres features code mobility.

The ADM described is just the "basic" one straightforwardly supported by *st*ReSpecT tuple centres. Other ADM could be defined on top of the basic one by exploiting *st*ReSpecT itself to adequately program a tuple centre. For instance, distance-limited communication could be implemented by simply *(i)* attaching to any exchanged tuple a space value, and *(ii)* programming tuple centres so as to check such value, then stopping communication when due, continuing otherwise.

Even the Transmission Granularity property could be adapted by need. For instance, broadcast (multicast) communication could be achieved by exploiting the transactional semantics of *st*ReSpecT reactions: if any tuple centre is programmed to spread any incoming information to any (some) other one, the whole broadcast (multicast) communication is perceived as a single, atomic communicative act by the initiator (agent, tuple centre) of the original operation.

A special remark is due w.r.t. the last property of the *st*ReSpecT ADM: in fact, describing as "code mobility" the feature provided by the meta-coordination operations of the *st*ReSpecT programming model is somehow restrictive. In fact, not only ReSpecT allow specification tuples to be moved between tuple centres: also, they can be added and deleted, their presence/absence tested—and, most importantly, this can

[1]Not reported in TABLE I for the sake of simplicity. Here, we assume meta-coordination predicates with the form ⟨*TCForgePredicate*⟩ ::= ⟨*TCPredicate*⟩`_s`—see [8] for the full syntax.

be done *at run-time*. Coupling this with the spatio-temporal awareness of the *st*ReSpecT model enables not only spatio-temporal-aware coordination, but also *spatio-temporal-aware meta-coordination*—that is, the ability of the coordination abstraction to adapt its coordination policies as time passes and the spatial context changes.

C. Meeting "T-Program" Requirements

Given the *st*ReSpecT syntax reported in TABLE II, we can now associate the new constructs to the classes of operators described in Subsection I-B. One should keep in mind that we are concerned with the *minimal API* our *st*ReSpecT-based Spatial Computing VM should provide to SCL, so as to better separate the "primitive" spatial mechanisms from the higher-level "composite" behaviours/patterns they could be used to build—similarly to the approach taken in [16] for classifying and comparing self-organising design patterns.

- *Measure Space* — A combination of three *Observation Predicates* is given to measure spatial properties:
  - `current_place` — Measuring where the tuple centre executing the current *st*ReSpecT reaction is.
  - `event_place`, `start_place` — Measuring respectively where the *direct cause* and *start cause* [8] of the event triggering the current *st*ReSpecT computation took place.
- *Manipulate Space* — Given that the modification of spatial properties is necessarily bound to the facilities provided by the host device, this can be addressed by the original situatedness constructs of ReSpecT (Subsection III-B):
  - ⟨*EnvResId*⟩ `<-` `env`(⟨*Key*⟩, ⟨*Value*⟩) — Precisely meant to be used as an interface to device actuators, allowing an agent to dispatch commands to a device ⟨*EnvResId*⟩ at the most appropriate level of abstraction—that is, as a part of the environment managed through the coordination medium.
- *Compute* — While this category is orthogonal w.r.t. the spatial (temporal) dimension, some predicates may be considered here as they ease the process of *"computing over spatial information"*—in particular, guards:
  - `at`(⟨*Place*⟩) — Triggers a reaction when the reacting tuple centre is at a given location.
  - `near`(⟨*Place*⟩, ⟨*Radius*⟩) — Triggers a reaction when the reacting tuple centre is near a given location.

Finally, it should be noted that some constructs were left out, in particular those in ⟨*TCEvent*⟩. We believe they belong to the fourth category of "operators", since they allow the Abstract Device – that is, the *st*ReSpecT VM – to *perceive motion events* generated either autonomously or on demand by the physical device hosting the VM. Respectively, `from`(⟨*Place*⟩) is the spatial event generated by the *st*ReSpecT VM when the host tuple centre *starts moving* (leaving a location), whereas `to`(⟨*Place*⟩) is the spatial event generated when it *stops moving* (approaching a location). Such events are meant to reify a *change of state* in the "spatial dimension of computation": therefore, no events have to be generated while the VM is staying still, since there is no state change to reify.

D. Sketching "T-Program" Stages

To better demonstrate how a *st*ReSpecT tuple centre could act as a Spatial Computing virtual machine, in the following we sketch a few reactions showing how the spatial constructs could be used to support the different stages of the "T-Program" benchmark—whereas not directly solving it, which could become cumbersome and should be left to a higher-level SCL.

Coordinate System: Setting a local coordinate system basically amounts at *(i)* choosing an origin node, *(ii)* making it spread a "vector tuple" to neighbours, then (recursively) *(iii)* making them increment such vector, and *(iv)* forwarding it to neighbours. Thus, the basic mechanism needed by the VM at the application level is what we call *neighborhood spreading*. Assuming a *physical* neighborhood relation is used, the following reactions – installed on every node – could help:

```
% Check range then forward.
reaction( out(nbr_spread(msg(Msg),nbr(Dist),req(ID))),
  ( completion, success ),
  ( no(req(ID)), out(req(ID)), % Avoid flooding
    current_place(Me), event_place(Sender),
    within(Me,Sender,Dist), % Prolog computation
    out(msg(Msg)),
    rd(nbrs(Nbrs)), % Neighbours list
    out(forward(Msg,Dist,req(ID),Nbrs))
) ).
% Delete multicast request.
reaction( out(nbr_spread(msg(Msg),nbr(Dist),req(ID))),
  ( completion, success ),
    in(spread(msg(Msg),nbr(Dist),req(ID)))
).
% Forward to every neighbour.
reaction( out(forward(Msg,Dist,req(ID),[H|T])), % Some Nbrs
  ( intra, completion ),
  ( H ? out(spread(msg(Msg),nbr(Dist),req(ID))), % Forward
    out(forward(Msg,Dist,req(ID),T)) % Iterate
) ).
% Delete iteration tuple.
reaction( out(forward(Msg,Dist,req(ID),Nbrs)),
  ( intra, completion ),
  in(forward(Msg,Dist,req(ID),Nbrs)) % Delete it anyway
).
```

Reactions 2-10 and 12-15 manage spreading requests: the former checks if request has been already served, gets reacting node position, sender node's one, checks if it's in the desired range, and if so stores the tuple (`Msg`) then starts forwarding it; the latter simply removes the request. Reactions 17-21 and 23-26 manage the forwarding process, that is, iterate neighbours forwarding the spreading comman—where neighbourhood is set here at the VM level through a tuple `nbrs([nbr_1,...,nbr_N])`, but could also be set at the middleware level by using a coordination middleware such as TuCSoN [17].

a) "T-shape": To arrange nodes (tuple centres) so as to form a "T-shaped" structure, we need to *(i)* define spatial constraints representing the T (how much "tall", "fat", etc.), then *(ii)* make every node move so as to satisfy them. Thus, the basic mechanism needed at the VM level is *motion monitoring and control*:

```
% Compute motion vector then start moving.
reaction( out(move(Constraints)),
  ( completion, success ),
  ( current_place(Here),current_time(Now),Check is Now+1000,
    direction(Constraints,Here,Vec), % Prolog computation
    out_s(
      % Reaction 12-22
    ),
    set(engine,'on'), set(dir,Vec) % Start actuators
) ).
% Motion constraints monitoring.
reaction( time(Check),
  internal,
  ( current_place(Here),
    rd(move(Constraints)),
    ( check(Here,Constraints), % Prolog computation
      set(engine,'off')
    ;
      current_time(Now), Check is Now+1000,
      out_s(
      % Reaction 12-22
) ) ) ).
% Arrival clean-up.
reaction( to(Dest),
  internal,
  in(move(Constraints))
).
```

An interesting feature of *st*ReSpecT is exploited in the code above. Beside reacting to a motion request by properly controlling actuators, Reaction 2-10 performs a *meta-coordination operation*, by inserting a new coordination law in the tuple centre (Reaction 12-22), which is responsible for arrival check. This is precisely what we defined as *space-aware meta-coordination* at the end of Subsection IV-B.

b) Focal Point: To first compute the focal point (FC) of the "T-shape", then draw a sphere around it, we need two basic mechanisms, both similar to the *neighborhood spreading* previously shown: *(i)* a *bidirectional* neighborhood spreading to collect replies to sent messages – allowing to aggregate all the node's coordinates and counting them – and *(ii)* a *spherical multicast* to draw the ring pattern. Nevertheless, the code for spherical multicast (not shown here for the sake of brevity) is almost identical to neighborhood spreading, but with a fundamental difference (besides the `req(ID)` test to avoid flooding), that is, the use of observation predicate `start_place` instead of `event_place`. This replacement (almost) alone stops the spreading process and completely change the *spatial properties* of communication, demonstrating what we meant with *minimal API*.

V. Conclusion

In this paper we brought some results from the Coordination field into the Spatial Computing perspective: in particular, we presented the *st*ReSpecT space-aware extension to the ReSpecT language & virtual machine as a suitable Abstract Device Model for Spatial Computing Languages. We then described how *st*ReSpecT programmability can be exploited to *(i)* adapt *st*ReSpecT ADM to application-specific needs (even at run-time), and *(ii)* meet "T-Program" benchmark requirements. Finally, we sketched a few lines of code showing how *st*ReSpecT model provides VM-level support to SCL.

VI. Acknowledgments

This work has been partially supported by the EU-FP7-FET Proactive project SAPERE – Self-aware Pervasive Service Ecosystems, under contract no. 256874.

References

[1] J. Beal, S. Dulman, K. Usbeck, M. Viroli, and N. Correll, "Organizing the aggregate: Languages for spatial computing," *CoRR*, vol. abs/1202.5509, 2012.

[2] J. Beal, "A basis set of operators for space-time computations," in *Proceedings of the 2010 Fourth IEEE International Conference on Self-Adaptive and Self-Organizing Systems Workshop (SASOW 2010)*. Washington, DC, USA: IEEE Computer Society, 2010, pp. 91–97.

[3] M. Viroli, J. Beal, and M. Casadei, "Core operational semantics of Proto," in *26th Annual ACM Symposium on Applied Computing (SAC 2011)*, M. J. Palakal, C.-C. Hung, W. Chu, and W. E. Wong, Eds., vol. II: Artificial Intelligence & Agents, Information Systems, and Software Development. Tunghai University, TaiChung, Taiwan: ACM, 21–25 Mar. 2011, pp. 1325–1332.

[4] M. Viroli, D. Pianini, and J. Beal, "Linda in space-time: an adaptive coordination model for mobile ad-hoc environments," in *Coordination Languages and Models*, ser. LNCS, M. Sirjani, Ed. Springer-Verlag, Jun. 2012, vol. 7274, pp. 212–229, proceedings of the 14th Conference of Coordination Models and Languages (Coordination 2012),Stockholm (Sweden), 14-15 June.

[5] J. Pauty, P. Couderc, M. Banatre, and Y. Berbers, "Geo-linda: a geometry aware distributed tuple space," in *Advanced Information Networking and Applications, 2007. AINA '07. 21st International Conference on*, may 2007, pp. 370 –377.

[6] M. Mamei and F. Zambonelli, "Programming pervasive and mobile computing applications: The TOTA approach," *ACM Transactions on Software Engineering and Methodology*, vol. 18, no. 4, Jul. 2009.

[7] M. Viroli, M. Casadei, S. Montagna, and F. Zambonelli, "Spatial coordination of pervasive services through chemical-inspired tuple spaces," *ACM Transactions on Autonomous and Adaptive Systems*, vol. 6, no. 2, pp. 14:1–14:24, June 2011.

[8] A. Omicini, "Formal ReSpecT in the A&A perspective," *Electronic Notes in Theoretical Computer Science*, vol. 175, no. 2, pp. 97–117, Jun. 2007, 5th International Workshop on Foundations of Coordination Languages and Software Architectures (FOCLASA'06), CONCUR'06, Bonn, Germany, 31 Aug. 2006. Post-proceedings.

[9] A. Omicini, A. Ricci, and M. Viroli, "Time-aware coordination in ReSpecT," in *Coordination Models and Languages*, ser. LNCS, J.-M. Jacquet and G. P. Picco, Eds. Springer-Verlag, Apr. 2005, vol. 3454, pp. 268–282, 7th International Conference (COORDINATION 2005), Namur, Belgium, 20–23 Apr. 2005. Proceedings.

[10] M. Casadei and A. Omicini, "Situated tuple centres in ReSpecT," in *24th Annual ACM Symposium on Applied Computing (SAC 2009)*, S. Y. Shin, S. Ossowski, R. Menezes, and M. Viroli, Eds., vol. III. Honolulu, Hawai'i, USA: ACM, 8–12 Mar. 2009, pp. 1361–1368.

[11] A. Omicini and E. Denti, "Formal ReSpecT," *Electronic Notes in Theoretical Computer Science*, vol. 48, pp. 179–196, Jun. 2001, declarative Programming – Selected Papers from AGP 2000, La Habana, Cuba, 4–6 Dec. 2000.

[12] ——, "From tuple spaces to tuple centres," *Science of Computer Programming*, vol. 41, no. 3, pp. 277–294, Nov. 2001.

[13] A. Ricci, M. Viroli, and A. Omicini, "The A&A programming model and technology for developing agent environments in MAS," in *Programming Multi-Agent Systems*, ser. LNCS, M. Dastani, A. El Fallah Seghrouchni, A. Ricci, and M. Winikoff, Eds. Springer, Apr. 2008, vol. 4908, pp. 89–106, 5th InternationalWorkshop (ProMAS 2007), Honolulu, HI, USA, 15 May 2007. Revised and Invited Papers.

[14] A. Omicini, A. Ricci, and N. Zaghini, "Distributed workflow upon linkable coordination artifacts," in *Coordination Models and Languages*, ser. LNCS, P. Ciancarini and H. Wiklicky, Eds. Springer, Jun. 2006, vol. 4038, pp. 228–246, 8th International Conference (COORDINATION 2006), Bologna, Italy, 14–16 Jun. 2006. Proceedings.

[15] R. Menezes, A. Omicini, and M. Viroli, "On the semantics of coordination models for distributed systems: The LogOp case study," *Electronic Notes in Theoretical Computer Science*, vol. 97, pp. 97–124, 22 Jul. 2004, 2nd International Workshop "Foundations of Coordination Languages and Software Architecture" (FOCLASA 2003), Marseille, France, 2 Sep. 2003. Proceedings.

[16] J. Fernandez-Marquez, G. Marzo Serugendo, S. Montagna, M. Viroli, and J. Arcos, "Description and composition of bio-inspired design patterns: a complete overview," *Natural Computing*, pp. 1–25, 2012.

[17] A. Omicini and F. Zambonelli, "Coordination for Internet application development," *Autonomous Agents and Multi-Agent Systems*, vol. 2, no. 3, pp. 251–269, Sep. 1999, special Issue: Coordination Mechanisms for Web Agents.

Modeling Inertia in an Amorphous Computing Medium

Alyssa S. Morgan and Daniel N. Coore

Abstract— **The Growing Point Language (GPL) is a programming language that can be used to specify self-organizing topological patterns on an amorphous computer. GPL uses a distributed diffusion process (metaphorically represented as the secretion of a pheromone) to break the spatial symmetry that is inherent in the coordinate-agnostic property of the amorphous computing model. (The result of a diffusion is a radially symmetric, spatial function that decreases monotonically with the distance from the source.) Tropisms, which guide the movement (metaphorically propagation) of growing points, are specified relative to the level curves of one or more pheromones. Tropisms that are orthogonal to the level curves of a pheromone are implemented by a node identifying its neighbour with a local minimum (or maximum) for that pheromone's concentration. The monotonic property of the diffusion ensures that growing points with such tropisms move in the same direction on each propagation step. On the other hand, tropisms that are tangential to a pheromone's level curves reduce to a node, which chooses a neighbour with an (approximately) equal pheromone concentration; but the direction in which such a neighbour lies is not constrained to a single ray. Moreover, even after making one such propagation step, a growing point does not have enough information to continue on the same tangent to the pheromone's level curve. In other words, there is no model of inertia built into the movement of a growing point. In this paper, we present a new GPL construct that provides propagation with inertia for growing points, and we discuss the methods that have been used to accomplish this in the past. We provide a quantitative comparison of the techniques to show that the new construct produces outcomes that are closer to geometric predictions than those of the previously used methods. We also discuss some drawbacks to this new construct and suggest ways in which it could be improved in a future implementation.**

Index Terms— **Algorithms, Amorphous Computing, Design, Experimentation, Programming Languages, Spatial Computing, Self-organising Patterns, Swarm Intelligence**

INTRODUCTION

An amorphous computer is made up of many, locally interacting parts (Abelson et al, 2000). Depending on the application, these parts could be nodes on a wireless sensor network, bacteria in a petri dish, computer processors and so on. Each unit of an amorphous computer is able to perform simple calculations, store some local state and communicate with its neighbours. Two nodes are neighbours if the distance between them is less than a user-defined radius, called the radius of communication. Through local interactions, the nodes of an amorphous computer work together to create a global result.

Developed by Coore (Coore, 1999), the Growing Point Language (GPL), is used to construct patterns of interconnect topologies on an amorphous computer. In the originally defined GPL, patterns are formed when the growing point, a phenomenon that effects behaviour in a node, leaves a trail of material as it moves through the medium from one node to the next. This movement is guided by virtual pheromones. Pheromones are diffused through the medium, starting from a source, assigning concentration values to all nodes in their reach. This creates level curves that are radially symmetric around the source. A growing point can sense the values of nodes in its neighbourhood, and move based on a tropism defined on it. When using tropisms that are tangential to the level curves of secretions, there are always two directions that are equally viable from a node's standpoint, so that growing points using this tropism can change its direction a number of times while propagating. In effect, a growing point has no inertia while moving through the amorphous computing medium. Currently, inertia can be modeled in GPL using the existing language features, but the inescapable uncertainty in the distribution of nodes leads to results that are sometimes geometrically counterintuitive. In this paper, we present a new GPL construct, called propagate-inertia, that explicitly implements a model of inertia in a growing point. We illustrate how it can be used, and we compare it with the methods that are already available. In order to quantify the comparisons, we have selected the problem of constructing a circular path as a specific example application where inertia is required. We provide three different implementations to solving this problem, and provide qualitative and quantitative comparisons to make the case that the new language construct provides a more reliable method of modeling inertia in GPL.

BACKGROUND

When a growing point moves from one neighbour to another, it is known as a propagation step. In the originally defined GPL, the propagation process first retrieves the concentrations of the relevant pheromones of all neighbours of the current point at which the growing point resides. The retrieval process returns the local ID of the relevant nodes, along with their pheromone values. The values are then assessed and a

This work was submitted for review on February 15, 2012.

Alyssa Morgan is an MPhil. Student in the Department of Computing at the University of the West Indies (Mona), Kingston, Jamaica. (e-mail: amorgan.edu@gmail.com).

Daniel Coore is a Senior Lecturer in the Department of Computing at the University of the West Indies (Mona), Kingston, Jamaica. (e-mail: daniel.coore@uwimona.edu.jm).

node whose value satisfies the tropism rule defined is chosen.

There are three types of tropisms defined in GPL to guide a growing point's movement, relative to one or more pheromone distributions. The tropism `ortho+ (ortho-)`, indicates movement orthogonal to the level curves of secretions, in a direction directly towards (away from) the source. The tropism `dia` causes a growing point to move tangentially to the level curve, essentially tracing it out. Radially symmetric secretions imply that in two dimensions, `dia` tropism causes a growing point to effectively orbit the source. A combination of these two tropisms, called `plagio+ (plagio-)` causes the growing point to move obliquely to a level curve, towards (away from) the source of the secretion.

The tropism `ortho+ (ortho-)` chooses the node with the greatest (smallest) value, directing the growing point directly towards or away from the source of the secretion. To satisfy the requirements for `dia`, the growing point chooses a node whose pheromone value is nearly equivalent to its own (within a defined tolerance). This causes a growing point to trace a given level curve, around the source. The tropism `plagio+ (plagio-)` combines the two tropisms mentioned above. First it collects all neighbours whose pheromone concentrations are larger (smaller) than the current node's then chooses the node with the median concentration among them. Once a successor is chosen, the propagation step is made to the chosen node. The choice can be marked by depositing a material, showing the growing point's path.

Growing points have no inertia. That is to say, there is nothing in the GPL computational state that maintains any directional information about where a growing point has visited, prior to its current position in the computational medium. For the purposes of this paper, we shall say that a growing point is in forward propagation, if it makes a step in the same direction as its previous step. The tropisms `ortho` and `plagio` exhibit a kind of pseudo-inertia because they ensure forward propagation of the growing point. The reason this occurs is that a secretion yields monotonically decreasing concentrations from the source, and therefore the neighbours of a node whose concentrations of a pheromone are larger than the propagating node's are necessarily closer to the source.

In contrast, the implementation of `dia` tropism identifies neighbours with approximately equal values of pheromone concentration. Those neighbours will lie on the same level curve as the propagating node, usually on either side of the node. Without any inertia, the predecessor of a node is equally likely a candidate for propagation as any other node would be with the same concentration level. This is most clearly illustrated by an attempt at drawing a circle in GPL by defining a growing point that has a dia tropism, relative to the pheromone secreted from a single distant point (which will act as the centre of the circle).

```
(define-growing-point (centre)
        (material red)
        (actions
                (secrete+ 15 circle-pheromone)))
```

The centre growing point will be coloured red, and its only action is to secrete a pheromone called circle-pheromone, for a distance of 15 hops. Note that the centre growing point does not move (there are no calls to propagate).

Naively, the code for the circumference would be defined as follows:

```
(define-growing-point (arc)
        (material blue)
        (tropism (dia circle-pheromone)
        (actions
         (propagate)))
```

The arc growing point will be coloured blue, and will move diatropically to a pheromone named circle-pheromone. The only computation performed at each node that the growing point visits is to determine its next location (i.e. it calls propagate as its only action).

Figure 1 illustrates the outcome of executing this GPL program. Observe that the growing point named arc gets stuck in oscillation between two points. This is because the pheromone concentrations of the two nodes are closer to each other than to that of any other node in either of their neighbourhoods. Each point then becomes the other's "best choice" for maintaining a constant pheromone concentration.

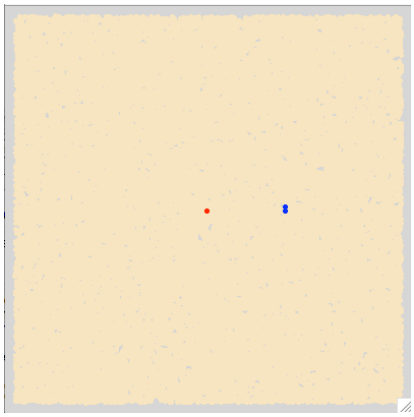


Figure 1 The results of using the originally defined propagate only, method M_o

In his PhD thesis, Coore showed how inertia can be modeled by giving a growing point a "self-repelling" tropism. Specifically, a growing point is given a tropism that is negatively orthotropic (`ortho-`) to a pheromone that the growing point itself secretes. (Coore, 1999). This method was also used by other authors as the way to model inertia in the originally defined GPL (D'Hondt, 2000) (D'Hondt & D'Hondt, 2001). The pheromone that it secretes is superposed on any phromone that has previously been laid down, so that there is more buildup of the pheromone along the line joining the predecessor and the current node of the growing point. This creates a natural preference for the growing point to propagate in the forward direction. This technique is shown below as a modification of the previously given code to draw a circle.

```
(define-growing-point (centre)
```

```
        (material red)
        (actions
                (secrete+ 15 circle-pheromone)))

(define-growing-point (arc)
        (material blue)
        (tropism (and (dia circle-pheromone)
                      (ortho- self-pheromone)))
        (actions
                (secrete+ 3 self-pheromone)
                (propagate)))
```

In order to give arc a self-repelling tropism along with its `dia`-tropism to the circle pheromone, the two tropisms were combined with the `AND` tropism operator. The `AND` combiner in GPL is not commutative. Each tropism is implemented as a function that takes a multi-set of pheromone concentrations and yields a subset that satisfies the appropriate criterion, sorted by the degree to which that criterion is satisfied. The `AND` operator applies its given specified tropism functions in reverse order, so that the first tropism given is the one that determines the final sorted order of the neighbourhood pheromone concentrations that qualify to be successor locations for the growing point. This means that there are two variations to implementing inertia purely with pheromones, which are derived by simply exchanging the order of the tropisms specified in the `AND` clause. The two methods would give rise to the following two variants of arc:

[1] Propagate method, $M_{diaortho}$

```
    …
    (tropism (and (dia circle-pheromone)
                  (ortho- self-pheromone)))
    …
```

[2] Propagate method, $M_{orthoda}$

```
    …
    (tropism (and (ortho- self-pheromone)
                  (dia circle-pheromone)))
    …
```

We shall use these two methods as references for comparing the behaviour of the new propagate-inertia primitive.

IMPLEMENTATION

The syntax of the new propagate-inertia command is the same as the original propagate command, namely it is given the values that should be passed to the growing point's next active site. Its semantics though, are that the next active site will be selected from among those neighbours that are located away from the direction in which the current active site's predecessor lies.

Apart from their pheromone concentrations, all nodes in a neighbourhood appear the same to a growing point when it is trying to propagate. For example, when using dia, all neighbours with pheromone concentrations approximately equal to that at the initial node's are suitable candidates, irrespective of direction. To introduce inertia, any spatial symmetry within pherommone values must be broken. This is done by creating a difference in the pheromone field surrounding the current active node, that depends on the direction the growing point came from. The idea is to take advantage of the difference in communication patterns of the nodes involved, to generate the desired inertia. Nodes that have previously communicated about propagating a growing point have different information from those hearing about the growing point for the first time. The new primitive, propagate-inertia, exploits this by effectively disabling nodes that have previously participated in propagating the growing point. Nodes are turned off after responding to a growing point, so that only newly responding nodes can be selected. This forces forward movement through the medium.

In its propagation, the growing point first requests a list of all its neighbours. One neighbour will be chosen as the successor, that is, the node that the growing point will now reside at. Once the node is chosen and the hop is made, the predecessor must be deactivated so that its successor does not return to it, stunting the growing point's movement. Furthermore, to prevent a change of direction in the movement of the growing point, all other neighbours that were being considered are deactivated as well. This ensures that only newly discovered nodes are considered.

Information of being deactivated is stored locally in each processor. When a processor responds to a query from a growing point, it stores that growing point's ID. It will then ignore any further queries for neighbours from that growing point instance. This ensures that only new neighbours respond to the query and so the successor to a growing point's active site will always be different from its predecessor, and moreover will likely be in the opposite direction..

In the simulator, this is done by creating a list of IDs belonging to the nodes that should be turned off for a particular growing point instance. Once a growing point has moved from one node to another all points considered during its propagation have their IDs added to the list of deactivated nodes.

To take its next step, the growing point once again makes a query for its neighbours. Its ID would have been stored locally by nodes that already responded to a query sent out by it, as a result, they will not respond to any more queries from it. This allows only newly discovered neighbours to take part in the propagation process. Using this method, previously considered neighbours are filtered from a successor's choices, forcing forward propagation of the growing point. With this method for the intended effect of a self-repelling pheromone is achieved by directly manipulating a node's computation. The following code fragment illustrates how our previous objective of drawing a circle would be implemented using the new propagate-inertia primitive:

```
define-growing-point (centre)
        (material red)
        (actions
```

```
                (secrete+ 15 circle-pheromone)))

(define-growing-point (arc)
        (material blue)
        (tropism (dia circle-pheromone)
        (actions
                (propagate-inertia)))
```

RESULTS

We set up a simulation of 30,000 nodes, each one of diameter 1 unit, in a square of 100x100 units. The radius of communication used was 2.5 units. A growing point set at (55.06, 49.94) was asked to trace a secretion with an extent of 15 hops, whose source was at (39.79, 49.84). The parameters for the simulator run are shown in **TABLE 1**.

TABLE 1 PARAMETERS OF SIMULATION RUN

Node of secretion, N_s	(39.79,49.9)
First point chosen	(55.06,49.94)
Radius of communication, r	2.5

We ran each method on a fresh simulation with exactly the same network and parameters. **Figure 2**, **Figure 3** and **Figure 7** show the the resulting pattern for each method of modeling inertia, using the same simulation parameters . The red dot shows the secreting node, and the blue dots show the nodes selected for propagation.

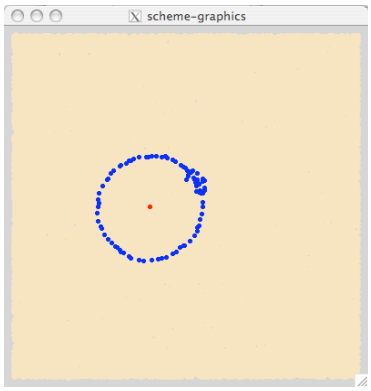


Figure 2 The results of using the originally defined propagate along with a self-repelling pheromone, method $M_{diaortho}$

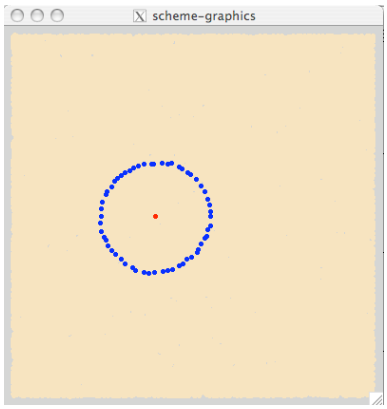


Figure 3 The results of using the newly implemented propagate, method $M_{inertia}$

Qualitatively, the overall pattern formed using the implemented propagate (**Figure 3**), gave approximately the same visual result as the method employing the self-repelling pheromone with preference to `dia` ($M_{diaortho}$, seen in **Figure 2**).

Figure 4, Figure 5 and Figure 6 show (in red) the angles that each point selected on the circumference made with the first point at the centre, for each method, on its run. It is plotted against the index (i) of the generated point (P_i), and the indices correspond to the order in which the points were generated. They also show (in blue) the distance from the centre point, of each point P_i against its index, i. The findings presented are typical of the results obtained on other runs.

Table 2, **Table 3** and **Table 4** show the minimum, maximum angle in degrees of the points generated, along with the standard deviation and average angle generated. The minimum, maximum, mean and standard deviation of the distance of a point from the centre are also shown. **Table 2** shows the nodes selected when using the newly implemented propagate, $M_{inertia}$. **Table 3** shows results for using the method $M_{orthoda}$; **Table 4** shows results when using method $M_{diaortho}$;

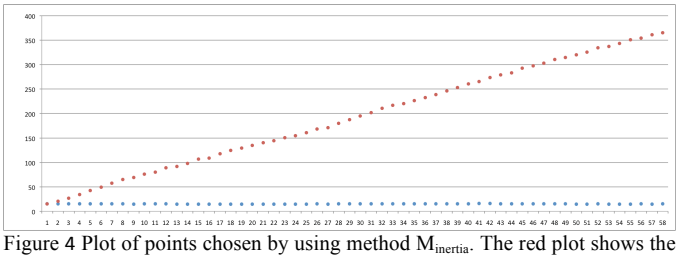


Figure 4 Plot of points chosen by using method $M_{inertia}$. The red plot shows the angle against the indexes and the blue plot shows the radius of the point against its index

Table 2 Results for method $M_{inertia}$.

	$\lvert P_i\text{-}N_s\rvert$	Difference in Angle
max	15.21	6.1
min	14.58	1.9
standard dev	0.31	1.84
mean	15.21	6.1

In analysing the data of **Table 2**, as well as the generated graph (**Figure 4**) we can see the points chosen for propagation using method $M_{inertia}$. This method completed a full trace of the level curve, getting stuck when it returned to its starting point. Also, the growing point never changed its direction. Variations in the distance of points from the center were minimal. The standard deviation obtained was 0.31, showing that the points were on a nearly perfectly circular locus.

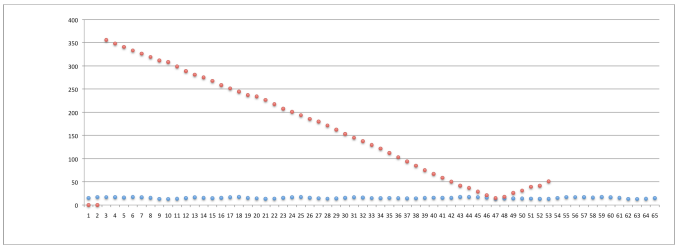


Figure 5 Plot of points chosen by using method $M_{orthodia}$. The red plot shows the angle against the indexes and the blue plot shows the radius of the point against its index

Table 3 Results for method $M_{orthodia}$.

	$\|P_i\text{-}N_s\|$	Difference in Angle
max	17.45	10.00
min	13.01	-9.49
standard dev	1.31	4.85
mean	15.37	7.6

Figure 5 shows the points generated using method $M_{orthoda}$. The average angle generated when using this method, is always larger that that of method $M_{diaortho}$ causing it to close its path in fewer points. This is because method $M_{orthoda}$ often cuts through the circle (**Figure 7**) instead of staying confined to the circumference. This causes larger deviations in the distance of points from the centre (**Table 3**), and a pattern that has a less circular shape.

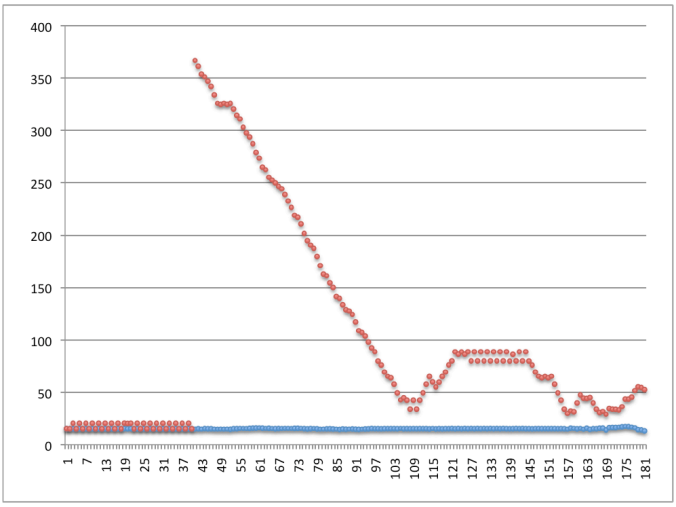


Figure 6 Plot of points chosen by using method $M_{diaortho}$. The red plot shows the angle against the indexes and the blue plot shows the radius of the point against its index

Table 4 RESULTS FOR METHOD $M_{diaortho}$

	$\|P_i\text{-}N_s\|$	Difference in Angle
max	17.4	351.9
min	13.2	-8.9
standard dev	0.5	27.0
mean	15.3	0.2

When a self-repelling pheromone was used along with the originally defined propagate (**Table 4**), the growing point changed its direction a number of times during the run as seen in Figure 6. While the growing point eventually completes its trajectory, it often goes backwards and forwards during the trace, until finally getting stuck at a neighbour that has no suitable options. Unidirectional movement, therefore, is not guaranteed with this method. This can first be seen in the generated nodes 1-39, where the growing point begins by oscillating between two nodes, changing its direction on every hop. As with $M_{inertia}$, the standard dev in the distance of points from the centre is relatively small (0.5), showing that the points are in a circular shape.

