DYNAM: Proceedings of the 1st International Workshop on Dynamicity

Lélia Blin, Yann Busnel

DYNAM
Proceedings of the 1st International Workshop on Dynamicity

Collocated with OPODIS 2011 / Toulouse, France

Editors: Lélia Blin and Yann Busnel

December 12, 2011

Contents

Part I

Foreword

Chapter 1

Scope

Dynamicity is a fundamental aspect of today's distributed applications. Large-scale systems and networks are now standard, yet designing, analyzing and taking into account their dynamicity raises many challenges.

The DYNAM workshop aims at fostering discussion on theoretical aspects of dynamicity among participants from different research areas like algorithm theory, distributed systems, networks and databases. The goal of DYNAM is to bring together researchers from such diverse fields of expertise.

Topics of interest cover all aspects of dynamicity, including, without being limited to:

- Large-scale dynamic distributed systems
- Modeling dynamic environment
- Distributed algorithms in dynamic networks
- Self-* dynamic computation
- Complexity measures and analysis of dynamic algorithms
- Performance modeling and prediction based on analytic approaches
- Application field of dynamicity: P2P, Sensor and Mobile networks, Social networks
- Fault-tolerance, reliability, availability in dynamic environment

The workshop is looking for contributions addressing research, ongoing work, ideas, concepts, and critical questions related to algorithmic and modeling aspects of dynamic computing. The final workshop program will include both talks presenting novel contributions and keynote presentations.

Chapter 2

Chairs and committee

General co-chairs

Lélia Blin, Universit d'Evry Val d'Essonne, LIP6, France

Yann Busnel, Universit de Nantes, LINA, France

Program Committee

Lélia Blin	Université d'Evry Val d'Essonne, LIP6
Yann Busnel	University de Nantes
Shantanu Das	The Technion-Israel Institute of Technology
Fabiola Greve	Federal University of Bahia (UFBA)
David Ilcinkas	CNRS Bordeaux
Anne-Marie Kermarrec	INRIA, Rennes Bretagne Atlantique
Mikel Larrea	University of the Basque Country
Leonardo Querzoni	Sapienza, Università di Roma
Etienne Rivière	Université de Neuchâtel
Gilles Tredan	Technische Universität Berlin

Chapter 3

Author Index

Part II

Papers

Toward a Solution to Partitionable Group Membership for MANETs

Léon Lim and Denis Conan
Institut Télécom, Télécom SudParis
UMR CNRS Samovar
{Leon.Lim,Denis.Conan}@telecom-sudparis.eu

Keywords: MANETs, dynamic partitionable systems, partitionable group membership, abortable consensus.

Context of the study

Ubiquitous computing environments are characterised by a diversity of mobile nodes and networks, and in particular, mobile ad-hoc networks. *Mobile Ad-hoc NETworks* (MANETs) are self-organising networks that lack a fixed infrastructure, and due to nodes arrivals, departures, crashes and movements, they are very dynamic networks. The topology changes occur both rapidly and unexpectedly and nodes (processes) can dynamically enter and leave the system. Thus, a distributed system built over MANETs can be partitioned. Distributed systems that are built over MANETs must be partition-tolerant, that is network partitioning may result in a degradation of services, but not necessarily in their unavailability. Fault-tolerant applications in partitionable system models generally rely on the two services of *Group Communication System*: *(1)* group membership service and *(2)* reliable multicast service. Informally, *group membership* specifies the *view* a process has on the current group it belongs to whereas *reliable multicast* provides reliable message diffusion within the same group. In this work, we focus on the group membership service, and more precisely, on group membership services for partitionable systems built over MANETs.

Group membership in dynamic and partitionable systems

Group membership consists of two sub-problems [10]: *(i)* determining the set of processes that are currently *up*, and *(ii)* ensuring that processes agree on the successive value of the this set. In the literature, two types of group membership services have emerged: primary-partition [9] and partitionable [5, 2]. Roughly speaking, *primary-partition* group membership maintains a single agreed view of the group. Such a group membership is intended for classical not-partition-tolerant systems. Since only a single view of the group can exist, primary-partition group membership requires strong assumptions on the system to satisfy the agreement property. These assumptions are strong enough to show that group membership is impossible in systems with crash failures [4]. In contrast to primary-partition ones, *partitionable group memberships* allow processes to disagree on the current membership of the group. Collaborative applications [3], resource allocation management [1], and distributed monitoring [7] are examples of applications that support permanent partitioning and thus go on running on multiple partitions. Such partitions may present some *eventual stability* [6] so that liveness of the computation can be guaranteed, that is a stability period lasts long enough so that eventually all the participant nodes of the partition are connected to each other and can communicate in a timely manner. Partitions can merge into larger partitions when the communication links between them are re-established. Thus, partitionable group memberships allow `split` and `merge` operations on partitions. By allowing disagreement, they escape from the impossibility result of [4]. However, they run into another fundamental issue. Specifying partitionable group membership for partitionable systems faces two orthogonal goals [4, 8]: *(1)* the specification must be weak enough to be solvable (implementable); and *(2)* it must be strong enough to simplify the design of fault-tolerant distributed applications in partitionable systems.

Open issues in the specification of partitionable group membership

To be useful, a partitionable group membership service must ensure at least the *virtual synchrony* property which is considered in the literature to be a basic property of partitionable membership services [5]. Two prominent specifications of partitionable group membership that ensure virtual synchrony are [5] and [2]. They sketch the two categories of partitionable group membership specifications which have been proposed in the literature and which differ about their liveness property: *(1)* liveness must hold only in stable partitions [5], and *(2)* liveness

must be ensured in every partition [2]. In [5], the authors define a completely stable partition as "a set of processes that are eventually alive and connected to each other, and the link from any process in this set to any process outside the set is down". The specification of [5] does not ensure liveness of the system when two processes p and q are forever intermittently reachable. This unstable case disappears in the specification of [2] with the consideration of fair channels, and thus [2] guarantees liveness not only in stable partitions. However, as shown in [8], these two specifications are not satisfactory. For instance, the specification in [5] can be satisfied by a "trivial but useless implementation" and the specification in [2] cannot be implemented without strong synchrony assumptions. Furthermore, the system model in both [5, 2] is static and the network is initially fully connected.

