A Tuple Space for Data Sharing in Robot Swarms

Carlo Pinciroli
Polytechnique Montréal
PO Box 6079, Succ.
Centre-ville Montréal
Québec, Canada H3C 3A7
carlo.pinciroli@polymtl.ca

Adam Lee-Brown
Royal Melbourne Institute of Technology
124 La Trobe St, Melbourne
VIC 3000, Australia
s3434074@student.rmit.edu.au

Giovanni Beltrame
Polytechnique Montréal
PO Box 6079, Succ.
Centre-ville Montréal
Québec, Canada H3C 3A7
giovanni.beltrame@polymtl.ca

ABSTRACT

In this paper, we present a system to allow a swarm of robots to agree on a set of `(key,value)` pairs. This system enables a form of information sharing that has the potential to be an asset for coordination in complex environments, such as globally optimized task allocation. Taking inspiration from the environment-mediated communication of social insects, we call the system *virtual stigmergy*. Experimental evaluation indicates that virtual stigmergy can work in a wide variety of running conditions including heavy packet loss, and can cope with random motion trajectories.

Categories and Subject Descriptors

I.2.11 [**Distributed Artificial Intelligence**]: Coherence and coordination; I.2.11 [**Distributed Artificial Intelligence**]: Multiagent systems; C.2.1 [**Network Architecture and Design**]: Distributed networks; C.2.2 [**Network Protocols**]: Applications; C.2.4 [**Distributed Systems**]: Distributed applications; I.2.9 [**Robotics**]: Autonomous vehicles

Keywords

Swarm robotics, Collective robotics, Information sharing, Stigmergy

1. INTRODUCTION

In nature, animal swarms display remarkable levels of coordination. Activities such as nest building and foraging in insect colonies are well-known examples of globally coordinated, complex behaviors that arise from purely local interactions between individuals and with the environment [9]. In nest building, the environment is used as a common medium to share information (i.e., the current state and progress of the nest). Based on the configuration of a certain area, insects decide to drop or modify the placement of the building material. In foraging, ants construct pheromone trails that produce globally optimized routes between nest and source [7]. In these examples, the key to coordination is employing the environment as a common 'information storage unit', which can be 'read' and 'written' in parallel. This form of indirect, environment-mediated communication modality is commonly called *stigmergy* [17].

Stigmergy has been used often in swarm robotics [4, 13] to achieve coordination. Examples include 'physical' implementations of the concept, such as the projection of light on the ground [14], smart material [1], and inert material detected and placed by the robots [31, 34]; 'logical' implementations that mimic the behavior of natural pheromone [10, 26]; and 'embodied' implementations whereby robots act as environmental markers [25].

We propose a new use for this concept, which we call *virtual stigmergy*[1]. Virtual stigmergy, at its essence, is a distributed tuple space similar to Linda [15] that allows robots to share a collection of `(key,value)` pairs. Virtual stigmergy is part of the Buzz programming language [27]. In this paper, we substantiate three statements:

1. Virtual stigmergy is a valuable concept for swarm robotics, which enables the implementation of a wide class of swarm behaviors in a simple way;

2. The peculiar aspects that characterize robot swarms (large numbers of individuals, limited bandwidth, limited computational capabilities, and dynamic topology) render existing designs of distributed tuple space impractical for swarms; and

3. The algorithms that embody the mechanisms of virtual stigmergy are efficient and robust, and suitable for real-world swarm applications.

The rest of the paper is organized as follows. In Section 2 we review related work on distributed tuple spaces. In Section 3 we present the design of virtual stigmergy. In Section 4 we analyze the robustness and scalability of virtual stigmergy in several experimental conditions. In Section 5 we conclude the paper and outline future research directions.

[1]The term 'virtual stigmergy' was also used in [22] to refer to an algorithm for information spreading loosely inspired by [26] which, combined with a PSO-based algorithm, was demonstrated in a search-and-cleanup task with multiple robots. This mechanism is unrelated to the algorithm presented in this paper.

BICT 2015, December 03-05, New York City, United States
Copyright © 2016 ICST
DOI 10.4108/eai.3-12-2015.2262503

2. BACKGROUND AND RELATED WORK

In this section, we set the context in which our work is placed. We first discuss the challenges in the design of a distributed tuple space for swarm robotics that derive from the communication modality that is dominant in robot swarms. Subsequently, we review past works that proposed the idea of using tuple spaces as a shared medium for inter-robot coordination. We relate these works to the mentioned challenges. Finally, we present a number of existing designs of distributed tuple spaces that focus on networks with volatile topology, highlighting their applicability limitations in robot swarms.

2.1 Communication Challenges

One of the cornerstones of coordination in swarm robotics is *situated communication* [32]. This communication modality is based on wireless devices that are capable of exchanging data payloads as well as detecting the location of message originators with respect to the receiver's frame of reference. To make mutual localization possible, data exchange is completed exclusively when two robots are in direct line-of-sight.

Robots capable of situated communication in 2D include the Kilobot [30], the e-puck [23], and the marXbot [8]; recently, a low-cost device for 3D communication was also proposed [12].