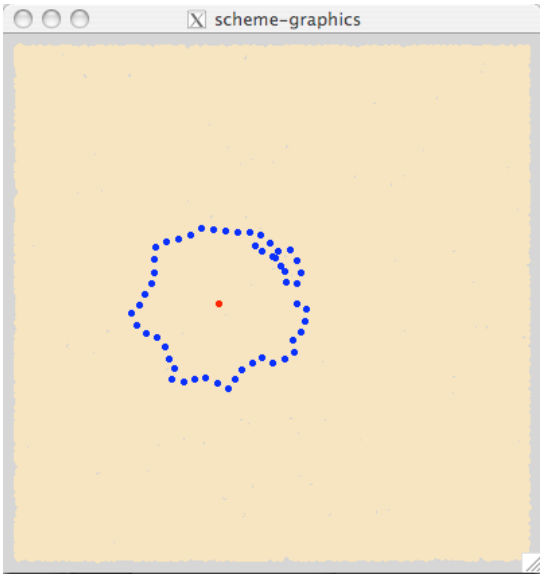


Figure 7 Results of using the method $M_{orthodia}$

We have shown use of a self-repelling pheromone to promote forward propagation in two ways. The first passes neighbouring points through the `dia` filters and sorters and then those of the `ortho-`. Using this method, it was shown that a circular trace was obtained, however, the growing point did not commit to one direction for its movement. This is due to the fact that the growing point will first filter out neighbours of equal values and then from that list, choose the node with the smallest value of the self-pheromone. Naming `dia` first means that precedence is given to satisfying its rule, and then to the chosen direction. The second method described, passes the points through the `ortho-` filter, and then the `dia` filter. It was shown that this forces unidirectional movement, but sacrifices a circular shape. By naming `ortho-` first, it is ensured that the growing point will move away from the secretions of its self-pheromone, resulting in unidirectional movement. After the directional requirement has been satisfied, a point of nearly equal value is chosen. However, since the node chosen by `ortho-` might not have been the optimal node to continue the circular shape, the global result shows a distorted circle. The method $M_{inertia}$ combines the positive aspects of these two methods, achieving both unidirectional movement and a circular trace.

DISCUSSION AND CONCLUSION

$M_{inertia}$ described in this paper forces the trace of a level curve and forward movement of the growing point and is an improvement on the previous method used to get a similar effect. The secretion of a self-repelling pheromone on every hop does encourage forward motion, but cannot guarantee it. That method increases the time to run the simulation as the growing point changes direction a number of times, before completing the trace. Furthermore, the use of a self-repelling pheromone is more taxing on the system, as a secretion has to be laid down on every step. The same global effect can be obtained using the newly implemented propagate, but with greater efficiency in resource management.

It is arguable that for some GPL programs, permanent deactivation of nodes towards a certain growing point would be unfavourable. If points are to be reactivated, they must change that state after the trace is complete. To test this, the implemented propagate was used, however, points stayed deactivated for only one hop, after which, they were reactivated. Results show that instead of getting stuck upon reaching the starting point, or even selecting a different node, a growing point selects the same point and proceeds to take exactly the same path as before, perpetually moving in a self-loop. This modification to the implementation is possible only in the simulator though, because it managed the deactivation and reactivation of nodes in a manner that was not necessarily consistent with the local requirements of amorphous computing (i.e. a node would become reactivated if its neighbour's neighbour successfully propagated). A realistic implementation of this variation of $M_{inertia}$ would probably use a small timeout to implement the temporary disabling of a node – much like the way neurons operate in the brain. However, GPL currently has no model for time, and therefore even this modification would have required extensive changes to the GPL interpreter. Given the results obtained in this paper, it seems worthwhile to implement such a temporal dimension to GPL in the future.

As currently implemented, $M_{inertia}$ can be used in programs that require a growing point to work with other growing points in order to form a pattern because nodes are selectively disabled only to a specific instance of a growing point. That means that those nodes may still respond to other growing points (even different instances of the same growing point that caused them to become unresponsive in the first place) and therefore facilitate almost all of the previously supported modes of interaction between growing point trajectories. Only patterns that relied upon a growing point interacting with its own trajectory would require a different implementation, if they were to take advantage of the new propagate-inertia command.

The current implementation causes a node's response to a growing point to be "digital", that is, either active or inactive. The purely pheromone based approach to modeling inertia was "analogue" in that it provided a means for a node to retain (by degrees) the potential to be an active site for a growing point, which might be desirable in certain situations (e.g. in order to make a pattern robust to node death or failure).

Our propagate-inertia primitive is reminiscent of the physical notion of inertia, in that growing points are strongly encouraged to keep moving in the same direction. However, a growing point still depends upon a tropism in order to move, unlike in Physics where inertia allows a particle with an initial velocity to move, in the absence of friction, in a straight line to infinity. But our propagate-inertia primitive is unable to make a growing point move indefinitely because a growing point requires the presence of a pheromone to which its tropisim is sensitive in order to make a step. This is one important difference between our model of inertia and physical inertia. Interestingly, in Physics, the phenomenon of friction is modeled as a force that is present only when there is velocity (or at least an attempt at initiating it). In GPL, the propagate-inertia primitive provides inertia only in the presence of a pheromone. It will probably be worthwhile investigating whether these physical analogies in GPL can be deepened to the point of using Physical laws to make interesting predictions about GPL programs.

REFERENCES

[1] Abelson, H., Allen, D., Coore, D., & Hanson. (2000). Amorphous computing. *Communications of the ACM , 43* (5), 74-82.

[2] Coore, D. (1999). Botanical computing: a developmental approach to generating interconnect topologies on an amorphous computer. *Phd. Thesis* . Massachusetts Institute of Technology.

[3] D'Hondt, E. (2000). Exploring the amorphous computing paradigm. *Master's thesis in computer science, Vrije University, Brussel* .

[4] D'Hondt, E., & D'Hondt, T. (2001). Experiments in amorphous geometry. *2001 International Conference on Artificial Intelligence*, 285-290..

Declarative Multidimensional Spatial Programming

John Plaice[1,2] Jarryd P. Beck[1] Blanca Mancilla[1]
[1]School of Computer Science and Engineering, UNSW, Australia
[2]Department of Computer Science and Software Engineering,
Concordia University, Canada

***Abstract*—We extend the declarative multidimensional language TransLucid so that TransLucid systems become first-class values. These systems can then be placed in a multidimensional grid, thereby enabling multidimensional spatial computing. TransLucid systems can appear as functions inside other systems, or as standalone systems which are programmed by receiving new declarations from other standalone systems.**

I. Introduction

This paper presents the *TransLucid system*, a mechanism for defining reactive systems which can be arranged in a multidimensional grid. The proposed mechanism extends the TransLucid programming language, in which variables denote multidimensional arrays of arbitrary rank and extent. A TransLucid system is a reactive system which, at each discrete instant, calculates finite arrays as output from finite arrays as input, using a system of equations. TransLucid systems can be defined in a functional manner or in an object-oriented manner; in the latter case, the systems can program each other by mutually sending new declarations for future instants, and a runtime system is needed to manage these interactions.

In the 2006 Dagstuhl seminar introducing the concept of *space-oriented computation* [4], now known as spatial computing, it was concluded that there is a need to grasp the distribution of computation through space, be it at a global level or at the level of an individual die. For it to be possible to put a handle on this concept, then there is a need for space to be structurable using an arbitrary set of dimensions, be they physical dimensions defining location and altitude/depth to an arbitrary level of detail, or other, virtual dimensions providing information more specific to a given task.

The intuition which drives this paper is that the structuring of space for distributing computations should be the same as the structuring of data in TransLucid, and so to prepare for spatial computing, it suffices to extend this language. Specifically, reactive systems need a notion of time, which here will be a *clock dimension*, passed as parameter to systems; and object-oriented systems need a mechanism to exchange their messages with each other, which here will be a *declaration multiplexer*, also passed as parameter.

Before we can imagine extending TransLucid, we must understand the nature of variables and their evaluation, as well as the behavior of functions. In TransLucid (*cf.* §II), a variable X is understood to vary, conceptually, in *all* possible dimensions; this means, say, that if X is defined using two dimensions d_1 and d_2, and Y is defined using two dimensions d_2 and d_3, then both can be considered to vary in all three dimensions: the "variance" of X in d_3 is constant, as is the "variance" of Y in d_1. As for their sum, $X + Y$, it varies in all three dimensions. (We say that the *ranks* of X, Y and $X + Y$ are, respectively, $\{d_1, d_2\}$, $\{d_2, d_3\}$ and $\{d_1, d_2, d_3\}$.) This approach is consistent with the use of dimensions in differential equations, in which one only writes down the dimensions of relevance, and with the use of attributes in the universal relation model used to define the semantics of relational databases [6].

The evaluation of an expression in TransLucid is always undertaken with respect to a runtime multidimensional context, an unordered set of $(dimension, ordinate)$ pairs, in which any atomic value may play the role of dimension. Should a variable appear in an expression, it is evaluated lazily, and is indexed by the context. To define the variance of variables with respect to the context, the latter can be perturbed by changing some of the $(dimension, ordinate)$ pairs defining it. The runtime context permeates an entire program, and dimensions are queried using dynamic binding.

TransLucid is a functional language, hence functions are first-class values, and variables can denote entire arrays of functions, which correspond to families of functions in mathematics. Entire variables can be passed to functions by name. Dimensions and constant values, on the other hand, are passed to functions by value. As for the body of a function, the key question is whether it should be evaluated with respect to the context when the function is created or to the context when the function is applied. It turns out that most of the time the application context is what is needed, but the programmer can bind the ordinates of a set of named dimensions from the context at the time of creation of an abstraction.

To extend TransLucid to address the problem of spatial computing, we propose to extend the notion of function: we add two new kinds of abstraction, together called *system abstractions*, where 1) the input arrays of *finite*—as opposed to arbitrary—rank and extent are passed by value, 2) the dimensions and extent of these arrays are passed by value, and 3) the named dimensions whose ordinates are to be bound correspond to the dimensions structuring space.

The system abstractions are, like other TransLucid abstractions, first-class values, so it is possible to define variables whose values are multidimensional arrays of systems. Hence, simply by adding new forms of abstraction to TransLucid, i.e.,

the system abstractions, spatial computing comes for free.

Of course, some dimensions do have an immediate physical interpretation. This is the case, for example, for the `time` dimension, which must always be causal, i.e., at no point may one refer to a future instant, with respect to `time`. We envisage that in the future, with more applications, there will be a need to add other physical dimensions.

The first system abstraction (§III), designed to appear directly in the system of equations of another system, has a completely functional interface: the system of equations defining it is given once, and cannot be changed. When an inner system is created by the outer system, a causal time dimension is passed by the outer system to the inner system, which then uses it as its own, internal `time` dimension. That inner system then becomes a function mapping streams of finite arrays to streams of finite arrays.

The second system abstraction (§IV), designed for standalone use or for peer-to-peer interactions, has an object-oriented interface, following the intuition of Alan Kay in his design of Smalltalk [5]: an object is a mapping from a stream of input messages to a stream of output messages. In TransLucid, the 'messages' are TransLucid declarations, tagged with a range defining which systems will be recipients of the message.

For the moment, interaction between systems is considered to be synchronous. At each instant, a declaration-multiplexer value collects all of the messages output from the different systems, and passes them on to the appropriate recipients before the beginning of each instant.

This second kind of system abstraction is a generalization of the current implementation of a TransLucid standalone system, as is incorporated in the current TransLucid interpreter.

II. Background

Development of the TransLucid programming language began in late 2005. It is the latest descendant of Ashcroft and Wadge's Lucid [1]. A historical recollection of the developments from Lucid to TransLucid is given in [8]. The denotational semantics and implementation of TransLucid, including full higher-order functions over arbitrary-dimensional arrays, are presented in [7]. The problem of memoizing arbitrary-dimensional data, as needed by TransLucid, is solved in [3]. The current interpreter and some running examples can be found at `translucid.web.cse.unsw.edu.au`.

Given that the evaluation of TransLucid expressions is unusual, we spend some time explaining it. An expression, such as '42', '#.0 + #.1' or '#.1 + X', varies conceptually in all dimensions, although any given variable will have finite rank. If the rank of X is $\{0\}$, i.e., X is a 0-stream (a stream varying in dimension 0) of the form $\langle x_0, x_1, x_2, \ldots \rangle$, then here is the partial variance of these three expressions with respect to dimensions 0 and 1.

		0	→								
42		0	1	2	3	4	5	6	7	8	⋯
1	0	42	42	42	42	42	42	42	42	42	⋯
↓	1	42	42	42	42	42	42	42	42	42	⋯
	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋱

		0	→								
#.0 + #.1		0	1	2	3	4	5	6	7	8	⋯
1	0	0	1	2	3	4	5	6	7	8	⋯
↓	1	1	2	3	4	5	6	7	8	9	⋯
	2	2	3	4	5	6	7	8	9	10	⋯
	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋱

		0	→				
#.1 + X		0	1	2	3	4	⋯
1	0	x_0	x_1	x_2	x_3	x_4	⋯
↓	1	$1 + x_0$	$1 + x_1$	$1 + x_2$	$1 + x_3$	$1 + x_4$	⋯
	2	$1 + x_0$	$2 + x_1$	$2 + x_2$	$2 + x_3$	$2 + x_4$	⋯
	⋮	⋮	⋮	⋮	⋮	⋮	⋱

An expression is evaluated with respect to a runtime context, a set of $(dimension, ordinate)$ pairs. If we consider the three above examples, the symbol '#' corresponds to the current context, and '#.1' applies the current context to the value 1, i.e., queries the 1-ordinate; expression '42' has rank $\emptyset$, and has value 42 in all contexts; expression '#.0 + #.1' has rank $\{0, 1\}$, and has value $\kappa(0) + \kappa(1)$ in all contexts κ where $\{0, 1\} \subseteq \mathrm{dom}(\kappa)$; and expression '#.1 + X' has rank $\{1\} \cup \mathrm{rank}(X)$, and has value the sum of $\kappa(1)$ and the value of X in κ, assuming $1 \in \mathrm{dom}(\kappa)$.

The methodology of programming in TransLucid is still under development. We have found, nevertheless, that programming with conceptually infinite data structures offers interesting possibilities. For example, consider calculating the factorial of 6 in a two-dimensional triangular grid:

		d	→								
$fact$.6		0	1	2	3	4	5	6	7	8	⋯
t	0	1	1	2	3	4	5	6	1	1	⋯
↓	1	1	6	20	6	1	1	1	1	1	⋯
	2	6	120	1	1	1	1	1	1	1	⋯
	3	720	1	1	1	1	1	1	1	1	⋯

The sequence 1..6 is embedded in a sea of 1s in order to create an infinite sequence. Then, in 3 iterations (the integer logarithm, base 2, of 6), a new sequence is created by multiplying successive pairs of integers from the previous sequence. Here is the TransLucid program, using higher-order functions.

```
fun fact.n = tournamentOp_1.d.n.times
                 (default_1.d.1.n.1 (#.d))
where
  dim d ← 0
end
fun default_1.d.m.n.val X = Y
where
  var Y [d : m..n] = X
  var Y [d : int] = val
end
fun tournamentOp_1.d.n.f X = Y
where
  dim t ← ilog.n
  var Y = fby.t X (f.(LofPair.d Y).(RofPair.d Y))
end
```

$$\begin{aligned}
&\texttt{fun } \mathit{fby}.d\ X\ Y = \texttt{if } \#.d \equiv 0 \texttt{ then } X\\
&\qquad\qquad\qquad\texttt{else } Y\ \texttt{@}\ [d \leftarrow \#.d - 1]\ \texttt{fi}\\
&\texttt{fun } \mathit{LofPair}.d\ X = X\ \texttt{@}\ [d \leftarrow \#.d * 2]\\
&\texttt{fun } \mathit{RofPair}.d\ X = X\ \texttt{@}\ [d \leftarrow \#.d * 2 + 1]
\end{aligned}$$

In TransLucid, one can use identifiers for dimensions (`dim`) or for variables (`var`); functions are a kind of variable. A dimension identifier will, during evaluation, be replaced by unused dimensions, which are atomic values. Variables, on the other hand, are multidimensional objects.

Function $\mathit{fby}.d\ A\ B$ assumes that A and B are d-streams, and prepends, in the d direction, the first element of A to B. Function $\mathit{LofPair}.d\ A$ (respectively, $\mathit{RofPair}.d\ A$) assumes that A is a d-stream, and returns, in the d direction, every second even (respectively, odd) element of A.

The $\mathit{default}_1.d.m.n.\mathit{val}\ X$ function assumes that X is a d-stream, and creates a new d-stream, where elements m through n are recopied from X and all the other elements have value val.

The $\mathit{tournamentOp}_1.d.n.f\ X$ function assumes that X is a d-stream, and that the first $2^{\lceil \log_2(n) \rceil}$ elements of X will be used to calculate a result. There will be $\lceil \log_2(n) \rceil$ iterations ($\mathit{ilog}.n$ in the code) in the t direction. In iteration $[t \leftarrow i]$, element $[t \leftarrow i, d \leftarrow j]$ of Y is equal to function f applied to elements $[t \leftarrow i-1, d \leftarrow 2*j]$ and $[t \leftarrow i-1, d \leftarrow 2*j+1]$. The line

$$\texttt{dim } t \leftarrow \mathit{ilog}.n$$

declares local dimension identifier t, and the initial ordinate for the corresponding dimension will be $\mathit{ilog}.n$.

The `fun` $\mathit{fact}.n$ function creates an infinite stream by embedding the sequence $1..n$ in a sea of 1s and then applies tournament computation, where function f is multiplication.

Note that some arguments are preceded by a period (.), while others are preceded by a space. The arguments preceded by a period are *base parameters* (typically dimensions and constants), which are the parameters of *base functions*; the arguments are passed by value and the body is not evaluated with respect to the application context. The arguments preceded by a space are *name parameters*, which are the parameters of *name functions*; the arguments are passed by name and the body is evaluated with respect to the application context.

As we have seen, functions can be first-class values, hence it is possible to have arrays of functions. In the next example, the n-th power function is created by extracting it from a stream, varying in local dimension d, of the successive power functions, starting from the 0-th power.

	dim d $\rightarrow$			
P	0	1	2	$\cdots$
	$\lambda^{\mathrm{b}}\ m \rightarrow m^0$	$\lambda^{\mathrm{b}}\ m \rightarrow m^1$	$\lambda^{\mathrm{b}}\ m \rightarrow m^2$	$\cdots$

Here is the text in TransLucid:

```
fun pow.n = P
where
  dim d ← n
  var P = fby.d (λ^b m → 1) (λ^b {d} m → m × P.m)
end
```

There are two anonymous base-function abstraction creators here. The second one, '$(\lambda^{\mathrm{b}}\ \{d\}\ m\ \rightarrow\ m \times P.m)$', has two kinds of parameter: the m is the value which will be raised to a certain power, and the $\{d\}$ corresponds to the set of dimensions—in this case, the single dimension d—whose ordinates are bound at the time of creation of the application.

From these examples, it follows that the idea of a function in TransLucid is not simple. There are in fact three separate aspects: 1) Is the body of the abstraction evaluated with respect to the application context, or to an empty context? 2) Is there a normal parameter passed as argument? and 3) If there is a normal parameter, is it passed by value or by name? It turns out that only four combinations are useful:

1) base fns (λ^{b}): empty context, value parameter;
2) value fns (λ^{v}): application context, value parameter;
3) name fns (λ^{n}): application context, name parameter;
4) intensions ($\uparrow$): application context, no normal parameter.

In all four cases, a set of dimensions whose ordinates should be bound at the time of creation of the application can be passed as well. (See [7] for more details.) In the above examples, we have seen base functions and name functions.

In the current implementation, a TransLucid system consists of a set of declarations which grows over (discrete) time. There is a special dimension, called `time`, which enumerates the instants. At each instant, a new set of declarations can be added to the system.

In fact, a system is entirely constructed from declarations. There are declarations for dimensions (`dim`), for operators appearing in expressions (`op`), for inductively defined datatypes (`data` and `constructor`), and for variables and functions (`var` and `fun`). The `op` declaration, for example, introduces a new lexeme for the parser, and specifies its associativity and precedence, as well as its mapping to a function in the host language, currently C++. (See [2] for more details.)

All symbols and identifiers may have multiple declarations. As a result, a symbol or identifier, at the syntactic level, can be understood to denote a multidimensional array of declarations, also indexed by the context. When an expression is being evaluated in a given context, and a symbol or an identifier appears in the expression, then the *bestfit* declaration for that symbol or identifier, with respect to the current context, is selected. Bestfitting takes into account the time validity of declarations, and all other things being equal, will take the latest declaration.

Hence, adding new declarations to a system can change the meaning of all operators and identifiers, effectively reprogramming the system. However, the past of the system cannot be changed. An attempt to recalculate a result from a previous instant will use the declarations available to the system at that previous instant.

The goal of this paper thus becomes to formalize this existing notion of system, and then to make it possible for a system to have further systems within it, or to set up a number of systems to be mutually programmed, in such a way that these systems are first-class TransLucid objects that resemble the existing abstraction mechanisms of TransLucid.

III. The simple system

A simple TransLucid system, called a *functional TransLucid system*, maps streams of input arrays to streams of output arrays, where the rank and extent of all arrays are finite. For this to work, a system acts as a discrete reactive system which, at each instant $i \in \mathbb{N}$, takes as input, say, n inputs X_{ij}, $j = 1..n$, and produces an output Y_i. In that sense, the behavior of the system for each instant i is then like that of a base function, except that the inputs are finite arrays that are used as ordinary TransLucid arrays inside the equations of the system.

Several new concepts are introduced for the functional TransLucid system to be definable. First, if a functional system is to be used within another, then the time dimension of the inner system cannot be expected to be the same as the time dimension of the outer system. Since it is the outer system that would call the inner system, it follows that the outer system must provide to the inner system its time dimension as a parameter.

Second, if, within an instant, the functional system is to effectively act as a base function, this means that the input arrays for that instant need to be bundled up so that they can be passed in as complete input blocks to the system. This requires mechanisms both at the abstraction and the application levels to define the arrays that must be built up. The same holds true for defining the output of the system.

Before we can write down the interface for the functional system, we therefore need two new features. First is a *clock dimension*, or causal dimension, which can be used as the clock of a system. The ordinates of a clock dimension must always be natural numbers, and a change of context during the evaluation of an expression can never refer to an ordinate for the clock dimension that is greater than the current ordinate. A clock dimension is introduced with the syntax:

$$\texttt{clock}\ t$$

declaring identifier t to be a clock dimension.

Second is the *block array calculation*, which forces an expression to be evaluated over an entire multidimensional region. Its syntax is of the form:

$$E\ \$\ [region]$$

where the $region$ defines a multidimensional region using the same syntax as for guards in guarded equations (see variable Y in the definition of function $default_1$, §II).

We present the functional system abstraction through an example:

$$\texttt{fun}\ S\ \hat{}ck.d.n.\big(X\$[d:0..n]\big) = E$$

which could also be written:

$$\texttt{var}\ S = \lambda^{\text{fs}}\ ck \to \lambda^{\text{b}}\ d \to \lambda^{\text{b}}\ n \to \lambda^{\text{b}}\ \big(X\$[d:0..n]\big) \to E$$

There are three kinds of parameters:

1) ck, introduced by a $\hat{}$, is a clock dimension parameter;
2) d and n, introduced by a period, are base (call-by-value) parameters; and
3) $X\$[d:0..n]$, introduced by a period, is a base parameter, annotated with its rank and extent, which is used inside the system as an intension.

As for the output E, it must be of the form

$$E'\ \$\ [region]$$

In all cases, both the inputs and the outputs must be finite.

To use this abstraction, we would write an expression of the following form:

$$\texttt{var}\ B = S\ \hat{}t.d_1.n_1.\big(A\$[d_1:0..n_1]\big)$$

where

1) t, which must be a clock dimension, is the actual parameter corresponding to formal parameter ck;
2) d_1 and n_1 are the actual parameters corresponding, respectively, to formal parameters d and n; and
3) $A\$[d_1:0..n_1]$ is the actual parameter corresponding to formal parameter $X\$[d:0..n]$.

It is supposed that the rank of A is $\{t, d_1\}$, so the current t-ordinate and the region $[d_1:0..n_1]$ are sufficient to compute any element of A.

In the above example, A would be a t-stream of arrays of rank and extent $[d_1:0..n_1]$, and B would be a t-stream of arrays whose rank and extent would be defined by the output of the S abstraction.

For a functional system to be usable in a spatial computing context, there needs to be a mechanism for multiple, similar, functional systems to be defined. For this to be possible, we can reuse the "bound dimension" feature of the other abstractions (*cf.* the pow example in §II), in which the ordinates of a named set of dimensions from the context at the time of creation of an application are bound.

We can adapt the above example so that the outer system has two "space" dimensions, d_r and d_c. Then

$$\texttt{fun}\ S\ \hat{}\ \{d_r, d_c\}\ ck.d.n.\big(X\$[d:0..n]\big) = E$$

allows the creation of a $\{d_r, d_c\}$ grid of S systems, where the output expression E could take into account dimensions d_r and d_c to create slightly different systems, depending on where they appear in the grid.

Once it becomes possible to define an arbitrary-sized multi-dimensional grid of functional systems, where the dimensions of the grid can correspond either to physical or virtual attributes, we need to experiment. The obvious example is that of *systolic arrays*. Here we present a systolic matrix multiplier, in which a function incorporates a network of functional systems:

$$\begin{array}{l}
\texttt{fun}\ systolicmul.d_1.d_2.n\ X\ Y = \\
\quad S\hat{}\,t.X''.Y''\ @\ [t \leftarrow 3 \times n] \\
\texttt{where} \\
\quad \texttt{clock}\ t \\
\quad \texttt{var}\ X' = default_2.d_1.0.(n-1).d_2.0.(n-1).0\ X \\
\quad \texttt{var}\ Y' = default_2.d_1.0.(n-1).d_2.0.(n-1).0\ Y \\
\quad \texttt{var}\ X'' = X'\ @\ [d_2 \leftarrow \#.t - \#.d_2 - \#.d_1] \\
\quad \texttt{var}\ Y'' = Y'\ @\ [d_1 \leftarrow \#.t - \#.d_1 - \#.d_2] \\
\quad \texttt{fun}\ S\ \hat{}\ \{d_1, d_2\} ck.U.L = A \\
\quad \texttt{where} \\
\quad\quad \texttt{var}\ A = fby.ck\ 0\ \big(A + next.ck\ (U \times L)\big) \\
\quad \texttt{end} \\
\texttt{end}
\end{array}$$

In systolic algorithms, the data processing units (DPUs) are arranged in a two-dimensional grid, and the data moves through the grid in a lock-step clocked manner. In TransLucid, three variables are needed for the definition of a systolic array: one for the grid of DPUs (here S), one for the data flowing horizontally (here X), and one for the data flowing vertically (here Y).

The $systolicmul.d_1.d_2.n\ X\ Y$ assumes that both X and Y are $n \times n$ matrices varying in dimensions d_1 and d_2. Each of X and Y is embedded in a two-dimensional sea of 0s to produce, respectively, X' and Y'. Then X' (resp. Y') is staggered in time with respect to dimension d_2 (resp. d_1) to ensure the correct arrival of the elements of X (resp. Y) at the processing systems. Each system is trivial, doing a multiply of its two inputs and adding the product to a running total. The results are ready once $3 \times n$ instants have elapsed.

IV. The programmable system

A programmable TransLucid system, called an *object-oriented TransLucid system*, maps streams of *input declarations* to streams of *output declarations*. It is designed to be a generalization of a functional TransLucid system, so it also maps streams of input arrays to streams of output arrays, where the rank and extent of all arrays are finite. The input declarations are used to change the internal behavior of the system, i.e., the set of equations defining the system, over time, and the output declarations are used to affect the internal behavior of other systems.

The intuitions leading to the invention of the object-oriented system are twofold. First, the existing implementation for TransLucid already works with the idea that a user programs a *system*, in which, at every discrete instant t, new declarations may be added to the system by the programmer, and all of the valid declarations for that instant, including ones from previous instants as well as the new ones, are taken into account for building up the system of equations for that instant.

Second, the semantics for object-oriented programming, as envisaged by Alan Kay for Smalltalk, was that an object is a function mapping a stream of input messages to a stream of output messages, and that the stream of internal states of the object was simply a byproduct of this function.

Combining these two intuitions, and maintaining declarativity, requires that the updates to the set of declarations defining a system be synchronous and discrete, i.e., updates take place in discrete transactions.

Assuming that the input declarations have been passed to a system, handling them is straightforward.

The key problem to be solved, therefore, is how to generate output declarations from the evaluation of a declarative system of equations, and then distributing those output declarations to other systems as input declarations for the latter.

The solution is also twofold. First, the output declarations are built up by generating side-effects during the evaluation of the expressions required to produce output during an instant. Since evaluation of TransLucid is done in a demand-driven manner, only those side-effects needed will be generated, and, further, multiple copies of the same side-effect are equivalent to one copy, hence re-evaluation of an expression can have no negative consequences.

Side-effects are generated in TransLucid by using the ";" operator. The expression $E_1; E_2$ means evaluate E_1, then E_2, and return the value from E_2. So, if E_1 is a side-effect, then it will be generated, and computation continues with the evaluation of E_2.

Second, a new kind of value is introduced: the *declaration-multiplexer* value. Here is an example declaration:

$$\texttt{multidecl}\ dm\ ck\ \{d_r, d_c\}$$

which states that identifier dm is a declaration-multiplexer. At each instant t of clock dimension ck, for every point $[d_r \leftarrow i, d_c \leftarrow j]$, there is a set of declarations $D = dm_{t,i,j}$. The set D has two components: $D.in$ and $D.out$ are respectively the set of input declarations and the set of output declarations at instant t for the system indexed by $[d_r \leftarrow i, d_c \leftarrow j]$.

A family of object-oriented systems is introduced by the syntax:

$$\texttt{fun}\ S\tilde{}\{d_r, d_c\}\ dmpar = E$$

which could also be written

$$\texttt{var}\ S = \lambda^{\text{OOS}}\ \{d_r, d_c\}\ dmpar \rightarrow E$$

which means that the declaration multiplexer formal parameter $dmpar$ is used by the family S of systems, and the two "space dimensions" d_r and d_c are used to parameterize a grid of object-oriented systems.

Adding to $dmpar.out$ is done by evaluating expressions of the form $dmpar.addDecl(guard, idOrSymb, declExpr)$, where $guard$ designates which of the systems in the multi-dimensional grid defined by the bound dimensions d_r and d_c will be informed in the next instant of the $decls$; $idOrSymb$ is the identifier or symbol being declared, and $declExpr$ is some kind of expression actually providing the declaration.

At the end of instant t, each system indexed by $[d_r \leftarrow i, d_c \leftarrow j]$ will have produced its $dmpar.out$ component. The multiplexer dm will then run a builtin functional system which, reading the guards, produces the set of input declarations for each system from the set of output declarations from all of the systems.

The infrastructure proposed above allows for the declarative programming of object-oriented systems, with no explicit change of state at any level, only the passage of time. In fact, it becomes possible to define a multidimensional grid of synchronous multi-agent systems, where at each instant, the different systems can assess their current state and provide new declarations both for themselves and for other systems.

Given the novelty of this proposed model of programming, many programming experiments will need to be undertaken before a methodology is developed.

Nevertheless, we do provide the following toy example. There are two systems, S_1 and S_2, where S_2 is setting the precedence of the plus operator in S_1 to be the ordinate of its clock at each instant. The result computed by S_1 may therefore change from one instant to the next, because its expression will be reparsed at each instant. Note that both

S_1 and S_2 bind dimension d, so that S_1 reads from dm at d taking value 1, and S_2 reads from dm at d taking value 2. In this way, system S_2 simply needs to write to its parameter $dmpar$ at location $[d \leftarrow 1]$ to program system S_1.

```
clock t
multidecl dm t {d}
fun S1~ {d}dmpar^ck = 1 + 2 − 3 × 4 + 5**6
fun S2~ {d}dmpar^ck =
dmpar.addOp([d : 1], " + ",
                OpInfix."plus".false.AssocLeft.(#.ck))
var demand [slot : 1] = S1~dm^t @ [d ← 1, t ← #.time]
var demand [slot : 2] = S2~dm^t @ [d ← 2, t ← #.time]
```

In the textual interpreter, variable $demand$ is a special variable keeping track of all of the current demands. Dimension $slot$ is used to order the different demands.

V. Conclusion

The guiding principle of this article is that if spatial computing is to have broad success, then space needs to be structured, and our approach is to use multiple dimensions.

To add spatial computing to TransLucid, we simply extend the idea of the multidimensional array to the building of "systems", so that there can be multidimensional families of systems.

For functional systems, we add a primitive for packaging arrays of finite rank and extent into single chunks, and introduce a clocked dimension, which acts in a causal manner.

For object-oriented systems, we add the ability to generate side-effects while evaluating an expression, and ensure that if the same side-effect were to be generated several times, that it only be counted once. We also introduce the declaration multiplexer, which distributes the declarations generated from within a family of systems as output to all of the systems that should receive these declarations in the next instant as input.

The solutions proposed in this paper need to be implemented, and much experimentation with programming methodology needs to be undertaken.

References

[1] Edward A. Ashcroft and William W. Wadge. Lucid, a Nonprocedural Language with Iteration. *Communications of the ACM*, 20(7):519–526, 1977.

[2] Jarryd P. Beck, Blanca Mancilla, and John Plaice. TransLucid user documentation, version 0.3.0, August 2012. `http://translucid.web.cse.unsw.edu.au/TL-doc-0.3.0.pdf`.

[3] Jarryd P. Beck, John Plaice, and William W. Wadge. Multidimensional Infinite Data in the Language Lucid. Report UNSW-CSE-TR-2013-06, School of Computer Science and Engineering, The University of New South Wales, Sydney, Australia, 2013. Submitted for publication. `http://plaice.web.cse.unsw.edu.au/archive/JAP/U-CSE-201306.pdf`.

[4] André DeHon, Jean-Louis Giavitto, and Fréderic Gruau. 06361 Executive Report – Computing Media Languages for Space-Oriented Computation. In André DeHon, Jean-Louis Giavitto, and Frédric Gruau, editors, *Computing Media and Languages for Space-Oriented Computation*, number 06361 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany, 2007. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany.

[5] Stuart I. Feldman and Alan C. Kay. A conversation with Alan Kay. *ACM Queue*, 2(9):20–30, 2004.

[6] David Maier, Jeffrey D. Ullman, and Moshe Y. Vardi. On the foundations of the universal relation model. *ACM Trans. Database Syst.*, 9(2):283–308, 1984.