Partitionable group membership as a sequence of abortable consensus

In our work, we propose a system model that characterises the dynamic behaviour of stable partitions in MANETs. Nodes that stay in a partition during a period that lasts enough are said to be *stable*, and *unstable* otherwise. In our model, the liveness property of partitions can be guaranteed even if they are not completely stable. To this means, we have defined a weak stability condition based upon the application-dependant parameter α which is a threshold value used to capture the liveness property of a partition. In each partition, α stable processes are required to execute distributed computations. Then, we propose a way to solve group membership in partitionable systems built over MANETs by adapting the Paxos protocol for such systems. This results in a specification of a form of consensus for partitionable systems called abortable consensus. *Abortable consensus* ($\mathcal{AC}$) is a combination of two abstractions: eventual α partition-participant detector ($\Diamond\mathcal{PPD}^{\alpha}$) and eventual register per partition ($\Diamond\mathcal{RPP}$). $\Diamond\mathcal{PPD}^{\alpha}$ is specified to abstract liveness in a partition whereas $\Diamond\mathcal{RPP}$ encapsulates safety in the same partition. The role of $\Diamond\mathcal{PPD}^{\alpha}$ is to make trade-offs between agreement and progress by eventually detecting the *stability condition* of α processes in a partition and eventually providing the leader among them. The participating nodes are selected among reachable nodes by some stability criterion that may satisfy the stability condition. A *stability criterion* is a parameter that is used to determine which nodes are the most stable ones. Finally, $\Diamond\mathcal{RPP}$ provides a distributed storage in a partition with a write-once semantics. The acts of locking and storing a value in the register can fail in two cases: *(1)* in case of contention or *(2)* in case of non-satisfied stability condition. The first case is the same as in the original Paxos algorithm: a proposer abandons a proposal if some proposer has begun trying to issue a higher-numbered one, but the consensus instance is not necessarily abandoned. In the second case, the proposer not only abandons the proposal but also the consensus instance if there is no α stable processes in its partition.

Then, the partitionable group membership problem is solved by a transformation into a sequence of $\mathcal{AC}$, where each $\mathcal{AC}$ is executed by the participant nodes in the current view. When the decision returned by the abortable consensus is a set of processes α-Set, these processes are the members of the next view. However, unlike a regular consensus, the returned value is not necessarily a value that was proposed by some process. It could be a specific value *abort* meaning that the consensus has aborted because the stability condition is not satisfied. In this case, processes re-compute their own α-Set, and then begin executing a new $\mathcal{AC}$ instance.

References

[1] T. Anker, D. Dolev, and I. Keidar. Fault tolerant video on demand services. In *Proc. 19th IEEE ICDCS*, pages 244–252, 1999.

[2] Ö. Babaoğlu, R. Davoli, and A. Montresor. Group Communication in Partitionable Systems: Specification and Algorithms. *IEEE Transactions on Software Engineering*, 27(4):308–336, April 2001.

[3] K.P. Birman, R. Friedman, M. Hayden, and I. Rhee. Middleware support for distributed multimedia and collaborative computing. *Software: Practice and Experience*, 29(14):1285–1312, 1999.

[4] T.D. Chandra, V. Hadzilacos, S. Toueg, and B. Charron-Bost. On the impossibility of group membership. In *ACM PODC*, pages 322–330, 1996.

[5] G.V. Chockler, I. Keidar, and R. Vitenberg. Group Communication Specifications: A Comprehensive Study. *ACM Computing Surveys*, 33(4):427–469, December 2001.

[6] A. Mostefaoui, M. Raynal, C. Travers, S. Patterson, D. Agrawal, and A. El Abbadi. From Static Distributed Systems to Dynamic Systems. In *Proc. 24th IEEE SRDS*, pages 109–118, Florianpolis, Brazil, October 2005.

[7] P. Murray. A Distributed State Monitoring Service for Adaptive Application Management. In *Proc. IEEE DSN*, pages 200–205, 2005.

[8] S. Pleish, O. Rütti, and A. Schiper. On the Specification of Partitionable Group Membership. In *Proc. 7th EDCC*, pages 37–45, May 2008.

[9] A. Schiper. Brief announcement: dynamic group communication. In *ACM PODC*, page 113, 2003.

[10] A. Schiper and S. Toueg. From Set Membership to Group Membership: A Separation of Concerns. *IEEE Transactions on Dependable and Secure Computing*, 3(1):2, 2006.

A Dynamic mechanism for solving Interfering Adaptations in Ubiquitous Computing Environment

Sana Fathallah Ben Abdenneji[1], Stéphane Lavirotte[1], Jean-Yves Tigli[1], Gaëtan Rey[1], Michel Riveill[1]

[1]I3S Laboratory, 930 Route des Colles, 06903 Sophia-Antipolis France

{fathalla, stephane.lavirotte, jean-yves.tigli, gaetan.rey, michel.riveill}@unice.fr

Abstract. Dynamic adaptation is a central need in Ubiquitous computing. In this kind of system, it is frequent to see the appearance of interference, when there are several adaptations to be applied on the application. In this paper, we present an automatic mechanism for dynamic interference resolving. Applications and adaptations will be represented by graphs; then we apply graph transformation rules to compute at runtime the solution.

Keywords: self-adaptation, software composition, interference resolution, graph transformation.

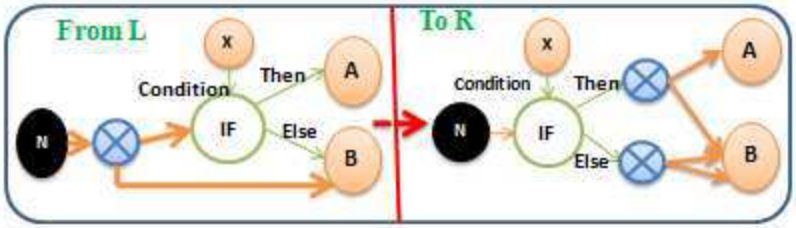
1 Introduction

Ubiquitous computing relies on objects of our daily life. These devices are able to communicate together and they constitute the software infrastructure, on which the ubiquitous system is based. In such kind of systems, applications are designed based on components assembly or orchestration of services. They can be represented using ***graphs*** where *nodes* are the entities (components or services) and *edges* model the interaction between these entities. Because of devices mobility, the graph modeling the application must be continually adapted to consider the changes of the environment (appearance/disappearance of devices). So, the system should react *automatically* and *transparently* by applying adaptations. An adaptation, which can also be modeled as a graph, specifies behaviors to be integrated into the initial graph (application). As a consequence, an adaptation changes the graph structure of the application by adding or modifying links and/or nodes. In addition, adaptations have to be written without *a priori knowledge* of the other adaptations of the system. So, during the step of adaptation, it is possible to have some modifications which can be attached in common points of the graph: we call these ***interferences.***

Researchers in self adaptation applications have used different techniques to deal with interferences. Zhang *et al.* [2] specify adaptations precedence at modeling level. However, interferences occur at runtime; so the resolution must be applied at runtime. Dinkelaker *et al*. [1] propose to dynamically change the adaptation composition strategies according to the runtime state of application. In this approach, if we add a new adaptation to the system, the developer should study its relationship with the other adaptations, which is a complex task. From these works, we can conclude that the proposed techniques cannot remove all interferences. To alleviate this problem, we propose to merge automatically interfering adaptations without preventing interferences explicitly. Our mechanism guarantee the independence between adaptations that can be composed whatever their order, and that can be added or removed easily to the system.