Current devices for situated communication suffer from two main problems:

1. The useful payload that can be exchanged is very small, typically on the order of just a few bytes. For instance, the Kilobot and the marXbot exchange 12-byte messages, and the e-puck up to 4 bytes.
2. Message loss is a significant issue. We are not aware of thorough studies on this aspect; however, our personal experience is that more than 1 in 4 messages are typically lost at each control step, with bursts of lost messages occurring relatively infrequently.

These issues alone render implementing an effective distributed tuple space for a robot swarm a challenging task.

In addition, robots move in ways that are unpredictable in the general case, because they depend on the task being executed. The combination of small payloads, message loss and dynamic network topology poses serious challenges for the maintenance of data integrity across the swarm.

2.2 Tuple Spaces in Swarm Robotics

An interesting use of tuple spaces to achieve dynamic, space-aware computation is illustrated in [33]. This study is presented from a purely theoretical standpoint. The authors formalize a calculus for the possible operations that can be performed, but no real-world implementation of the concept is shown.

Karma [11] and Voltron [24] are programming languages whose run-time platform incorporates a tuple space to achieve efficient task allocation in single-robot single-task scenarios [16]. Both languages separate the environment into disjoint areas that must be visited by a robot. The states of the areas (visited/not visited/being visited) are stored in the tuple space. The tuple space is implemented as a hash map storing one entry for each area.

In Karma, the tuple space is stored in a centralized server. The robots are assumed unable to communicate between each other and interact solely by updating the tuple space, which effectively acts as a stigmergic communication medium. While this system shows that task allocation can be achieved without direct communication, a centralized implementation constitutes a single point of failure. In addition, the system requires the robots to physically return to the 'nest' where the server is located, which limits the update rate of the tuple space and the applicability of this approach to more complex scenarios.

In Voltron, the robots communicate through a traditional WiFi network. The authors propose two functionally equivalent versions of the tuple space: one centralized, analogous to Karma's, and one distributed, based on virtual synchrony [6], a tuple space widely used in cloud computing. Being based on a traditional WiFi network, the system is not scalable above about a dozen robots; also, the bandwith requirements of virtual synchrony exceed what situated communication can typically offer.

2.3 Tuple Spaces for Dynamic Topologies

The communication modalities of robot swarms push towards gossip-based, low-bandwidth, and drop-resistant designs of tuple spaces.

In [18], Heer *et al.* discuss the issues of realizing distributed hash tables (DHTs) in mobile ad hoc networks (MANETs). The main assumption in DHTs is that each node possesses a view of the complete system, and that the system is capable of maintaining a body of metadata such as lists of participating nodes and routing tables. While remarkable results can be obtained with clever management of this information [21], these assumptions are difficult to realize with robot swarms due to the low bandwidth and the high rate of mobility of the robots.

Cell Hash Routing [2] is a low-bandwith DHT based on the assumption that nodes know their position in the environment. Using this information, the nodes are organized in groups that aggregate and manage information collectively. Positional knowledge is a hard assumption to respect in robot swarms, making the applicability of this design problematic in real-world scenarios.

3. VIRTUAL STIGMERGY DESIGN

In the design of virtual stigmergy, we realized that bandwidth limitations make it impossible to share the entire tuple space at any time. Instead, in virtual stigmergy, data is exchanged only when it is accessed in reading or writing. This design is fundamentally different from existing implementations of distributed hash tables, as discussed in Section 2.3.

Operations. Virtual stigmergy offers six fundamental operations:

```
# Executed at init time
function init() {
  # Create a vstig
  VSKEY = 1
  vs = stigmergy.create(1)
  # Set onconflict manager
  vs.onconflict(function(k,l,r) {
    # Return local value if
    # - Remote value is smaller than local, OR
    # - Values are equal, robot of remote record is
    #   smaller than local one
    if(r.data < l.data or
      (r.data == l.data and
       r.robot < l.robot)) {
      return l
    }
    # Otherwise return remote value
    else return r
  })
  # Initialize vstig
  vs_value = id
  vs.put(VSKEY, vs_value)
}

# Executed at each time step
function step() {
  # Get current value
  vs_value = vs.get(VSKEY)
}
```

Figure 1: The Buzz code executed on the robots to test the performance of virtual stigmergy.

- `put(key,value)` writes a tuple `(key,value)`;
- `get(key)` returns the `value` corresponding to `key`, or `nil` if no matching tuple exists;
- `has(key)` returns `true` if a tuple indexed by `key` is present in the structure, and `false` otherwise;
- `size()` returns the number of `(key,value)` pairs in the structure;
- `onconflict()` is a user-defined function called when a write conflict occurs (more on this below);
- `conflictlost()` is a user-defined function called when a robot published an update that conflicted with another one, and the update was rejected (more on this below).

Local storage. Each robot maintains a local copy of the virtual stigmergy data. The data is stored in a hash map indexed by `key`. Each data entry consists of a record that contains the corresponding `value`, as well as additional metadata used to maintain integrity: `timestamp` and `robot_id`. The `timestamp` field is a Lamport clock [20] that induces a temporal ordering on the updates; the `robot_id` field stores the numerical identifier of the robot that originated the stored `value`, and is used to detect conflicts arising from simultaneous updates by different robots.