[7] John Plaice and Jarryd P. Beck. Higher-order Multidimensional Programming. Report UNSW-CSE-TR-2012-15, School of Computer Science and Engineering, The University of New South Wales, Sydney, Australia, 2012. Submitted for publication. `http://plaice.web.cse.unsw.edu.au/archive/JAP/U-CSE-201215.pdf`.

[8] John Plaice, Blanca Mancilla, and Gabriel Ditu. From Lucid to TransLucid: Iteration, dataflow, intensional and Cartesian programming. *Mathematics in Computer Science*, 2(1):37–61, 2008.

Computing Activity in Space

Martin Potier*, Antoine Spicher*, Olivier Michel*

*LACL, Université Paris-Est Créteil,
61 av. du Général de Gaulle 94010 Créteil, France
Email: {martin.potier,antoine.spicher,olivier.michel}@u-pec.fr

***Abstract*—In the field of modeling and simulation of dynamical systems, an interesting challenge consists in the *spatial* computation of the *activity*, and its topology, exhibited by their discrete event simulation. We believe that activity and regions of activity can help in optimize, analyze and model complex systems.**

In a previous work, we have introduced a limited form of computation of activity. We extend here the algorithm to take into account any kind of MGS program. We apply the algorithm to optimise the pattern matching of MGS. A process of diffusion-limited aggregation is used as a benchmark to provide encouraging preliminary results.

I. Introduction

Complex simulation models are usually defined in terms of a large number of interacting parts. In common modeling tools and languages it is not easy to extract abstractions of the dynamics of the whole system, during, before or after its simulation. When outputs are available, their analysis is long and meticulous. To our knowledge, no spatial method exists for finding or using *patterns of interactions* in system structures, during a simulation. In the simulation context, *activity* is usually used as a stage of the system under study.

We call *active regions* contiguous active parts. These regions are evolving according to the laws governing the model. They are highly dynamical, usually changing at each time step. We are interested in giving a topological definition of the active regions and to handle them intentionally [1], as we would with any other element of the system. We believe that the definition of activity and activity regions [2], [3] can be used as a central guiding concept to enhance the dynamics of sub-components in time, space, and states. Moreover, we believe that it could open the route to *multi-level modeling* considering multiple levels of descriptions of phenomenon.

MGS is an experimental programming language dedicated to the modeling and the simulation of a special kind of discrete dynamical systems: *dynamical systems with a dynamical structure*, or $(\text{DS})^2$ [4]. Dynamical systems with a dynamical structure arise when the state space is not fixed *a priori* but is jointly computed with the current state during the simulation. In this case the evolution function is often given through local rules that drive the interaction between some system components. MGS offers a new kind of data structure, *topological collections*, to describe the state of a dynamical system, and a new kind of control structure, *transformation*, to express local and discrete evolution laws. These two notions permit an easy specification of $(\text{DS})^2$ and capture most of unconventional computation models, such as cellular automata, multi-agents systems, Lindenmayer systems, chemical computations, Paŭn systems, etc.

We focus in this paper on the *spatial computation* of activity and activity regions for programs in MGS involving matching patterns of arbitrary radius. Section II introduces a generic method for tracking an active region throughout the simulation of a dynamical system. Using the *link* operator from combinatorial topology, it allows one to decompose the domain into *active* and *quiescent* sub-domains. Active regions are used to optimize the efficiency of MGS's pattern matching algorithm and preliminary results of the optimization are given in section III. Conclusions are drawn in the last section.

II. Spatial Computing of Activity in MGS

We are interested in investigating the concept of *activity* in MGS. Activity [3] is a measure of event occurrences or state changes in a simulation. This measure can be used for different purposes like optimizing simulation or giving a better understanding of the dynamics. In the context of MGS, activity will allow us (1) to develop a more efficient pattern matching algorithm, and (2) to identify higher level structures in a model and to track them in simulations. The former will be illustrated in the next section, the latter will be discussed in the conclusion.

In this section, we present a generic way to track an active region throughout the simulation. The reader not familiar with the MGS concepts should refer to our previous publications [5], [6], [7].

A. Context

For the sake of simplicity, we restrict ourselves only to patterns that involve at most two interacting elements. For example, pattern `x, y, z` (*i.e.*, matching three elements where `y` is a neighbor of `x` and `z` is a neighbor of `y`) is out of the scope of the following study. However we conclude the section with a paragraph about the generalization to any pattern.

We illustrate our idea with a simplistic but paradigmatic example, a model of forest fire spread where *fire* spreads through a *forest* leaving burnt matter in the form of *ashes*. This can be described by a 3-state cellular automaton easily encoded in an MGS transformation as follows:

```
trans fire_spread = {
  'Forest as x / member('Fire, neighbors x)
       => 'Fire;
```

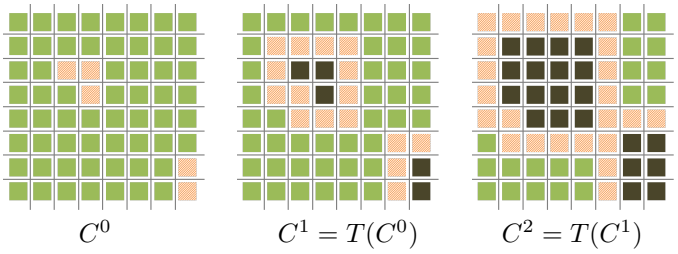


Figure 1. Evolution relation, first three steps of a forest fire simulation. Green, orange (hatched) and black represent Forest, Fire and Ashes respectively.

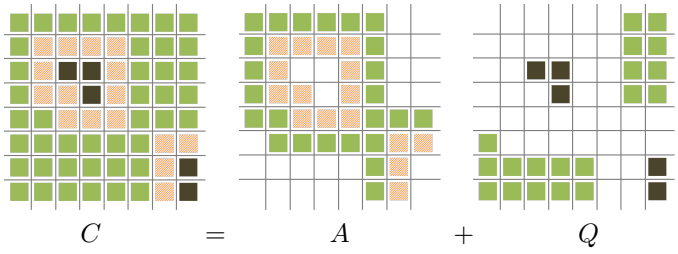


Figure 2. Decomposition of a topological collection into a set of active cells and quiescent cells. Green, orange (hatched) and black represent Forest, Fire and Ashes respectively.

```
  'Fire => 'Ashes;
}
```

States are represented by three symbols: `'Forest`, `'Fire` and `'Ashes`. The first rule specifies how forest gets into fire when it is neighbor of some fire and the second rule specifies that fire leaves ashes after burning. Figure 1 shows three consecutive evolution steps of this automaton on a square grid topological collection.

In the example of Figure 1, activity is located in a small subpart of the domain – only cells in fire and forest in their neighborhoods evolve – and progresses from neighbor to neighbor. However the usual MGS pattern matching process needs to iterate over the whole collection at each application of the transformation `fire_spread`. Given the small number of cells in interaction, the matching process is here quite inefficient. The following study of activity will allow us to target specifically evolving cells during pattern matching. Moreover, activity enlightens an important emergent structure of fire spread models, the *fire front*.

B. Activity and Interactions

The interpretation of activity in MGS corresponds to the number of interactions occurring at each time step of a simulation. Since interactions happen whenever a rule of a transformation matches a subcollection, activity splits naturally a collection into two parts: the *active* subcollection as the subcollection where interactions *can* occur and the *quiescent* subcollection where they cannot.

Let us define the active and quiescent subcollections in a formal and general way, using the construct of topological collections. Let C be a collection and T be a transformation. We define the *matching function* $\mathbb{M}_T$ of transformation T as the function which maps a collection C onto the set of all subcollections of C matched by a pattern of T. In other words, the matching function $\mathbb{M}_T$ represents a call to the pattern matching process. The *active subcollection* A of C is then defined as

$$A = \bigcup_{S \in \mathbb{M}_T(C)} S$$

that is the merge[1] of all matched subcollections of C. Thus, the *quiescent subcollection* Q corresponds to the complementary of A in C:

$$Q = C - A$$

Figure 2 shows the decomposition into active and quiescent subcollections for the fire spread example.

C. Activity Dynamics

In order to track activity during the simulation of a system, we define a way to compute the evolution of the active region (grow, shrink, carve, etc.) based on the topological properties of the active subcollection.

Let us consider $(C^0, C^1, \dots)$ the trajectory of collections due to the successive applications of a transformation T, where $C^{i+1} = T(C^i)$ for $i \geq 0$. Using the active-quiescent decomposition on $C^i = A^i + Q^i$, one can notice that the application of transformation T only acts on the active part and leaves Q^i unchanged so that the pattern matching can be restricted to subcollection A^i only:

$$C^{i+1} = T(C^i) = T(A^i|F^i) + Q^i \tag{1}$$

The notation $T(A^i|F^i)$ reflects that the restricted pattern matching process may require some information from Q^i: visiting the neighborhood of some matched element (*e.g.*, in transformation `fire_spread`: `member('Fire, neighbors x))` does not imply that the visited neighbors are in A^i. This information is denoted F^i which is the subcollection whose support consists of all quiescent cells with a neighbor in A^i:

$$F^i = \mathrm{Lk}\, A^i$$

The *link* operator Lk denotes the set of all neighbor cells of some cell σ. We use here its natural extension to the set of active cells.

The decomposition of C^{i+1} proposed in Equation (1) does not coincide with the definition of A^{i+1} and Q^{i+1}. In fact, some cells of Q^i may become active and *vice versa*. However the tracking of the active-quiescent frontier during a simulation can be refined by making use of subcollection F^i. Indeed, assuming that transformation rules involve at most two neighbor elements, any quiescent cell, at some iteration of the simulation, having no active neighbor cell will observe no change in its environment and then will remain quiescent at the next iteration step. More formally, we obtain:

$$Q^i - \mathrm{Lk}\, A^i \subset Q^{i+1} \qquad A^{i+1} \subset T(A^i|\mathrm{Lk}\, A^i) + \mathrm{Lk}\, A^i \tag{2}$$

[1]The union operator is used instead of the addition since some subcollections of $\mathbb{M}_T$ may have overlapping support.

The second statement is trivially obtained using complementary and means equivalently that the active part cannot expand further than F^i. Equation (1) is then rewritten

$$C^{i+1} = \left[T(A^i|\mathrm{Lk}\,A^i) + \mathrm{Lk}\,A^i\right] + \left[Q^i - \mathrm{Lk}\,A^i\right] \qquad (3)$$

D. Optimized Pattern Matching

In the light of Equations (2), it is obvious that Equation (3) over-approximates the active-quiescent decomposition of C^{i+1}. As an example, on Figure 3, bold lines represent the limit of expansion of the active part at next step but some cells in this subcollection become quiescent. These cells can be identified as the cells of $T(A^i|\mathrm{Lk}\,A^i)+\mathrm{Lk}\,A^i$ which cannot be involved in any interaction at time $i+1$. In other words they are not selected by the pattern matching process:

$$\mathbb{M}_T(C^{i+1}) = \mathbb{M}_T(T(A^i|\mathrm{Lk}\,A^i) + \mathrm{Lk}\,A^i)$$

and the computation of sequence $(A^i)_{i\in\mathbb{N}}$ follows

$$\left\{ \begin{array}{lcl} A^0 & = & \bigcup_{S\in\mathbb{M}_T(C^0)} S \\ A^{i+1} & = & \bigcup_{S\in\mathbb{M}_T(T(A^i|\mathrm{Lk}\,A^i)+\mathrm{Lk}\,A^i)} S \end{array} \right.$$

Notice that this recursive definition of A^i does not refer to the quiescent subcollection Q^i anymore, since the neighbor operator Lk exactly targets the interesting cells. As a consequence, activity tracking during the simulation allows us to focus on a reduced part of the collection for pattern matching. The induced optimization is discussed and illustrated in Section III on a more complex example.

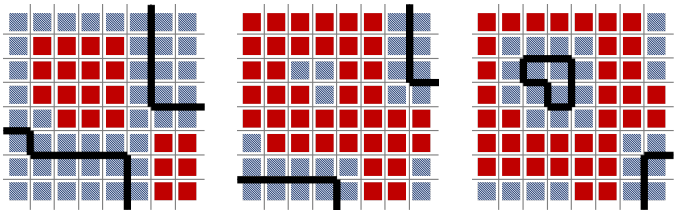


Figure 3. Evolution of the active-quiescent frontier for the three steps of Figure 1. Active cells in red, quiescent cells in light blue. The bold lines represent the decompositions induced by Equation (3).

E. Topological Characterization

The previous study can be generalized to transformations involving more than two interacting elements by reconsidering how far the active subcollection can expand at each time step. Let r be the *radius* of an interaction, *i.e.*, the minimum distance (in terms of hops) between interacting elements. The previous restriction was to only consider transformations of radius $r = 1$, which was considered in our previous work [8]. Let us now consider an arbitrary radius $r \in \mathbb{N}$.

The expansion F_r^i of A^i for a radius r is given recursively by

$$\left\{ \begin{array}{lcl} F_0^i & = & 0 \\ F_{r+1}^i & = & \mathrm{Lk}(A^i + F_r^i) + F_r^i \end{array} \right.$$

This definition is analogous to the definition of the *wave operator* $W(n)$ in [9] and reveals the topological nature of activity. The wave operator is used for the elaboration of a combinatorial Morse theory; Morse theory is a mathematical tool for studying topology of spaces. Roughly speaking, this theory deals with *Morse functions* – a way to flood the space with some "liquid" – and *critical points* – where the liquid reveals basins, passes and peaks of the topography of the space. We are currently investigating the analogy between Morse theory and activity tracking: Morse functions will correspond to activity propagation and critical points to space-time positions where independent activity zones segregate or collide, pointing out important events in the model.

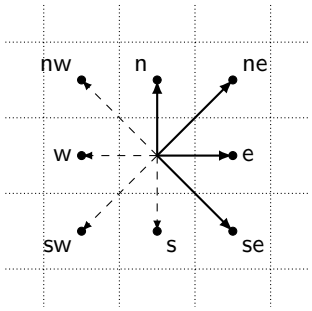


Figure 4. A GBF defining a Moore neighborhood, with four generators e, n, ne and se and two constraints n + e = ne, e − n = se. The directions w, s, sw and nw are defined as the inverses of the generators.

III. Preliminary Results

In this section, we consider a more elaborate example: we chose to implement a model of a system presenting *diffusion-limited aggregation* (DLA). Contrary to the forest fire spread model, the DLA model presents real interactions between two particles, and not just reactions of a particle depending on the state of its neighbors. Following results of Section II, information provided by activity is used to speed up the simulation by applying the rules of the transformations *only* to the active regions.

A. The Example of DLA simulation

DLA phenomenon is a well-known example of aggregate particules clustering together to form Brownian trees [10]. It can be implemented [11] on a Cellular Automaton (CA) and so we implemented it in MGS.

1) State Representation in MGS: The regular lattice used in the CA is represented using a GBF topological collection representing a Moore neighborhood on a cyclic 2D square grid.

In MGS, such a collection is specified from a presentation of the group of displacements. The definition of a Moore neighborhood grid, requires four basic displacements `n` (north), `e` (east), `ne` (north-east) and `se` (south-east) as shown on Figure 4:

```
gbf Grid = < n, ne, e, se ;
             50 n = 0, 50 e = 0,
             ne = e + n, se = e - n >
```

The four additional equations specify that the grid is cyclic of size 50×50, and define relations between diagonal and

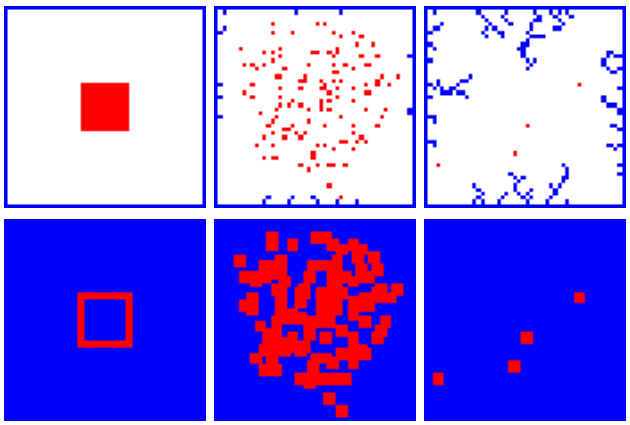


Figure 5. DLA at 3 different iterations, from left to right, of a simulation with mobile particles starting at the center and static particles on the borders on a 50×50 grid. In the top serie, `'mobile` cells are in red, `'static` in blue, and `'empty` in white. In the bottom serie, active cells are in red, and quiescent are in blue.

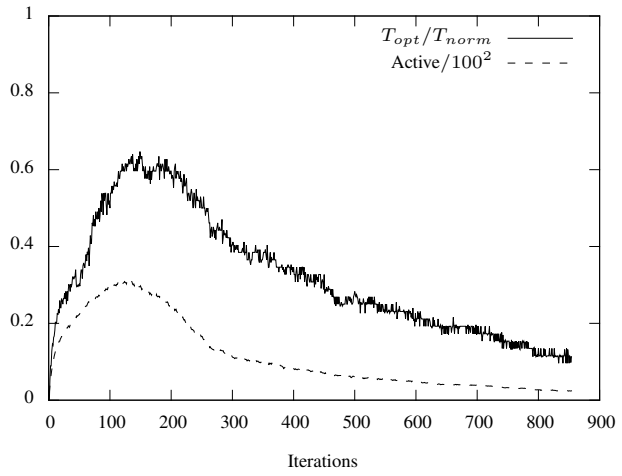


Figure 6. Amount of active cells, and computation time for optimized algorithm per iteration step on a 100×100 grid. Values have been normalized using the total size of the grid (10 000 cells), and the computation time of a normal run per iteration step respectively.

non-diagonal directions while the reverse directions are automatically considered. Then, each cell of the GBF is labeled by its state, which varies along the iterations of the simulation.

```
type cell = 'empty | 'mobile | 'static ;;
```

The label `'mobile` refers to a cell containing a particle that may randomly move out at a next iteration; the label `'static` is used for cells where there is a particle that will remain still for the rest of the simulation, and the label `'empty` means that there is no particle on the cell therefore it is available to host one particle at the next iteration.

2) Dynamics Specification in MGS: Specifying the dynamics of the DLA in a transformation amounts to specifying two rules which are applied following a maximal parallel strategy:

```
trans evolution = {
  'mobile, 'static => 'static, 'static ;
  'mobile, 'empty =>  'empty,  'mobile ;
} ;;
```

The first rule of transformation applies if two neighbor cells are in states `'mobile` and `'static`, and replaces them by a couple of cells which states are both `'static`. The second rule of transformation matches if two neighbor cells are in states `'mobile` and `'empty` in which case they exchange their states to simulate a particle "jumping" from one spot to another. Note that the priority is given to the first rule allowing a cell labeled `'mobile` surrounded by many `'empty` and at least a `'static` to become `'static` instead of continuing its walk. This transition function iterates over all cells and updates their labels. Figure 5 shows simulations of the model at different iterations.

B. Benchmark

Using the classical pattern matching algorithm, the entire topological collection must be iterated over at every update. For a CA on a square grid of side length n, the process must go through the n^2 cells leading to an update step with a constant time since all cells must be visited at each iteration. However, only a fluctuating number of cells have their values changed on the update – which correspond to only a fraction of the grid.

In this section, we use activity information to reduce the cost of pattern matching. The example of DLA was rewritten to have the transformation rules only apply to the active regions and skip the quiescent part. As with the theoretical view presented in Section II, the cellular automaton space is split between active cells taking part in a transformation, meaning that they are caught in an interaction, and quiescent cells having no role to play whatsoever in the current iteration. This results in a speed-up of the whole computation.

Figure 6 presents a plot of the count of active cells for each iteration, and of the computation time by iteration for the optimized run of the simulation. The occasional noise comes from the fluctuating workload on the test computer.

As shown in Figure 5, at the beginning of the simulation, very few cells are active; there is only the "crown" around the initial square of mobile particles which are particles with some freedom of movement. Their amount increases dramatically with the random walk of the particles starting to fill the domain to only lower once they pair more and more with a static particle. As with a classical cellular automaton, the computation time is linear with the number of cells: the more cells there are, the more time it requires with the same computation time by cell. Moreover, the computation time of the non-optimized run is roughly the same for every iteration: the pattern matching mechanism must explore all of the 10 000 cells to verify whether a rule applies. Whether a rule matches or not hardly changes the time required to compute the application of a transformation rule.

From the graph, we can notice that activity and computation time of the optimized run coincide since the latter directly depends on the amount of cells to explore. Then, it is only

natural that the shape of the curve is nearly identical to the variation of the amount of active cells.

Cells are never all active at once. Actually, for the presented run, the maximum amount of active cells is very low and stays below 30% of the total amount. So, even at its peak, the optimized run always requires less computation time by iteration than the non-optimized one. The cost of maintenance can be read from the difference between the computation time of the optimized run and the active cell count. In our case, it is rather elevated, one reason would be that the activity optimizations were implemented on top of MGS, and not yet included in the language itself. This optimization is data-dependent, had we changed the initial values, the graph would have been altered too. Therefore, a general or global speed-up does not make sense. More generally, there is no guarantee of speed-up but there cannot be a slow-down since the worst case scenario would correspond to a pattern matching on the whole structure, that is the normal run. Regarding the DLA, such a speed-up is dependent on the type of implementation; in an agent-based implementation, activity tracking would have allowed a focus on the mobile particles only and thus an optimal speed-up.

IV. Conclusion

This paper focuses on the computation in space of activity and activity regions in the context of the domain-specific language MGS. We have extended the topological framework introduced in [8] to compute these two notions on matching patterns of arbitrary radius. Activity regions have been successfully used to reduce the cost of pattern matching in the simulation of an example of diffusion-limited aggregation. Here activity appears to be a data-dependent optimization technique that can be easily combined with other techniques. It is important to stress that, due to the topological approach followed by MGS, the computation of activity regions describe in this paper are valid for *any kind of topological collections* available in the language.

The perspectives opened by this work are numerous. At the runtime level, the optimization of the MGS runtime is a long term effort. We have to internalize in the runtime the computation of the active regions. Indeed, we believe that it is not the role of the programmer to seek for complex and obfuscating optimizations but to the language (or the execution support) to provide automatic high-level techniques to reduce the execution time.

At the language level, activity regions are not first-class objects: they are computed at each time-step of the simulation, and there is no link between regions computed at two different time steps. We plan on *reifying* the regions to first-class objects and to add operators to handle these new objects. Moreover, as we pointed out in section II-E, we have to investigate the analogy between Morse theory and activity tracking. Those two directions will be the key to level up the language to multi-level modeling of complex dynamical systems that exhibits behaviors at multiple levels of descriptions (like in biology the molecular level, the cellular level, the organic level, etc.).

Acknowledgments

We are grateful to J.-L. Giavitto at CNRS-IRCAM for being part of the MGS project since its beginning in 2001.

The work on activity would not have "grown" without the participation of, and the fruitful interactions with many colleagues. The authors are especially grateful to the community born during the Cargese Interdisciplinary Seminar in 2009 and more specifically to A. Muzy at the CNRS, B. P. Zeigler at the University of Arizona and D. RC Hill at the University Blaise Pascal - Clermont-Ferrand.

This research is supported in part by the French ANR grant "SynBioTIC" 2010-BLAN-0307-03, the Université Paris-Est Créteil, the IRCAM (CNRS, UMR STMS 9912), the Université of Évry and the Complex System Institute - Paris.

References

[1] A. Spicher, O. Michel, and J.-L. Giavitto, "Spatial computing as intensional data parallelism," in *Third IEEE International Conference on Self-Adaptive and Self-Organizing Systems - Spatial Computing Workshop*, San Franciso, United States, 14–18 September 2009.

[2] A. Muzy, L. Touraille, H. Vangheluwe, O. Michel, M. Kaba Traoré, and D. R.C. Hill, "Activity Regions for the Specification of Discrete Event Systems," in *Spring Simulation Multi-Conference Symposium On Theory of Modeling and Simulation - DEVS Integrative M&S Symposium (DEVS'10)*, USA, 2010, p. Accepted for publication, ACM, SCS CNRS.

[3] A. Muzy, F. Varenne, B. P. Zeigler, J. Caux, P. Coquillard, L. Touraille, D. Prunetti, P. Caillou, O. Michel, and D. R. Hill, "Refounding of activity concept? Towards a federative paradigm for modeling and simulation," *SIMULATION: Transactions of The Society for Modeling and Simulation International*, vol. 89, no. 2, pp. 156–177, 2012.

[4] J.-L. Giavitto, C. Godin, O. Michel, and P. Prusinkiewicz, *Modelling and Simulation of biological processes in the context of genomics*. Hermes, Jul. 2002, ch. "Computational Models for Integrative and Developmental Biology", also republished as an high-level course in the proceedings of the Dieppe spring school on "Modelling and simulation of biological processes in the context of genomics", 12-17 may 2003, Dieppes, France.

[5] A. Spicher, O. Michel, and J.-L. Giavitto, "Spatial computing as intensional data parallelism," in *Third IEEE International Conference on Self-Adaptive and Self-Organizing Systems - Spatial Computing Workshop*, San Franciso, United States, 14–18 September 2009.

[6] L. Bigo, A. Spicher, and O. Michel, "Spatial programming for music representation and analysis," in *Fourth IEEE International Conference on Self-Adaptive and Self-Organizing Systems - Spatial Computing Workshop*, Budapest, Hungary, 27 September 2010.

[7] A. Spicher, O. Michel, and J.-L. Giavitto, "Arbitrary nesting of spatial computing," in *Spatial Computing Workshop 2012 at AAMAS*. Valencia, Spain: AAMAS, June 2012.

[8] M. Potier, A. Spicher, and O. Michel, "Topological computation of activity regions," in *Submitted to the Conference on Principles of Advanced Discrete Simulation (PADS'13)*, 2013.

[9] U. Axen, "Topological analysis using morse theory and auditory display," Ph.D. dissertation, University of Illinois at Urbana-Champaign, Champaign, IL, USA, 1998.

[10] T. A. Witten and L. M. Sander, "Diffusion-limited aggregation, a kinetic critical phenomenon," *Phys. Rev. Lett.*, vol. 47, pp. 1400–1403, Nov 1981.

[11] A. Spicher, N. A. Fatès, O. Simonin *et al.*, "From reactive multi-agent models to cellular automata - Illustration on a diffusion-limited aggregation model," *Proceedings of ICAART'09*, 2009.

Application of Force-Directed Graphs on Character Positioning

Christine Talbot and G. Michael Youngblood
Game Intelligence Group
University of North Carolina at Charlotte
Email: {ctalbot1, youngbld}@uncc.edu

***Abstract*—Mixed human-agent control and positioning is a challenge in virtual environments, such as the theater. The main impacts are found when human-controlled characters choose not to follow a predefined script, or play-script.**

Force directed graphs have been utilized to provide clean layouts for large and complex graphs and their relationships. Here, we utilize a force-directed graph approach to spatially organize where our characters are while on-stage. We analyze our approach's success against the basic positioning of the characters on-stage, using our predefined goals for the force-directed graphs from our previous work.

I. Introduction

There is a need for greater automation of character positioning within virtual environments, especially around visualizations of play-scripts, or production notes. This becomes more critical when arranging a mix of human- and agent-controlled characters. Humans do not always follow predictable patterns, and virtual characters must be able to react appropriately (spatially) within the environment.

A simple example of this is within theatre productions as a virtual environment. In real life, actors arrange themselves on the stage according to both basic rules of the theatre, as well as with respect to the positioning of the other actors on-stage. Humans may not always hit their mark like they should, may move when they are not supposed to, or may not even move at all during the play. This presents issues with the blocking within the play, as the other characters on-stage are assuming that the human followed the script. If the agent-controlled characters do not adjust, they could create unrealistic positionings of the characters based on the standard rules of thumb for theatre, but also could obstruct visibility to themselves or the human-controlled character for the audience. In video games, there is also a desire to adjust the positions of the agent-controlled characters based on where the human-controlled character is, in order to provide better visibility (or less visibility) of those characters.

In this paper, we look at force-directed graphs to assist with positioning the agent-controlled characters based on the movements of the human-controlled character within a theatre environment. Force directed graphs, also known as string embedders, utilize the information contained within the structure of the graph when placing nodes on the screen. They have been shown to create both aesthetically pleasing and symmetric graphs, and have been used for many years to display complex graphs. Some have been used for visualizing networks, such as a social network, or even to better understand relationships between entities.

We review some key force-directed graph algorithms and extract some key features provided by them, which can be utilized for positioning our characters on-stage. Results of implementing our proposed algorithms from our previous work [1] are presented, showing reasonable results for six of our eight targeted requirements for force-directed graphs.

II. Background

Prior work has centered around the emotional and one-on-one interactions of characters with humans. The placement of characters on a stage has not been heavily pursued. Many researchers focus on the conversational and nonverbal domains, such as Thespian [2], Virtual Storyteller [3], and Stability and Support Operations (SASO) [4]. However, these do not emphasize the spatial aspects of the interactions between multiple characters.

To appropriately apply spatial logic, there are some key works done by groups around personal space and conversational space. For instance, Jan and Traum describe six different forces that affect when / why a person may shift position when in a group of people:

- one is listening to a speaker who is too far and or not loud enough to hear
- there is too much noise from other nearby sound sources
- the background noise is louder than the speaker
- one is too close to others to feel comfortable
- one has an occluded view or is occluding the view of others [5]

Additional research shows that friendship and attraction can affect the spatial distances between people (decreases as attraction increases), while negative attitudes may not have much effect on the spatial distances [6]. People also prefer to be across from one another than next to each other in most situations, but there is importance to the environment for determining what distance is comfortable [7]. According to studies reviewed by Sundstrom, comfortable face-to-face distances for speaking while sitting is approximately five feet and comfortable face-to-face conversation while standing is approximately three feet [6]. There is also a discussion around the effects of spatial invasion on character behaviors and movements within Sundstrom's review.

Robotics focuses more on spatial reasoning, such as verbal instructions. For instance, Langley, Schermerhorn, and Scheutz provide an approach to human-robot interaction which allows for communicating complex tasks which can be translated into procedures for the robot [8]. Matuszek and Herbst take natural language and robotic perceptions and translate it into a robot control language for following route directions [9]. Dzifcak, Scheutz, and Baral also utilize natural language to determine actions and goals for the robot [10]. All of these incorporate telling a robot what to do or where to go.

In addition, Brooks's work attempted to train a robot to be an actor by utilizing verbal directions [11]. This is helpful, but we require less explicit directions for our work. David Lu and Bill Smart's work with robots in theatre focused on pre-recording real actors' movements and replicating them on robots to perform specific scenarios [12]. The focus in their work is on believability; however, this work is based more on a motion capture-like style of replaying actions done by a human and does not address our concerns with dynamically positioning multiple characters without pre-recording.

In our previous work, we focused on what had been explicitly written as an annotation (in natural language) within a play-script [13]. We showed that combining play-scripts, natural language, and a rules engine can correctly position characters, on average, 89% of the time [14] with respect to a well-known production of Hamlet [15]. Movements identified by our natural language processor were fed into our rules engine to adjust the motion based on these rules:

- r_1: Characters should face the audience as much as possible, and avoid turning their back to the audience
- r_2: Characters should face the person speaking
- r_3: Characters with higher importance or larger roles should be placed slightly closer to the audience relative to lesser role characters
- r_4: Characters should try to stay closer to center line as much as possible to improve visibility for the maximum portion of the audience
- r_5: Characters should avoid unnatural movements by adhering to basic frame coherence rules, such as not having their gaze or orientation jump from left to right immediately
- r_6: Characters should maintain appropriate personal space based on inter-character relationships within the play
- r_7: Characters should be next to an item they wish to pick up [14]

However, all of this prior work was done with computer-controlled characters only. We had not introduced the variable of a human-controlled character, where we must adjust our rules in a more real-time manner. Therefore, we proposed the use of force-directed graphs for positioning characters in our prior work [1], and provided some key algorithms to assist with translating force-directed graphs into a positioning network on-stage. Our focus is on ensuring that all the characters on-stage (including the human-controlled character) are appropriately positioned, even with an incorrect human movement (or lack thereof).

III. Related Work

Crowd modeling at first thought appears to be an appropriate approach to positioning characters. Upon further investigation, it can be seen that crowd modeling focuses more on modeling people's behaviors as opposed to the close-knit intricacies of the relationships between the characters onstage. It does not focus on spatially pleasing arrangements as a whole, but rather looks at each individual's contribution independent of the others. In theatre, the goal is to have the actors work together as a whole, not as independent entities, and thereby is not suitable for a theatre-type environment.

The next logical approach involves the use of force-directed graphs and their repellent and attractive forces to spatially arrange the characters. The connected nodes (or characters in our situation) are arranged as a whole to be aesthetically pleasing, meaning that all edge lengths should be the same length, and it should maximize symmetry over the entire graph layout. They have been used for many different purposes, such as social networks, like Bannister et al's work. Their work attempts to centralize vertices that are more theoretically central in the graph [16]. This is interesting because of its close relationship to our work—visualizing relationships between nodes.

Network visualizations use force-directed graphs to help identify information about different clusters, and arranges graphs into symbolic shapes to help recognize the relative size of the clusters. These allow viewers to be able to estimate overall sizes of the graphs, as well as recall the layout of the graph at a high level. It does best with clusters of about eight vertices, and may not do well scaling to sparse clusters [17]. This could prove useful in specifically shaping clusters of characters into particular shapes such as a semi-circle when discussing.

One of the first force-directed graph drawing methods was provided by Tutte in 1963 [18]. In his algorithm, he guarantees a crossings-free drawing if all faces of the drawing are convex for a 3-connected planar graph. The forces in this model are proportional to the distance between vertices, with no repulsive forces, and places each free vertex at the barycenter (center of mass) of its neighbors. This is useful in our work since we are concerned with obstructing the audience's view of all the characters onstage. However, there are some results of this algorithm that produce a graph with infinite area [19], or would not place our characters within our stage's confines. Also ensuring three-connectedness and a convex drawing may be challenging in a dynamic environment with a human-controlled character, such as the theatre's stage.

In 1991, Fruchterman and Reingold introduced an algorithm which provides an equalization of vertex distributions. It calculates the forces between adjacent vertices as well as between all pairs of vertices. It also introduces the concept of temperature to reduce the amount of movement of vertices as the layout improves. This algorithm was targeted for small graphs, such as those with 40 or fewer vertices, not unlike what

we expect to have within the theatre. Its cooling of movement via temperature is a specialized use of simulated annealing, which helps to limit oscillations of the layout. However the forces are based on the size of the grid that is to be drawn on, and therefore tries to maximize the real estate used. [20]

Another algorithm, by Kamada and Kawai, tries to minimize the distance of vertices from their corresponding, underlying graph distances [21]. This method is intriguing because of the intent to keep closely related nodes together. However, it requires more computation ($O(|V|^3)$ time) and more storage space ($O(|V|^2)$ space) since it requires a shortest distance calculation on every vertex before running its minimization function [19]. Even though we could calculate the underlying shortest distances for the graph ahead of time, we would need to adjust this each time a character is introduced into the scene or creates a new association to a targeted position onstage. Also, we have some key relationships which encourage a character to hit their mark(s) and remain there until their next movement in the play. Kamada and Kawai's method would equally distribute the characters from each other as well as their marks, which is undesirable in the theatre.