2 Merging interfering adaptations using Graph Transformation

Since adaptations are designed independently, they can interfere. Our solution is to merge them. This is possible due to the knowledge of the semantic of some nodes of the graph. Our graphs have two classes of nodes. **Whitebox** nodes (are operators that explain their semantics) and **Blackbox** node (are devices; they encapsulate the functionalities that can be only accessed by their ports, without knowing their semantics). The interference resolution process can be divided into two steps. The first step is the interference detection. The input of this step is the graph G which is obtained from the superposition of all graphs of adaptations on the graph of the initial application. On the graph G, we add a specific component ⊗ (*Fus*) to mark all interference problems. The second step is interference resolution. Since we work at graph level, the resolution will be a transformation of the graph G to the graph G'. Therefore, we need to define graph transformation rules that specify how to merge all known semantic nodes. We have defined a set of merging rules which derived from previous works [3]. A graph transformation rule has the form of $r:L \rightarrow R$. It can be performed if there is an occurrence of L in the host graph G. The rule execution implies to: (1) remove L and (2) add the R graph. The figure 1 represents an example of the graph transformation rule. It shows the merging of the semantic known operator IF (the conditional behavior) and a message send. The conditional behavior "*IF*" is specified by three parts. "*X"* node represents the condition to be evaluated. When "X" is ***True***, we execute the node *"A*", otherwise "*B*" will be executed.



Our composition mechanism is independent from application's implementation because it occurs on the graph G, which abstracts all details. The main property of our composition is the ***symmetric***. This property consists of three sub-properties: *associativity*, *commutativity* and *idempotency*. It means that there is ***no order*** in which composition process should be applied. As a consequence, adaptations are independent of each other and it can be composed in an unanticipated manner. These properties allow the adaptation process to be deterministic. To enable the merging of these interfering rules with the previous properties, we constrain the adaptation language. Whatever the language used to write the adaptation, it must be based on a limited set of operators with a well-known semantic that can be merged. In our implementation, 6 operators were defined; which implies the definition of 16 graph transformation rules.

References

1. Dinkelaker T., Mezini M., and Bockisch C.: The art of the meta-aspect protocol. In Proceedings of the 8th ACM international conference on Aspect-oriented software development, pp. 51–62. ACM, (2009).
2. Zhang J., Cottenier T., Van Den Berg A., and Gray J. : Aspect composition in the motorola aspect-oriented modeling weaver . In Journal of Object Technology , (2007).
3. Tigli J. Y., Lavirotte S., Rey G., Hourdin V., Cheung-Foo-Wo D., Callegari E., and Riveill M. WComp Middleware for Ubiquitous Computing: Aspects and Composite Event-based Web Services. In Annals of Telecom, pp. 197-214, (2009).

Understanding community evolution in Complex systems science

Qinna Wang[a] and Eric Fleury[b]
[a] *qinna.wang@ens-lyon.fr*
[b] *eric.fleury@inria.fr*

LIP ENS-LYON
D-NET INRIA
Université de lyon
46 Allée d'Italie Lyon 69364 France

Complex systems is a new approach in science that studying organized behaviours in computer science, biology, physics, chemistry, and many other fields. By collecting articles containing topic keywords relevant for the field of complex networks from ISI Web of knowledge during 1985-2009, we construct a science network, which connects ~ 215000 articles according to the proportion of shared references. Moreover, articles' publication time makes it dynamically evolve in time. We here use a two-step approach [3] to explore community evolution and study underlying information behind community changes. We firstly detect communities by applying Louvain algorithm [2] on each snapshot graph, and secondly construct relationships between partitions at successive snapshot graphs [1].

Communities may change, like fusion, split, disappearance and emergence. To construct relationships between communities, we use *community predecessor and successor*: given a community $C_i(t)$ found at time t, its predecessor is community $C_j(t-1)$, which has the maximum overlap size among all communities at time $t-1$, such as $C_j(t-1) = \arg\max_{C_k(t-1)\subseteq\mathcal{P}(t-1)} |C_k(t-1) \cap C_i(t)|$; its successor is community $\mathcal{C}_j(t+1)$, which has the maximum overlap size among all communities at time $t+1$, such as $C_j(t+1) = \arg\max_{C_k(t+1)\subseteq\mathcal{P}(t+1)} |C_k(t+1) \cap C_i(t)|$. Given a pair of clusters (X, Y), we use $X \to Y$ to denote that Y is X's successor while $X \leftarrow Y$ to denote that X is Y's predecessor. Besides, we define community $C_i(t)$'s *survival* is community $C_j(t+1)$ such as $C(i)(t) \leftrightarrows C_j(t+1)$, if and only if $C_i(t) \to C_j(t+1)$ and $C_i(t) \leftarrow C_j(t+1)$.

We use the survival to describe one community evolving stable. Furthermore, we also use community predecessor and successor to identify community dynamic events: given a community $\mathcal{C}(t)$, if it has more than one predecessors, then $\mathcal{C}(t)$ is a merged community; if it has more than one successors, then $\mathcal{C}(t)$ split in the next time step; if it has no predecessor, then $\mathcal{C}(t)$ is a new community; If it has no successor, then $\mathcal{C}(t)$ will vanish; otherwise, it evolves stable. A diagram (see Fig 1) shows several cases involving community dynamic events. We observe nearly all types of community changes: community $\mathcal{C}_2$ emerges at $t = 2$, community $\mathcal{C}_3$ disappears at $t = 3$, community $\mathcal{C}_2$ merges into $\mathcal{C}_1$ at $t = 4$ and community $\mathcal{C}_4$ is split from $\mathcal{C}_3$ at $t = 3$. For community $\mathcal{C}_1$, it evolves stable across the total four time steps although it is related to a merge event. It is like the change of $\mathcal{C}_2$ rather than $\mathcal{C}_1$.

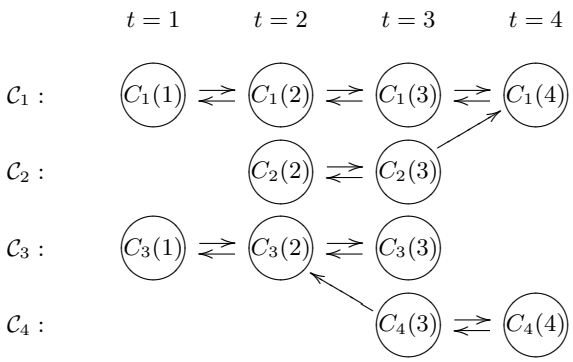


Fig. 1. Diagram of four communities over four time points, featuring continuation, emerge, disappearance, merge and split community events.

After applying our method on Complex Systems Science's snapshots: $G(1985-1995), G(1990-2000), G(1995-2005), G(2000-2010)$, we observe most of communities evolve stable over time, especially communities representing theoretical sciences like chaos theory(CHAOS), systems ecology(ECOLOGY), systems neuroscience(NEURAL NETs)[1][2]. These stable communities are also involving many change events, especially merges between function applications and science domains such as ISING-MODEL and PHOTOSYSTEM-II merged into chemistry during $1990-2000$. Results on community evolution suggest that theoretical sciences are the foundations of Complex System Science.

References

1. S. Asur, S. Parthasarathy, and D. Ucar. An event-based framework for characterizing the evolutionary behavior of interaction graphs. In *Proceedings of the 13th ACM SIGKDD on Knowledge discovery and data mining*, page 921. ACM, 2007.
2. V. D. Blondel, J. L. Guillaume, R. Lambiotte, and E. Lefebvre. Fast unfolding of communities in large networks. *JSTAT*, 2008.
3. M. Spiliopoulou, I. Ntoutsi, Y. Theodoridis, and R. Schult. Monic: modeling and monitoring cluster transitions. In *Proceedings of the 12th ACM SIGKDD on Knowledge discovery and data mining*, pages 706–711. ACM New York, NY, USA, 2006.