Writing. Whenever a robot needs to update an entry, the change is first made locally. If the structure does not already contain an element for `key`, a new tuple `(key,value, 1,id)` is created, in which `id` denotes the numerical identifier of the current robot. Conversely, if the structure already contains an entry `e`, the new entry is `(key,value, timestamp,id)` where `timestamp` is set to `e.timestamp+1`. Subsequently, the robot broadcasts a message `<PUT,key, value,timestamp,id>` in its neighborhood. Nearby robots who receive the message compare the locally known `timestamp` for the entry that matches `key`. If the local `timestamp` is lower, they update the entry and propagate the message; if the local `timestamp` is higher, they ignore the message to prevent an infinite flood of messages throughout the network.

Conflict management. It might happen that two robots publish conflicting data on the same `key` simultaneously, i.e., with the same `timestamp`. In this case, an *update conflict* occurs. More in general, a conflicting update on a `key` is characterized by two conditions: *(i)* The update is marked with a `timestamp` that is equal to the locally known one, and *(ii)* The `robot_id` of the originator is different from the locally known one. When a robot detects a conflicting update, it executes `onconflict()`. This user-defined function accepts as arguments the locally known entry and the conflicting update, returning the entry that must be kept; such entry is subsequently stored. If the `robot_id` of the discarded entry corresponds to the identifier of the current robot, then the `conflictlost()` function is called to allow the robot to react to a rejected update (e.g., retry an update with a new `timestamp`).

Reading. A robot *R1* that needs to read data from the virtual stigmergy accesses the locally known tuple. If the tuple is unknown, the return value of this command is `nil` (see [27]). After reading, the robot broadcasts a message `<GET,key,value1,timestamp1,robot1_id>` to ask neighbors whether the data is up-to-date. A nearby robot *R2* that receives the message compares `timestamp1` with the locally known `timestamp2`. If `timestamp1` and `timestamp2` coincide, *R2* compares `robot1_id` and the local `robot2_id`. If these match, *R2* does nothing; otherwise, it performs conflict management as explained above. If `timestamp1` is lower than `timestamp2`, then *R1* needs to be updated. Therefore, *R2* broadcasts `<PUT,key,value2,timestamp2,robot2_id>`. Conversely, if `timestamp2` is lower than `timestamp1`, then *R2* needs to be updated. In this case, *R2* stores the new entry, and then broadcasts `<PUT,key,value1,timestamp1, robot1_id>`. This broadcast mechanism allows robots to recover data integrity after temporary disconnections or random message drops occurred.

Bandwidth usage. The limited bandwidth available on the robots pushes towards minimal message exchange. In virtual stigmergy, the robots never perform a 'full' update of the structure—rather, messages are generated only when a specific tuple is written or read. This means that virtual stigmergy dedicates resources only on 'hot' (i.e., recently used) data. This is a reasonable choice in many scenarios, such as area-based task allocation in the style of Karma and Voltron, in which, once a tuple is marked as 'completed', there is not need to process it further. The fact that reading on virtual stigmergy triggers the generation of a message might produce long message queues on a robot. To alleviate this problem, the message queue retains only the most up-to-date message for a `key`, i.e., that with the highest `timestamp`.

4. EXPERIMENTAL EVALUATION

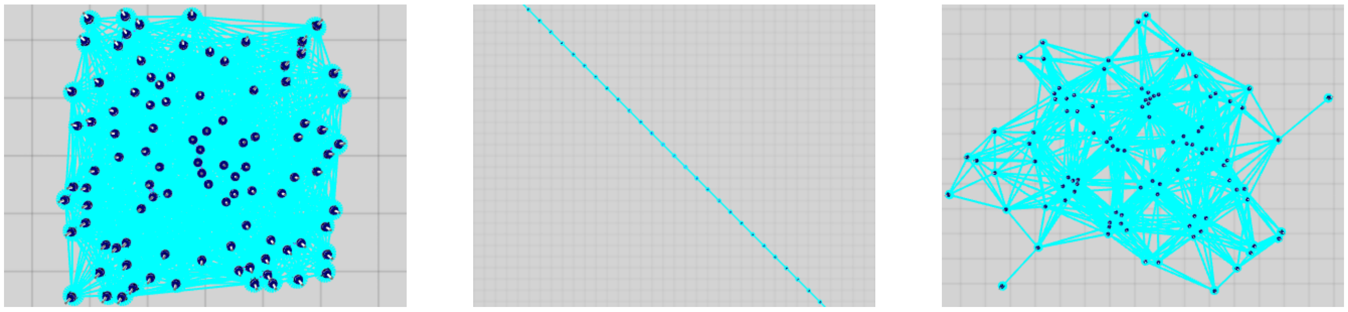

(a) Cluster (b) Line (c) Scale-free

Figure 2: Robot distribution for different topologies in the experiments discussed in Section 4.1. The cyan lines connect robots within each others' communication range and in direct line-of-sight. Screenshot taken with the ARGoS multi-robot simulator.