Other more complex force-directed graph drawing algorithms exist that can accommodate tens and hundreds of thousands of vertices. These attempt to break down the graph into simpler structures, like Hadany and Harel [22] or Gajer, Goodrich, and Kobourov [23]. They often involve three-dimensional drawing of the graphs and zooming in order to provide visibility to the nodes of the graph. However, we are focused on very small numbers of vertices and a planar drawing area, so these do not provide much use for our current work.

IV. Approach

To utilize force-directed graphs for positioning characters, we applied the algorithms based on Fruchterman and Reingold's algorithm, as discussed in our prior work [1]. The algorithms included the details on constructing the graph's nodes and vertices to reflect the relationships between the characters, pawns, and the audience. They centered on the addition of characters, the movement of characters, the movement of the human-controlled character, the leaving of characters, and the elapsing of time.

The main algorithm utilized the forces and this graph for repositioning characters on-stage whenever the human-controlled character moved. Each character relationship (edge) had its own unique forces that pushed or pulled the AI characters (vertices) around the stage. Some vertices are setup to be unmoveable, such as the human-controlled character and the targets/pawns. For instance, the relationship between the AI character and its mark/target would pull the AI character closer to the mark, depending on how long it had been since the character moved to that location. The targets are identified by the play-script, with the assumption that all characters (including the human-controlled one) hit their marks correctly and on-time.

The goal for the non-moveable vertices is to act as attractors, but not repellers for the moveable characters. This can be seen in the table of forces in Table I. The attraction and repelling of the vertices is setup to be a quadratic function of the distance of the two vertices. This ensures a stronger pull or push between the vertices as they get further or closer together, respectively. The special vertices of the audience to the character helps to attract the characters to the front of the stage as much as possible, while the center point is intended to act as a barycenter (or mass center point) for the characters onstage. By providing the center point a strong attraction to the audience, it forces the group of characters to form a semi-circle facing the audience.

	Force Type	AI Char	Center Pt
AI Character	Attract	$\|\delta\|^2 - \alpha^2$	
AI Character	Repel	$-\|\delta\|^2 + \alpha^2$	
Human Character	Attract	$\|\delta\|^2/2 - \alpha^2$	
Human Character	Repel	0	
Audience	Attract	$\|\delta\|^2 - L^2/4$	$\|\delta\|^2 - L^2/16$
Audience	Repel	$-\|\delta\|^2 + L^2/4$	0
Center Point	Attract	$\|\delta\|^2 - \alpha^2$	
Center Point	Repel	$-\|\delta\|^2 + \alpha^2$	
Target / Pawn	Attract	$\beta\|\delta\|^2 - \alpha^2$	
Target / Pawn	Repel	0	

TABLE I: Attractive and Repellent Forces
$|\delta|$ = Separation Distance, α = Desired Separation Distance

The key requirements that are met by these particular forces and graph structures are described in more detail in our prior work [1]. However, to provide appropriate context, they are briefly described here as well. They are based on the typical rules of the theatre and conversational groupings as described in the Background section and our prior work on a rules engine [14].

Even Vertex Distribution – Provide a sense of balance and symmetry with the character positioning onstage

Small Number of Vertices – To maintain a small number of vertices within the graph, less than 40 vertices

Crossings-Free Drawing – Avoid occlusions of characters from the audience perspective, if three-connected graph with convex faces

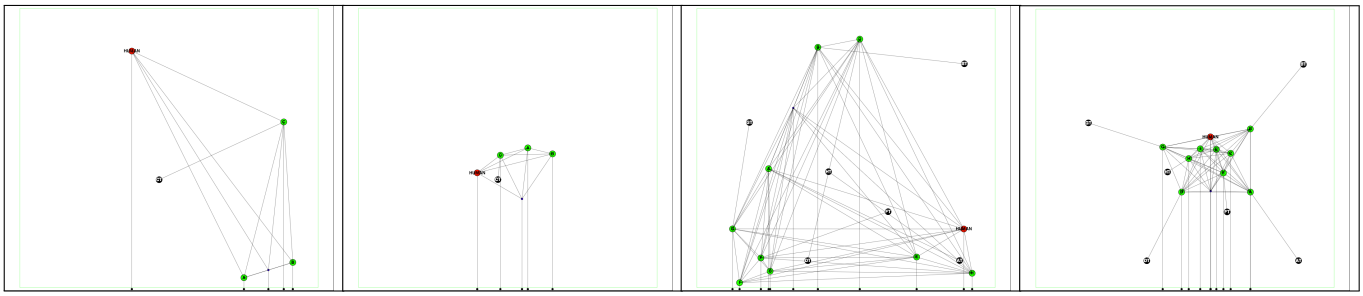
Fixed Vertices – Ability to include unmoveable vertices to assist with characters hitting their marks

Oscillation-Free Arrangements – Avoid oscillation of positions for a single arrangement of characters with the use of a cooling factor

Strength of Relationships Over Time – Ability to adjust the strength of relationships between characters and targets over time

Centering and Encircling Groups – Ensure characters produce a consistent conversational grouping arrangement (semi-circle)

Varying Attracting and Repellent Forces – Ability to have



(a) Random Layout of Three Characters | (b) Adjustment of Figure 1a Using Force-Directed Graphs | (c) Random Layout of Nine Characters | (d) Adjustment of Figure 1c Using Force-Directed Graphs

Fig. 1: Character Positioning Using Force-Directed Graphs—Red=Human; Green=AI Character; Black=Pawn; Blue=Center Pt

different relationships between the vertices that control the forces incurred

Validating the accomplishment of these requirements, we will need to measure performance in three separate, but related, threads. The first is to validate the positioning from randomized states of the play, which is done in this paper. Most of the requirements can be measured during this experimentation. Next, the positioning of the characters must be validated over the duration of an entire scene or play. This test focuses on the oscillation-free arrangements and the strength of relationships over time, in conjunction with human-controlled character movements both correct and incorrect. Finally, it is important to validate these positionings with respect to a human audience. Is the play reasonably blocked, and is it a believable arrangement of the characters?

This approach can be applied to predefined trajectories of characters, if the target destination is considered the current location of the character during the adjustment calculations. It will not address the orientation of the characters at each of these positions. However, our prior work with a rules engine can be utilized to adjust gazes appropriately.

V. Experimentation

To test our approach, we implemented the algorithms described in our previous work [1] in a JSGameSoup javascript application. Here, the stage was represented as a box within the screen, and characters as circles with connecting lines representing their relationships. One character (the human-controlled character) could be moved by dragging it across the screen, to represent the human-controlled character. Numerous scenarios were tested by randomly placing characters, pawns, and the human character on the screen and applying the force-directed graph drawing algorithms to arrange them on the screen, as seen in Figure 1.

Each AI character (in green) is connected to every other AI character on the stage, the human-controlled character, and the audience. They are also sometimes connected to a target position, or pawn, to indicate their correct "mark" on the stage for this moment in the play. These target point connections have no repellent forces, but very strong attractive forces (β) applied to them. Also, if there is at least two AI characters on the stage, they are also connected to a center point which only has attraction forces applied to it. The human character is also tied to this center point (if it exists), the audience, and every character on the stage. The relationship between the human-controlled character and the AI characters has a weaker attraction force than the between-AI character forces. The center point is given a stronger tendency to be in the front quarter of the stage than any of the AI or human-controlled characters to help force the semi-circular arrangement on the stage, as well as an opening in the grouping which faces the audience. Each force is described further in Table I.

These forces allowed for a relatively consistent positioning distance of about 3.14 (SD=1.54) feet between the different characters, which provided our "Even Vertex Distribution" as described above. Even with 12 characters plus the one human-controlled character on the screen, we had at most 40 vertices in our graph, which kept us within reasonable limits for the Fruchterman and Reingold algorithm approach. We observed that the fixed points (also known as the AI characters' target destination) pulled the characters toward them, which helped to minimize movement of the character from their mark. This distance averaged at 494.56 drawing units apart, which represents about 3.30 (SD=1.52) feet of spacing, and was relatively consistent across the different sized character groupings tested (1-12) as seen in Table II.

The forces were varied between the audience, the center point, the target pawns, the AI characters, and the human character, with the target pawn connections being the strongest attraction (with no repelling forces), and the AI character interrelationships being the strongest repellent forces. This, in conjunction with the center point, provided balance with the positioning and provided semi-circle positioning for the smaller number of characters on the stage, as seen in Figure 1b. However, for the larger character groups, they often formed a more circular arrangement, with the audience side not quite being enclosed, as seen in Figure 1d. More work may be required to better balance the grouping arrangements of the characters.

More work needs to be pursued to test the varying relationship strengths over time, as well as the impact of oscillations between arrangements. Some preliminary testing indicated is-

Number of Characters	Character Connected To:	Average Distance (in Drawing Units)	Average Distance (in Feet)
1	audience	370.63	2.47
1	human	526.30	3.51
1	target	496.80	3.31
2	audience	515.92	3.44
2	center	270.80	1.81
2	char	400.74	2.67
2	human	488.35	3.26
2	target	456.22	3.04
3	audience	449.66	3.00
3	center	355.34	2.37
3	char	414.75	2.77
3	human	662.09	4.41
3	target	448.98	2.99
4	audience	528.01	3.52
4	center	296.00	1.97
4	char	424.55	2.83
4	human	468.30	3.12
4	target	555.88	3.71
5	audience	419.66	2.80
5	center	378.33	2.52
5	char	529.04	3.53
5	human	466.43	3.11
5	target	611.72	4.08
6	audience	425.97	2.84
6	center	368.86	2.46
6	char	482.32	3.22
6	human	484.72	3.23
6	target	464.94	3.10
7	audience	436.75	2.91
7	center	348.79	2.33
7	char	469.19	3.13
7	human	502.33	3.35
7	target	428.91	2.86
8	audience	415.04	2.77
8	center	401.58	2.68
8	char	564.30	3.76
8	human	462.93	3.09
8	target	487.51	3.25
9	audience	498.57	3.32
9	center	439.40	2.93
9	char	511.48	3.41
9	human	502.24	3.35
9	target	489.51	3.26
10	audience	482.81	3.22
10	center	376.99	2.51
10	char	483.21	3.22
10	human	480.38	3.20
10	target	521.40	3.48
11	audience	496.43	3.31
11	center	394.77	2.63
11	char	478.84	3.19
11	human	495.07	3.30
11	target	480.68	3.20
12	audience	476.52	3.18
12	center	403.22	2.69
12	char	477.78	3.19
12	human	498.51	3.32
12	target	493.62	3.29

TABLE II: Average Distances (in Feet) Between Characters and Pawns

sues with oscillations of character positioning where characters would swap places, but still maintain the overall layout on the stage. Additional force manipulations may achieve better results than found during this particular experiment.

VI. Conclusion

The results from this work are encouraging, but may require some additional fine-tuning to better optimize the balance between the different forces. We showed that force-directed graphs can create a balanced and centralized grouping of characters on the stage, and maintain relatively consistent distances between characters versus other connections on the stage. This resulted in meeting the following six out of the eight criteria discussed in this paper:

- Even Vertex Distribution
- Small Number of Vertices
- Fixed Vertices
- Oscillation-Free Arrangements
- Centering and Encircling Groups
- Varying Attracting and Repellent Forces

Future work will involve sequentially positioning characters through a single scene to enable measurement of time-based relationships, such as target relationships and paired character entrances. Also, user studies to analyze the human perspective of the resulting positioning will be key in determining the overall success of this work.

References

[1] C. Talbot and G. M. Youngblood, "Positioning Characters Using Forces," in *CAVE*, 2013.

[2] M. Si, S. C. S. Marsella, D. V. Pynadath, and M. Rey, "Evaluating Directorial Control in a Character-Centric Interactive Narrative Framework," *Proceedings of 9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2010)*, pp. 1289–1296, 2010.

[3] M. Theune, S. Faas, and D. Heylen, "The virtual storyteller: Story creation by intelligent agents," *Proceedings of the Technologies for Interactive Digital Storytelling and Entertainment TIDSE Conference (2003)*, 2003.

[4] P. Kenny, A. Hartholt, J. Gratch, W. Swartout, D. Traum, S. Marsella, and D. Piepol, "Building Interactive Virtual Humans for Training Environments," *The Interservice/Industry Training, Simulation & Education Conference (I/ITSEC)*, vol. 2007, no. -1, 2007. [Online]. Available: http://ntsa.metapress.com/index/awk02100355170v6.pdf

[5] D. Jan and D. R. Traum, "Dynamic Movement and Positioning of Embodied Agents in Multiparty Conversations," in *Proceedings of the Workshop on Embodied Language Processing*, ser. Embodied NLP '07. Stroudsburg, PA, USA: Association for Computational Linguistics, 2007, pp. 59–66.

[6] E. Sundstrom and I. Altman, "Interpersonal Relationships and Personal Space: Research Review and Theoretical Model," *Human Ecology*, vol. 4, no. 1, pp. 47–67, 1976.

[7] R. Sommer, "The Distance for Comfortable Conversation: A Further Study," *Sociometry*, vol. 25, no. 1, pp. 111–116, 1962.

[8] N. Trivedi, P. Langley, P. Schermerhorn, and M. Scheutz, "Communicating, Interpreting, and Executing High-Level Instructions for Human-Robot Interaction," in *Proceedings of the 2011 AAAI Fall Symposium on Advances in Cognitive Systems*, Arlington, VA, November 2011.

[9] C. Matuszek, E. Herbst, L. Zettlemoyer, and D. Fox, "Learning to Parse Natural Language Commands to a Robot Control System," in *Proceedings of the 13th International Symposium on Experimental Robotics (ISER)*, June 2012.

[10] J. Dzifcak, M. Scheutz, C. Baral, and P. Schermerhorn, "What to do and How to do it: Translating Natural Language Directives into Temporal and Dynamic Logic Representation for Goal Management and Action Execution," *Proceedings of the 2009 IEEE international conference on Robotics and Automation*, pp. 3768–3773, 2009.

[11] A. G. Brooks, "Coordinating Human-Robot Communication," Ph.D. dissertation, MIT, Jan. 2006.

[12] D. V. Lu and W. D. Smart, "Human-robot interactions as theatre," in *RO-MAN 2011*. IEEE, 2011, pp. 473–478.

[13] C. Talbot and G. M. Youngblood, "Spatial Cues in Hamlet," in *Intelligent Virtual Agents*, Y. Nakano, M. Neff, A. Paiva, and M. Walker, Eds. Springer, 2012, pp. 252–259.

[14] ——, "Shakespearean Spatial Cues," in *International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2013, Saint Paul, Minnesota*. IFAAMAS, 2013.

[15] B. Colleran, J. Gielgud, W. Shakespeare, R. Burton, H. Cronyn, A. Drake, and E. Herlie, "*Hamlet*, Electronovision, Inc." 1964. [Online]. Available: http://www.youtube.com/watch?v=NelO_Jm5O6w

[16] M. J. Bannister, D. Eppstein, M. T. Goodrich, and L. Trott, "Force-Directed Graph Drawing Using Social Gravity and Scaling," *CoRR*, vol. abs/1209.0748, 2012.

[17] R. Shannon, A. Quigley, H. Australia, and P. Nixon, "Graphemes: Self-Organizing Shape-Based Clustered Structures for Network Visualisations," in *Proceedings of the 28th of the International Conference Extended Abstracts on Human Factors in Computing Systems*, 2010, pp. 4195–4200.

[18] W. T. Tutte, "How to Draw a Graph," *Proc. London Math. Soc*, vol. 13, no. 3, p. 743768, 1963.

[19] S. G. Kobourov, "Spring Embedders and Force Directed Graph Drawing Algorithms," *CoRR*, vol. abs/1201.3011, 2012.

[20] T. M. J. Fruchterman, Edward, and E. M. Reingold, "Graph Drawing by Force-Directed Placement," *Software: Practice and Experience*, vol. 21, no. 11, pp. 1129–1164, 1991.

[21] T. Kamada and S. Kawai, "An algorithm for drawing general undirected graphs," *Inf. Process. Lett.*, vol. 31, no. 1, pp. 7–15, Apr. 1989.

[22] R. Hadany and D. Harel, "A multi-scale algorithm for drawing graphs nicely," in *Proceedings of the 25th International Workshop on Graph-Theoretic Concepts in Computer Science*, ser. WG '99. London, UK, UK: Springer-Verlag, 1999, pp. 262–277.

[23] P. Gajer, M. T. Goodrich, and S. G. Kobourov, "A Fast Multi-Dimensional Algorithm for Drawing Large Graphs," *8th Symp. on Graph Drawing (GD)*, pp. 211–221, 2000.

[24] C. Talbot and G. M. Youngblood, "Lack of Spatial Indicators in Hamlet," in *FLAIRS*, 2013.

[25] C. Manning, "Force layout," https://github.com/mbostock/d3/wiki/Force-Layout. [Online]. Available: https://github.com/mbostock/d3

[26] A. Modeling, "Anchor modeling | an agile modeling technique for evolving information," http://www.anchormodeling.com/. [Online]. Available: http://www.anchormodeling.com/

[27] A. Quigley, "Large scale force directed layout (spring algorithm): For information visualization: Clustering and abstraction," http://rp-www.cs.usyd.edu.au/~aquigley/3dfade/, 2003. [Online]. Available: http://rp-www.cs.usyd.edu.au/~aquigley/3dfade/

Engineering Confluent Computational Fields: from Functions to Rewrite Rules

Mirko Viroli
Alma Mater Studiorum – Università di Bologna, Italy
Email: mirko.viroli@unibo.it

***Abstract*—Among the many models used to specify spatial computations in terms of (computational) fields, those based on some form of rewrite rules have been proposed for their ability to support the development of open and situated systems—e.g., on top of distributed tuple space infrastructures. However, due to lack of modularity abstractions, one such approach hardly tackles multi-level combination of different fields, as typically happens in non-trivial self-organisation patterns. In this paper we introduce a functional notation to express computational fields (and their combination), and define its semantics by a translation into a rewrite-based model. The proposed language, which is reminiscent of Proto, includes a somewhat narrow set of constructs, yet providing a good trade-off between expressiveness and formal tractability. In fact, we state for it the confluence property, namely, so-called "don't care non-determinism": this implies – among the others – that specifications satisfying certain monotonicity properties result in fields that stabilise in linear time to a state of easily predictable shape.**

I. Introduction

Chemical-oriented computational models have a large tradition, beginning with the work of Gamma [2], the chemical abstract machine [8], and then higher order chemical computing [1]. They are based on the idea of structuring computation in terms of rewrite rules for multiset of data-terms. Recent works extend this approach to spatial computing, allowing such rules to span information on a node and its neighbourhood [23], [20]. A natural engine for this kind of approach is the distributed tuple space architecture [13]: a tuple injected in a node gets diffused and aggregated by locally-applied rules, thus resulting in whole computational fields (of tuples)—e.g. gradients [5]. This approach finds notable applications in the context of pervasive computing applications, as recently being developed in the context of SAPERE EU project [24], [15].

However, when expressing complex self-organisation patterns [10], in which spatial structures result from the combination of many computational fields, the rule-based approach hardly scales, for it lacks the desired level of modularity. Additionally, it makes it very difficult to properly engineer computational fields, e.g., investigating the conditions under which we can formally predict their stability and evolution over time—simulation remains the only viable approach [20], [9], [16]. So, while some form of chemical-like rewrite rules can serve as a sort of "bytecode" language for spatial computing in open and situated systems, we believe that different approaches can be envisaged that rely on higher-level programming metaphors and can possibly be compiled into such bytecode.

The functional approach adopted in Proto [14], [4], [19] is a first choice in this direction, for its expressiveness and ability of naturally filling the gap between aggregate-level specification of behaviour and single-device interactions. However, it cannot be taken "as is" at this stage of our research for basically two reasons: first, it is still a too large and expressive language, hampering the ability of reasoning about programs; and second, it does not perfectly fit the rewrite nature of chemical computing. Although future works will possibly aim at solving these problems, in this paper we proceed identifying a tiny functional language to express computational fields (which in fact resembles in many ways a fragment of Proto), with arguable good trade-off between expressiveness and tractability. It is based on the following few constructs: *(i)* functional composition, *(ii)* function definition and call (without recursion), *(iii)* pointwise operators, *(iv)* spread of information (by which field evolution is achieved), and *(v)* corresponding aggregation (based on a "minimum" function). This language is shown to allow expressing a rather wide set of useful computational fields.

On top of this language we provide two contributions. First, we formalise its semantics by a translation into a model of rewrite rules inspired by the work in [15], [24], thus enabling a rather direct implementation on top of distributed tuple spaces, as well as shading light towards the relationship between the functional and rewrite spatial computing metaphors. Second, we prove that under certain monotonicity conditions (not very restrictive for the proposed language, actually) any computational field expressed with the proposed language has the *confluence property* [12]. This property gives us a proof technique to state stabilisation of computational fields, and to easily characterise what the final state would be [1].

The remainder of this paper is organised as follows: Section 2 motivates, presents and exemplifies the proposed language, Section 3 describes the underlying rewrite model we consider, Section 4 defines how the proposed language can be translated into corresponding rewrite rules, Section 5 presents the confluence property and its implications, and Section 6 concludes discussing related and future works.

[1]Proof sketches are not reported here for the sake of brevity. The interested reader can read them in [18]

II. A HIGH-LEVEL LANGUAGE OF COMPUTATIONAL FIELDS

A. Motivation

This work starts from the consideration that there is a correspondence between the following two ways of generating a hop-count gradient structure. The first, which we call the "rewrite style", is made of a system of two (chemical-like) rewrite rules of the kind:

$$x + y \Rightarrow \mathit{min}(x, y) \qquad x \Rightarrow (x+1)^{\rightsquigarrow}$$

where x and y are "molecules" (possibly implemented as tuples) residing in the same space, function *min* takes the one with minimum distance value, $(x+1)$ creates a molecule similar to x but with increased distance, and operator $\rightsquigarrow$ denotes that the molecule will be spread to neighbours [20]. The second, which we call the "functional style", is constructed by the Proto specification

```
(rep d (inf) (mux (sense 1) 0
                  (min-hood (nbr (+ d 1))))))
```

which defines a field that is initially infinity everywhere, and evolves by setting the initial value 0 in the source (where sensor 1 returns positive value), and the minimum of neighbourhood values plus one everywhere else.

Of course, the two systems have many differences and underlying assumptions, including: the message-caching technique sustaining `nbr` construct in the functional style, which has no counterpart in the rewrite style; the initial value of the field (infinity in the functional style and "undefined" in the rewrite-style approach); the openness assumption of the rewrite style, in which the initial 0 value in the source is assumed to be externally provided; and finally the need of properly imposing an ordering/rate of rewrite rules in the rewrite approach. In this paper we hence try to investigate on the relationship between the two styles, and also on the idea that it can be proved, because of a certain monotonicity structure of the above specifications, that the resulting field surely stabilise to the gradient pattern.

We proceed by identifying a functional language inspired by Proto, but releasing a number of assumptions for the sake of simplifying the language (and consequently, reducing its expressiveness in favour of enhancing tractability):

- While the idea of defining and combing "functions" is retained, we neglect constructs explicitly representing state and its evolution (namely, `rep`), since handling variables introduces many technical difficulties (especially when they are nested). We shall still use variables for function formal parameters, though.
- In our model, evolution is intertwined with interaction, and is never explicitly mentioned. Instead we have a notion of spreading the values of a field in neighbours and eventually re-aggregate them into a single one in each node. As suggested in [3], we shall allow only aggregation by extraction of the minimum value (according to any user-defined ordering of values).
- We do not have the powerful restriction operator of Proto: we rely on a simple mechanism rooted on the idea that fields are partial mappings, namely, they can be undefined in certain regions of the network.
- There are no special constructs to model the "environment" (time passing, topological structure): inspired by the work in [21], [23] such information, like "sensors" in Proto, is assumed to be externally provided in the form of "environment fields".

B. Syntax

In the following we let meta-variable f range over built-in function names, x over variable names, d over user-defined functions, and σ over environment fields. In the surface syntax we introduce here, they will all be expressed as literals: f by non capitalised characters, x by capitalised characters, d starts by one capitalised character, and σ starts with symbol `#`. We use notation $\overline{x}$ to mean a (possibly empty) comma-separated list of variables, also written $x_1, \ldots, x_n$.Such an overbar notation will be used for other elements as well (fields $\overline{F}$, atoms $\overline{a}$, and so on). The syntax of the language is expressed by the abstract grammar:

a	::=	`number` | `boolean` | `string`	Atom
v	::=	`#` | a | $\langle \overline{a} \rangle$	Value
V	::=	x | v	Var-value
A	::=	V | $\langle \overline{V} \rangle$	Var-atom
F	::=	A | σ | $f(\overline{F})$ | $\{F\}$ | $[F]$ | $d(\overline{F})$	Field
D	::=	`def` $d(\overline{x})$ `is` F	Definition

A variable atom A is an atom, symbol `#` means no-value, a variable, or a tuple $\langle V_1, \ldots, V_n \rangle$ of these elements. A field F, a (possibly partial) mapping from locations to (sets of) values v, can be: the constant field A, the environment field σ, a functional combination $f(\overline{F})$ of fields $\overline{F}$ by built-in function f – possibly using prefix notation for unary functions, infix notation for binary ones, and notation $F\texttt{?}F\texttt{:}F$ for standard ternary conditional operator –, the *spreading field* $\{F\}$ obtained by spreading the local value of F to all neighbours, the *aggregating field* $[F]$ obtained from (previously spread field F) by taking the minimum value in each node, and finally the functional combination $d(\overline{F})$ of fields $\overline{F}$ by user-defined function d. Functions are defined by standard sentences of the kind "`def` $d(\overline{x})$ `is` F", where $\overline{x}$ are formal parameters and F (possibly including parameters) is the expression body. Built-in functions include mathematical ones, and other utility functions such as unary functions *1st*, *2nd*,..., to access i-th element of a tuple, and n-ary *tup* to create a tuple of n values. We shall use infix operator "$\wedge$" for the minimum of two values—according to an externally-provided ad-hoc minimum function.

We assume that a program made of a set of definitions and one top-level field is well-structured if the graph of user-defined function usage is acyclic (i.e., we do not permit recursion). Additionally, we have the following constraints on occurrences of spreading and aggregating fields: *(i)* each

```
def Restrict(FIELD,WHERE) is WHERE?FIELD:#
def LBFilter(FIELD,BOUND) is FIELD<=BOUND?FIELD:#

def Gossip(VAL) is [{VAL}]
def Gossip-max(VAL) is -[-{VAL}]
def Gossip-val(SRC,VAL) is Gossip(Restrict(VAL,SRC))

def GradRNG(SRC,RANGE) is [{SRC}+RANGE]
def Grad-count(SRC) is GradRNG(SRC,1)
def Grad(SRC) is [{SRC}+#RNG]
def Grad-zero(SRC) is Grad(SRC*0)

def Grad-bound(SRC,BND) is [LBFilter({SRC}+#RNG,BND)]
def Grad-obs(SRC,OBS) is [Restrict({SRC}+#RNG,not(OBS))]

def Dist(SRC,DEST) is Gossip(Restrict(Grad(SRC),DEST))
def Channel(SRC,DEST,WIDTH) is
    (Grad(SRC)+Grad(DEST)+WIDTH)<Dist(SRC,DEST)
def GradChannel(SRC,DEST,WIDTH) is
    Restrict(Grad(SRC),Channel(SRC,DEST,WIDTH))

def addTo1st(T,V) is tup(1st(T)+V,2nd(T))
def GradCast(SRC,VAL) is [addTo1st({<SRC,VAL>},#RNG)]
```

Fig. 1. Example definitions, inspired by related Proto programs

spreading field is enclosed in one aggregating field (e.g., we do not accept field $\{\texttt{\#a}\}$), and *(ii)* each aggregating field includes exactly one (top-level) spreading field (e.g., we do not accept $[\{\texttt{\#a}\} + \{\texttt{\#b}\}]$). A valid field is e.g. $[\{1 + [1 + \{\texttt{\#a}\}]\}]$. The reasons for this constraints is that we need to strongly and directly couple spreading and aggregating fields to create the proper feedback loop.

C. Examples

Example definitions of useful fields are reported in Figure 1. Function `Restrict` applied to `FIELD` and `WHERE` creates a new field that is `FIELD` where `WHERE` is true, and is undefined everywhere else. Similarly `LBFilter` creates a new field identical to `FIELD` where it is not greater than `BOUND`, and is undefined everywhere else.

`Gossip` accepts a field `VAL` and yields a new constant field holding the minimum value of it—in each node the value is spread to neighbours, and the result is the minimum value received from neighbours. `Gossip-max` is similar but gossips the maximum value: this is achieved by spreading around the negated value, taking the minimum of the received ones and then negating again. `Gossip-val` spreads the minimum value of `VAL` considered where field `SRC` is true—as an example, `Gossip-val(#sns,1)` gossips value 1 from where environmental field `#sns` is true.

`GradRNG` creates a gradient from the source `SRC` (defined only in the source nodes where the gradient value is fixed), and increasing in each node by the local value of `RANGE`. `Grad-count` is hence trivially an hop-count gradient. `Grad` is instead a gradient based on an externally provided distance increase value in each node, provided and maintained by the environment field denoted `#RNG`. Should we want to use this field to represent an estimation of physical distances (like construct `nbr-range` in Proto) we should consider one physical location per physical device with value 0 for `#RNG`, and a virtual one per directional connection between two devices with the estimated distance as value for `#RNG`. Finally, `Grad-zero` makes sure that we fix value 0 for the gradient where `SRC` is defined—as an example, `Grad-zero(#sns)` is a field of distances from the nearest node where `#sns` is defined.

`Grad-bound` and `Grad-obs` use (directly or indirectly) `Restrict` to limit the region where gradient propagates: the former limits propagation as the gradient reaches a bound value `BND`, the latter propagates everywhere obstacle `OBS` is false.

`Dist` creates a constant field with the minimum distance from where `SRC` and `DEST` are defined—note such a distance is obtained by restricting the gradient of `SRC` where `DEST` is defined. `Channel` is a boolean field that is true in all nodes whose distance from the minimum path from `SRC` and `DEST` is `WIDTH`. Finally `GradChannel` creates a gradient from `SRC` which propagates only inside a channel of width `WIDTH` directed to `DEST`—a sort of slide area from `DEST` to `SRC`.

The last case is `GradCast` which spreads a gradient from `SRC` additionally carrying the value `VAL` stored where `SRC` is defined. Assuming lexicographic ordering of tuples is applied, this makes sure that in each node the value coming from the nearest source will establish: if `VAL` have distinct values where `SRC` is defined, namely in so-called *source nodes*, a partition of the network emerges with exactly one region per source node, tagged by the corresponding value of `VAL`.

III. A REWRITE-ORIENTED PLATFORM

We now explain the computational model of rewrite rules adopted in this paper as target for the translational semantics. Among the many forms rewrite rules can take, our model is inspired by previous works of ours in the context of chemical tuple spaces [20] and their application to the SAPERE approach in the context of pervasive computing [15].

A. Architecture

We shall assume that each location (or node) of the network holds one local *annotation space*, and has direct interaction abilities with locations in the neighbourhood. We use the terminology of [15], calling the data-terms stored in the space *annotations* and not tuples, to avoid confusion with the different concept of a tuple as a possible kind of value, introduced in previous section. Each annotation is a pair of a field unique identifier i and the local value v, written $[i : v]$. Note that the field with identifier i is made by all annotations with id i available in the system, and that annotation spaces are actually sets of annotations—duplication is not permitted. Agents residing in each location and interacting with the annotation space are in charge of keeping updated the annotations corresponding to all environment fields.

Each annotation space carries a fixed set of rewrite rules (the same set occurs in each space) and executes them according to the following dynamics: it *(i)* sleeps for a given amount of time, *(ii)* wakes up and clean the annotation space except environment fields, which get possibly updated, *(iii)* transfers the content of the message queue into the annotation space,

(iv) executes all rewrite rules until a fix-point is reached, *(v)* sends all annotations marked as "to-be-spread" to neighbours, and then go back to sleep—a similar computation round is present in [14], [21].

To tackle the difficulty of providing a fully synchronous firing model for nodes, thus tolerating the case in which a node lacks messages coming from neighbours for it fires too frequently, we also assume that input messages are cached as in [14]. New messages overwrite the cache, and a message is actually dropped from the cache only if the underlying platform perceives a permanent failure of the connection with the message source.

B. Rewrite Model

We let metavariable W generalise over V by adding to it expressions of the kind $f(\overline{V})$. We abuse the overbar notation writing, $[\overline{i} : \overline{W}]$ as a shorthand for $[i_1 : W_1] + \ldots + [i_n : W_n]$, and similarly for $[\overline{i} : \overline{W}]^+$, which denotes an annotation to be spread to neighbours. A rewrite rule R takes the form:

$$[\overline{i} : \overline{V}] \rightarrowtail^{p}_{name} [\overline{i}' : \overline{W}] + [\overline{i}'' : \overline{W}']^+$$

Tag *name* is the rule name, while p is an indication of priority: we use $*$ to mean high priority, and leave it as blank for low priority—in a nature-inspired chemical model they would mean high rate and low rate [20]. For a rule to be valid, each variable on the right-hand side ($\overline{W}$ and $\overline{W}'$) should also occur on the left-hand side ($\overline{V}$). Rules are applied in a given location as follows:

- We look for one substitution of the variables $\overline{x}$ in $\overline{V}$ to values $\overline{v}$ forming a set of annotations $[\overline{i} : \overline{V}\{\overline{v}/\overline{x}\}]$ which exist in the space, and accordingly remove them.
- We apply substitution $\{\overline{v}/\overline{x}\}$ to the right-hand side, and then evaluate the resulting expressions (which are ground by construction) to obtain products $[\overline{i}' : \overline{v}']$ and $[\overline{i}'' : \overline{v}'']^+$. The set of annotations $[\overline{i}' : \overline{v}']$ are immediately injected in the space, while annotations $[\overline{i}'' : \overline{v}'']$ are added to the queue of outgoing messages. Addition of a new annotation in the space (or queue of outgoing messages) has no effect if an equivalent one already exists. We say that at a given time a rule is *enabled* if it can be applied and its application would change the space.
- Application of rules is non-deterministic, with the only constraint that a low-priority rule is never fired if an enabled high-priority rule exists. Application of rules carry on until no rule is enabled.

IV. Translational Semantics

In this section we fill the gap between the proposed functional language and the underlying rewrite-rule semantics described in previous two sections, orderly.