[1] CHAOS, ECOLOGY and NEURAL NETs are the most popular key words shared by articles in the found communities while chaos theory, systems ecology, systems neuroscience are disciplinarians by considering popular keywords shared by articles in the found communities. For instance, the community nominated by CHAOS representing chaos theory have popular keywords like DYNAMIC, CHAOS, SYSTEM, NONLI-, COMPL-, MODEL, STABI- and so on.

[2] Through our method, we found many matching communities sharing the same popular keywords and the same disciplinarians, like CHAOS and ECOLOGY, and many matching communities sharing different popular keywords but representing the same disciplinarians, like biology:nervous system(RAT, BRAIN) whose articles focus on biology functions about humans' brain in place of rats' as time going.

Towards Temporal Mobility Markov Chains

Sébastien Gambs
Université de Rennes 1 - INRIA / IRISA ;
Campus Universitaire de Beaulieu
35042 Rennes, France

Marc-Olivier Killijian
Miguel Núñez del Prado Cortez
CNRS ; LAAS, 7 avenue du Colonel Roche
F-31077 Toulouse, France
Université de Toulouse ; UPS, INSA, INP, ISAE ; LAAS
marco.killijian@laas.fr

I. Mobility Markov Chains

In [1], we defined a type of mobility model that we coined as *mobility Markov chain (MMC)*, which can represent in a compact yet precise way the mobility behaviour of an individual. In short, a MMC is a probabilistic automaton in which states represent points of interest (POIs) of an individual and transitions between states corresponds to a movement from one POI to another one. The automaton is probabilistic in the sense that a transition between POIs is non deterministic but rather that there is a probability distribution over the transitions that corresponds to the probability of moving from one POI to another (edges are directed). Note that Markov models are a popular technique that have been used in the past for the study of motion (for instance see [2] for a recent work using hidden Markov networks to extract POIs from geolocated data).

More formally, a MMC is a transition system composed of:

- *A set of states* $P = \{p_1, \ldots, p_n\}$, in which each state p_i corresponds to a POI (or a set of POIs). These POIs may have been learned for instance by running a clustering algorithm on the trail of mobility traces from an individual or simply by collecting the locations that he has posted on a geolocated social network such as Foursquare or Gowalla. Each state (*i.e.* POI) is therefore associated with a physical location.Moreover, it is often possible to attach a *semantic label* to the states of the mobility Markov chain, such as for instance "home" instead of simply p_1" or "work" instead of p_2.
- *A set of transitions*, $T = \{t_{1,1}, \ldots, t_{n,n}\}$, where each transition $t_{i,j}$ represents a movement from the state p_i to the state p_j. Each transition $t_{i,j}$ has a probability assigned to it that corresponds to the probability of moving from state p_i to state p_j. Sometimes an individual can move from one POI, go somewhere else (but not to one of his usual POIs) and come back later to the same POI. For example, an individual might leave his house to go wash his car in a facility near his home and come back 30 minutes later. This type of behaviour is materialized in the mobility Markov chain by a transition from one state to itself.

A MMC can be represented either as a *transition matrix* or as a *directed graph* in which nodes correspond to states and there is a directed weighted edge between two nodes if and only if the transition probability between these two nodes is non-null. The sum of all the edges' weights leaving from a state is equal to 1.

For instance, consider for illustration purpose, an individual, that we refer thereafter as "Alice", who has a set of 4 important POIs that she visits often plus some other POIs that are less important to her mobility. In order to represent her mobility behaviour, we could define a MMC composed of 5 states, one for each important POI plus a last one that will contain all the infrequent POIs. Suppose now that we have been able to collect her trail of mobility traces (*e.g.*, by tracking her mobile phone [3]), then possibly we could have learnt the following MMC (Figure 1). For more details, about MMC we refer the reader to [1].

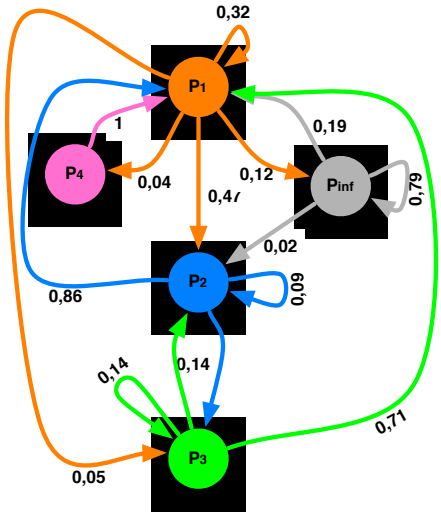


Fig. 1. Alice's Mobility Markov Chain.

II. Introducing Time into Mobility Markov Chains

In this section, we describe an algorithm for learning Temporal Mobility Markov Chains (TMMC). This algorithm is an extension of the one described in [1] to which we add the temporal aspect. Basically, the gist of this algorithm is to decompose time into n different discrete timeslices. For example, with $n = 4$, we can observe mobility in one of the following timeslices: *morning*, *afternoon*, *evening* and *night*. Each POI from the original MMC will thus be represented by n different states in the TMMC, one for each timeslice. If we

Algorithm 1 Temporal mobility Markov chain algorithm

Require: Trail of (mobility) traces M, merging distance d, speed threshold ϵ, time interval threshold $mintime$, time slices definition $timeslices$

1: Run a clustering algorithm on M to learn the most significant clusters
2: Merge all the clusters that are within d distance of each other
3: Let $listPOIs$ be the list of all remaining clusters
4: **for** each cluster C in $listPOIs$ **do**
5: Compute the $time\_interval$ and the $density$ of C
6: **end for**
7: **for** each cluster C in $listPOIs$ **do**
8: **if** $C.time\_interval > mintime$ **then**
9: Add C to $freqPOIs$ (the list of frequent POIs)
10: **else**
11: Add C to $infreqPOIs$ (the list of infrequent POIs)
12: **end if**
13: **end for**
14: Sort the clusters in $freqPOIs$ by decreasing order according to their densities
15: **for** each cluster C_i in $freqPOIs$ (for $1 \leq i \leq n-1$) **do**
16: **for** each time slice t in $timeslices$ **do**
17: Create a state $p_{i/t}$ in the mobility Markov chain
18: **end for**
19: **end for**
20: Create a state $p_{infrequent}$ representing all the clusters within $infreqPOIs$
21: Let M' be the trail of traces obtained from M by removing all the traces whose speed is above ϵ
22: **for** each mobility trace in M' **do**
23: **if** the distance between the trace and the state p_i is less than d and the state p_i is the closest state **then**
24: determine t the appropriate time slice for the trace
25: label the trace with "$p_{i/t}$"
26: **else**
27: label the state with the value "unknown"
28: **end if**
29: **end for**
30: Squash all the successive mobility traces sharing the same (time and space) label into a single occurrence
31: Compute all the transition probabilities between each pair of states of the Markov chain
32: **return** the mobility Markov chain computed

take Alice's home for example, which is state p_1 on Figure 1., it will be represented by the states $P_{1morning}$, $P_{1afternoon}$, $P_{1evening}$ and P_{1night}. Remark that the scale of the timeslicing can be tuned to match the required level of detail. For example, the timeslices can also be $winter, spring, summer, autumn$ or even months or years, ...