Experimental evaluation aims to assess the performance of virtual stigmergy in diverse conditions. We conducted our experiments in realistic physics-based simulations performed with the ARGoS multi-robot simulator.[2] The models present in ARGoS have been validated, and robot controllers developed with this simulator have been ported to real platforms in numerous works [28].

Performance measure. The most important performance measure of virtual stigmergy is the time an update takes to reach every robot in the swarm. We want convergence time to be as short as possible, to allow the robots to react swiftly to data updates. In the evaluation, we employ the number of control steps as a measure of time, and the total time for every robot to agree on a certain `(key,value)` pair as a performance measure. In our simulations, a control step lasts 0.1 s.

Communication model. We use the *range-and-bearing* sensor of the marXbot robot [29]. This device is modeled in ARGoS as an entity capable of broadcasting a message within a predefined range. A robot within range and in direct line-of-sight with the message originator has a probability P to ignore the message (also called *packet drop*). If, at a certain location, multiple messages can be received by a robot, each of them is tested for dropping individually. All robots perform independent tests on every message in range. In our experiments we consider values for P in $[0, 0.25, 0.5, 0.75, 0.95]$. Regarding bandwidth, each exchanged message is 8 bytes long and structured as follows:

- 1 bit to encode `PUT` or `GET`;
- 15 bits for the `key`;
- 2 bytes for the `value`;
- 2 bytes for the `timestamp`;
- 2 bytes for the `robot_id`.

The arena and the task. The experimental arena is a square of side L in which N robots are distributed. In all our experiments, the robots must agree on the highest numerical identifier in the swarm. This task is representative of a large class of scenarios in which the robots must agree on the highest (or lowest) value of a quantity (typically detected through the sensors) as a prerequisite to, e.g., construct a swarm-wide gradient or elect a leader. The Buzz [27] code executed by the robots is reported in Figure 1.

[2]`http://www.argos-sim.info/`

4.1 Scalability and Topology Dependence

We performed a set of experiments to assess the dependency of convergence time from different types of topologies. In these experiments, once placed, the robots are not allowed to move. To test for scalability, we chose N in the set $[10, 100, 1000]$. We identified the three interesting classes of topologies depicted in Figure 2.

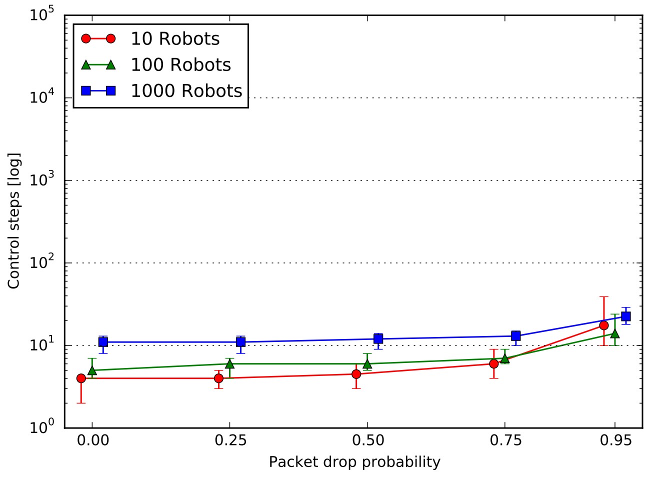
Cluster topology. The robots form a tight aggregate in which every individual has many nearby robots. Robots in the middle of the aggregate have up to 8 neighbors; robots on the borders typically have less. This topology is an extremely 'positive' case, in which connectivity problems are unlikely to increase convergence time. To generate clusters, the position (x, y) of a robot is chosen uniformly from $\mathcal{U}(-L/2, L/2)$.[3] To ensure comparable results across multiple experiments, we keep the density of the robots constant. We define the density D as the ratio between the 'communication area' occupied by all robots and the total area in the environment. The 'communication area' of a robot is a circle centered in the robot with a radius R corresponding to its communication range (for marXbots, $R = 3\,\text{m}$). To keep D constant across different values of N and ensure a tight cluster, we set $D = 5$ and calculate the arena side L as follows:

$$D = \frac{\text{tot comm area}}{\text{arena area}} = \frac{N\pi R^2}{L^2} \Rightarrow L = \sqrt{\frac{N\pi R^2}{D}}.$$

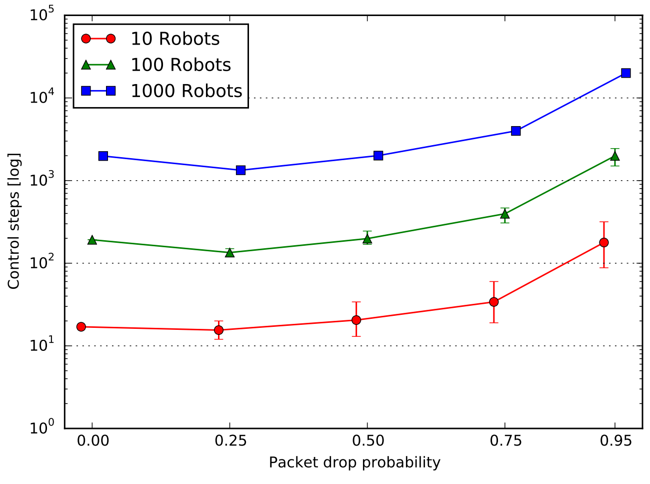
Line topology. The robots form a straight line. With the exception of the robots at the ends of the line, every robot has exactly two neighbors; the robots at the ends of the line only one. This topology is an extremely 'negative' case, in which every packet drop results in a convergence delay. To form the line, each robot $i \in [1, N]$ is placed at position (i, i) in the arena.