A. Core Source Language

As far as describing the translation details is concerned, it is not necessary to consider the entire surface language proposed, for the following reasons: *(i)* constant fields can be dropped from the language as they can be seen as 0-ary built-in functions; *(ii)* since we do not deal with recursive functions, user-defined functions can always be inlined in the top-expression, so the concept of function definition and call may be simply skipped and we can just focus on built-in functions; *(iii)* we can consider the composition of built-in functions as being built-in functions as well; *(iv)* due to the constraints on the mutual occurrence of spreading and aggregating functions, we can assume they are always present in expressions of the kind $[\ldots\{F\}\ldots]$, and can hence simply consider syntax $[g(\{F\}, \overline{F})]$ for some built-in function g. Accordingly, from the remainder of the paper we restrict our attention to the following core syntax:

$$F \quad ::= \quad \sigma \mid f(\overline{F}) \mid [g(\{F\}, \overline{F})]$$

where f, g belong to an extended set of built-in functions. As an example, `GradCast(#S,#V)` is equivalent to $[g(\{f(\texttt{\#S}, \texttt{\#V})\}, \texttt{\#RNG})]$ where g is function `addTo1st` and f is `tup`.

B. A set of universal rules

To support the translation, in this paper we shall focus only on the following kinds of rule:

$$\begin{aligned}
\mathcal{F}(i, \overline{i}, f) &= [\overline{i} : \overline{x}] \rightarrowtail_F [\overline{i} : \overline{x}] + [i : f(\overline{x})] \\
\mathcal{C}(i_s, i_c, \overline{i}, g) &= [i_s : x] + [\overline{i} : \overline{x}] \rightarrowtail^*_C [\overline{i} : \overline{x}] + [i_c : g(x, \overline{x})] \\
\mathcal{A}(i_c) &= [i_c : x] + [i_c : x'] \rightarrowtail^*_A [i_c : x \wedge x'] \\
\mathcal{O}(i_c, i, i_o) &= [i_c : x] + [i : x'] \rightarrowtail_O [i : x'] + [i_o : x \wedge x'] \\
\mathcal{S}(i_o, i_s) &= [i_o : x] \rightarrowtail_S [i_o : x] + [i_s : x]^+
\end{aligned}$$

Function rule $\mathcal{F}(i, \overline{i}, f)$ is used to produce annotation with id i out of function F and input annotations $\overline{i}$. Contextualisation rule $\mathcal{C}(i_s, i_c, \overline{i}, g)$ turns annotation i_s into one i_c obtained by applying to i_s function g also considering as arguments the content of annotations $\overline{i}$. Aggregation rule $\mathcal{A}(i_c)$ takes two annotations i_c and retains the one with smaller value. Output rule $\mathcal{O}(i_c, i, i_o)$ concludes the spreading/aggregation process by turning the resulting annotation i_c into one i_o whose value is the minimum between i_c's value, x, and the input annotation i's value, x'. Finally, spreading rule $\mathcal{S}(i_o, i_s)$ spreads the value of i_o's annotation into field i_s. Although $\mathcal{O}(i_c, i, i_o) = \mathcal{C}(i_c, i_o, i, \wedge)$ we prefer to keep aggregation and output rules distinct for the sake of presentation.

C. Translation

The basic idea behind the translation is that each purely functional field $f(\overline{F})$ is created out of parameters by rule $\mathcal{F}$, while spreading-aggregation field $[g(\{F\}, \overline{F})]$ is achieved by the remaining four rules: $\mathcal{C}$ and $\mathcal{A}$ execute until all incoming messages (i_s) have been contextualised by function g and then min-aggregated (into i_c), then the minimum value from the result of contextualisation and the result of F (i) yields the result by $\mathcal{O}$ (i_o), which is also spread by $\mathcal{S}$ (i_s again). As an example, field `Grad-zero(#S)`, that is `[{#S*0}+#RNG]`, is handled by the following rules, which create field i_o out of

#S and #RNG, and using temporary fields i_i (the local input), i_s (the spread field), and i_c (the aggregating field):

$$[\texttt{\#S} : x] \rightarrowtail_F [\texttt{\#S} : x]+[i_i : x * 0]$$
$$[i_s : x_s]+[\texttt{\#RNG} : x_r] \rightarrowtail^*_C [\texttt{\#RNG} : x_r]+[i_c : x_s + x_r]$$
$$[i_c : x]+[i_c : x'] \rightarrowtail^*_A [i_c : x \wedge x']$$
$$[i_c : x]+[i : x'] \rightarrowtail_O [i : x']+[i_o : x \wedge x']$$
$$[i_o : x] \rightarrowtail_S [i_o : x]+[i_s : x]^+$$

This translation can be formalised as a ternary relation $|F| \stackrel{i}{=} \overline{R}$ between a field F, its identifier i, and the rules that produce it, recursively defined by the following deduction system:

$$\frac{-}{|\sigma| \stackrel{\sigma}{=} \cdot} \quad \text{[ENV]}$$

$$\frac{\mathit{fresh}(i) \qquad |\overline{F}| \stackrel{\bar{i}}{=} \overline{R} \qquad R_F = \mathcal{F}(i, \bar{i}, f)}{|f(\overline{F})| \stackrel{i}{=} (\overline{R}\ R_F)} \quad \text{[FUN]}$$

$$\frac{\begin{array}{c}\mathit{fresh}(i_s, i_c, i_o) \quad |F| \stackrel{i}{=} \overline{R} \quad |\overline{F}| \stackrel{\bar{i}}{=} \overline{R}' \\ R_C = \mathcal{C}(i_s, i_c, \bar{i}, g) \quad R_A = \mathcal{A}(i_c) \\ R_O = \mathcal{O}(i_c, i, i_o) \quad R_S = \mathcal{S}(i_o, i_s)\end{array}}{|[g(\{F\}, \overline{F})]| \stackrel{i_o}{=} (\overline{R}\ \overline{R}'\ R_C\ R_A\ R_O\ R_S)} \quad \text{[NBR]}$$

Deduction rule [ENV] generates no rule to handle environment fields, and use their name σ as field identifier. Deduction rule [FUN] generates all the rules necessary to produce arguments ($\overline{R}$), and add function rule R_F that produces a field with the freshly generated id i. Similarly, [NBR] generates all the rules necessary to produce arguments $\overline{F}$ and F, and then add four instances of the remaining rules, by properly generating the fresh identifiers i_s, i_c and i_o.

V. Confluence

We consider a system composed of a directed graph of locations, capable of the following kinds of transition: *(i)* the whole computation of a round by a node n, called "firing of n", causing a change in the set of annotations and dispatch of messages to neighbours; *(ii)* a new (communication) link can be added to the graph; *(ii)* an existing link of the graph can be removed; *(iv)* some value of an environment field can change.

Definition 1 (Fair evolution). *A system is said to have* fair evolution *if in any computation path, at each step of the path any node n fires again.*

Definition 2 (Transitory evolution). *A system is said to have* transitory evolution to network topology $\mathcal{T}$ *if it starts from a graph topology $\mathcal{T}' \subseteq \mathcal{T}$, no links outside $\mathcal{T}$ are ever added, and it eventually reaches a state in which there is topology $\mathcal{T}$, and from then no changes to topology and to environment fields occur.*

Definition 3 (Well-monotonicity). *A field F (along with the functions it uses, and the order relation it bases function min upon) is said to be* well-monotonic *if:* (i) *each function g is such that forall $\overline{v}$, $g(v, \overline{v}) \geq v$ and $g(v, \overline{v}) \geq g(v', \overline{v})$ if $v \geq v'$; and* (ii) *there is no infinite ascending chain of values that can be constructed where an element is obtained by previous one v by formula $g(v, \overline{v})$ for some g and $\overline{v}$—chosen arbitrarily at each point in the chain.*

Note that well-monotonicity would typically imply to impose an upper bound to the set of values (preventing fields from diverge to infinity)—a condition not very hard in practice.

Theorem 1 (Confluence). *A system with fair and transitory evolution running a well-monotonic field is* confluent *[12].*

Definition 4 (Stability). *A system in state $\mathcal{S}$ is said to be* stable *if firing at any node does not change the state of annotation spaces.*

Confluence property is key: it implies that if we are able to find one computation path leading to a stable state, then this stable stable is eventually reached in any path. A key brick is the unveiling of outcomes of spreading and aggregation, explained by the following theorem.

Theorem 2 (Outcome of spreading-aggregation). *A system with fair and transitory evolution and running the well-monotonic field $F_s = [g(\{F\}, \overline{F})]$ (where $F, \overline{F}$ are assumed to be constant over time), stabilises to a field mapping node n to the value v computed as the minimum value it would take in any smaller network made of a single directed path from a node in the domain of F to n. Let $\pi = n_0, \ldots, n_k$ be such a path, the value that the final field F_s takes in any node is recursively computed as:*

$$\begin{array}{rcl} F_s(n_0) & = & F(n_0) \\ F_s(n_j) & = & g(F_s(n_{j-1}) \wedge F(n_j), \overline{F}(n_j)) \end{array}$$

Additionally, and assuming all nodes are given the same rate of firing, the time to reach the stable value in a node n_j is linear in j.

More generally, we can state that a system running the combination of stabilising fields $\overline{F}$ stabilises to a field obtained by combing the stable state reached from each element in $\overline{F}$. Concerning a spreading-aggregation field $[g(\{F\}, \overline{F})]$, it states that one can reason about its final shape by assuming that $F, \overline{F}$ are immutable—have already reached their stable state.

Theorem 3 (Compositionality and Stability). *A system with fair and transitory evolution and running a well-monotonic field F reaches a stable state written $\Downarrow F$. Abusing the notation considering $\Downarrow F$ itself as a non-evolving field, we can write:*

$$\begin{array}{rcl} \Downarrow f(\overline{F}) & = & f(\Downarrow \overline{F}) \\ \Downarrow [g(\{F\}, \overline{F})] & = & \Downarrow [g(\{\Downarrow F\}, \Downarrow \overline{F})] \end{array}$$

VI. Conclusions

This paper investigates the problem of engineering computational fields (from specification down to implementation) in such a way that their evolution and final state is formally predictable. It provides the following contributions: *(i)* a tiny functional language able to express a wide set of useful

computational fields, *(ii)* a translation of it into an easy-to-implement rewrite-model, and *(iii)* a set of sufficient conditions to make the expressed computational fields (compositionally) stable—via the notion of confluence.

To the best of our knowledge, the only work aiming at a mathematical proof of stabilisation for computational fields is [5]. There, a self-healing gradient algorithm called CRF (constraints and restoring forces) is introduced to estimate physical distance in a spatial computer, where the neighbouring relation is fixed to unit-disc radio, and node firing is strictly connected to physical time. Compared to our approach, the work in [5] tackles a more specific problem, and is highly dependent on the underlying spatial computer assumptions. Our confluence-based results instead, although used on a rather rigid computational model, apply to a wide set of scenarios, since it makes no assumption on the network topology, rather weak ones on node firing, and can be applied to various fields.

The work proposed in this paper leads to several future directions of investigation. First, we aim at weakening the sufficient conditions we found for stabilisation, namely, addressing a wider set of computational fields. For instance, it would be interesting to relax the need of disallowing infinite chains of node updates (used in the definition of Well-montonicity), and topological conditions of transient evolution.

Second, we believe some of our results can be applied to many spatial computing languages [6], there included Proto, as recently it has been given an operational semantics [19]. We speculate that stability might be stated for Proto programs featuring the use of `nbr` and `rep` constructs only in the following combination:

```
(rep var loc (f (min-hood (nbr (g var)))))
```

where `loc` is a local value, `f` any pointwise function, and `g` a monotonic pointwise function in the sense described in Definition 3. The solution of the stability problem can also be a starting point for proving the related (but not equivalent) problem of convergence of a field evolution as the network density and firing rates tend to infinite [7].

Third, due to the mapping to rewrite rules, the present work can apply also to the SAPERE framework [24]. Namely, it suggests that using only five general *eco-laws* adhering to the five kinds presented in Section IV-B, and adding the monotonicity constraints, can likely guarantee again the stability of the pervasive ecosystem dynamics [22].

Finally, it would surely be interesting to look at other reasearch areas where the notion of confluence and stability have been used, e.g., in term rewriting systems [12], chemical-oriented computing [1], sensor networks [11], and distributed algorithms [17], to bridge with some existing technique so as to possibly investigate a more general approach.

ACKNOWLEDGMENT

This work has been supported by the EU-FP7-FET Proactive project SAPERE Self-aware Pervasive Service Ecosystems, under contract no.256873

REFERENCES

[1] J.-P. Banâtre, P. Fradet, and Y. Radenac. A generalized higher-order chemical computation model. *Electr. Notes Theor. Comput. Sci.*, 135(3):3–13, 2006.

[2] J.-P. Banâtre and D. Le Métayer. Gamma and the chemical reaction model: Ten years after. In *Coordination Programming*, pages 3–41. Imperial College Press London, UK, 1996.

[3] J. Beal. A basis set of operators for space-time computations. In *Spatial Computing Workshop*, 2010. Available at: `http://www.spatial-computing.org/scw10/`.

[4] J. Beal and J. Bachrach. Infrastructure for engineered emergence in sensor/actuator networks. *IEEE Intelligent Systems*, 21:10–19, 2006.

[5] J. Beal, J. Bachrach, D. Vickery, and M. Tobenkin. Fast self-healing gradients. In *Proceedings of ACM SAC 2008*, pages 1969–1975, 2008.

[6] J. Beal, S. Dulman, K. Usbeck, M. Viroli, and N. Correll. Organizing the aggregate: Languages for spatial computing. In *Formal and Practical Aspects of Domain-Specific Languages: Recent Developments*, chapter 16, pages 436–501. IGI Global, 2013.

[7] J. Beal, K. Usbeck, and B. Benyo. On the evaluation of space-time functions. *The Computer Journal*, 2012. Online first.

[8] G. Berry and G. Boudol. The chemical abstract machine. *Theoretical Computer Science*, 96(1):217–248, apr 1992.

[9] M. Casadei, L. Gardelli, and M. Viroli. Simulating emergent properties of coordination in Maude: the collective sort case. volume 175(2) of *ENTCS*, pages 59–80. Elsevier Science B.V., 2007.

[10] J. L. Fernandez-Marquez, G. D. M. Serugendo, S. Montagna, M. Viroli, and J. L. Arcos. Description and composition of bio-inspired design patterns: a complete overview. *Natural Computing*, 12(1):43–67, 2013.

[11] C. Fischione and U. Jonsson. Fast-lipschitz optimization with wireless sensor networks applications. In *Proceedings of IPSN 2011*, pages 378–389. IEEE, 2011.

[12] G. Huet. Confluent reductions: Abstract properties and applications to term rewriting systems: Abstract properties and applications to term rewriting systems. *J. ACM*, 27(4):797–821, Oct. 1980.

[13] M. Mamei and F. Zambonelli. Programming pervasive and mobile computing applications: The tota approach. *ACM Transactions on Software Engineering Methodologies*, 18(4):1–56, 2009.

[14] MIT Proto. `http://proto.bbn.com/`, 2013.

[15] S. Montagna, M. Viroli, J. L. Fernandez-Marquez, G. Di Marzo Serugendo, and F. Zambonelli. Injecting self-organisation into pervasive service ecosystems. *Mobile Networks and Applications*, pages 1–15, September 2012. Online first.

[16] D. Pianini, S. Montagna, and M. Viroli. Alchemist: a chemical-oriented multi-agent based simulation framework. *Journal of Simulation*, 2013. In press. Appeared online with DOI: 10.1057/jos.2012.27.

[17] S. Tixeuil. Algorithms and theory of computation handbook. chapter Self-stabilizing algorithms, pages 26–26. Chapman & Hall/CRC, 2010.

[18] M. Viroli. Engineering confluent computational fields: from functions to rewrite rules – proof sketches of theorems. `http://apice.unibo.it/xwiki/bin/download/Publications/FieldsSCW2013/proofs.pdf`, 2013.

[19] M. Viroli, J. Beal, and K. Usbeck. Operational semantics of Proto. *Science of Computer Programming*, 2012. Online first.

[20] M. Viroli, M. Casadei, S. Montagna, and F. Zambonelli. Spatial coordination of pervasive services through chemical-inspired tuple spaces. *ACM Transactions on Autonomous and Adaptive Systems*, 6(2):14:1 – 14:24, June 2011.

[21] M. Viroli, D. Pianini, and J. Beal. Linda in space-time: an adaptive coordination model for mobile ad-hoc environments. In *Coordination Models and Languages*, volume 7274 of *Lecture Notes in Computer Science*, pages 212–229. Springer, 2012.

[22] M. Viroli, D. Pianini, S. Montagna, and G. Stevenson. Pervasive ecosystems: a coordination model based on semantic chemistry. In *Proceedings of ACM SAC 2012*, pages 295–302. ACM, 2012.

[23] M. Viroli and G. Stevenson. On the space-time situation of pervasive service ecosystems. In *Workshop on Spatial Computing*, Valencia, Spain, June 2012. Available at: `http://www.spatial-computing.org/scw12/`.

[24] F. Zambonelli, G. Castelli, L. Ferrari, M. Mamei, A. Rosi, G. D. M. Serugendo, M. Risoldi, A.-E. Tchao, S. Dobson, G. Stevenson, J. Ye, E. Nardini, A. Omicini, S. Montagna, M. Viroli, A. Ferscha, S. Maschek, and B. Wally. Self-aware pervasive service ecosystems. *Procedia CS*, 7:197–199, 2011.

Short Presentations

Composing gradients for a context-aware navigation of users in a smart-city

Sara Montagna and Mirko Viroli
Alma Mater Studiorum–Università di Bologna
Via Venezia 52, 47521 Cesena, Italy
{sara.montagna,mirko.viroli}@unibo.it

ABSTRACT

Recent works suggested the use of self-organising spatial patterns, enacted as computational fields, to the problem of steering users towards their desired destination in complex environments. In a rich and open scenarios, various contextual services can enter the system providing additional information that can be exploited to guide users by suggesting paths that more likely satisfy their preferences. This can be done composing new information with the steering services already available in the system.

Since the type and number of new services available can vary over time, such compositions must be identified dynamically, in an autonomous, spontaneous and unsupervised way. Moreover, in order to avoid the system to be overflooded by services that do not match any user preferences, a mechanism for identifying useless compositions and for removing them should be provided.

In this paper we investigate this problem and propose a *self*-composition approach envisioned to support the composition of new services together with basic services for crowd steering, such that, depending on the actual (spatial/temporal) context in which the composition is deployed, users are steered across the path that better fit their preferences.

1. INTRODUCTION

Context-aware navigation of users in complex environments is a typical problem in pervasive computing. There, many services typically enrich the system with information and functionalities that can be opportunely used for steering users based on their preferences and on contextual information. Such services, are main actors in pervasive computing, and are dynamically injected by humans, institutions, software systems and devices, without an a-priori knowledge of the structure and behaviour of those already available and those that will be injected later. For a context-aware navigation it can be useful to compose the information they provide with the basic navigation service, so as to provide a whole new set of "tailored" navigation services.

Making composition of services effective in such *open* and complex systems is a rather challenging task, which in this paper we aim at tackling by a *self-composition* approach. The term "self" is used to suggest that this process should happen automatically and in an unsupervised way, i.e., without specifying at design time which services are to be composed, and ensuring that the system is not overflooded by useless service compositions.

From [9, 11] we cite the big questions that self-composition brings: *what can make services compose if they were not explicitly designed to do so? how can we decide "whether" and "how" two or more services should compose? how can such decisions be continuously reconsidered depending on the actual (spatial/temporal context) in which the composition is deployed? can we make multi-level composition seemingly work as well?* These problems are far from being fully solved, though several surveys and partial solutions have been proposed in different contexts, classified in three main categories: *(i)* Service Composition in SOA where the static character of traditional composition approaches such as orchestration and choreography has been recently challenged by so-called dynamic service composition approaches, involving semantic relations [2], and/or AI planning techniques to generate processes automatically based on the specification of a problem [14]; *(ii)* Evolutionary techniques such as those based on Genetics Algorithms (GA) have been proposed as well for service composition, such as in [5], motivated by the need of determining the services participating in a composition that satisfies certain QoS constraints, which is known to be a NP-Hard problem; and *(iii)* Competition-based approaches where, in order to tackle the potentially high number of different compositions that can arise in an open system, a mechanism of *coordination reactions* for networked tuple spaces is proposed, showing how it can be applied to support competition and composition in a fully distributed setting [11, 12, 9].

In this paper we adopt the latter approach developing on preliminary results appeared in [9], focussing on the composition of computational fields [1, 7, 12]. We address the problem of automatically composing a gradient-based navigation service [7], with the distributed information provided by contextual sensors available around, detecting presence of traffic, pollution, points of interests, availability of friends —in [9] the unique information of crowded areas were considered. We propose the adoption of a strategy for optimising service composition based on competition of different compositions according to their actual exploitation, hence promoting successful ones while decaying the useless ones, in what can appear as a *spontaneous* selection of the most proper *crowd steering* service.

The scenario is modelled within the framework of pervasive ecosystems [8]. In this computational model, the pres-

Appears in: *Proceedings of the 12th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2013), Ito, Jonker, Gini, and Shehory (eds.), May, 6–10, 2013, Saint Paul, Minnesota, USA.*

ence and activities of all system components is reified in terms of tuple-like items called *Live Semantic Annotations* (LSAs), which are stored across all computational devices of the pervasive computing system. Overall system behaviour is regulated by a set of laws, called *eco-laws*, which act locally on each node and its neighbourhood, combining and manipulating annotations using a chemical style and semantic pattern matching [8]. Accordingly, each service in the system is represented as an LSA. Eco-laws are then in charge of composing these basic LSAs creating new services. When involving the gradient service, this composition realises gradient contextualisation with local information provided by sensors. Each composition is associated with a quality value that can be increased by user positive feedbacks or decreased over time via eco-laws. If a service has a low quality it is removed by the system according to the competition principle: services are used based on their quality, which is the solely means by which the quality level can avoid decreasing to zero.

The remainder of this paper is organised as follows. Section 2 introduces the framework of pervasive ecosystems; Section 3 describes our vision on how self-composition can be achieved in such a framework; the approach application to the context-aware navigation of users is illustrated in Section 4, showing how gradient can be composed with other services using a simulation scenario; and finally Section 5 concludes with a discussion of future works.

2. FRAMEWORK

The framework we rely upon for describing and experimenting our approach is the one of pervasive ecosystem. A detailed description of the framework is provided in [8, 13]. Pervasive ecosystems [15] are characterised by two main features, which influence their underlying abstract architecture and model. On one hand, they should be situated, namely, the activity of any software agent and the data it produces are tightly bound to the agent's physical location: this is because any behaviour should be intrinsically aware of and affect the surrounding context. Situatedness is achieved by infrastructures reifying data, knowledge, and events in the precise point (or region) of space where they pertain, and by promoting interactions based on proximity. Accordingly, a cornerstone of pervasive ecosystems is that a uniform representation is required for the various software agents living within them (whether they run on smartphones, sensors, actuators, displays, or any other computational device). We term such a representation a "Live Semantic Annotation" (LSA) for it should continuously represent the state of its associated component (live), and it should be implicitly or explicitly connected to the domain in which such information is produced, interpreted and manipulated (semantic). The LSAs of each agent are reified in a distributed space (called an "LSA-space") acting as the fabric of the ecosystem, located in the computational device hosting the agent. On the other hand, pervasive ecosystems should be adaptive, exhibiting properties of autonomous adaptation and management to survive contingencies without human intervention and/or global supervision. This is achieved following the natural inspiration [15, 4], by designing system rules that – by acting locally – make global properties emerge dynamically. So, while agents enact their individual behaviour by observing their context and updating their LSAs, global behaviour (i.e., global system coordination) is enacted by self-organising manipulation rules of the LSA-space, which we call eco-laws. They can execute delete/update/create actions applied to a small set of LSAs within the same locality. Moreover they can move or copy LSAs among neighbouring locations, allowing local information to become global. Following [12] such eco-laws are structured as chemical-resembling reactions over LSAs. To have in mind an architectural view of the system, one should think of a very large and mobile set of devices connected to each other based on proximity, creating a distributed "space" – ideally a pervasive continuum – where LSAs form spatial structures that evolve over time. The eco-law engine, accordingly, has to be seen as a distributed one uniformly working on all LSA-space—though we will show that a feasible approach amounts at developing local eco-law engines in each node.

3. THE SELF-COMPOSITION APPROACH

The self-composition approach we adopt here is inspired at *competition-based* approaches [11, 12], already introduced in Section 1. There potentially high number of different compositions that can arise in an open system is managed via a mechanism of competition and composition that is envisioned in a fully distributed setting, guaranteeing that services opportunistically compose in those regions of the network where this results in a more competitive service. While all compositions can possibly take place, thanks to a positive/negative feedback only successful ones are meant to "survive" in the system, the others fading until becoming never actually selected. This is achieved by matching services with requests through a self-regulating dynamics inspired by prey-predator model of population dynamics [3], and by diffusing service tuples using a computational-field approach similar to the one presented in [1].

The incarnation of this approach to the context-aware crowd steering scenario mainly consists in creating new gradients composing the basic gradient – that only contains a measure of distance to the source – with other services providing also contextual information about the different path the user can follow towards the source. These new gradients have a different shape from the basic one, *i.e.* distance value is modified and contextualised. Such modification, according to user profiles, encourages or discourages the path across certain areas whose characteristics are described by those services the gradient is composed with. In order to perform competition among all these composite gradients available, at each of them is associated a value that represents the satisfaction of users that exploited it. This value decays over time, so that if it is not reinforced by new good feedbacks from users it will decay until fading. Once satisfaction is in fact lower then a certain threshold the composition is removed from the system. This process should ensure that only "appreciated" compositions will survive in the system.

4. A SELF-COMPOSITION MODEL FOR CONTEXT-AWARE CROWD STEERING

The framework described so far can be adopted for modelling crowd steering scenarios. In particular we claim that self-composition can be successfully adopted for the context-aware navigation of users to their destinations inside a city. Our scenario is built upon two crucial services: a service that builds a gradient out of a specified source and a crowd

steering service that provides users with a direction to follow. These two basic services can be composed with data produced by other sensors hosted in nodes spread in the city. These services are, in principle, unknown and can vary in time without any predictable plan. They can equip the system with, for instance, the following information: *(i)* traffic levels along the street of the whole city map; *(ii)* measures of smog; *(iii)* measures of the street slopes; *(iv)* information about the street characteristics (is it across a pedestrian area, a park, an industrial zone, a residential district and so on?); *(v)* a map of the city; *(vi)* information about stuff for kids; *(vii)* information about 30 days weather forecast; *(viii)* information about levels of pollution or crime in a city. All these data can be composed together and with the two basic services.

The first goal of this scenario is to demonstrate that in a crowd steering application enriched with a self-composition algorithm, given information about what users prefer to come across, given data provided by sensors, as well as the basic gradient and the crowd steering service, the more satisfiable path can be identified for each user, composing all these services and information together. The second goal is to demonstrate that it is possible to guarantee only useful service to survive and to make this process dynamic, *i.e.* continuously reversible.

4.1 The model

Our application is developed across a city, whose map can ideally be a real map of a city, but in this first example is a fake map. Sensors are located here and there to produce information about the characteristics of different areas of the city. In each node of the network is available at least one basic gradient, for each point of interest in the city, and the crowd steering service. Users specify a set of preferences that characterises their profile.

The model can be discussed in two parts, one that performs self-composition and that can be valid in any scenario, the second one that is more application specific and that describes those services for making crowd steering. The main behaviours of the model for the self-composition part are:

1. Sensor data are reified in the system via LSAs in the closest node/nodes.

2. Self-composition happens through an eco-law that creates from two services A and B a new one that contains all the information of A and B; it can be applied recursively so that we have a multilayer composition. This mechanism can be applied at any service, being completely unaware, at design time, of the specific characteristics of the application. At each composition is associated a quality level SV.

3. If SV is lower than a threshold value, the new service is removed.

4. SV can decrease for two reasons: evaporation – if a service is not used by users, SV evaporates with a certain rate – or negative feedbacks by users.

5. SV is increased by positive feedbacks coming from users or from software components – such as reasoners – possibly available in the system, if the composition is recognised to be crucial.

While the crowd steering components can be modelled as follow:

1. A **gradient service** can be built out of each point of interest (source) identified in the "fake" map above. Composing gradient with one or more services provides a new gradient source tagged with contextual information provided by other services. They are propagated over the system so that in each point a set of gradients is available, each one tagged in a different way.

2. The **crowd steering service** should identify the gradient with the shorter distance and follow it. If at some point the crowd steering service finds a local minimum of that gradient it should look for a new gradient to follow which is again the one with shorter distance.

3. The **gradient contextualisation service** locally modifies the gradient distance according to the user profile. We represent user preferences as a set of pairs preference-(value k_i), where $k_i \in [0, \text{inf}]$. Each k_i estimates how much user is interested to service i with respect to the maximum deviation he/she is inclined to accept. If the preference is not specified we can decide to not consider the correspondent tag (namely $k_i = 0$) or to take a random number for k_i. Distance D of the composite gradient is then updated with the following function:

$$D = D - \sum_{i=0}^{N_{tag}} \frac{k_i}{(D - D_{min})} \quad (1)$$

where N_{tag} is the number of tags stored in the gradient, and D_{min} is the value of distance of the shortest path towards the desired point of interest. The physical distance to travel along the shorter path of a tagged gradient is then decreased as much as the preference i of a tag is strong (value k_i) and as much as the distance to travel is close to the distance of the shortest path. This ensures that if a composite gradient requires to travel a distance too much bigger with respect to the shorter one, its choice will be discouraged. At the same time if the preference is not that strong the decreased factor is not very significative even in the case of small $D - D_{min}$.

To summarise, the effect of gradient distance modification is that the best path perceived by the user – the one with the minimal distance value – will not necessarily be the physically shorter one, but the one that maximises the user satisfaction minimising the deviation, from the physically shorter path, needed to ascend a composite gradient. The distance value of composite gradients will then no more represent only the physical distance but will also include a measure of user thoughts against the service the composite gradient is tagged by, opportunely modulated with respect to the deviation needed to exploit different paths. At the same time, it still remains a real value which can be exploited by the crowd steering service, that, once all the gradient distances have been updated, simply identifies the one with smaller D as the gradient that will steer the user. This behaviour is shown in Figure 2 where users with the same preference are steered along different paths (continuos lines) according to their initial position: the ones on the extreme left are for instance guided in a path that cross the park (*i.e.* the area

$$\langle \texttt{sensor}, Desc, Value, Id\rangle \mapsto \langle \texttt{sensor}, Desc, Value, Id\rangle, \langle \texttt{service}, [Id;], initSV\rangle \quad (1)$$

$$\langle \texttt{service}, A, SV1\rangle, \langle \texttt{service}, B \textbf{ has not } [A], SV2\rangle \mapsto \langle \texttt{service}, A, SV1\rangle, \langle \texttt{service}, B, SV2\rangle, \langle \texttt{service}, B \textbf{ add } [A], initSV\rangle \quad (2)$$

$$\langle \texttt{service}, grad, T, Id, Tag \textbf{ has not } [L], D, SV2\rangle, \langle \texttt{service}, L, SV1\rangle \mapsto \langle \texttt{service}, L, SV1\rangle, \langle \texttt{service}, grad, T, Id, Tag, D, SV2\rangle, \langle \texttt{source}, T, Tag \textbf{ add } [L], D, initSV\rangle \quad (3)$$

$$\langle \texttt{source}, Type, ID, Tag, Distance, SV\rangle \mapsto \langle \texttt{source}, Type, ID, Tag, Distance, SV - \Delta SV\rangle \quad (4)$$

$$\langle \texttt{source}, Type, ID, Tag, D, SV <= 0 >\rangle \mapsto \quad (5)$$

$$\langle \texttt{preference}, Desc, V, K\rangle, +\langle \texttt{sensor}, Desc, Value, Id1\rangle, \langle \texttt{diffLength}, \Delta D\rangle, +\langle \texttt{source}, Type, Id, Tag \textbf{ has } [Id1], Distance, SV\rangle, \langle \texttt{feedback}, Id\rangle \mapsto +\langle \texttt{sensor}, Desc, Value, Id\rangle, +\langle \texttt{source}, Type, ID, Tag, Distance, SV + K/(\Delta D + 1)\rangle \quad (6)$$

Figure 1: Laws describing the crowd steering application

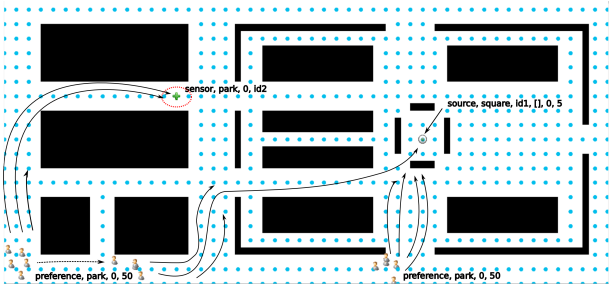


Figure 2: Representation of model behaviour

where a sensor reifies the LSA $\langle \texttt{sensor}, park, 0, id1\rangle$) even though the physically shorter one from their position would follow an other path (dashed line); the one in the middle will instead not cross the park, meaning that the k value associated at the park preference is not sufficient for ignoring the path extension from their initial position.

4.2 A formalisation

We follow the annotation described in [10]. All the information exchanged is in form of annotations, simply modelled as tuples $\langle v_1, \ldots, v_n\rangle$ (ordered sequence) of typed values, which could be for example numbers, strings, structured types, or function names. Rules are expressed in form of chemical-resembling laws, working over patterns of annotations. One such pattern P is basically an annotation which may have some variable in place of one or more arguments of a tuple, and an annotation L is matched to the pattern P if there exists a substitution of variables which applied to P gives L. Moreover we introduce here operators **has** and **has not** applied to a list, meaning, in the conditions, that the element or sublist has not to be contained, and operators **add** and **remove** that forces, in the actions, the element to be added or removed from the list. A law is hence of the kind $P_1, \ldots, P_n \overset{r}{\mapsto} P'_1, \ldots, P'_m$, where: *(i)* the left-hand side (reagents) specifies patterns that should match annotations $L_1, \ldots, L_n$ to be extracted from the local annotation space; *(ii)* the right-hand side (products) specifies patterns of annotations which are accordingly to be inserted back in the space (after applying substitutions found when extracting reagents, as in standard logic-based rule approaches). To allow interaction between different nodes (hence, annotation spaces), it is possible to use a *remote pattern*, written $+P$, which is a pattern that will be matched with an annotation occurring in a neighbouring space.