In a nutshell, Algorithm 1 starts (line 1) by applying a clustering algorithm on a trail of traces of an individual in order to identify clusters of locations that are significant. Then, in order to reduce the number of resulting clusters, the algorithm merges clusters whose medoids are within a predefined distance d of each other (line 2). Each resulting cluster is considered as a POI, and the medoid of the cluster is considered to be the physical location of the POI. For each cluster (lines 4 to 6), we compute the number of mobility traces inside the cluster (which we call the *density* of the cluster) and the *time interval* (measured in days) between the earliest and the latest mobility traces of the cluster (line 5). The POIs (*i.e.* clusters) are then split (lines 7 to 13) into two categories; the *frequent POIs* that correspond to POIs whose time interval is above or equal to a certain threshold $mintime$ and the *infrequent POIs* whose time interval is below this threshold $mintime$. In the set of frequent POIs (line 14), we sort the POIs by decreasing order according to their densities. Therefore, the first POI will be the denser and the last POI the less dense.

Now, we can start to build the temporal mobility Markov chain by creating a state for each tuple $< POI, timeslice >$ within the set of frequent POIs (lines 15 to 19) and also a last state representing the set of infrequent POIs (line 20). Each state is then assigned a weight that is set to its density. Afterwards (line 21), we come back to the trail of traces that have been used to learn the POIs and we remove all the moving points (whose speed is above ϵ, for ϵ a small predefined value). Then, we traverse the trail of traces in a chronological order (lines 22 to 29) labeling each of the mobility traces either with the tag of closest POI and the appropriate timeslice (line 25) or with the tag "unknown" if the mobility trace is not within d-distance of one of the frequent or infrequent POIs (line 27). From this labeling, we can extract sequences spatio-temporal chunks that have been visited by the individual in which all the successive mobility traces sharing the same label are merged into a single occurrence (line 30). For example, a typical day could be summarized by the following sequence "$p_{home/morning} \Rightarrow p_{work/morning} \Rightarrow p_{work/afternoon} \Rightarrow p_{sport/afternoon} \Rightarrow p_{sport/evening} \Rightarrow$ "$unknown$" $\Rightarrow p_{home/evening}$". From the set of sequences extracted, we compute the transition probabilities between the different states of the MMC by counting the number of transitions between each pair of states and then normalizing these probabilities (line 31). If we observe a subsequence in the form of "$p_{i/t} \Rightarrow$ "unknown" $\Rightarrow p_{i/t}$" then we increment the count from the state $p_{i/t}$ to itself (which translates in the graph representation by a self-arrow).

REFERENCES

[1] S. Gambs, M.-O. Killijian, and M. N. del Prado Cortez, "Show me how you move and i will tell you who you are," *Transactions on Data Privacy*, vol. 4, no. 2, pp. 103–126, 2011.

[2] Z. Yan, D. Chakraborty, C. Parent, S. Spaccapietra, and K. Aberer, "SeMiTri: A Framework for Semantic Annotation of Heterogeneous Trajectories," in *14th International Conference on Extending Database Technology (EDBT)*, 2011.

[3] M.-O. Killijian, M. Roy, and G. Trédan, "Beyond san francisco cabs : Building a *-lity mining dataset," in *Workshop on the Analysis of Mobile Phone Networks (NetMob 2010)*, 2010.

A distributed algorithm for path maintaining in dynamic networks

F. El Ali[(1)(2)], B. Ducourthial[(1)(2)]
(1) Université de Technologie de Compiègne
(2) CNRS Heudiasyc UMR6599, BP 20529, 60205 Compigne Cedex, France
(corresponding author: Bertrand.Ducourthial@utc.fr)

***Abstract*— Routing becomes obsolete in ad hoc dynamic networks. Path maintaining becomes necessary in order to maintain a communication that already started between two entities. In this paper we present a path maintaining algorithm.**

***Index Terms*— Ad hoc, dynamic, path maintaining**

I. Introduction and related works

Ad hoc dynamic networks represent a challenging domain of application due to their specifications. Still we find many applications on these networks nowadays, like the vehicular networks, drone networks or even pedestrian networks. The general method to study these networks starts by studying their dynamics. The dynamic changes are closely related to the frequency of neighborhood changes, and the functioning and requirements of the protocols.

Unicast communications in these networks require a continuous communication between the source of the messages and the destination even if they are separated by multiple hops. Many routing protocols exist for mobile networks, like AODV, OLSR and geographic routing among others. But these protocols are not efficient if the dynamic of the network increases to exceed the admitted dynamicity. Not to forget that these protocols require broadcast in general, location services [5] [4] or infrastructure bases, which make them inadequate to highly mobile networks. We should also mention that the need for global knowledge in these protocols increases the control over the network, in order to build or to update the structures needed to maintain routes.

II. Contribution

Due to their specificities, dynamic networks do not require routing as those used in networks with fixed topologies or low dynamics. In fact, in mobile networks, the search for a distant destination might be obsolete since the nodes are frequently moving, meaning that the destination might have changed its location by the time the source receives its response. The search for the destination implies the broadcast of messages in order to identify its position. And in case some geographic protocols admit the knowledge of the position of the destination, this means that a location service has already identified this position using broadcasted messages. As explained in [2], the need for unicast communication is in order to maintain a communication that started when the source and the destination were neighbors, and that is needed to pursued for a certain duration. This problem is called path maintaining. This said, the aim of the path maintaing application is to maintain a communication initiated with a path of length 1 and to avoid the flooding of the network with unnecessary messages.

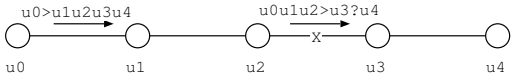


Fig. 1: Local periodic messages

In this paper, we describe the path maintaining algorithm, that we denote PTH. PTH is a path maintaining algorithm that relies on local adjustments of the path. Local adjustment is used in order to avoid having a global knowledge of the network, which implies to flood the network with too much control messages in order to build a global knowledge that might become unusable because of the dynamic. PTH adjusts the path between the source and the destination using neighbor nodes of the broken links in the path. Starting from a direct communication between the source and the destination, PTH interferes when these two entities move apart, and the direct communication between them becomes impossible. At this point, the neighbor nodes of the broken link are used to relay the messages. The same mechanism is used for all the broken links of the path between the source and the destination. PTH relies on three mechanisms: local periodic update for the nodes of the path only, path reduction and path extension. These mechanisms are detailed in the next section.

III. The PTH algorithm

In this section, we describe our algorithm named PTH for solving the path maintaining problem. It is mainly based on three mechanisms, that we describe in this section: local periodic update, path extension and path reduction.