Scale-free topology. The robots are distributed so as to form multiple clusters connected by thin lines of robots.

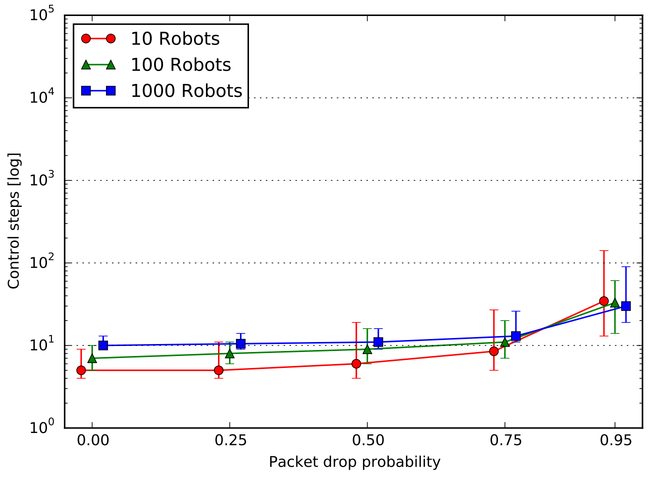
[3]When a coordinate choice causes physical overlap with already placed robots, a new coordinate is picked until no overlap occurs.



(a) Static cluster topology.



(b) Static line topology.



(c) Static scale-free topology.

Figure 3: Convergence time (in control steps) for static topologies in different experimental configurations. The plot reports the median, max, and min values of the distributions obtained for each experimental configuration $\langle N, P, \text{topology}\rangle$ over 50 runs. The markers are slightly offset to make them visible.

This case logically falls between the two extremes and it corresponds to the 'most likely' topology a swarm might form over time. To place the robots in a scale-free topology, we utilized the Barabási-Albert preferential attachment algorithm [3]. The first robot is placed in the origin of the simulated arena. Subsequent robots are placed by first choosing randomly an already placed robot to connect to (a.k.a. the 'pivot'), and then picking a random location within the communication range of the pivot.[2] Each robot can become a pivot with a probability proportional to the number of connected robots in the arena.

Results. The results are reported in Figure 3. The cluster topology (Figure 3a) displays a graceful degradation of performance with the increase of packet drop probability. Up to $P = 0.75$, convergence time does not increase; only with the extreme value $P = 0.95$ performance drops significantly. This result can be explained by noticing that, in the cluster topology, every robot has on average more than 4 neighbors. With $P = 0.75$, 3 out of 4 messages are lost, but the high number of neighbors allows messages to circulate anyway. Convergence time is also not significantly affected by the swarm size—for $\langle N = 10, P = 0.75\rangle$ the average is 6 control steps, while for $\langle N = 1000, P = 0.75\rangle$ it is 13 control steps. In contrast, the line topology (Figure 3b) displays severe dependence from both message drop probability and number of robots. This is intuitive, as each failure to deliver a message entails a delay in convergence time. The scale-free topology (Figure 3c) behaves almost analogously to the cluster case. The main difference between cluster and scale-free topology is in the max-min span of the data distribution. For instance, when $\langle N = 1000, P = 0.75\rangle$, the cluster topology convergence time is in the range $[10, 15]$ control steps; for the scale-free topology, it is in the range $[11, 26]$ control steps.

4.2 Motion

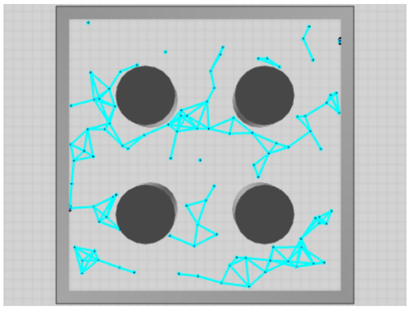
In this set of experiments we test the role of motion on convergence time. To highlight the role of motion, the experimental arena is also endowed with obstacles meant to divide the environment in several 'regions'. The robots diffuse throughout the arena following a simplified version of the algorithm presented in [19].

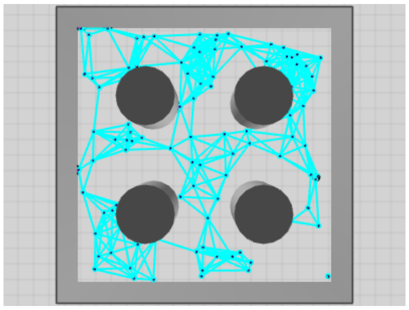
The arena is divided in 9 regions as shown in Figure 4 by placing 4 identical columns with radius $L/10$. The robots are distributed in the environment in a uniform way. To assess the performance of virtual stigmergy, we studied two parameters: the robot density and their maximum forward speed.