Each sensor in the system produces data that can be represented as tuples of the kind:

$$\langle \texttt{sensor}, Desc, Value, Id\rangle$$

where variable $Desc$ contains a description of the service (such as "measure of smog" or "measures of slope" or "presence of a park" etc), variable $Value$ possibly the numeric measures acquired by the service, and finally Id couples with the service an univocal identificator. Associated services are then represented as

$$\langle \texttt{service}, List, SV\rangle$$

where variable $List$ collects the set of ids of sensor data it is composed by and SV stores a measure of satisfaction of users exploiting that service. Therefore the condition required for two services to be composed is that they are represented with an LSA matching this pattern. Service composition is then obtained with eco-laws (1) and (2) of Figure 1: the first one associates at each sensor datum a description in terms of service LSA, while the second realises the composition of services adding a new element to the list of ids.

A gradient is represented with the following couple of LSAs for the source and the gradient service

$$\langle \texttt{source}, Type, Id, Tag, Distance, SV\rangle$$
$$\langle \texttt{service}, grad, Type, Id, Tag, Distance, SV\rangle$$

where $Type$ and $Distance$ are typical attribute for a gradient, used respectively to characterise the gradient source and to define the distance of the node to the source. Id, Tag and SV are attributes introduced at the purpose of composition of gradient with other services. They have the same meaning as before. The list of ids contained in the Tag list will be used to contextualise each gradient according to user preferences. Eco-law (3) allows gradient service composition with sensor services: a new source for a gradient with an enriched list of tags is created, out of which a new tagged gradient will be built. Eco-law (4) evaporates, of a parameter ΔSV, the SV value of gradient sources, that will be deleted, via eco-law (5), if it becomes smaller or equal to 0.

Finally each person is modelled as a node hosting the LSA:

$$\langle \texttt{preference}, Desc, V, K\rangle$$

that describes the service the user is looking for ($Desc$), if necessary specifies the value desired for that service (V) and, inside K, quantifies the interest of the user for that service. Reactions that model person behaviour mainly are

those for contextualising gradients and for steering people ascending the chosen gradient. Given their complexity these behaviours are not modelled in terms of eco-laws but with agents that perform the computation described in Section 4.1. Moreover the mechanism for giving feedback to the system about the gradient followed is modelled via eco-law (6). It is built upon two new LSAs:

$$\langle \texttt{feedback}, Id \rangle$$
$$\langle \texttt{diffLength}, \Delta D \rangle$$

the first one is produced by the crowd steering service once a local minimum is found and simply claims that the user followed the gradient Id, the second one is produced by the gradient contextualisation service and in variable ΔD stores the measure of deviation required to follow that gradient (namely $D - D_{min}$ of Equation 1). This eco-law then ensures that once users reach the source of the gradient, they provide feedbacks augmenting gradient's SV of the factor $K/(\Delta D + 1)$, *i.e.* how much the users were interested to that service modulated with how much the deviation costed.

4.3 Simulation results

Simulations are performed using the ALCHEMIST simulator [10], created to smoothly target pervasive ecosystems. It implements an optimised version of the Gillespie's SSA – which has in principle been developed to model the dynamic of chemical solutions – namely the Next Reaction Method [6], that is known for its high performances. To face the model requirements, this algorithm has been properly extended with the possibility to have dynamic reactions, *i.e.*, system transitions that can be added or removed during simulation due to network mobility and unpredicted situations. It is equipped with a Domain Specific Language to flexibly formulate the model in terms of LSAs, eco-laws and environment structure and topology, and with a user interface to load, execute and inspect the simulation.

In Figure 3 are provided a set of snapshots of the simulation we performed. Snapshot *(a)* shows the initial setting: a fake city map with buildings, parks and streets for cars or pedestrians; people are randomly distributed across the map, bringing information about their preferences —they for instance wish to cross a park, to see a lake, to avoid sloping or polluted streets. Snapshot *(b)* shows the overlay network of nodes and with a "plus" symbol those nodes where a service is available – they advertise level of smog and slope, and presence of parks, lakes and churches – while with a "circle" symbol the central area – the square – of the city towards which all the people move, *(i.e.* the main gradient source. From snapshot *(c)* it is possible to observe the content of each node, *i.e.* the set of gradients alive in that moment and the eco-laws resembling the ones of Figure 1 that can possibly be fired inside the node. Moreover, for each eco-law, the scheduling time is provided. Among them also *infinity* appears for some reactions, meaning that their conditions are not satisfied so that they cannot be executed. Finally in snapshot *(d)* people movement towards the city centre is shown. Moreover attention should be paid at the number of services still alive, which is clearly decreased from the initial condition of snapshot *(b)* due to the self-composition algorithm selection.

In Figure 4 measures for the qualitative behaviour of the system are then plotted. The upper one shows how the number of composite gradients available in the whole system varies with time. The second measures how the overall satisfaction of users varies in the same window of time. It is computed as the sum of SV values stored in composite gradient sources. We can observe that while the number of available compositions significantly decrease after a first period of adaptation, the quality of the system still remains high – it begins with low values as soon as no user feedbacks are already available, then it increases until reaching an average value around which it oscillates due to the decay and feedback components –, meaning that clearly only those compositions that satisfy users, survive.

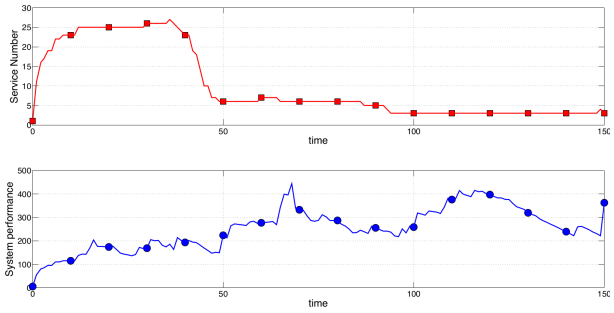


Figure 4: Number of services (up) and system performance (down) over time

5. CONCLUSIONS

In this paper we described an approach for self-composition of services in pervasive system, with the purpose of developing context-aware crowd steering applications. An illustrative model, framed in the pervasive ecosystem framework, is given. Experiments have been performed for validating the approach, and first results are promising.

Future works are devoted to: *(i)* Adopt a composition recommender or an evolutionary technique that selects only a subset of all possible compositions, identifying the most useful according to the actual state of the system. The adoption of heuristic algorithms for instance, may restrict the number of paths available and may provide faster convergence in the identification of the best path *(ii)* Introduce the concept of "dangerous" services. For instance, we can imagine services that advertises levels of pollution or crime in a city. The composition results in higher services identifying areas of the city that the crowd steering service should force to avoid, regardless of user preferences.

6. ACKNOWLEDGMENTS

This work has been supported by the EU FP7 project "SAPERE - Self-aware Pervasive Service Ecosystems" under contract No. 256873.

7. REFERENCES

[1] J. Beal and J. Bachrach. Infrastructure for engineered emergence on sensor/actuator networks. *IEEE Intelligent Systems*, 21(2):10–19, 2006.

[2] M. Beek, A. Bucchiarone, and S. Gnesi. A survey on service composition approaches: From industrial standards to formal methods. In *Technical Report 2006TR-15, Istituto*, pages 15–20. IEEE CS Press, 2006.

(a) Initial setting

(b) Nodes and PoIs

(c) Inspecting the content of one node

(d) PoIs decayed and people moved

Figure 3: Snapshots from ALCHEMIST of the crowd steering application

[3] A. A. Berryman. The origins and evolution of predator-prey theory. *Ecology*, 73(5):1530–1535, October 1992.

[4] J.-P. Bonâtre and D. Le Métayer. Gamma and the chemical reaction model: Ten years after. In *Coordination Programming*, pages 3–41. Imperial College Press London, UK, 1996.

[5] G. Canfora, M. Di Penta, R. Esposito, and M. L. Villani. An approach for qos-aware service composition based on genetic algorithms. In *Proceedings of the 2005 conference on Genetic and evolutionary computation*, GECCO '05, pages 1069–1075, New York, NY, USA, 2005. ACM.

[6] M. A. Gibson and J. Bruck. Efficient exact stochastic simulation of chemical systems with many species and many channels. *J. Phys. Chem. A*, 104:1876–1889, 2000.

[7] M. Mamei and F. Zambonelli. Programming pervasive and mobile computing applications: The tota approach. *ACM Trans. Softw. Eng. Methodol.*, 18(4):1–56, 2009.

[8] S. Montagna, M. Viroli, J. L. Fernandez-Marquez, G. Di Marzo Serugendo, and F. Zambonelli. Injecting self-organisation into pervasive service ecosystems. *Mobile Networks and Applications*, September 2012. Online first, available through DOI: 10.1007/s11036-012-0411-1.

[9] S. Montagna, M. Viroli, D. Pianini, and J. L. Fernandez-Marquez. Towards a comprehensive approach to spontaneous self-composition in pervasive ecosystems. In F. De Paoli and G. Vizzari, editors, *Proceedings of the 13th Workshop on Objects and Agents*. CEUR-WS, 12 September 2012.

[10] D. Pianini, S. Montagna, and M. Viroli. Chemical-oriented simulation of computational systems with alchemist. *Journal of Simulation*, January 2013. Advance online publication.

[11] M. Viroli. A self-organising approach to competition and composition of pervasive services. *Science of Computer Programming*, pages 1–29, oct 2012. In press. Appeared online with DOI: 10.1016/j.scico.2012.10.002.

[12] M. Viroli, M. Casadei, S. Montagna, and F. Zambonelli. Spatial coordination of pervasive services through chemical-inspired tuple spaces. *ACM Transactions on Autonomous and Adaptive Systems*, 6(2):14:1–14:24, June 2011.

[13] M. Viroli and G. Stevenson. On the space-time situation of pervasive service ecosystems. In *Workshop on Spatial Computing*, pages 53–59, Valencia, Spain, June 2012. Informal Proceedings.

[14] Z. Wu, A. Ranabahu, K. Gomadam, A. Sheth, and J. Miller. Automatic composition of semantic web services using process and data mediation. In *Proc. of the 9th Intl. Conf. on Enterprise Information Systems*, pages 453–461, 2007.

[15] F. Zambonelli and M. Viroli. A survey on nature-inspired metaphors for pervasive service ecosystems. *International Journal of Pervasive Computing and Communications*, 7(3):186–204, 2011.

Teams to exploit spatial locality among agents

James Parker and Maria Gini
Department of Computer Science and Engineering, University of Minnesota
Email: [jparker,gini]@cs.umn.edu

***Abstract*—In many situations, task allocation is highly dependent on the spatial locality of the agents and the tasks. In this paper we explore task allocation in complex spatial environments, where the cost of completing a task can vary over time. This work focuses on a subset of cost functions for tasks that grow over time and presents the real-time Latest Finishing First (RT-LFF) algorithm, which strikes a balance between the optimal zero travel time solution and minimizing the amount of time agents spend transferring to tasks in different locations. We also present how spatial knowledge can be used to infer missing knowledge in partially observable spaces.**

I. Introduction

Multi-agent systems offer robustness, flexibility, and efficiency over single-agent systems. The fields of wide-area surveillance, exploration and mapping, transportation, and search & rescue are all being enriched by incorporating multiple agents. In all these examples, not only must agents travel through the spatial topology of the environment to accomplish their goals, but the topology itself can effect how the agent's goals can change over time. However, the benefits of multi-agent systems increases the burden on coordination.

In dynamic environments where the set of agents might change over time, as agents fail or new agents are added to the system, and where new tasks can appear and old tasks can expire, task allocation has to be done in real-time to handle the uncertainty as to location, completion requirements and size of tasks. Also agents travel through unknown and possibly unsafe areas to reach the tasks. Unlike most task allocation problems [1], we focus on domains where the amount of resources that a task requires changes over time.

We present the *Latest Finishing First (LFF)* algorithm for task allocation in domains where the cost of tasks increases with time (Section IV). These types of problems can occur in nature, such as invasive species or forest fires, where if a task is not completed quickly, then it can become difficult or impossible later. We then extend our LFF algorithm to a real-time heuristic version for partially observable dynamic environments where new tasks can appear as time progresses (Section V). Finally, we propose a novel way of incorporating partial information in RoboCup Rescue to accurately predict a model for clustered tasks (Section VI), and we evaluate experimentally our work against other methods (Section VII).

II. Related Work

Multi agent task allocation is well known to be an NP hard problem, giving rise to many different techniques for finding an approximate solution. One example is swarm robotics, where agents often use threshold based methods to individually assess the constraints and their ability to complete each task. If an agent's abilities surpass a threshold on the constraints, then the agent allocates itself to the task. If not, the agent passes the information to other agents. An example is [2], which uses distributed constraint optimization (DCOP) as a basis for task allocation. A comparison between DCOP and other swarm techniques is provided in [3]. In comparison, market inspired auction methods typically require more communication and are more centralized. Zhang et al. [4] present an auction based approach to form executable coalitions, allowing multiple agents from different locations to reach a task and compete it efficiently. Recently, decentralized applications have been designed that add flexibility to the system (e.g., [5]). Our work strikes a balance between swarm and centralization, each agent is directed to an area by a central authority, but upon reaching the destination, agents act on their own individual abilities.

Other approaches have been developed, such as modeling task allocation as a potential game [6]. Sandholm et al. [7] present a generalized coalition formation algorithm which produces solutions within a bound from the optimal via pruning a subset of the search tree. Dang et al. [8] improve on it by further pruning, but the search time is still exponential. The work in [9] focuses on tasks that require more than one agent to do them, while simultaneously trying to use efficiently the agent's resources and time. Our approach also assumes multiple agents are required, but we also allow the requirements of tasks to change over time.

Our work specifically applies to urban search & rescue using the RoboCup Search and Rescue Simulator [10], providing simulations based off street and building maps of real cities. Emergency situations are very time critical and often lacking in information, as demonstrated by [11]. Most notably, when the emergency occurs agents are spatially spread out in a disorganized fashion and must quickly coordinate and form teams to accomplish tasks.

III. Problem description

Our problem is task allocation for multi-agent systems, where agents must travel on designated pathways. The agents are scattered at random initially and need to converge on tasks. These tasks require more work to be completed as time progresses, which necessitates multiple agents being allocated to them. Thus, simply going to the nearest task can cause overallocation and other tasks might grow out of control.

We denote the set of homogeneous agents by $A = \{a_1, \ldots, a_{|A|}\}$ and the set of tasks by $B = \{b_1, \ldots, b_{|B|}\}$. The set of assignments of agents to a task is denoted by $N(t) = \{n_1(t), \ldots, n_{|B|}(t)\}$, where $n_i(t)$ is the set of agents from A that are assigned to task b_i at time unit t. An agent $a_i \in A$ can only work on one task, b_j, at a time and at most one assignment $n_j(t) \in N$ can contain a_i. All agents and tasks

have a spatial location in the environment and the travel time between two locations is assumed to be computable.

Each agent provides the amount of work w per time unit towards the task it is assigned. Every task $b_i \in B$ has a cost function $f_i^t : (f_i^{t-1}, |n_i(t-1)|) \to \mathbb{R}$ with the relationship:

$$f_i^{t+1}(f_i^t, |n_i(t)|) = f_i^t(f_i^{t-1}, |n_i(t-1)|) + \Delta f_i^t \quad (1)$$

where Δf_i^t has the form:

$$\Delta f_i^t = g_i(f_i^t) - w \times |n_i(t)| \quad (2)$$

and $g_i : \mathbb{R}_{>0} \to \mathbb{R}_{>0}$ is a monotonically increasing function with initial value f_i^0. For simplicity, we will treat f_i^t as a value rather than a function if the values of f_i^{t-1} and $|n_i(t-1)|$ are clear in the context. This means f_i^t is strictly monotonically increasing when $g_i(f_i^t) > w \times |n_i(t)|$ and strictly monotonically decreasing when $g_i(f_i^t) < w \times |n_i(t)|$. When the cost function f_i^t reaches or passes zero at some time $t_i(0)$, the task is considered complete and is removed from the problem. In our problem the cost of completing a task increases over time, therefore the goal of task allocation is to minimize the time the last task is completed.

IV. Latest Finishing First

With perfect knowledge about how a task grows, we can predict what allocation of agents would be ideal. This ideal solution is not reachable due to travel time being required to reach tasks, but even in this ideal setting task allocation is not as simple as it seems. Even neglecting the spatial limitations, when tasks grow over time some allocations of agents are better than others. Taking into account the spatial limitations, we derive the Latest Finishing First (LFF) algorithm, which we present after first describing properties of the dynamic cost functions we will use. Throughout this section, we assume that time t is sufficiently small to enable us to treat t as continuous and we assume a solution exists. $TT(x, y)$ is defined to be the travel time between location x and location y.

A solution is found at some time t_s if and only if all $b_i \in B$ have $f_i^{t_s} = 0$. Using (2), we can write the previous equation as $f_i^0 + \sum_{t<t_s} \Delta f_i^t = 0$. Since this is true for every $b_i \in B$ a solution is reached if and only if:

$$\sum_{b_i \in B} \left(f_i^0 + \sum_{t<t_s} \left(g_i(f_i^t) - w \times |n_i(t)| \right) \right) = 0 \quad (3)$$

Note that $\sum_{b_i \in B} f_i^0$ is a constant and $\sum_{b_i \in B} \sum_{t<t_s} w \times |n_i(t)| = \bar{p} \times t_s \times w \times |A|$, where $\bar{p}$ is the overall percent of time the agents are working. We can then rewrite (3) as:

$$\sum_{b_i \in B} f_i^0 + \sum_{b_i \in B} \sum_{t<t_s} g_i(f_i^t) - \bar{p} \times t_s \times w \times |A| = 0 \quad (4)$$

This means that $\sum_{b_i \in B} \sum_{t<t_s} g_i(f_i^t)$ is the amount of work added to the system from time $t = 0$, which we call R_{t_s} or the global regret. We define $\Delta R_t = R_t - R_{t-1} = \sum_{b_i \in B} g_i(f_i^t)$.

Theorem 1: Minimizing R_{t_s} is an optimal solution when $TT(x, y) = 0 \ \forall x, y$.

Proof: When $TT(x, y) = 0 \ \forall x, y$ we can assume $\bar{p} = 1$, since agents can instantly move between tasks. Suppose a better solution $\hat{f}_i^t$ exists, then $\hat{R}_{t_{\hat{s}}} = \sum_{b_i \in B} \sum_{t<t_{\hat{s}}} g_i(\hat{f}_i^t)$ where $\hat{S} < S$ with $\hat{R}_{t_{\hat{s}}} > R_{t_s}$. Rewriting (4) for R_{t_s} yields:

$$R_{t_s} = w \times |A| \times t_s - \sum_{b_i \in B} f_i^0$$

Solving (4) for $\hat{R}_{t_{\hat{s}}}$ in terms of t_s:

$$\hat{R}_{t_{\hat{s}}} + w \times |A| \times (t_s - t_{\hat{s}}) = w \times |A| \times t_s - \sum_{b_i \in B} f_i^0$$

Using our two assumptions, we can then write:

$$R_{t_s} + w \times |A| \times (t_s - t_{\hat{s}}) < \hat{R}_{t_{\hat{s}}} + w \times |A| \times (t_s - t_{\hat{s}})$$

which implies that R_{t_s} satisfied (3) at time $t_{\hat{s}}$. This is a contradiction. ∎

Theorem 2: Minimizing ΔR_{t_s} for every time unit t also minimizes the global regret, R_{t_s}, when 1. $TT(x, y) = 0 \ \forall x, y$, 2. $g_i = g_j \ \forall i, j$, 3. $\lim_{x \to 0^+} g(x) = 0$, and 4. $\frac{\partial^2}{\partial t^2} g_i(f_i^t) \geq 0$.

Proof: Since $g_i = g_j \ \forall i, j$, we will denote g_i with g to simplify notation. Assume there exists a better solution $\hat{F}_i = \{\hat{f}_i^1, \hat{f}_i^2, \ldots\}$ which differs from the greedy minimization $F_i = \{f_i^1, f_i^2, \ldots\}$. This means that at some time t_d the better solution must assign agents differently than the greedy solution. By definition the greedy minimization is a minimum ΔR_{t_d} at time t_d, thus $\Delta R_{t_d} \leq \Delta \hat{R}_{t_d}$, where $\Delta \hat{R}_t = \sum_{b_i \in B} g(\hat{f}_i^t)$. After time t_d, the greedy minimization will allocate agents exactly in the same way as the better solution and because $TT(x, y) = 0 \ \forall x, y$ this is possible from any configuration. Next we prove by induction that $\sum_{b_i \in B} f_i^t \leq \sum_{b_i \in B} \hat{f}_i^t$. At time t_d, $\sum_{b_i \in B} f_i^{t_d} = \sum_{b_i \in B} \hat{f}_i^{t_d}$ combined with the fact that $f_i^x = f_i^0 + \sum_{t<x} \Delta f_i^x$ along with F_i is a greedy choice implies $\sum_{b_i \in B} f_i^{t_d+1} \leq \sum_{b_i \in B} \hat{f}_i^{t_d+1}$, which is the base case in the induction. If we write $\hat{f}_i^t = f_i^t + c_i$, then we can conclude $\sum_{b_i \in B} c_i \geq 0$. We can then compute for f_i^{t+1} as:

$$f_i^{t+1} = f_i^t + g(f_i) - w \times |n_i(t)|$$

and $\hat{f}_i^{t+1}$ as ($n_i(t)$ is the same since assignments are copied):

$$\hat{f}_i^{t+1} = (f_i^t + c_i) + g(f_i + c_i) - w \times |n_i(t)|$$

If we use the monotonicity of g, then we can see $g(f_i + c_i) = g(f_i) + \delta_i \times c_i$ for some $\delta_i > 0$. This means $\hat{f}_i^{t+1} - f_i^{t+1} = c_i + \delta_i \times c_i$ or rearranging: $f_i^{t+1} + c_i + \delta_i \times c_i = \hat{f}_i^{t+1}$. Since $\frac{\partial^2}{\partial t^2} g_i(f_i^t) \geq 0$ we know $\delta_i \geq \delta_j$ when $c_i > c_j \ \forall i, j$. Thus, $\sum_{b_i \in B} \delta_i \times c_i \geq 0$ since $\sum_{b_i \in B} c_i \geq 0$. We can conclude that $\sum_{b_i \in B} f_i^{t+1} \leq \sum_{b_i \in B} \hat{f}_i^{t+1}$, the inductive step. Using a similar logic to extracting the definition of ΔR_t from (3), we drop $\sum_{b_i \in B} f_i^0$ and $\sum_{t<t_s} \sum_{b_i \in B} w \times |n_i(t)|$ from both sides and get:

$$R_{t_s} \leq \hat{R}_{t_s}$$

where $\hat{R}_{t_s} = \sum_{b_i \in B} \sum_{t<t_s} g(\hat{f}_i^t)$. This is a contradiction to the fact that $\hat{F}$ is a better solution, therefore minimizing ΔR_t at every time t also minimizes R_{t_s}.

Before concluding the proof, we must consider the cases when F completes a task that $\hat{F}$ did not and vice versa. If F completes a task that $\hat{F}$ did not, then the greedy minimization

will assign agents from an already completed task to a random unfinished tasks. This does not invalidate any of the inequalities above. When $\hat{F}$ completes a task that F has not, this task will never be completed by direct mimicry from the greedy minimization solution, instead this will be finished by the random assignment described above. There is no discontinuity in the sums since we require $\lim_{x\to 0^+} g(x) = 0$, so when a task is completed it simply disappears from the equations. ■

Unfortunately, the assumption that $TT(x, y) = 0 \forall x, y$ is rather strong. When agents require time to move between different task clusters, every time an agent is allocated to a different task, $\bar{p}$ decreases in (4). To address this issue, we introduce Latest Finishing First (LFF) in Algorithm 1. The inspiration behind LFF is to create a stable assignment that will try to maintain $\bar{p}$ close to 1 to maximize the overall output of agents in the system. To do this, each task is prioritized and the closest agent to the highest priority task is assigned to that task. After an agent has been assigned, the priority of the highest priority task is recomputed and this process is repeated.

The priority of a task is the expected time to complete that task, so tasks that take longer have a higher priority. Since g_i increases work needed to complete a task, all tasks initially start out never completing (or $t_i(0) = \infty,\ \forall i$). In case of ties, the task which has the largest current cost is allocated the closest agent. This causes the completion times of all the tasks to be as balanced as possible, which in turn reduces the amount of travel needed for agents.

Algorithm 1 Latest Finishing First

while $\exists a_i \in A$ and $\nexists j$ such that $a_i \in n_j(0)$ **do**
  Find b_i for $\underset{b_i \in B}{\operatorname{argmax}}\ t_i(0)$ {Task b_i ends last}
  if $\exists b_i, b_j$ where $t_i(0) = \infty$ and $t_j(0) = \infty$ **then**
    Find b_k for $\underset{b_k \in B}{\operatorname{argmax}}\ f_k^0$ with $t_k(0) = \infty$
    Assign $\underset{a_j \in A \text{ and } a_j \text{ unassigned}}{\operatorname{argmin}}\ TT(a_j, b_k)$
  else
    Assign $\underset{a_j \in A \text{ and } a_j \text{ unassigned}}{\operatorname{argmin}}\ TT(a_j, b_i)$
  end if
end while

V. Real-Time Latest Finishing First

This section extends LFF outlined in Algorithm 1 to a decentralized real-time solution for dynamic partially observable environments in Algorithm 2. The LFF algorithm assumed all the tasks were known initially, but without this assumption it becomes impractical to try and minimize agent movement. When new tasks are discovered, it is crucial that agents are reallocated to the new tasks. The Real-Time Latest Finishing First (RT-LFF) algorithm assumes that new tasks are identified from an oracle, and when a new task is identified agents are reallocated to try finish all tasks at the same time.

When the task allocation first starts, the normal LFF is used to compute the assignments of agents. While time is progressing, if a new task, b_j, is identified then RT-LFF uses a greedy heuristic to reallocate agents to b_j. Tasks are ordered based on spatial proximity to b_j, and tasks that are closest to the new task attempt to reallocate agents first. Let $t_i^-(0)$ be the time for task b_i to complete with one fewer agent and $t_j^+(0)$ be the time for task b_j to complete with one additional agent including the delay from the agent traveling. An agent is only reallocated from b_i to b_j if $t_i^-(0) < t_j^+(0)$, namely a greedy heuristic which only transfers agents if the original task will still finish first even after reallocating an agent.

If the oracle informs all agents of the new task, each agent can compute this algorithm and agree on which agents should transfer without any communication. In the case where agents have limited computational power, a single more powerful agent at or nearby each task could direct this algorithm. If the oracle only informs the agents closest to the newly identified task, these agents would run the RT-LFF algorithm, since they are the highest priority, and after any assignments pass on updated information to the next closest agents.

Theorem 3: RT-LFF has at most $|B| - 1$ misallocated agents from LFF at any time assuming zero travel time.

Proof: We show that using RT-LFF a task b_1 will never be assigned more than one agent when using LFF by breaking it down into two cases. First, assume that we have two tasks b_1 and b_2 and let $t_j^{++}(0)$ be the time task b_j is completed with two additional agents assigned to it and similarly $t_i^{--}(0)$ be the time task b_i is completed with two less agents assigned to it. Without loss of generality assume $t_2(0) < t_1(0)$. If RT-LFF did not reallocate an agent from b_2 to b_1, then $t_2^-(0) > t_1^+(0)$. We notice that reallocating two agents from b_2 to b_1 means $t_2^{--}(0) > t_2^-(0)$ and $t_1^{++}(0) < t_1^+(0)$, which is a contradiction to the RT-LFF algorithm since b_1 would allocate an agent back to b_2 since $t_2^-(0) > t_1^+(0)$.

Next we show that a task b_1 would never receive a single agent from more than one cluster. The argument is similar to the first part, only it now involves task b_3. If b_2 and b_3 did not reallocate an agent to b_1 under RT-LFF, then $t_1^+(0) < t_2^-(0)$ and $t_1^+(0) < t_3^-(0)$. Suppose LFF did have both b_2 and b_3 reallocate an agent to b_1, then without a loss of generality assume $t_2^-(0) < t_3^-(0)$. Then $t_1^{++}(0) < t_1^+(0) < t_2^-(0) < t_3^-(0)$. This is a contradiction to how LFF works. Currently $t_3^-(0)$ is the last finishing and $t_1^{++}(0)$ is the fastest finishing, and from the equation above if b_1 gave back an agent to b_3 then $t_1^+(0) < t_2^-(0)$. This means the transfer back has reduced the time of latest finishing task, thus under RT-LFF b_1 will be not allocated an agent from more than one cluster compared to LFF. This implies a task in RT-LFF cannot have more than one agent under the allocation of LFF for each time step. The worst case is when $|B| - 1$ tasks have one agent fewer than LFF's allocation with the last task over allocated. ■

Algorithm 2 Real-Time Latest Finishing First

Use LFF for initial allocation
while $t < t_s$ **do**
  for $b_i \in B$ **do**
    if $\exists b_i \neq b_j$ where $t_i^-(0) < t_j^+(0)$ with $\underset{b_i}{\operatorname{argmin}}\ TT(b_i, b_j)$ **then**
      Reassign $a_k \in N_i(t)$ to b_j with $\underset{a_k}{\operatorname{argmin}}\ TT(a_k, b_j)$
    end if
  end for
end while

VI. Modeling Cost of Task Completion for Extinguishing Fires

In this section we focus only on the subproblem of dealing with fires in RoboCup Rescue and present a way of modeling how fire spreads and assigning agents to fires. The RoboCup Rescue simulator is designed for urban search & rescue after an earthquake, which fires to start in buildings. The environment is large (see Figure 1) with upward of 100 agents. The full simulator uses heterogeneous agents, but for this work we focus only on the agents that can extinguish fires, i.e. firetrucks. The other agents simulate the oracle and constantly search for new fires, and relay the location and estimated size when a new fire is found.

Fires are the most dangerous hazard in RoboCup Rescue. While a single building on fire can be dealt with quickly, if too many buildings ignite the fire becomes very difficult to tackle both due to its size and re-ignitions of buildings. We present two novel contributions. First, fire clusters are modeled as single tasks that have a cost which increases as time passes. Second, we present a method to estimate the number of buildings on fire in a cluster when only a few buildings from that cluster have been observed. The RT-LFF algorithm is then shown to out-perform more naïve heuristics.

A. Growth of Fire Clusters

Each building on fire individually has a chance to ignite nearby buildings based primarily on distance. This means the rate of growth, g, is proportional to the number of current buildings on fire, x:

$$\text{(a)} \quad \frac{\delta x}{\delta t} = g \times x \qquad \text{(b)} \quad x = C \times e^{g \times t} \tag{5}$$

Ten simulations with a single fire starting in various locations were used to empirically evaluate g to be roughly 0.0687. This is a first order linear differential equation that can be explicitly solved by separation of variables to yield the well known exponential growth function given by (5b). A constant C is introduced by integration to satisfy the initial conditions. Eq. (5a) can be modified to incorporate fire agents extinguishing effect on a fire. If there are n fire agents assigned to a fire cluster that each extinguish at a rate w, then the rate of growth of a fire of a fire cluster will be reduced by $n \times w$:

$$\frac{\delta x}{\delta t} = g \times x - n \times w \tag{6}$$

The constant w was also empirically calculated to be about 0.184 by running ten simulations with a single fire agent and tracking the total number of fires extinguished after 100 cycles. Matters are further complicated since the intensity of the fire has an effect on the amount of time it takes to extinguish, thus, w is biased higher than the real value. Agents can extinguish small fires much more quickly than larger fires, which often causes the agent to repeatedly put out small fires as they are reignited from nearby larger fires. Nevertheless, w gives a reasonable estimate for the agent's capabilities as shown in Section VII. Since both n and w are constants, this still is a linear differential equation which simplifies to a slightly modified exponential growth function shown in (7). This satisfies all the conditions in Theorem 2 except the zero travel-time assumption.

$$x = \frac{n \times w}{g} + C \times e^{g \times t} \tag{7}$$

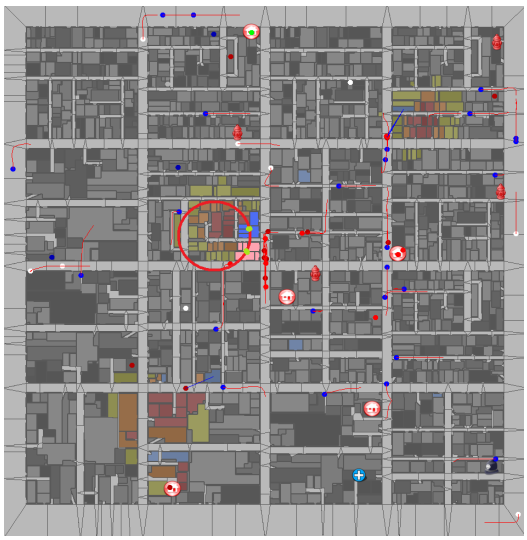


Fig. 1. Pink and blue buildings' average position determine the two points on the red fire cluster circle.

B. Estimating the Size of a Cluster

Clustering is commonly done to group similar objects into a single abstract object to reduce the complexity and to enable a macro-level analysis. Running a probabilistic model for every single building to predict the fire spread would require a large amount of computation and have a low probability for every possible state. Furthermore, the model will change based on which fires agents are assigned to extinguish, so ideally the model should be recomputed after every assignment. This method will not scale and is too complex for a scenario in which a large amount of information is already missing. For that reason fires are abstracted into clusters and the macro-level behaviors of these clusters are analyzed instead.

The lack of full information in RoboCup Rescue makes clustering difficult and requires some assumptions to be made. If a large number of agents were available to circle around the fire cluster and monitor its growth, then direct clustering using (6) with current known information would be a good estimate. However, normally agents are not able to dedicate this much time to information gathering, so it is necessary to come up with a way of estimating the size of a fire cluster while only being able to see a few buildings. To do this we need to make one assumption: on average fire propagates equally in all directions.

From this assumption, a circle is a good estimate of the area on fire. This circle is identified by two points on its edge in the following manner: First, the most recently seen building and the 9 closest burning buildings (or all known nearby burning buildings if fewer than 9 are on fire) are included in a set. Then k-means clustering with $k = 2$ is run to find two subsets. The average position of each subset is then used as the two points to fit the circle upon, shown in Figure 1.

The method for finding the radius can be extended from the exponential model given in Section VI-A. If x is the number of buildings on fire, then we can estimate $\pi \times r^2 = A \times x$, where r is the circle radius and A is the average building area (estimated over all buildings on the map). This can be rewritten

as: $x = \frac{\pi \times r^2}{A}$ which can then be substituted into (5a) yielding:

$$\frac{\delta r}{\delta t} = \frac{g}{2} \times r, \text{ and } r(t) = D \times e^{\frac{g}{2} \times t} \quad (8)$$

To overestimate the radius of the circle, we initially assume the fire started at the beginning of the simulation. For example if a fire is found at time $t = 50$, we would assume this fire started as a single burning building, $D = 1$, at time $t = 0$. Thus the radius would simply be $r(50)$. With a radius and two points on a circle, there are two possible circles to choose from. The circle with the highest ratio of burning buildings to total buildings is the one chosen as the fire cluster.