The first mechanism – **local periodic update** – allows to deal with neighborhood changes and failures. It relies on periodic diffusion of messages in the neighborhood of the sender. These messages contain some local information regarding the path. Such information is composed by the list of members of the path (eg. $\mathtt{u}_0\mathtt{u}_1\mathtt{u}_2\mathtt{u}_3\ \ldots$) (see Figure 1), some flags allowing to determine the sender and the willing receiver of the current message ($\mathtt{u}_0\mathtt{u}_1{>}\mathtt{u}_2\mathtt{u}_3$ if u_1 is the sender and u_2 the receiver) and an uncertainty flag regarding a member of the path ($\mathtt{u}_0\mathtt{u}_1{>}\mathtt{u}_2\mathtt{?u}_3$ (see Figure 1) if u_1 has a doubt on its successor u_2). Such a message informs both the predecessor and the successor of the local state of the

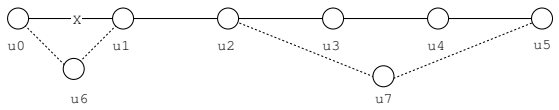


Fig. 2: Extension of the path through u6 and reduction through u7

path, which in turns can then update their own information. Other nodes not belonging to the path may propose to belong to, in order to repair or shorten it.

The second mechanism – **path extension** – allows to deal with link breaking. This can appear when two neighbors go far each other (Consider Figure 2, with $\mathtt{u}_0 = \mathtt{u}_i$, $\mathtt{u}_1 = \mathtt{u}_j$ and $\mathtt{u}_6 = \mathtt{v}$ for instance). Consider that the link $[u_i, u_j]$ in the path breaks. As the node u_i sends periodically some messages containing $\mathtt{u}_i{>}\mathtt{u}_j$, it expects implicit acknowledgment of u_j, that is messages without uncertainty, containing $\mathtt{u}_i\mathtt{u}_j{>}$. This means that $\mathtt{u}_j$ receives the message of $\mathtt{u}_i$ and forwards it with the path $\mathtt{u}_i\mathtt{u}_j{>}$. If it does not receive such messages from u_j, it sets the uncertainty flag on its successor u_j in its next messages: $\mathtt{u}_i{>}\mathtt{u}_j?$. Either u_j receives messages of u_i with an uncertainty flag on itself (communication $u_i \leftarrow u_j$ broken) or it does not receive at all messages from u_i (communication $u_i \rightarrow u_j$ broken). In both cases, it will set the uncertainty flag on its predecessor u_i in its messages; they will contain: $\mathtt{u}_i?\mathtt{u}_j{>}$. Such uncertainty flags allows to neighbors of both u_i and u_j to propose repairing the path. If a neighbor node v notices such exchanges, it counts the received messages from u_i and u_j. When a given threshold is reached, v proposes to be relay of the communication by sending $\mathtt{u}_i?\mathtt{v}{>}\mathtt{u}_j?$. As several neighbors of both u_i and u_j may propose to be a relay, u_i (closer from the source) chooses the first (say v) and sends $\mathtt{u}_i{>}\mathtt{v}\mathtt{u}_j?$. Then v sends $\mathtt{u}_i\mathtt{v}{>}\mathtt{u}_j?$. Then u_j sends $\mathtt{u}_i\mathtt{v}\mathtt{u}_j{>}$ and the path is repaired (to the condition that a node v was present in the neighborhood of the breaking edge extremities).

The third mechanism – **path reduction** – consists in reducing the path when possible. This can be done by a node in the path, or neighbor of the path. Consider the case when the reduction between two given nodes u_i and u_j (where u_i precedes u_j in the path) can be done by a node v, neighbor of the path (Consider Figure 2, with $\mathtt{u}_2 = \mathtt{u}_i$, $\mathtt{u}_5 = \mathtt{u}_j$ and $\mathtt{u}_7 = \mathtt{v}$ for instance) . This node v, by receiving the messages sent by u_i and u_j simultaneously, observes that it can reduce the path by being a relay between u_i and u_j.

The proposition of the reduction is done by placing in the path the shortcut denoted by $\mathtt{u}_i\text{-}?\mathtt{v}\text{-}?\mathtt{u}_j$ (with the uncertainty flag on the links $[u_i, v]$ and $[v, u_j]$). The message sent by v is of the form: $\ldots \mathtt{u}_i\text{-}?\mathtt{v}\text{-}?\mathtt{u}_j \ldots \mathtt{v}{>}\mathtt{u}_j \ldots$. When u_j receives this message, it confirms to v the possibility of reduction, removing the uncertainty on the reduction link $[v, u_j]$. The node v then asks the permission of u_i to be a relay. At this point, u_i sends a validation message $\ldots \mathtt{u}_i\text{-}\mathtt{v}\text{-}\mathtt{u}_j{>} \ldots \mathtt{u}_j \ldots$ to its successor in the path; this message will be relayed until reaching u_j.

Indeed, as several overlapping reductions may occur simultaneously, the reduction $u_i - v - u_j$ has to be validated by all nodes from u_i to u_j in the path. This allows to avoid any split of the path into several smaller disconnected paths. If a node is already engaged with another reduction, it will not forward the validation message containing the reduction. In the converse case, the node forwards the message and becomes engaged. By this way, in case of two reductions in conflict for a common sub-path, the validated reduction is the one that first sent a validation message into the common sub-path. When u_j receives the validation message, it confirms the reduction by relaying from now messages of v instead of messages of its old predecessor u_{j-1}. An engaged node will either leave the path or receive a message without reduction from its predecessor; it stops then being engaged. The same process is used when the reduction is proposed by nodes u_i belonging to the path instead of a neighbor node.

IV. PTH VALIDATION AND PERFORMANCES

Whatever are the performance of a given algorithm, the dynamic of the network can lead to failure because it could compromise any communication. Our protocol is able to run even in highly dynamic network but the following requirement is necessary: a neighbor of the path is present during the path extension. This operation is very short (few local messages). By the way, we have a contract between the dynamic and the specifications of our protocol. This has been formalized in [3] under the term of best effort algorithm. A best effort algorithm does its best regarding the dynamic while still ensuring some useful properties for those (users or other protocols) relying on it. More precisely, whenever a *topological property* is fulfilled, a *continuity property* is ensured. Here, the topological property is the presence of a neighbor node when two nodes belonging to the path move far each other. The continuity property is the existence of the path from source to destination, the two initiators which were in the same neighborhoods at the beginning. Proof is omitted by lack of place.

PTH has been compared to a basic geographic routing with a location service, and to a traditional broadcast in the same scenario under the Airplug emulator [1] (with 31 cars moving from the UTC laboratory to the train station in Compiègne), and starting from the same initial state. Consider now N the number of messages sent by the source node in the network. While the broadcast flooded the network with 8.4×N messages, the geographic routing with its location service sent to the network 5×N messages, and PTH sent 2.4×N messages. Not to mention that when 31 cars were involved in sending messages for the broadcast or geographic routing, only 6 were involved when running the PTH algorithm.

REFERENCES

[1] A. Buisset, B. Ducourthial, F. El Ali, and S. Khalfallah. Vehicular networks emulation. Proceedings of ICCCN 2010, 2010.