Robot density. In these experiments, the robot density is defined as the fraction between the 'communication area' occupied by the robots (see Section 4.1) and the 'walkable area' in the arena. The 'walkable area' is the difference between the total area of the arena (L^2) and the total area occupied by the four columns ($4\pi(L/10)^2$). The robot density D is thus calculated with:

$$D = \frac{\text{tot comm area}}{\text{walkable area}} = \frac{N\pi R^2}{L^2 - 4\pi(L/10)^2}.$$

Setting $N = 100$, we experimented with D to identify three values that would produce *(i)* A sparse robot distribution,


(a) Density = 1.


(b) Density = 3.

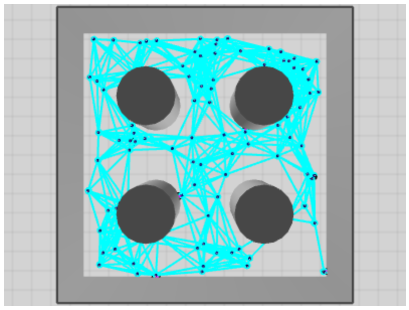

(c) Density = 5.

Figure 4: Robot distribution for different values of density in the experiments discussed in Section 4.2. The cyan lines connect robots within each others' communication range and in direct line-of-sight. Screenshot taken with the ARGoS multi-robot simulator.

in which small islands of unconnected robots exist and motion is necessary for information to spread ($D = 1$); *(ii)* A more compact robot distribution, in which most robots are connected, but a few must move to receive the data ($D = 3$); and *(iii)* A fully connected robot distribution, analogous to the cluster/scale-free case studied in Section 4.1 ($D = 5$). Similarly to the cluster topology in Section 4.1, for each configuration we calculated the value of L that would produce the desired D.

Maximum forward speed. To assess the role of motion, we identified three values for the maximum forward speed (S) of a robot. The marXbot is capable of moving with a speed of up to 20 cm/s; in our experiments, we considered $S \in [5, 10, 20]$ cm/s.

Results. The results are reported in Figure 5. The plots show that, when the robots are fully connected ($D = 5$) or almost ($D = 3$), the performance of virtual stigmergy degrades gracefully. Also, the faster the robots move, the more efficient information spreading is. In contrast, when the robots are too sparse ($D = 1$), information spreads slowly, with the forward speed being the limiting factor in convergence time. While the impact of S over convergence time is intuitive, given the simple diffusion logic employed in these experiments, it is interesting to notice that virtual stigmergy can ensure efficient information spreading even when a minority of the swarm is disconnected ($D = 3$).

5. CONCLUSIONS

We presented a distributed tuple space that enables efficient and robust information sharing in robot swarms with severe communication limitations. Results indicate that virtual stigmergy is efficient across several experimental conditions that vary in terms of swarm size, message loss, and network topology.

Future work will be devoted to augmenting virtual stigmergy with the possibility to affect the network topology to prevent catastrophic disconnection of large portions of the swarm. A promising approach in this direction is to employ spectral graph theory [5] to detect dangerous topologies and derive motion strategies to correct them [35].

Acknowledgments

This work was supported by an NSERC Engage Grant. Computations were made on the supercomputer Mammouth-Ms from Université de Sherbrooke, managed by Calcul Québec and Compute Canada. The operation of this supercomputer is funded by the Canada Foundation for Innovation (CFI), NanoQuébec, RMGA and the Fonds de recherche du Québec — Nature et technologies (FRQ-NT).

6. REFERENCES

[1] M. Allwright, N. Bhalla, H. El-faham, A. Antoun, C. Pinciroli, and M. Dorigo. SRoCS: Leveraging stigmergy on a multi-robot construction platform for unknown environments. In *Swarm Intelligence*, number 8667 in Lecture Notes in Computer Science, pages 158–169. Springer International Publishing, Berlin, Germany, September 2014.

[2] F. Araújo, L. Rodrigues, J. Kaiser, C. Liu, and C. Mitidieri. CHR: A Distributed Hash Table for Wireless Ad Hoc Networks. In *25th IEEE International Conference on Distributed Computing Systems Workshops*, pages 407–413. IEEE, 2005.

[3] A.-L. Barabási and R. Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, 1999.

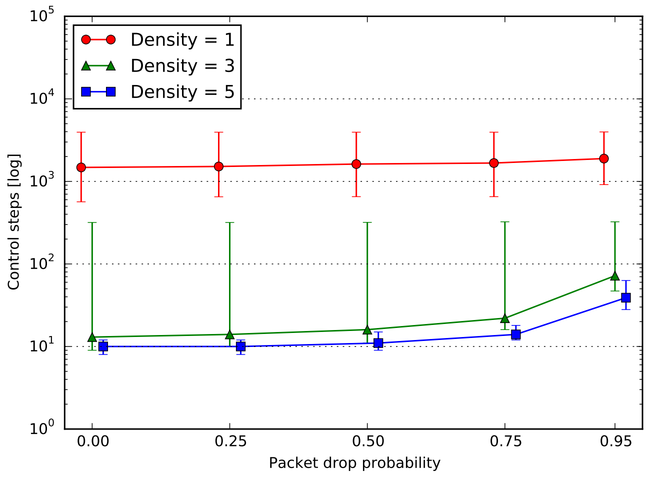
[4] G. Beni. From Swarm Intelligence to Swarm Robotics. *Swarm Robotics*, 3342:1–9, 2005.