Since this radius is the worst case estimate, we should incorporate current information to refine the estimate. For all buildings in the circle, we check their status at the last time viewed. If a building has never been seen, we neglect it. If there is a conflict between past information and the assumption that the fire started at time $t = 0$, we assume the center of the circle is still the same and recompute D for a radius that satisfies the information. This new initial condition gives a smaller radius to be fit on the two circle points and this time we choose the circle whose center was inside the old overestimated circle. This process is repeated until there is no conflicting information.

C. Assignment to Fire Clusters

The goal of assigning agents to clusters is to extinguish fires as quickly as possible. Eq. (7) can be solved for t when $x = 0$ as shown in (9). If the constant C is nonnegative, then the fire is growing faster than the n agents can extinguish it. Thus the fire will grow indefinitely and $t(0)$ is set to ∞. When C is negative, (9) will be computable and will give the time when the fire will be fully extinguished.

$$t(0) = (\ln \frac{n \times w}{g \times -C})/g \quad (9)$$

Algorithm 3 shows the implementation of the RT-LFF algorithm for the case when $g(x) = x$ in RoboCup Rescue. Normally in RoboCup Rescue no fire clusters are known initially, so when the first fire cluster is found all agents are assigned to that cluster. When a new fire cluster is found, multiple agents may need to be transfered from the old clusters. This is why any time an agent was transfered, we should run the algorithm again until no more useful transfers exist.

The order in which the fire clusters are labeled is important, since every cluster first attempts to transfer an agent to b_1. When a new fire is discovered, it will initially have no agents assigned and will need to receive agents from other clusters. Since this fire cluster is in the greatest need of agents, it is assigned b_1. All other clusters are then reordered based on their distance from cluster b_1, the newly discovered fire. Thus b_2 will be the closest fire cluster to b_1, b_3 the second closest and so forth. This helps reduce the travel time $TT(i, j)$ in order to more efficiently use fire trucks.

VII. Results

In this section, we show the validity of our exponential model and RT-LFF by empirical evaluation in RoboCup Rescue. First, we show how the exponential model accurately fits the real fire growth data. Then the model is used to estimate the time a fire is extinguished and compared against the real time it took for agents to extinguish the fire. Finally RT-LFF is compared against a random assignment method and a closest distance greedy heuristic.

Algorithm 3 RT-LFF for g(x) = x

Require: $C_i, C_j, n_i, n_j, w, g, t$ {C_i is the integration constant and $n_i(t)$ is the number of agents on task b_i, C_j and n_j correspond to similar things on task b_j, w is the task completion rate of agents, g is the growth rate of fires and t is the current time.}

1: $change \leftarrow$ **true**
2: **while** $change$ **do**
3: $change \leftarrow$ **false**
4: **for** $i = 1$ to k **do**
5: $t_i^-(0) \leftarrow (\ln \frac{(n_i(t)-1)\times w}{g \times -C_i})/g$ {Compute the time to extinguish fire i with one less agent than it currently is assigned.}
6: **for** $j = 1$ to k **do**
7: $\hat{x_j} \leftarrow \frac{n_j(t)\times w}{g} + C_j \times e^{g\times(t+TT(i,j))}$ {Before the agent from fire cluster i arrives to cluster j, compute the effect of the agents.}
8: $\hat{C_j} \leftarrow (\hat{x_j} - \frac{(n_j(t)+1)\times w}{g})/(e^{g\times(t+TT(i,j))})$ {Once the agent from cluster i arrives, we need to recompute C_j.}
9: $t_j^+(0) \leftarrow (\ln \frac{(n_j(t)+1)\times w}{g \times -\hat{C_j}})/g$ {Finally compute when fire j is fully extinguished.}
10: **if** $t_i^-(0) < t_j^+(0)$ **then**
11: Transfer an agent from i to j
12: $n_i \leftarrow n_i - 1$
13: $n_j \leftarrow n_j + 1$
14: $change \leftarrow$ **true**
15: **end if**
16: **end for**
17: **end for**
18: **end while**

A. Model Fitting

A single fire was tracked over 5 simulations on two maps (Virtual City and Berlin) and the actual number of fires is compared against the exponential function best fit in Figure 2. The exponential function slightly under estimates the fire cluster between $t = 40$ and $t = 60$ but is otherwise fairly accurate.

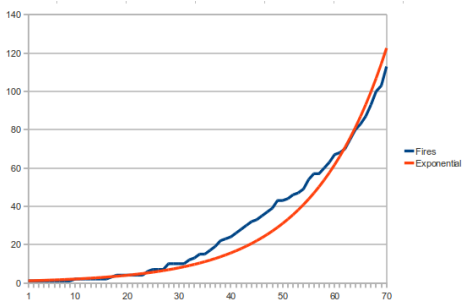


Fig. 2. Average fire spread estimated by best exponential fit to data.

The estimate of fire cluster size described in Section VI-B

was tested by assigning a static amount of fire trucks to extinguish the fire cluster and comparing the estimated and real extinguish time. The estimated size of the cluster is recomputed every time step and increases in accuracy as a larger number of buildings in the cluster is observed. For worst case analysis the real extinguish time is compared against the estimated extinguish time when the fire cluster is first discovered.

A histogram of 54 fires extinguished with the normalized error is shown in Figure 3. The error in estimation had a mean of 0.0268 and standard deviation of 0.2526. The distribution is close to Gaussian and fairly unbiased at overestimating or underestimating. Some error may be due to imprecise empirically derived constants in the modeling equation or a lack of incorporating the intensity in the model.

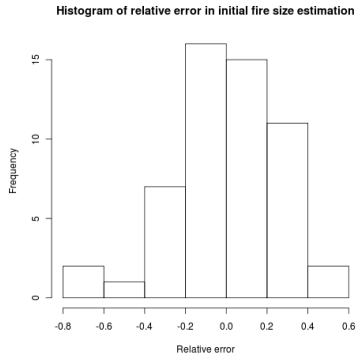


Fig. 3. Relative error of extinguish time when a cluster is first discovered.

B. Performance Comparison of Different Allocation Strategies

Each map was run with 5 simulations and the time the last fire was extinguished is reported in Figure 4. In each case, the oracle is simulated by a fixed number of agents randomly searching the environment for new tasks and broadcasting the location of the tasks when they are discovered. The fixed number of agents searching the environment do not interact with any of the tasks directly. RT-LFF method does much better in Virtual City (8.3% better than the closest heuristic approach) than in Berlin (4.0% better). This is probably due to Virtual City being an artificial map with similar building sizes and a compact building arrangement. Berlin has open spaces where there is a river or park that throws off the accuracy of the circle estimating method. There is also a larger discrepancy in building size in Berlin which can again effect the circle estimate. If a large building is chosen for one of the two clusters to estimate the two points on the circle, the buildings in this cluster will be more spread out and would give an inaccurate point estimate since the center of the large building is far away from the others. The random method performs poorly on both maps, due to lack of cooperation.

VIII. Conclusions and Future Work

This paper addresses task allocation with multiple agents, where each task has a cost that changes over time. This adds a substantial amount of complexity and requires more coordination between agents. We limited the investigation of dynamic cost tasks to a general family of functions in order to reduce the complexity. We presented the Latest Finishing

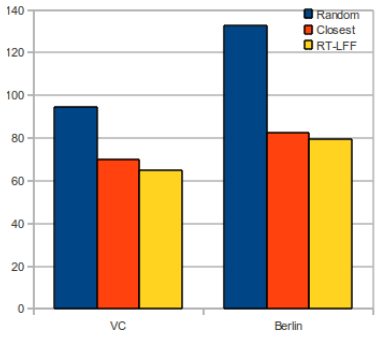


Fig. 4. Average finish time (no fires left) per map for each configuration.

First algorithm, which attempts to minimize the time when the last task finishes. We documented properties exhibited by dynamic tasks and went on to present a real-time solution, which approximates LFF for partially observable spaces. The algorithms currently work only for homogeneous agents with identical abilities. We hope to expand the current model to include other agent types. The choice of metric for evaluating an algorithm for dynamic cost functions is also an open question, especially when no solution exists. Completing fewer tasks at the expense of ignoring others might be beneficial more than keeping the current total cost low.

References

[1] S. D. Ramchurn, M. Polukarov, A. Farinelli, C. Truong, and N. R. Jennings, "Coalition formation with spatial and temporal constraints," in *Proc. Int'l Conf. on Autonomous Agents and Multi-Agent Systems*, 2010, pp. 1181–1188.

[2] P. Scerri, A. Farinelli, S. Okamoto, and M. Tambe, "Allocating tasks in extreme teams," in *Proc. Int'l Conf. on Autonomous Agents and Multi-Agent Systems*, 2005, pp. 727–734.

[3] P. R. Ferreira, Jr., F. dos Santos, A. L. C. Bazzan, D. Epstein, and S. J. Waskow, "RoboCup Rescue as multiagent task allocation among teams: experiments with task interdependencies," *Journal of Autonomous Agents and Multi-Agent Systems*, vol. 20, no. 3, pp. 421–443, 2010.

[4] Y. Zhang and L. E. Parker, "Task allocation with executable coalitions in multirobot tasks," in *Proc. IEEE Int'l Conf. on Robotics and Automation*, 2012.

[5] M. Nanjanath, A. Erlandson, S. Andrist, A. Ragipindi, A. Mohammed, A. Sharma, and M. Gini, "Decision and coordination strategies for robocup rescue agents," in *Proc. SIMPAR*, 2010, pp. 473–484.

[6] A. Chapman, R. A. Micillo, R. Kota, and N. Jennings, "Decentralised dynamic task allocation using overlapping potential games," *The Computer Journal*, 2010.

[7] T. Sandholm, K. Larson, M. Andersson, O. Shehory, and F. Tohmé, "Generation with worst case guarantees," *Artificial Intelligence*, vol. 111, no. 1–2, pp. 209–238, 1999.

[8] V. D. Dang and N. R. Jennings, "Generating coalition structures with finite bound from the optimal guarantees," in *Proc. Int'l Conf. on Autonomous Agents and Multi-Agent Systems*, 2004, pp. 564–571.

[9] X. Zheng and S. Koenig, "Reaction functions for task allocation to cooperative agents," in *Proc. Int'l Conf. on Autonomous Agents and Multi-Agent Systems*, 2008, pp. 559–566.

[10] H. Kitano and S. Tadokoro, "RoboCup rescue: A grand challenge for multiagent and intelligent systems," *AI Magazine*, vol. 22, no. 1, pp. 39–52, 2001.

[11] Álvaro Monares, S. F. Ochoa, J. A. Pino, V. Herskovic, J. Rodriguez-Covili, and A. Neyem, "Mobile computing in urban emergency situations: Improving the support to firefighters in the field," *Expert Systems with Applications*, vol. 38, no. 2, pp. 1255 – 1267, 2011.

Spatial Computing in an Orbital Environment: An Extrapolation of the Unique Constraints of this Special Case to other Spatial Computing Environments

Jeremy Straub

***Abstract*—The creation of an orbital services model (where spacecraft expose their capabilities for use by other spacecraft as part of a service-for-hire, barter system or single-owner network) requires effective determination of how to best transmit information between the two collaborating spacecraft. Existing approaches developed for ad hoc networking (e.g., wireless networks with users entering and departing in a pseudo-random fashion) exist; however, these fail to generate optimal solutions, as they ignore a critical piece of available information. This additional piece of information is the orbital characteristics of the spacecraft. A spacecraft's orbit is nearly deterministic if the magnitude and direction of its velocity vector is known (irregular drag and other factors can introduce error, particularly over an extended period of time, which can be corrected for). Because of this, it is possible for the communications routing algorithm to predict (with a high level of accuracy) future positions of origin, destination and prospective intermediate nodes and determine a communications path optimized based on this knowledge. Many other applications of spatial computing have similar constraints on the problem space. This paper considers the special case of spatially-aware communications traffic routing in an orbital environment and extrapolates from this to other spatial computing problems with problem space constraints.**

***Index Terms*—Spacecraft networking, orbital computing, cloud computing, spatial computing, orbital services model, service delivery**

I. Introduction

ANY spacecraft in orbit of a dominating mass moves in a deterministic pattern around that mass. This allows the determination of a future position of the spacecraft based on knowledge of the current magnitude and direction of the spacecraft's velocity relative to the dominating mass. While subtle orbital variations (caused by irregular atmospheric drag, the irregularities of the central body's mass distribution, solar pressure and other factors) will perturb this orbit over time, these can be effectively corrected for. This allows two spacecraft to determine when they will be in close enough proximity to communicate in the future.

A key challenge to facilitating the miniaturization of spacecraft and survivability of missions is capability sharing between multiple spacecraft. This allows spacecraft to become smaller, resulting in lower mass and cost (both in terms of construction and launch costs) for a given capability level or it allows increased capabilities at a given price point.

Large 'pristine-class' missions (e.g., most current large-scale Earth sensing and interplanetary satellites) generally are equipped with the capabilities required during the period of greatest need. This is dictated by the fact that they must operate largely on a stand-alone basis. Thus, they may have significant capacity most of the time, when these additional capabilities are not required for their primary mission.

Two models, thus, emerge for reducing space mission costs (analogous to models utilized for 'big data' processing on Earth): central service providers that make capabilities available to customer spacecraft or a peer-to-peer-based model where spacecraft make their unneeded resources available to others on a barter or fee-for-service basis.

Both versions of the orbital service model [1] require effective communications and transmission cost prediction. This is, of course, required when actually transmitting data. Projections of transmission windows can also be utilized when comparing multiple service provider bids for prospective jobs to determine if the collaboration will produce a suitably expedient result (based on processing time and the projected delay between the end of processing and result delivery). Projections may also be utilized to determine whether it is cost-effective to procure a transmission forwarding service to allow provider-to-client (and vice-versa) communications via an intermediary, when the two cannot communicate directly (due to orbital position).

Several models have been considered based on work related to various types of networks with entering, moving and leaving nodes. The value of considering the deterministic movement is presented and a limited extrapolated is made to other (more general case) applications where the problem space of a spatial computing problem can be constrained by environmental traits or other factors.

II. Background

The work presented herein benefits from two existing areas of knowledge. These areas are the work related to modeling

J. Straub is with the Department of Computer Science, University of North Dakota, Grand Forks, ND 58202 USA (e-mail:Jeremy.straub@my.und.edu).

ad hoc networks and orbital mechanics. The prior work related to modeling ad hoc networks can be utilized to create an effective (but generally non-optimal) solution to the orbital communications problem. These algorithms, generally, presume an unreliable network with nodes entering and leaving in a pseudo-random fashion. Orbital mechanics further informs this with specific times and durations of communications that can be determined before the fact and used to create optimal or near-optimal communications routing solutions. Previous work related to these areas is now presented.

A. Review of Ad Hoc Networks

Abolhasan, Wysocki and Dutkiewicz [2] review the numerous routing protocols available for mobile ad hoc networks (MANETs). They identify 28 prospective routing protocols and divide them into three classes (proactive and reactive and hybrid). Proactive approaches maintain a cache of the complete routing map of the network (e.g., they can determine the best way to get to any other node). Reactive approaches, alternately, maintain path details only for the active path; they locate a new path if one is needed between the same two nodes again in the future. They conclude that the hybrid approach seems to work the best, benefiting from features from across the 23 non-hybrid approaches.

Karande and Kulkarni [3] present work on routing for vehicular ad hoc networks (VANETs). VANETs share several common characteristics with orbital ones: they incorporate node movement and in some areas (e.g., a highway) the nodes follow a common path. Unlike orbital networks, in some areas numerous movement scenarios are incorporated (e.g., a parking lot) and nodes are less reliable, potentially not trusted and more likely to take unpredicted actions. In a VANET traffic routing can flow from node-to-node or from infrastructure to node.

Karande and Kulkarni consider two common ad hoc routing protocols, AODV (ad-hoc on-demand distance vector) and DSR (dynamic source routing) and determine that while they can be utilized they are not optimal due to the node mobility and network dynamism. Instead, they propose several other solutions including position based routing (PBR), geographic source routing (GSR), cluster-based routing (CBR), broadcast routing (BR) and geocast routing (GR).

PBR was found to work well in uncongested areas with limited obstructions; however, obstructions or congestion resulted in undesirable performance including routing loops and not-even-close-to-optimal paths being used. GSR utilizes map data for the operating area and creates a routing plan based on the direction of travel from message origin to destination. CBR requires a virtual network be created via grouping nodes together. However, the node movement made this problematic, as the virtual cluster formation was not fast enough to serve user needs in the quickly changing VANET environment.

BR sends messages to all adjacent nodes which repeat it to all nodes they are connected to. This was shown to be suitable for very small networks; however, congestion becomes a problem for anything larger than a small network. GR is a form of localized multicast, where a message is delivered to all nodes in a designated area. This was described as being well suited for notification of hazards and vehicles in distress.

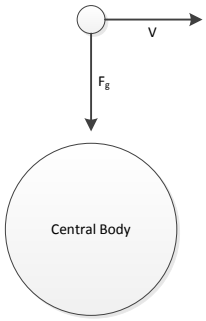
None of these approaches is suitable, on its own, for use in an orbital environment. However, the challenges presented suggest several possible considerations and inclusions to improve orbital network performance. Networks utilized for UAV-to-UAV communication and autonomous boat-to-boat communications can also be considered; however, many of these technologies are still designed around the concept of human-in-the-loop teleoperation. Thus, while the movement is more deterministic (e.g., controlled by a known routing algorithm), the communications approach is dissimilar.

Das, Barman and Sinha [4] present a clustering mechanism for sensornets which incorporates multi-hop ad hoc transmissions. The proposed approach creates a "virtual backbone" where, using an iterative convergence approach, a dominating set is discovered. Traffic flows through these key nodes, which have been identified. This was shown to work effectively and demonstrated to save power (however, there was shown to be a trade-off between the power utilized and system performance).

B. Introduction to Orbital Mechanics

Orbital mechanics is based on a single fundamental relationship between velocity and gravity [see 5]. An object remains in a stable orbit if and only if its velocity directly counter-balances the force of gravity at a particular distance from the central mass (i.e., the Earth is the central mass for an object in orbit around it; the Sun is the central mass for the orbits of the planets and other objects such as asteroids).

Figure 1. Counteraction of Velocity and Force of Gravity



While this is an over abstraction (as perturbing forces must be corrected for or counter balanced and may change over time), there is a single velocity suitable for each point in an orbit of a given altitude / semi-major axis. At this velocity (V), the object is moving fast enough tangent to the force of gravity (F_g) so as to be in constant free fall, but never approach the object (this, notably, is what creates the microgravity environment in orbit). This relationship is shown in Figure 1. At a slower speed, its orbit will decay resulting, eventually, in impact. At higher velocities, it will exit the orbit. Changes in velocity (ΔV) are thus used to change the altitude of an orbit.

Because of this relationship, three useful suppositions exist. These are:

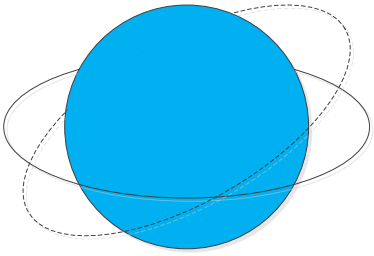
1) An object in a given orbit will maintain a constant path of movement. The velocity at a given point in an orbit will be the same from iteration-to-iteration. For circular orbits, the magnitude of this velocity is constant throughout the orbit.
2) A given orbit will take a fixed amount of time to complete, which is a function of the orbit's altitude / semi-major axis.
3) Two spacecraft in orbits of the same altitude will occupy the same points at the same times from iteration-to-iteration of orbits.

Note, however, that there is no guarantee with regards to objects in orbits of different altitudes having position consistency from orbit to orbit. These relative positions will change (unless the orbits are multiples of each other). Other mathematical relations between orbital periods will cause recurring patterns to occur. For simplicity of presentation, this paper deals with orbits that have the same altitude. However, actual implementations will generally require the determination of future communications opportunities that do not benefit from this simple relationship.

III. The Orbital Special Case

While being governed by the same fundamental physical laws, orbital movement is unlike movement on the surface of the Earth. Spacecraft are constantly moving at a relatively constant speed. The following two sections discuss how this impacts the problem space for routing communications traffic (section a) and how this impacts the determination of an optimal or near-optimal solution (section b).

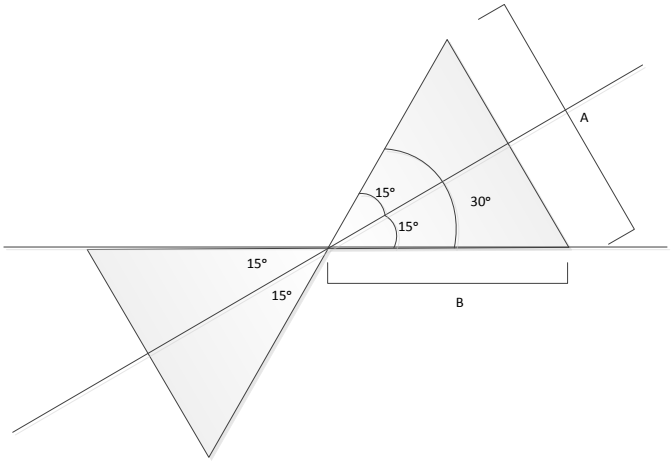
Figure 2. Equatorial and Inclined Orbit



A. *Constraints Imposed on the Problem Space*

The principles of orbital mechanics pose some severe constraints upon the problem space (making the problem easier to solve than a general for of the problem). Unlike ad hoc networks which must presume that a node could enter or leave (or for many applications, be shut off) at any time, orbital mechanics (and common spacecraft operation principles) makes the position of the assets known at all times (within a margin of error caused by orbital perturbation) and allows times and speeds of connectivity to be pre-determined. In addition, the general reliability of orbital assets means that there is little worry about an asset being shut off (though services being disabled for maintenance, etc. is certainly possible and must be handled). This, thus, becomes an exception to be handled rather than a common (ad hoc) occurrence that must be projected or predicted by a model.

Figure 3. Relative Angles of Orbits



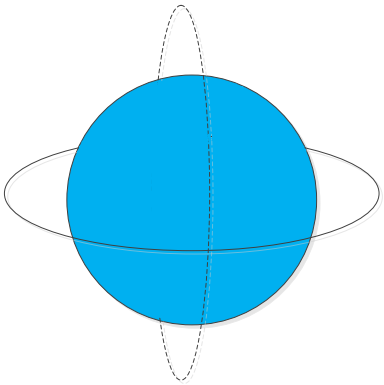
With a 30 degree inclination orbit and an elliptical orbit, the separation can be approximated with the equation:

$$\sin(15°) = \frac{\frac{1}{2}A}{B} \quad (1)$$

From this, the distance before and after the point where orbits cross where the spacecraft are in communication can be determined. Presuming (as an example) a 500 km maximum separation for effective communications, the two spacecraft can communicate from approximately 965.62 km before orbit crossing to 965.62 km after orbit crossing. With the 20,979 km circumference orbit requiring 90.52 minutes to traverse (and being traversed at a constant speed, as it is a circular orbit – the speed of traversal of ecliptic orbits varies), a communications period of approximately 8.33 minutes is available twice an orbit, for an average daily communications time of 265.03 minutes.

For the orbits shown in Figure 3 (continuing to presume a maximum communications distance of 500 km) each spacecraft is able to communicate with the other from 353.5 km before the point where their orbits cross to 353.6 km after their orbits cross. This occurs twice per orbit (on opposite sides of the earth). With an orbital circumference of 20,979 km and traversal of this taking 90.52 minutes, this 707.2 km traversal during which the spacecraft are able to communicate lasts 3.05 minutes, for a total of 6.1 minutes of communication per orbit (or 90.04 minutes of communication per 24-hour period). The amount of time that the spacecraft can communicate, for this example, scales up linearly with the maximum allowed range. For example 12.2 minutes of communication per orbit are available if the spacecraft is able to communicate at distances of up to 1000 km. Note that in each case, the spacecraft cannot cross the shared orbital crossing point at the same time at exactly the same altitude (As a collision would occur); however, the small impact of a slight spacing has been ignored to conceptually simplify the presentation.

Figure 4. Equatorial and Polar Orbit



Additional examples are beyond the scope of what can be presented herein (and do not add significantly to the analysis of this topic, even though they add significant computational complexity). Prospective scenarios that bear consideration include: orbits with different altitudes (and thus different periods where the relative position of the two craft differs on each successive orbit), orbits with different eccentricities (the eccentricity defines how elongated an orbit is) and the combination of eccentricity, inclination and altitude. It is important to note, however, that irrespective of the complexity of determining relative position, it can be determined for any time in the future (ignoring orbital perturbations).

In all cases, this can be reduced to a set of values related to the next times the various spacecraft will be within communications range of each other and the duration of that communications window. Tables 1 and 2 show an example of this for the three spacecraft considered (the spacecraft in equatorial orbit, spacecraft 1, is presumed to be the same across the two previously discussed scenarios).

Table 1. Example Future Communications Opportunities Between Spacecraft 1 and 2

Time	Duration
14	8.33
59.26	8.33
104.52	8.33
149.78	8.33
195.04	8.33
240.3	8.33
285.56	8.33

Table 2. Example Future Communications Opportunities Between Spacecraft 1 and 3

Time	Duration
19	6.1
64.26	6.1
109.52	6.1
154.78	6.1
200.04	6.1
245.3	6.1
290.56	6.1

From this, the amount of time before a message of a given duration can be completely transmitted can easily be determined. For example a 1 minute message could be transmitted within 20 minutes. An 8 minute message (presuming that spacecraft 1 can communicate with spacecraft 2 and 3 concurrently during the overlapping period) would take 66.36 minutes before transmission was complete.

This scenario is quite simple, as there is only one path for the transmission to take. If an additional node (spacecraft 4) was added that could also communicate with both spacecraft 2 and 3, then a routing determination could be made. For the purposes of this example, it will be presumed that spacecraft 4 is also in an equatorial orbit (at a different position). Tables 3 and 4 present the next times the various spacecraft will be within communications range of each other and the duration of that communications window.

Table 3. Example Future Communications Opportunities Between Spacecraft 2 and 4

Time	Duration
22	8.33
67.26	8.33
112.52	8.33
157.78	8.33
203.04	8.33
248.3	8.33
293.56	8.33

Table 4. Example Future Communications Opportunities Between Spacecraft 3 and 4

Time	Duration
26	6.1
71.26	6.1
116.52	6.1
161.78	6.1
207.04	6.1
252.3	6.1
297.56	6.1

With these added communications channels, the 8 minute message could be completed in 27.9 minutes, with the first 6.1 minutes being relayed by spacecraft 1 and the last 1.9 minutes being relayed by spacecraft 4.

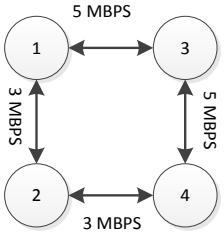
This can, obviously, get significantly more complex as additional nodes are added with each one creating (potentially) n-1 (where the node being added is node number n) additional connections across the network. The basic principles, however, remain the same. Numerous routing methodologies exist for best path determination; many can be augmented with the additional decision making parameter of time and duration of connection (in addition to other relevant parameters such as the link speed, which could conceivably vary from link to link).

B. Simplification of Solution Determination

The additional information related to when and for how long nodes will be in contact with each other complexifies the routing calculation somewhat. However, it can result in

significantly more optimal routing choices. This information also removes the need to add a random/pseudo-random distribution to attempt to model node entry to / exit from the network. Figure 6 presents an algorithm that utilizes this additional information effectively for route determination. This can be contrasted with Figure 7 which presents a routing algorithm that is not spatially aware. Figure 5 presents a diagram of the small network considered. Tables 5 through 8 contrast the performance of these two algorithms across three transfer scenarios, based on the four spacecraft for which the connection availability data is presented in Tables 1 through 4.

Figure 5. Connectivity Diagram



For simplicity, the network (shown in figure 5) groups the lower speed and higher speed connections into low and high speed paths (as combining them removes the ability to easily demonstrate the differences between the routing algorithms).

Table 5. Performance Using Algorithm 1

To Transfer	Routing	Speed	Units Time	Competed
25 MB	1-3-4	5/5 mbps	40	296.96
15 MB	1-3-4	5/5 mbps	24	167.48
5 MB	1-3-4	5/5 mbps	8	73.16

Table 6. Performance Using Algorithm 2, Single Path

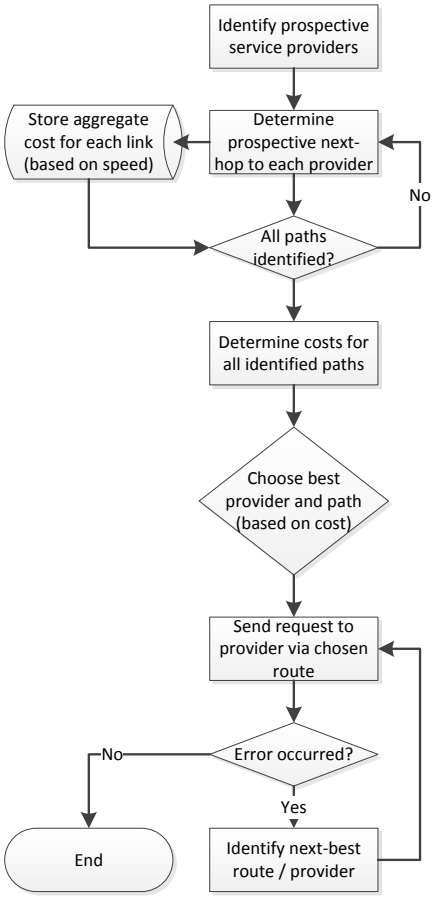
To Transfer	Routing	Speed	Units Time	Competed
25 MB	1-3-4	5/5 mbps	40	296.96
15 MB	1-3-4	5/5 mbps	24	167.48
5 MB	1-2-4	3/3 mbps	13.33	72.26

From tables 5 through 8, it is clear that the location aware algorithm outperforms the non-location-aware one in at least some circumstances (and never under-performs it). Table 7 demonstrates an approach where traffic is split across the two available paths, reducing the time significantly. Table 8 contains completion time specifics for this.

IV. Extrapolation

The details provided in Section IIA demonstrate that the challenges related to orbital communications are also relevant to terrestrial ad hoc mobile networks. There are, of course, significant differences (as discussed in Section IIA). These differences, however, become significantly less pronounced as the autonomous operations of vehicles becomes more commonplace. An autonomously operated vehicle operating on a roadway shares even more characteristics with the orbital nodes: its movement is largely predictable (and when it is not, it's an explicitly triggered action which can provide notification for change propagation – just like if a spacecraft conducts an orbit changing maneuver) and its reasonably reliable. Of course, autonomously operated vehicles will likely have to make more maneuvers (that spacecraft) to avoid human pedestrians or human-operated vehicles.

Figure 6. Non-Spatially-Aware Communications Traffic Routing Algorithm



There are thus two aspects of generalization relevant for consideration: first, there is the generalization to all deterministic or nearly-deterministic mobile (no-infrastructure) networks. Second, there is the generalization to any spatial computing application where the problem (and consequently the solution) space is significantly limited by constraints on the behavior of the nodes/agents.

V. Other Prospective Areas of Application

Prospective areas of application of this principle include all space networks (e.g., orbital communications, deep space exploration, etc.), most manufacturing processes (e.g., ad hoc networked robot workers moving around a production floor in a regular pattern and communicating with each other to coordinate actions) and construction robots. Less direct application can be found to applications which cause humans to move in a controlled pattern (e.g., queuing in a line at an amusement park, etc.).

Figure 7. Spatially-Aware Communications Traffic Routing Algorithm

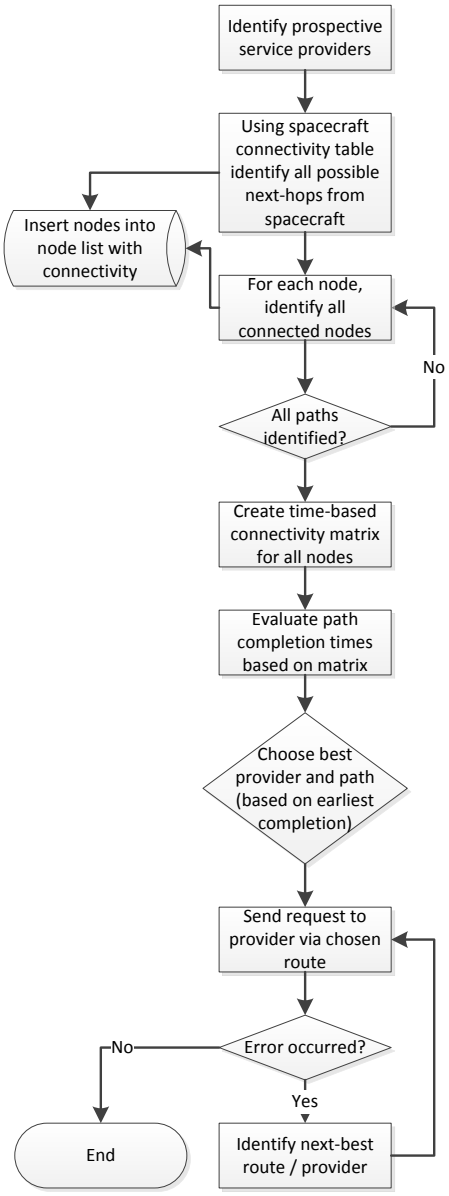


Table 7. Example of Using Algorithm 2, Multi-Path

Routing	Speed	Units Time	Session	TTL XFER
1-2-4	3/3 mbps	8.33	3.12375	3.12375
1-3-4	5/5 mbps	6.1	3.8125	6.93625
1-2-4	3/3 mbps	8.33	3.12375	10.06
1-3-4	5/5 mbps	6.1	3.8125	13.8725
1-2-4	3/3 mbps	8.33	3.12375	16.99625
1-3-4	5/5 mbps	6.1	3.8125	20.80875
1-2-4	3/3 mbps	6.69	2.50875	23.3175
1-3-4	5/5 mbps	2.69	1.68125	24.99875

The generalized principle of spatially constrained problems and solutions, based on robotic/agent behavior is even more widely applicable. Image mosaicking of remote sensing imagery with a loosely known relative camera position and orientation, for example, falls within this category. Unlike most mosaicking algorithms which presume that the images could begin in any relative orientation and skew, imagery with GPS and/or inertial measurement unit (IMU) difference values known can employ an expedited image feature matching process (that is less computationally intensive).

Table 8. Performance Using Algorithm 2, Multi-Path

Transferred	Completion
25 MB	164.47
15 MB	115.55
5 MB	29.5

VI. Conclusions

This paper has explored the value of incorporating orbital mechanics-derived communications window timing and duration information into an ad hoc spacecraft communications framework. It has shown that while many common routing algorithms will work in this environment, they do not necessarily generate optimal solutions. By incorporating this additional data, latency can be reduced (in some cases significantly) and system performance improved. Limited examples were presented to illustrate this for spacecraft in similar orbits. The effect becomes significantly more pronounced in orbits that are more divergant.