[2] B. Ducourthial and Y. Khaled. *Vehicular Networks: Techniques, Standards and Applications*, chapter Routing in Vehicular Networks: User Perspective. CRC Press (Taylor & Francis Group), 2009.

[3] Bertrand Ducourthial, Sofiane Khalfallah, and Franck Petit. Best-effort group service in dynamic networks. In *SPAA 2010, June 13-15, 2010, Thira, Santorini, Greece*, 2010.

[4] Michael Kasemann, Holger Fuler, Hannes Hartenstein, and Martin Mauve. A reactive location service for mobile ad hoc networks. 2002.

[5] Hanan Saleet, Otman Basir Rami Langar and, , and Raouf Boutaba. Proposal and analysis of region-based location service management protocol for vanets. *IEEE "GLOBECOM" 2008 proceedings*, 2008.

Member Classification and Party Characteristics in Twitter during UK Election

Antoine Boutet
INRIA Rennes, France
antoine.boutet@inria.fr

Eiko Yoneki
University of Cambridge, UK
eiko.yoneki@cl.cam.ac.uk

Abstract

In modern politics, parties and individual candidates must have an online presence and usually have dedicated social media coordinators. In this context, real time members classification and party characterization taking into account the dynamic nature of social media are essential to highlight the main differences between parties and to monitor their activities, influences, structures, contents and mood. This paper summarizes a case study of member classification and party characteristics in Twitter during the UK election.

Keywords — Social Media, User Classification

1 Introduction

Using social media for political discourse is becoming common practice, especially around election time [4]. Different off-line approaches have been proposed for classifying the political leaning of users [3, 2]. However, inferring the political orientation of members in real time needs (1) to avoid collecting and using a prohibitive quantity of data for classification and (2) to consider the highly dynamic nature of social media including changes in term of active users, content or social structure.

On the other hand, while several studies have addressed the characterization of user in social networks [1], few works have tried to characterize politic parties. This characterization permits to highlight the main differences between parties, their impact in term of activity and influence, their preferred content, their mood and sentiment, their structure and how they exchange between each other over time. For instance, this information can be useful for political parties or candidates to monitor their impact and to assess and adapt their strategy.

This paper summarizes a case study of member classification and party characterization in Twitter during the 2010 UK general election. By looking at temporal changes in activity, Twitter is shown to react instantly to UK election events. To highlight the main differences between parties, we first characterize the self-identified users as member of a specific party according to several categories. Then, we present a real-time method for deducing user's party affiliation using only basic textual and semantic analysis of the public stream of Twitter. Finally, we discuss key parameters to address the dynamic nature of social media in online classification.

2 Party Characteristics

To highlight the main differences between parties, we first characterize the self-identified populations (users self-identified as members of specific parties in their Twitter user description) according to several categories: (1) the activity reflects the commitment of the parties to send tweets and to take place in political debates ; (2) the influence gives an indication on the potential impact of each party ; (3) the social structure reflects both the cohesiveness of the party and the level of exchange and debate between them ; (4) the content shows how the party used hashtags and urls, and the volume of references did to a specific party or political figure ; (5) the sentiment analysis evaluates the sentiment of words in tweets through different directions (self-focus, cognitive, social, positive and negative).

We saw that only few characteristics differ one party to another and made the following observations:

- **Activity.** Except for a small fraction of users with particular behavior in term of generated tweets, retweets or mentions, the majority of members from all parties exhibited a similar activity.
- **Influence.** Labour members had a better influence, however, Conservative members were better organized to promote their party.
- **Structure.** The retweet graph presents a highly segregated partisan structure where very few links connected different parties. In contrast, the mention graph formed a more heterogeneous structure where no particular clusters were observed.
- **Content.** The usage of hashtags depended on the underlying neutrality of the tag. In addition, each party was more likely to cite certains websites than others especially for blogs which were more political orientated. The similarity in terms of hashtags used showed a high heterogeneity between users in all parties. Moreover, we demonstrated that party members were more likely to make reference to

their own party than another.

- **Sentiment.** No particular sentiment direction permitted the differentiation of parties. However we clearly saw that members were more likely to express more positive opinion when they referenced its own party.

3 User Classification

The goal of this classification is to identify the party to which each person belongs and their evolution from the pattern of tweets. We consider the observed tweet process by events $E \in \{E_n\}$ representing the current state of beliefs of the political affiliation of members. Each affiliation probability is represented by an event, $Am \in \{labour, libdem, conservative\}$ for each user where $labour + libdem + conservative = 1$, containing elementary events $A_m \cap E_n$. The conditional probabilities or evidences $P(E_n|A_m)$ are specified to define the affiliation probability. $P(A_m)$ is the uncertainty of model A_m. Before the first inference step, the initial prior probability is set uniformly: $\{P(A)\} = \{\frac{1}{3}, \frac{1}{3}, \frac{1}{3}\}$. For each affiliation probability Am, $P(A_m)$ is updated to $P(A_m|E_n)$. From Bayes' theorem:

$$P(A_m|E_n) = \frac{P(E_n|A_m)P(A_m)}{P(E_n)} \quad (1)$$

where $P(A_m|E_n)$ is the posterior, the uncertainty of A_m after E_n is observed ; $P(A_m)$ is the prior, the uncertainty of A_m before E_n is observed ; and $\frac{P(E_n|A_m)}{P(E_n)}$ is a factor representing the impact of E_n on the uncertainty of A_m.

We define two evidences based on the main characters identified in the party charateristics analysis. The first is based on the pattern of retweets and highlights the fact that people of similar political persuasion retweet roughly the same thing (highly segregated partisan structure of the retweet graph). Retweets can be easily identified in the tweets stream thanks to the keyword *RT*. This evidence is defined as the average affiliation probability of both people retweeted by the user or people retweeting the user during the period $[E_{n-1}, E_n]$:

$$P(E_n|A_m = a) = \frac{\sum_{R \in Retweets} P(A = a)_R}{\textit{Number of retweets}} \quad (2)$$

The second exploits the fact that party members are more likely to make reference to their own party than another. This second evidence is defined as the ratio of tweets referencing one party over the total number of references during the period $[E_{n-1}, E_n]$:

$$P(E_n|A_m = a) = \frac{Volume(A_m = a)}{\sum Volume(A_m)} \quad (3)$$

Approach		Accuracy
Bayesian	Retweet	0.70
	Volume	0.80

Table 1: Performance of our user classification

Finally we define the political affiliation of user according to the best probability in A_m (i.e. $[0.7, 0.2, 0.1]$ refers a person who is probably Labour). Ties are broken randomly in case of equiprobabilities.

To evaluate our classification algorithm, initial seeds are chosen from the 10% most active users in term of generated tweets from those which are self-identified as member of specific party. We then measure the accuracy by calculating the number of correct predictions of the remaining 90% over the total number of predictions. Table 1 shows the high accuracy produced by our member classification.

4 Discussions and Conclusions

We summarize here a case study of member classification and party characteristics from Twitter during the 2010 UK general election. The three main differences identified between political parties are (1) the retweet graph presented a highly segregated partisan structure where very few links connect different parties, (2) party members were more likely to make reference to their own party than another, and (3) members were more likely to express more positive opinion when they referenced their own party.