[5] A. Bertrand and M. Moonen. Seeing the bigger picture: How nodes can learn their place within a complex ad hoc network topology. *IEEE Signal Processing Magazine*, 30(3):71–82, 2013.

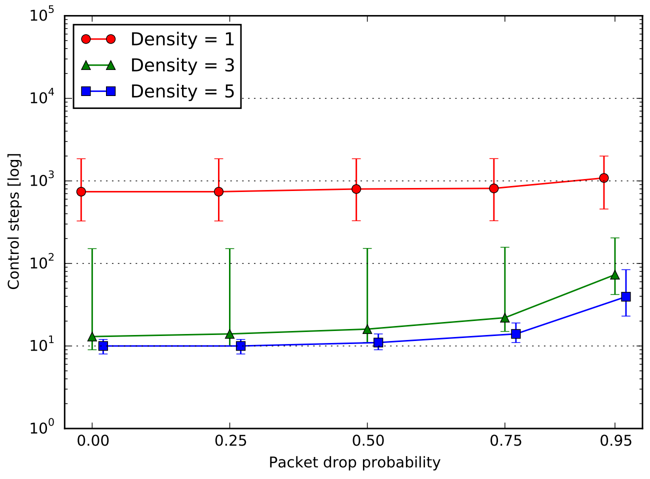
[6] K. Birman and T. Joseph. Exploiting virtual synchrony in distributed systems. In *Proceedings of the eleventh ACM Symposium on Operating systems principles (SOSP '87)*, volume 21, pages 123–138. ACM New York, 1987.

[7] E. Bonabeau, M. Dorigo, and G. Theraulaz. *Swarm Intelligence: From Natural to Artificial Systems.* Santa Fe Institute Studies in the Sciences of Complexity. Oxford University Press, New York, NY, 1999.

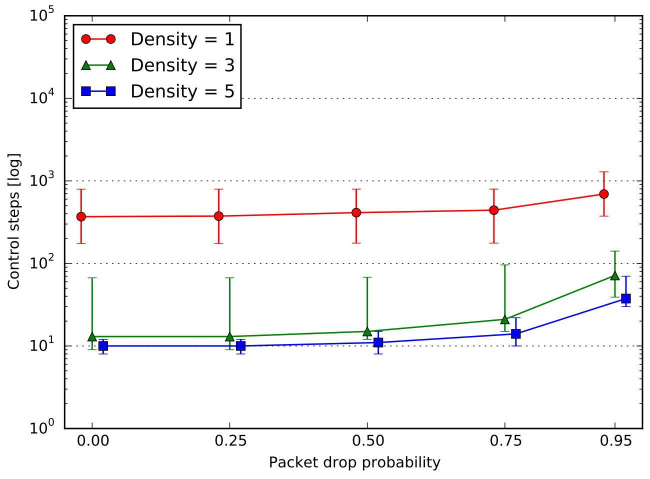
[8] M. Bonani, V. Longchamp, S. Magnenat, P. Rétornaz, D. Burnier, G. Roulet, F. Vaussard, H. Bleuler, and F. Mondada. The marXbot, a miniature mobile robot opening new perspectives for the collective-robotic research. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and*



(a) Max speed = 5 cm/s.



(b) Max speed = 10 cm/s.



(c) Max speed = 20 cm/s.

Figure 5: Convergence time (in control steps) for dynamic topologies in different experimental configurations. The plot reports the median, max, and min values of the distributions obtained for each experimental configuration $\langle N, P, S\rangle$ over 50 runs. The markers are slightly offset to make them visible.

Systems (IROS), pages 4187–4193. IEEE Press, Piscataway, NJ, 2010.

[9] S. Camazine, J.-L. Deneubourg, N. R. Franks, J. Sneyd, G. Theraulaz, and E. Bonabeau. *Self-Organization in Biological Systems*. Princeton University Press, 2003.

[10] A. Campo, A. Gutiérrez, S. Nouyan, C. Pinciroli, V. Longchamp, S. Garnier, and M. Dorigo. Artificial pheromone for path selection by a foraging swarm of robots. *Biological Cybernetics*, 103(5):339–352, 2010.

[11] K. Dantu, B. Kate, J. Waterman, P. Bailis, and M. Welsh. Programming micro-aerial vehicle swarms with Karma. In *Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems - SenSys '11*, pages 121–134, New York, New York, USA, 2011. ACM Press.

[12] O. De Silva, G. Mann, and R. Gosine. An ultrasonic and vision-based relative positioning sensor for multirobot localization. *IEEE Sensors Journal*, 15(3):1716–1726, 2014.

[13] M. Dorigo, M. Birattari, and M. Brambilla. Swarm robotics. *Scholarpedia*, 9(1):1463, 2014.

[14] S. Garnier, F. Tache, M. Combe, A. Grimal, and G. Theraulaz. Alice in pheromone land: An experimental setup for the study of ant-like robots. In *Swarm Intelligence Symposium (SIS 2007)*, pages 37–44. IEEE, 2007.