The general principle of acknowledging spatial constraints upon a problem and solution space is also demonstrated by this work. While this seems quite simple conceptually, many applications fail to acknowledge the additional data that they have at their disposal. Instead they utilize a general-purpose solution that, while functional, does not maximize resource utilization and other metrics. Application performance, particularly in time-sensitive implementations, can be improved in many cases via utilizing this (possibly) pre-existing data.

Acknowledgment

Small spacecraft work at the University of North Dakota is or has been supported by the North Dakota Space Grant Consortium, North Dakota NASA EPSCoR, the University of North Dakota Faculty Research Seed Money Committee, North Dakota EPSCoR, the Department of Computer Science, the John D. Odegard School of Aerospace Sciences and the National Aeronautics and Space Administration.

References

[1] J. Straub, A. F. Mohammad, “Above the cloud computing: creating an orbital service model using cloud computing techniques,” acc. Proceedings of the SPIE Defense Security + Sensing Conference, Baltimore, MD, 2013.

[2] M Abolhasan, “A review of routing protocols for mobile ad hoc networks,” Ad Hoc Networks, vol. 2, pp. 1-22, 2003.

[3] N. D. Karande, “Efficient routing protocols for vehicular adhoc network,” Int. J. Eng. Res & Tech., vol. 2, no. 1, Jan, 2013.

[4] S. Das, S. Barman, J. D. Sinha, “Energy efficient routing in wireless sensor network,” Procedia Technology, vol. 6, 2012.

[5] H. D. Curtis, *Orbital Mechanics for Engineering Students*. Burlington, MA: Elsevier, 2009.

Spatial Computation and Algorithmic Information content in Non-DNA based Molecular Self-Assembly

Germán Terrazas
ICOS Research Group
School of Computer Science
University of Nottingham, UK
gzt@cs.nott.ac.uk

Leong Ting Lui
ICOS Research Group
School of Computer Science
University of Nottingham, UK
ltl@cs.nott.ac.uk

Natalio Krasnogor
ICOS Research Group
School of Computer Science
University of Nottingham, UK
nxk@cs.nott.ac.uk

***Abstract*—Porphyrins are molecular units with fourfold symmetry and suitable for solid substrate deposition. The chemical structure of a porphyrin molecule reveals four structural units which can be synthesised with different substituent functional groups. The adequate selection of functional groups plays a central role in defining the correct intermolecular bindings that lead to a precisely tuned spatial self-assembled pattern. In this paper we explore the state-space of self-assembled programmable patterns. This is done by modelling the porphyrins molecular units using a kinetic Monte Carlo approach. Furthermore we analyse our simulations by both deriving discrete computational automata and in terms of algorithmic information content.**

I. Introduction

Self-assembly is a natural phenomenon observed at all scales where autonomous entities build complex structures without external influences or centralised control. Research and practise of self-assembly systems often encounters three key problems (a) the "prediction problem" that tries to anticipate the final product of a self-assembly process for a set of entities under given environmental conditions and the natural laws prevalent at the scale these operate, (b) the "programmability problem" which addresses how the entities (and their environment) should be designed in such a way that the final outcome of the self-assembly process is a specific pre-ordained one, and (c) the "yield problem" which relates to production and control of the intended self-assembled objects expected from a particular self-assembling system [1, 2].

Spatial computing metaphors based on multi-agents systems have already been exploited for the automatic development of problem-specific heuristics for solving combinatorial optimisation problems [3]. In fact, earlier ideas on automated program constructions are reported in [4] where the Automated Self-Assembly Programming Paradigm (ASAP2) is introduced as a self-assembly model for unconventional spatial computing. ASAP2 is defined in terms of gas molecules within a 3-dimensional volume embodying portions of software sampled from man-made programs that interact with one another subject to different values of temperature, pressure, "binding sites" compatibility, etc. giving rise to a variety of emergent programs architectures.

Closer to the molecular domains related to our investigations, we observe that linking self-assembly and computation provides a powerful new approach to addressing profound questions about the controllability of complex physico-chemical nano systems. We could ask, for example [5–8] *what are the least complex molecular tiling 'motifs' which may be exploited in the programming of self-assembly 2D lattices with specific geometries?* It is important to remark that the idea is to use computation in such a way as to program nano entities so they self-assemble, with exquisite detail, into specific patterns. That is, computation embedded designs allows for an enhanced control of the self-assembling entities. This approach has been successfully employed with DNA tiles [9, 10] and here we aim at studying non-DNA tiles since they are both smaller and work in different environment. The connection between self-assembly and computation is the subject of our work.

In the present work, we investigate the "prediction" and "programmability" problem for non-DNA molecular self-assembly. By a detailed kinetic Monte Carlo simulation we explore the range of spatial patterns that can, in principle, be obtained with porphyrin "tiles". We then investigate "programmability" by analysing the algorithm information content and discrete computational structures embedded within the observed spatial patterns.

II. Porphyrin-tiles Kinetic Monte Carlo System

This section presents a succinct description of the kinetic Monte Carlo model we have developed [11] for the simulation of porphyrin molecules spatial self-assembly.

A. Entities of the System

Porphyrins are molecules with a predominant bi-dimensional structure (the third structure is much smaller and hence can be ignored), fourfold symmetry and a chemical structure comprising four structural units which can be synthesised with substituent functional to obtain different intermolecular bindings such as hydrogen bonding and van der Waals forces. We have currently synthesise porphyrins with iodine, carboxylic acid, pyridine, bromine and nitro functional

groups; in particular, the dimension of a porphyrin is 2.89nm^2 whereas DNA-tiles are 12.6nm wide [9]. We choose Wang tiles [12] as abstract model of porphyrins since these are two-dimensional squares with labelled edges and undergo tile-to-tile interactions. Moreover, these have been used before to model DNA tiles in [9]. Thus, Wang tiles have a natural morphological correspondence to functionalised porphyrin molecules. We refer to such embodiment as *porphyrin-tile* which could be defined as either *iso-functionalised* when its four sides are set with the same functional group and as *hetero-functionalised* when its four sides are set with different functional groups. Fig. 1 depicts the model correspondence and structural parts of an hetero-functionalised porphyrin molecule.

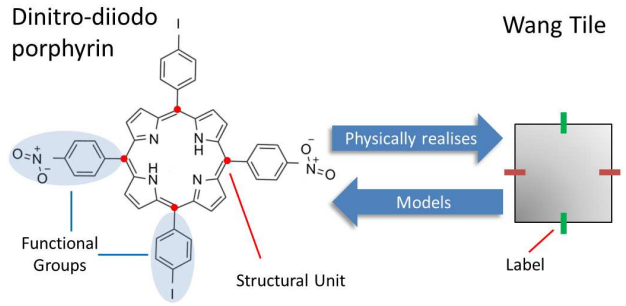


Fig. 1. A functionalised porphyrin molecule with structural units set with nitrogen and iodine embodies a Wang tile, the labels of which correspond to different functional groups. The Wang tile abstracts and models the dinitro-diiodo porphyrin, while the latter is a physical instantiation of the Wang tile.

B. Interaction Methods and Environment

The substrate where molecules are deposited and interact with one another forming supra molecular aggregates is modelled as a two-dimensional square site lattice. Such lattice is set with periodic boundary conditions where each position is occupied by only one porphyrin-tile at a time. The neighbourhood of a porphyrin-tile is von Neumann type and energy interactions among neighbouring molecules are at the core of the system dynamics for capturing *deposition*, *motion* and *rotation* of a molecule on substrate. In particular, deposition models the arrival of a molecule onto an empty position of the substrate. Motion models the translation of a molecule to one of its four neighbouring empty positions on the substrate by considering three cases: the diffusion of a molecule across the lattice without interacting with neighbouring molecules as shown in Fig. 2 (b), diffusion along an aggregate as depicted in Fig. 2 (c) or departure of a molecule from an aggregate as illustrated in Fig. 2 (d). Rotation models spinning of a molecule on its centre of mass, i.e. the ± 90 degrees gyration of a porphyrin-tile on its geometrical midpoint.

The Monte Carlo family of methods are generally good when a coarse grained modelling of molecular entities and a fast approximation of the overall behaviour of a physico-chemical nano system is needed [13–18]. We have then designed and developed a *porphyrin-tiles kinetic Monte Carlo*

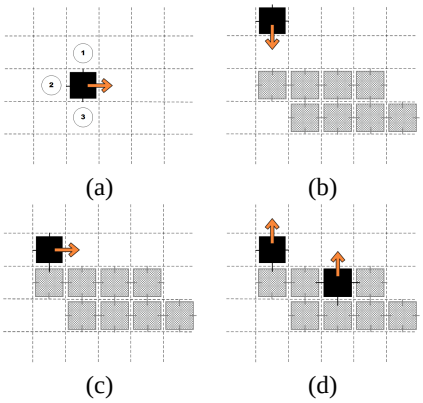


Fig. 2. Symbolic examples of a molecule hop from position (i, j) to position $(i, j+1)$ together with its neighbouring positions (a), diffusion of a molecule across the lattice without interacting with neighbouring molecules (b), diffusion of a molecule along an aggregate (c) and departure of a molecule from an aggregate (d).

(kMC) [11] in which a list with the possible transitions of the system, i.e. deposition, motion and rotation of a porphyrin molecule together with their correspondent rates is compiled for each time step. Depositions take place at a constant deposition rate (R_{Dep}) whereas diffusions and rotations are performed according to a diffusion rate (r_{ijkl}) calculated as:

$$r_{ijkl} = \exp(\frac{-E_{ijkl}}{TT0}) \quad (1)$$

where E_{ijkl} is the activation energy a molecule needs to jump from position (i, j) to position (k, l) and $TT0$ is a fixed parameter capturing the temperature of the system and the Boltzmann constant. The E_{ijkl} is calculated in terms of the sum of intermolecular bindings of a molecule and its binding with the substrate. Once the list of all possible transitions of the system and their rates is compiled, a Monte Carlo selection process follows in which the transition with the best weighted choice is performed. The chance of a transition is given according to the value of its associated rate linked to the activation energy, i.e. the more neighbouring molecules, the smaller the chance for a porphyrin-tile to diffuse or rotate. After a transition is performed, the list is updated and the process repeats for a fixed number of time steps.

III. Experiments in-silico

In the experiments presented in this section we systematically program porphyrin-tiles with different functional groups in order to see how different configurations impact on the resulting self-assembled patterns. We consider two types of independent settings for the kMC multi-agent system described in Section II.

A. Two Porphyrin-tiles Species

Across this set of experiments, the porphyrin-tiles kMC was configured with a lattice of 256×256 positions, $E_r = 1.3\,eV$, $TT0 = 28 \times 10^{-3}$, $R_{Dep} = 5 \times 10^{-5}$, a maximum coverage of 25%, a given binding energy between molecule and substrate

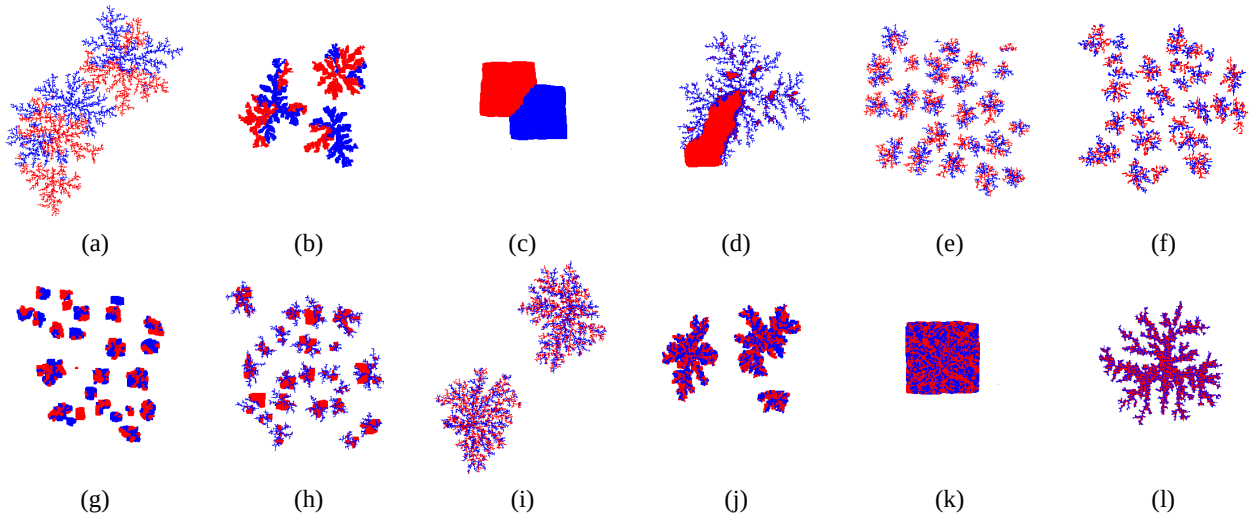


Fig. 3. Some of the many self-assembled complex spatial patterns achieved with two species of iso-functionalised porphyrin-tiles. A wide variety is observed in terms of shape, topological characteristics, dimension and internal structure.

(E_s), and two species of iso-functionalised porphyrin-tiles as shown in Fig. 4 (a). Since these species are systematically programmed with different functional groups, intermolecular binding among their instances takes place either between identical functional groups (e.g. E_{11} in Fig. 4 (b)) or between different functional groups (E_{12} in Fig. 4 (d)). Although porphyrin-tiles programmability is achieved in the lab by changing the associated functional groups, in the model this is done by setting different values to E_{11}, E_{22} and E_{12}.

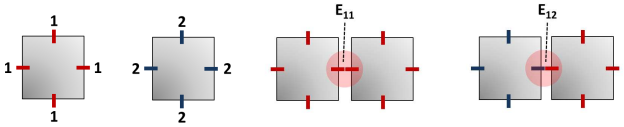


Fig. 4. The two iso-functionalised porphyrin-tile species with functional groups labelled as 1 and 2 (a). Intermolecular binding takes place throughout adjacent functional groups, in this case between identical functional groups (E_{11} or E_{22}) or between different functional groups (E_{12}) (b-c).

With these two porphyrin-tile species we then consider two sets of experiments. The first one involves functional groups for which the intermolecular bindings are always positive. Therefore, all the possible combinations among E_{11}, E_{22}, E_{12} and E_s were systematically tried: the first three energies taking values from $[0.1\,eV, 0.2\,eV, \ldots, 1.0\,eV]$ with the latter taking values from $[0.5\,eV, 0.6\,eV, \ldots, 1.0\,eV]$. The second set of experiments is designed to explore the impact of repulsive forces between different functional groups. Thus, all the possible combinations among E_{11}, E_{22}, E_{12} and E_s were systematically tried: the first two taking values from $[0.1\,eV, 0.2\,eV, \ldots, 1.0\,eV]$, the third from $[-0.1\,eV, -0.2\,eV, \ldots, -1.0\,eV]$ and the latter from $[0.5\,eV, 0.6\,eV, \ldots, 1.0\,eV]$.

B. Six Porphyrin-tiles Species

Across this set of experiments, the porphyrin-tiles kMC was configured with a lattice of 64×64 positions, $E_r = 1.3\,eV$, $TT0 = 28 \times 10^{-3}$, $R_{Dep} = 5 \times 10^{-5}$, a maximum coverage of 25%, a given binding energy between molecule and substrate (E_s), and six species comprising two iso-functionalised and four hetero-functionalised porphyrin-tiles as shown in Fig. 5.

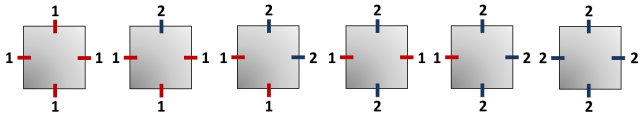


Fig. 5. The six porphyrin-tiles species comprising two iso-functionalised porphyrin-tiles and four hetero-functionalised porphyrin-tiles with functional groups labelled as 1 and 2.

The experiments involving these six porphyrin-tile species consider functional groups for which the intermolecular bindings are always positive. Therefore, all the possible combinations among E_{11}, E_{22}, E_{12} and E_s were systematically tried: the first two taking values from $[0.1\,eV, 0.2\,eV, \ldots, 0.5\,eV]$, E_{12} from $[0.1\,eV, 0.2\,eV, \ldots, 0.5\,eV]$, with the latter taking values from $[0.5\,eV, 0.6\,eV, 0.7\,eV]$.

IV. Results

For each experimental configuration, our kMC initially considered deposition, motion or rotation until maximum coverage is achieved. Then, only motion or rotation took place for $200 \times R_{Dep} \times$ lattice positions time steps. Once finished, the final arrangement of the porphyrin-tiles on the lattice was captured into an image. A summary of the input parameters and their range together with the resulting images of experiments conducted with the two porphyrin-tile species and the six porphyrin-tile species are respectively collected in Table 1 and Table 2 of http://www.cs.nott.ac.uk/~gzt/sipndbmsa.

A. Two Porphyrin-tiles Species

The symmetry of the underlying lattice and range of values explored by all possible combinations of E_s, E_{11}, E_{22}, E_{12} suffice to produce wide variety of self-assembly patterns in terms of shape, dimension and internal structure (see samples depicted in Fig. 3). This fact is related to the role played by each of the four variable parameters of the system. As reported in [11], the principal role of E_s is to determine the number of resulting aggregates across the lattice. E_{12} exerts influence on the segregation between species within aggregates and also on the geometrical diversity of the structures. Different combinations of E_{11} and E_{22} are linked to morphological aspects as well as to internal structural features. More importantly, we are interested in understanding the programmability of this system and thus explore next which discrete computational processing determine specific spatial self-assembled patterns.

We show next cases in which specifically programmed porphyrin-tile species gave rise to particular self-assembled structures via embedding discrete processes of computation order, some of which linked to simple finite automata theory.

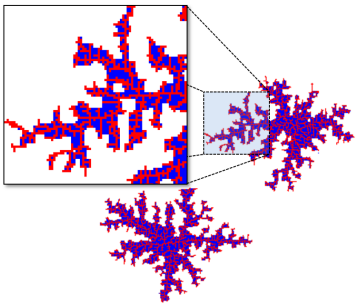


Fig. 6. Self-assembled aggregates formed with E_s=0.5, E_{11}=1.0, E_{22}=0.2 and E_{12}=0.2. Instances of one species form backbones (red structures) whilst instances of the other self-assemble within (blue patches) those backbones.

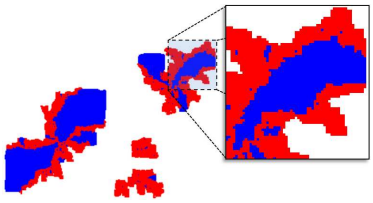


Fig. 7. Self-assembled aggregates formed with E_s=0.6, E_{11}=0.4, E_{22}=0.2 and E_{12}=0.1. The physical limits on blue porphyrin-tiles self-assembly are controlled by the edges of the red backbone. That is, blue porphyrin-tiles act as counters the positions of which are "seeded" via red porphyrin-tiles.

Instances of aggregates embedding discrete process of computation are shown in Figs. 6 and 7. The internal structures of these reveal that it is possible to program two porphyrin-tiles species in such a way that instances of one spontaneously form backbones (see red structures) whilst instances of the other self-assemble within (see blue patches) those backbones. In particular, the emerging backbone not only encloses very well defined areas but also directs up to which physical extent supplies of blue porphyrin-tiles will self-assemble. A similar result has been defined as *counter* in [19, 20] where a self-assembled two-dimensional structure of height N=2^K systematically emerge from a user defined structure (seed) of width K by means of repeated accretion of hand-tailored tiles. Our results indicate that while these DNA-tiles might be hard to hand craft [19, 20] they are readily designable in a simulated porphyrin based molecular setting. Remarkably, these porphyrin-tiles self-assemble with such a precision and control that the structure they form never goes beyond the edges of the longest of the surrounding backbones.

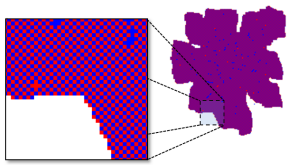


Fig. 8. Self-assembled aggregate formed with E_s=0.5, E_{11}=0.1, E_{22}=0.1 and E_{12}=0.4. A systematic order (checkers pattern) within an irregular shape springs out from specifically programmed red and blue porphyrin-tiles self-assembly.

An example of internally highly ordered self-assembled structures is shown in Fig. 8. In this case, there is a specific arrangement among instances of two porphyrin-tile species from where a checkers board pattern (visualised in Fig. 8 inset) springs out as a spontaneous result of self-assembly. This is an interesting example of a globally complex (i.e. dendritic) shape with a locally simple organisational pattern. Seen as a consecutive collection of diagonally-oriented red and blue stripes, leads to a finite state machine like the one illustrated in Fig. 9. This is the simplest probabilistic automaton defined in terms of two states each of which writing a diagonal of different symbols one after another, i.e. red symbols in state q_1 and blue symbols in state q_2, with an associated small error probability ε to write red instead of blue or vice versa, and with λ transitions to change between states when a new diagonal begins. One could define an automata by scanning column-wise or row-wise instead of diagonally. But that results in a more complex automata.

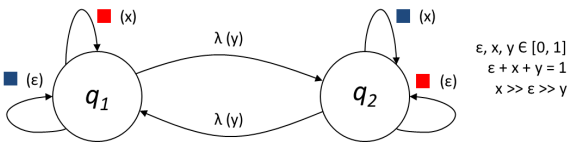


Fig. 9. The simplest two-states probabilistic automaton that generates the alternating diagonal stripes of blue and red symbols observed in Fig. 8 and Fig. 10 (b), error value ε is the probability of mistaking the symbol whereas λ transitions change between states when a new diagonal begins.

Considering ordered or disordered arrangement among porphyrin-tiles as a micro level feature and the shape of the associated aggregate as a macro level feature, it results that it is possible to program the spatial interaction among porphyrin-tiles in order to obtain different order/disorder combinations for the local/global scales. For instance, different porphyrin-tiles in Fig. 8 and Fig. 10 (a) achieved a structure with ordered

arrangements within a regular and irregular shape respectively. On the other hand, porphyrin-tiles in Fig. 10 (b - c) self-assembled into disordered arrangements within irregular and regular shapes.

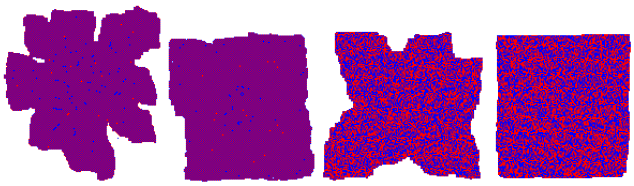


Fig. 10. Differently programmed porphyrin-tile lead to irregular assemblies with ordered internal structure (a), regular assemblies with ordered internal structure (b), irregular aggregates with disordered internal structure (c) or regular aggregates with disordered internal structure (d).

Similar analyses can be done to the simulation results ran with negative input parameter values for which other automata can be derived.

B. Six Porphyrin-tiles Species

In this section we explore results with a more complex set of programmable porphyrins. The self-assembled aggregates obtained with the six species of porphyrin-tiles reveal that their topological diversity is mostly ruled by E_s and E_{12}. In particular, it is observed a certain degree of structure complexity and global order (disorder) across simulation results associated to high (low) values of E_{12} for a given E_s. Thus, one could ask:

Is it possible to measure and characterise the algorithmic complexity observed across the final self-assembled complex patterns ?

In order to answer this question, we apply a Kolmogorov complexity-based technique to assess the simulation results in terms of algorithmic information content as this proved to be an effective approach for characterising and quantifying emergent behaviour in complex systems [11, 21, 22]. The Kolmogorov complexity of an object is defined as the size of the shortest computer program that could generate a complete description of the object [23]. Since the Kolmogorov complexity is an uncomputable function, existing approximations make use of lossless compression algorithms. Thus, if it is a highly compressed object, then it has high regularities and if it is a less compressed object, then it has low regularities.

Across all simulation results achieved with the six species of porphyrin-tiles, the most significant variety regarding topology of self-assembled aggregates is observed across experiments ran with combinations of E_s=0.5, E_{11},$E_{22}\in[0.1,\ldots,0.5]$, $E_{12}\in[0.1,\ldots,0.5]$. Since the most remarkable contrast is found between subsets obtained with E_{12}=0.1 and E_{12}=0.5, we decided to explore E_{11},E_{22} with higher resolution aiming at a more comprehensive and insightful complexity analysis. Therefore, extra simulations were ran with combinations of E_s=0.5, E_{11},$E_{22}\in[0.125, 0.150\ldots, 0.475]$, $E_{12}\in\{0.1, 0.5\}$. All results are collected in Table 3 of http://www.cs.nott.ac.uk/~gzt/sipndbmsa. We applied Portable Network Graphics (PNG) data compression algorithm to the images, the size of the compressed images (K) is an approximation in bits of their associated Kolmogorov complexity. Our findings reveal that low values of K link to simulation results containing assemblies like the one depicted in Fig. 11 (a), i.e. big aggregates, whereas high values of K relate to simulation results presenting small assemblies with porphyrin-tiles scattered across the lattice as shown in Fig. 11 (d).

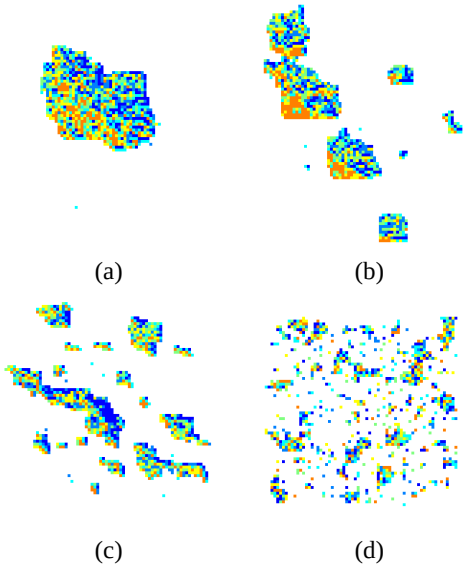


Fig. 11. Self-assembled complex patterns samples achieved with six species of porphyrin-tiles. K values (Kolmogorov complexity) increase from (a) to (d).

All calculated K values are collected in Fig. 12. For E_{12}=0.1 (Fig. 12 (top)), low intermolecular bindings between identical functional groups (E_{11} or E_{22}) give place to weak spatial interactions making self-assembly difficult and, consequently, a highly disordered final configuration (e.g. Fig. 11 (d)) is produced. Values of K quickly drop from 1665 bits to 966 bits as E_{11} or E_{22} increase and bigger self-assembled aggregates are formed with more ordered final configurations (e.g. Fig. 11 (a)).

On the other hand, simulations ran with E_{12}=0.5 generate a more homogeneous map of Kolmogorov complexity in Fig. 12 (bottom). The reason behind is that strong intermolecular bindings between different functional groups are better exploited in the presence of weak intermolecular bindings between identical functional groups, thus different species have a high chance to self-assemble (e.g. Fig. 11 (a)). Values of K slightly change from 1240 bits to 1089 bits as E_{11} or E_{22} increase revealing no significant contribution to the overall topology of the self-assembled aggregates and, moreover, a conservation of order across associated final configurations.

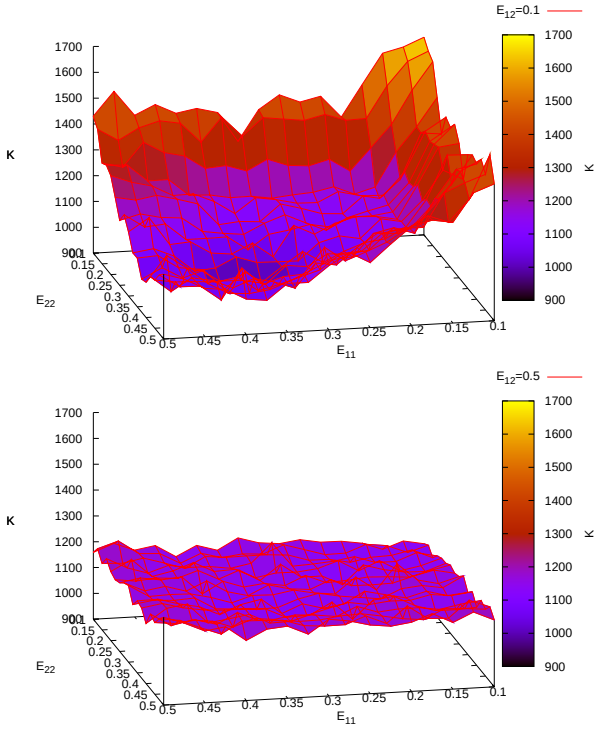


Fig. 12. K values for simulation results ran with E_s=0.5, E_{11},$E_{22} \in [0.1, 0.125, \ldots, 0.5]$ and E_{12}=0.1 (top) or E_{12}=0.5 (bottom). Plots are shown symmetric over E_{11} and E_{22} for illustration purposes.

V. Conclusion

In this work we have presented a spatial computing approach to modelling dynamics of a particular physico-chemical nano system based on porphyrin tiles which are smaller than DNA tiles and can operate in different environments. We have studied the resulting self-assembled complex patterns from two different perspectives. First, we focused on the internal structure of aggregates obtained with two species of iso-functionalised porphyrin-tiles. In this case, we have demonstrated that it is possible to observe discrete processes of computation driving the self-assembly among specifically programmed porphyrin-tiles. In addition, we have also shown that differently programmed porphyrin-tiles are able to arrange themselves into a highly ordered structure from disordered random initial conditions by linking self-assembly process with discrete computational steps performed by a probabilistic finite automata. Second, we measured the Kolgomorov complexity of simulation results obtained with six species of porphyrin-tiles. This has contributed not only with a high level characterisation of the aggregates associated to the operating intermolecular bindings, but also with a general description of the degree of order observed within the system.

Acknowledgment

This work is supported by EPSRC grant EP/H010432/1 Evolutionary Optimisation of Self-Assembly Nano-Design.

References

[1] J.A. Pelesko. *Self Assembly: The Science of Things that Put Themselves Together*. Chapman & Hall/CRC, 2007.

[2] N. Krasnogor, S. Gustafson, D.A. Pelta, and J.L. Verdegay. *Systems self-assembly: multidisciplinary snapshots*, volume 5. Elsevier Science, 2008.

[3] G. Terrazas, D. Landa-Silva, and N. Krasnogor. Towards the design of heuristics by means of self-assembly. In *Developments in Computational Models*, volume 26, pages 135–146. EPTCS, 2010.

[4] L. Li, J. Garibaldi, and N. Krasnogor. Automated self-assembly programming paradigm: initial investigation. In *IEEE International Workshop on Engineering of Autonomic and Autonomous Systems*, pages 25–26. IEEE, 2006.

[5] D. Soloveichik and E. Winfree. The computational power of benenson automata. *Theoretical Computer Science*, 344(2-3):279–297, 2005.

[6] L.M. Adleman, Q. Cheng, A. Goel, M.-D. A. Huang, D. Kempe, P.M. de Espanés, and P.W.K. Rothemund. Combinatorial optimization problems in self-assembly. In *Symposium on Theory of Computing*, pages 23–32. ACM, 2002.

[7] L.M. Adleman, Q. Cheng, A. Goel, and M.D. Huang. Running time and program size for self-assembled squares. In *Symposium on Theory of computing*, pages 740–748. ACM, 2001.

[8] P.W.K. Rothemund and E. Winfree. The program-size complexity of self-assembled squares. In *Symposium on Theory of computing*, pages 459–468. ACM, 2000.

[9] E. Winfree, F. Liu, L.A. Wenzler, and N.C. Seeman. Design and self-assembly of two-dimensional dna crystals. *Nature*, 394(6693):539–544, 1998.

[10] R. Schulman and E. Winfree. Programmable control of nucleation for algorithmic self-assembly. *SIAM Journal on Computing*, 39(4):1581–1616, 2009.

[11] G. Terrazas, H. Zenil, and Krasnogor N. Exploring programmable self-assembly in non-dna based molecular computing. *Journal of Natural Computing*, 2013. (to appear).

[12] H. Wang. Proving theorems by pattern recognition. *Bell Systems Technical Journal*, 40:1–42, 1961.

[13] E. Flenner, L. Janosi, B. Barz, A. Neagu, G. Forgacs, and I. Kosztin. Kinetic monte carlo and cellular particle dynamics simulations of multicellular systems. *Phys. Rev. E*, 85(3):031907, 2012.

[14] P. Bruschi, P. Cagnoni, and A. Nannini. Temperature-dependent monte carlo simulations of thin metal film growth and percolation. *Phys. Rev. B*, 55(12):7955–7963, 1997.

[15] F. Silly, U.K. Weber, A.Q. Shaw, V.M. Burlakov, M.R. Castell, G. A. D. Briggs, and D.G. Pettifor. Deriving molecular bonding from a macro-molecular self-assembly using kinetic monte carlo simulations. *Phys. Rev. B*, 77(20):201408, 2008.

[16] B. A. Hermann, C. Rohr, M. Balbás Gambra, A. Malecki, M. S. Malarek, E. Frey, and T. Franosch. Molecular self-organization: Predicting the pattern diversity and lowest energy state of competing ordering motifs. *Phys. Rev. B*, 82(16):165451, October 2010.

[17] Y. Li and N. Lin. Combined scanning tunneling microscopy and kinetic monte carlo study on kinetics of cu-coordinated pyridyl-porphyrin supramolecular self-assembly on a au(111) surface. *Phys. Rev. B*, 84(12):125418, Sep 2011.

[18] U. K. Weber, V. M. Burlakov, L. M. A. Perdigão, R. H. J. Fawcett, P. H. Beton, N. R. Champness, J. H. Jefferson, G. A. D. Briggs, and D. G. Pettifor. Role of interaction anisotropy in the formation and stability of molecular templates. *Phys. Rev. Lett.*, 100(15):156101, April 2008.

[19] Q. Cheng, A. Goel, and P. Moisset. Optimal self-assembly of counters at temperature two. In *Foundations of nanoscience: self-assembled architectures and devices*, 2004.

[20] P. Moisset. Computer aided search for optimal self-assembly systems. In K. Natalio, S. Gustafson, D.A. Pelta, and J.L. Verdegay, editors, *Systems Self-Assembly Multidisciplinary Snapshots*, volume 5 of *Studies in Multidisciplinarity*, pages 225 – 243. 2008.

[21] P. Siepmann, G. Terrazas, and N. Krasnogor. Evolutionary design for the behaviour of cellular automaton-based complex systems. In *Adaptive Computing in Design and Manufacture*, pages 199–208, 2006.

[22] G. Terrazas, P. Siepmann, G. Kendall, and N. Krasnogor. An evolutionary methodology for the automated design of cellular automaton-based complex systems. *Journal of Cellular Automata*, 2(1):77–102, 2007.

[23] M. Li and P.M.B. Vitányi. *An Introduction to Kolmogorov Complexity and Its Applications*. Springer Verlag, 2008.