[15] D. Gelernter. Generative communication in Linda. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 7(1):80–112, 1985.

[16] B. P. Gerkey and M. J. Matarić. A formal analysis and taxonomy of task allocation in multi-robot systems. *International Journal of Robotics Research*, 23(9):939–954, 2004.

[17] P. Grassé. La reconstruction du nid et les coordinations inter-individuelles chez bellicositermes natalensis et cubitermes sp. la théorie de la stigmergie: Essai d'interprétation des termites constructeurs. *Insects Sociaux*, 6:41–83, 1959.

[18] T. Heer, S. Götz, S. Rieche, and K. Wehrle. Adapting Distributed Hash Tables for Mobile Ad Hoc Networks. In *Fourth Annual IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOMW'06)*, pages 173–178. IEEE, 2006.

[19] A. Howard, M. Matarić, and G. Sukhatme. Mobile sensor network deployment using potential fields: A distributed, scalable solution to the area coverage problem. In *Proceedings of the International Symposium on Distributed Autonomous Robotic Systems (DARS)*, pages 299–308. Springer, New York, 2002.

[20] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, July 1978.

[21] T. Li, X. Zhou, K. Brandstatter, D. Zhao, K. Wang, A. Rajendran, Z. Zhang, and I. Raicu. ZHT: A Light-weight Reliable Persistent Dynamic Scalable Zero-hop Distributed Hash Table. In *2013 IEEE 27th International Symposium on Parallel & Distributed Processing*, pages 775–787. IEEE Computer Society Press, 2013.

[22] Y. Meng and J. Gan. Livs: Local interaction via

virtual stigmergy coordination in distributed search and collective cleanup. In *Proceedings of the 2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1371–1376. IEEE, 2007.

[23] F. Mondada, M. Bonani, X. Raemy, J. Pugh, C. Cianci, A. Klaptocz, J.-C. Zufferey, D. Floreano, and A. Martinoli. The e-puck , a Robot Designed for Education in Engineering. In P. J. S. Gonçalves, P. J. D. Torres, and C. M. O. Alves, editors, *Proceedings of Robotica 2009 – 9th Conference on Autonomous Robot Systems and Competitions*, volume 1, pages 59–65. IPCB, Castelo Branco, Portugal, 2006.

[24] L. Mottola, M. Moretta, K. Whitehouse, and C. Ghezzi. Team-level Programming of Drone Sensor Networks. In *SenSys '14 Proceedings of the 12th ACM Conference on Embedded Network Sensor SystemsSystems*, pages 177–190. ACM New York, NY, 2014.

[25] S. Nouyan, A. Campo, and M. Dorigo. Path formation in a robot swarm. *Swarm Intelligence*, 2(1):1–23, 2008.

[26] D. Payton, R. Estkowski, and M. Howard. Pheromone Robotics and the Logic of Virtual Pheromones. *Swarm Robotics*, 3342:45–57, 2005.

[27] C. Pinciroli, A. Lee-Brown, and G. Beltrame. Buzz: An extensible programming language for self-organizing heterogeneous robot swarms. Available online at http://arxiv.org/abs/1507.05946, 2015.

[28] C. Pinciroli, V. Trianni, R. O'Grady, G. Pini, A. Brutschy, M. Brambilla, N. Mathews, E. Ferrante, G. Di Caro, F. Ducatelle, M. Birattari, L. M. Gambardella, and M. Dorigo. ARGoS: a modular, parallel, multi-engine simulator for multi-robot systems. *Swarm Intelligence*, 6(4):271–295, 2012.

[29] J. F. Roberts, T. S. Stirling, J.-C. Zufferey, and D. Floreano. 2.5d infrared range and bearing system for collective robotics. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2009).*, pages 3659–3664. IEEE, 2009.

[30] M. Rubenstein, C. Ahler, and R. Nagpal. Kilobot: A low cost scalable robot system for collective behaviors. *2012 IEEE International Conference on Robotics and Automation*, pages 3293–3298, May 2012.

[31] T. Soleymani, V. Trianni, M. Bonani, F. Mondada, and M. Dorigo. Bio-inspired construction with mobile robots and compliant pockets. *Robotics and Autonomous Systems*, 2015. In press.

[32] K. Støy. Using situated communication in distributed autonomous mobile robots. In *Proceedings of the 7th Scandinavian Conference on Artificial Intelligence*, pages 44–52. IOS Press, 2001.

[33] M. Viroli, D. Pianini, and J. Beal. Linda in Space-Time: An Adaptive Coordination Model for Mobile Ad-Hoc Environments. *Coordination 2012, LNCS 7274*, pages 212–229, 2012.

[34] J. Werfel, K. Petersen, and R. Nagpal. Designing collective behavior in a termite-inspired robot construction team. *Science*, 343(6172):754–758, 2014.

[35] M. M. Zavlanos, M. B. Egerstedt, and G. J. Pappas. Graph Theoretic Connectivity Control of Mobile Robot Networks. *Proceedings of the IEEE*, 99(9):1525 – 1540, 2011.