To be able to act in real time, a classification system must not require a prohibitive amount of data. Furthermore, due to the dynamic nature of social media, active users and content can change quickly. In this context, collecting the social structure of every new user or performing linguistic analysis for each new hashtag is inconceivable. The proposed member classification exhibits good performance, can be easily used in real time and fits perfectly to the dynamic nature of social media.

References

[1] Fabrício Benevenuto, Tiago Rodrigues, Meeyoung Cha, and Virgílio Almeida. Characterizing user behavior in online social networks. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, IMC '09, pages 49–62, New York, NY, USA, 2009. ACM.

[2] Qiaozhu Mei Daniel Xiaodan Zhou, Paul Resnick. Classifying the political leaning of news articles and users from user votes. In *Proc. 5th International AAAI Conference on Weblogs and Social Media (ICWSM)*, 2011.

[3] Ana-Maria Popescu Marco Pennacchiotti. Democrats, republicans and starbucks afficionados: User classification in twitter. In *KDD*, 2011.

[4] Andranik Tumasjan, Timm O Sprenger, Philipp G Sandner, and Isabell M Welpe. Predicting elections with twitter : What 140 characters reveal about political sentiment. *Word Journal Of The International Linguistic Association*, pages 178–185, 2010.

Towards dynamic graph analysis: a position note

Erwan Le Merrer
Technicolor, Rennes

Gilles Trédan
CNRS, Toulouse

In this note, we elaborate on directions for efficient graph analysis, taking into account the dynamic nature of complex networks.

1 Introduction and problematic

Tailored network analysis is at the core of some of the most successful Internet stories, as Google search engine for instance [2]. Understanding complex networks is a challenge that receives a growing interest. Interestingly, this challenge is tackled not only by computer scientists, but also by biologists and physicists. Together, these efforts led to the definition of several important methodological tools to extract information from a complex network, such as centralities, sampling, or scale-free property analysis.

However, most of these tools have been defined in a static way, by only providing insights on crawled networks, at some fixed point in time [3]. This is notably counter intuitive, as the complex networks observed are by nature evolving constantly [4]; predictions are doomed to be inaccurate.

At the same time, researchers started to develop models that capture graphs over time. For instance the concept *time varying graphs* (TVGs) [1] which has been recently recognized as a suitable model to describe the evolution of a network in time, pushed by the development of delay-tolerant and opportunistic networks. The TVG model may then constitutes the next step of understanding complex networks, observing them on a dynamic time dimension.

Adapting the aforementioned static tools to TVGs may seem easy. A trivial solution is to simply take snapshots of the network at fixed intervals in time, and to run the analytical computation on each of these snapshots. This is of course cumbersome if the time interval is small, and even simply not computationally feasible if the observed network is large; this prohibits online use by number of resources limited devices or applications for instance.

Since the scale of each snapshot already forbids the use of expensive algorithms, it is clear to us that analyzing TVGs calls for developing methods to reduce the size of the input dataset[1]. The question we ask in this note is then the following one: **"How to reduce the computational/memory costs when analyzing TVGs?"**.

2 Possible directions

We foresee 4 directions to reduce the global cost in computing characteristics of TVGs: *i)* observe a subset of *relevant* nodes over time, *ii)* concentrate on a subset of *relevant* snapshots, *iii)* observe the evolution over time of a set of graph measures and *iv)* develop/adapt graph measures to TVGs. We will hereafter briefly detail each approach.

Sample nodes Sampling has been extensively studied and provides accurate results in static contexts. This is probably a very efficient approach to gather the evolution of aggregate values, such as the evolution of the average age of social network users. However unbiased sampling is difficult, and adding the time dimension complexifies the problem.

The question of newcomers is pending: should we only sample nodes originally selected at random? Moreover, such an approach can not answer questions regarding the evolution of the graph structure itself.

[1] `Google+` graph has now more than 40e6 users. At the time of this writing, its most famous user receives more than 3000 followers each day.

Sample snapshots Here, the objective is to reduce the analysis complexity by discarding snapshots that bring little information to concentrate on those conveying more meaning. However, selecting which snapshot is relevant to keep is a non trivial task.

The difficulty is precisely tied to the arbitrarily set interval at with a snapshot should be kept for analysis. If a snapshot is created at each topological change, *i.e.* appearance/disappearance of a node or edge (called *graph centric evolution* in [1]), then the sampling rate of those snapshots should be related to their creation. It might as well be of a real interest to sample more snapshots when topological changes occur in the core of the network (or on nodes or edges that have been previously marked as important), than when events occur at the fringe.

Evolution of graph measures The approach here is to run standard static graph analysis tools on each snapshot independently, and to interpret the obtained time serie. Reusing the static graph analysis toolbox is probably a very good starting strategy. Similarly to node sampling, this strategy has the drawback of only studying the evolution of static graph properties.

Keeping only an history of past analysis concerning a network may drastically reduce memory needed to track evolution in time, when compared to the snapshot-based approach. Starting from an initial network analysis, algorithms are providing outputs that are compared to previously observed results (one can for instance keep track of network diameter [4]). Only one network snapshot is then required in memory, for each evolution analysis, as compared to an array of snapshots as in the previous solution. The negative side being of course the loss of information when only conserving past measures instead of full network snapshots.

Adapted analytical tools The approach here is to develop new algorithms suited to TVGs and their natural information redundancy. For instance, in many social networks, such as the co-authorship graph in DBLP, links are never removed. Each snapshot is thus a subset of the following one. Recycling the results of previous snapshots to empower the current snapshot analysis could then lead to significant optimizations.

There is probably no generic solution for transforming static algorithms into their dynamicity-aware counterparts. For most of known tools for graph analysis, even the slightest update of the initial graph might have unpredictable consequence on the algorithm.

Most of all, we believe that some very important TVG metrics are only definable in a dynamic context. Even very simple but interesting questions such as "how long does a link takes to be reciprocal, for a triangle to be closed?" can not be addressed by simply adapting our static toolbox to a dynamic setup.

When collecting data from evolving structures, researchers are often bound to the limitations of the API or available datasets. Thus, it is also important to think about the robustness of these complexity reduction techniques against incomplete datasets. For instance, time sampling is sometimes not under control because of crawling time and delays. It is sometimes hard to ensure the full graph is collected. Moreover, crawling times might lead us to question the association of one snapshot to one point in time.

We however believe that each of these directions could be valuable to be explored. It is still unclear which methods are above the others while considering costs, but probably all of them lead to the observation of different interesting phenomena. We yet have to understand their respective relevance to specific contexts.

References

[1] A. Casteigts, P. Flocchini, W. Quattrociocchi, and N. Santoro. Time-varying graphs and dynamic networks. In *ADHOC-NOW*, pages 346–359, 2011.

[2] M. Franceschet. Pagerank: standing on the shoulders of giants. *Commun. ACM*, 54:92–101, 2011.

[3] A.-M. Kermarrec, E. Le Merrer, B. Sericola, and G. Trédan. Second order centrality: Distributed assessment of nodes criticity in complex networks. *Comput. Commun.*, 34:619–628, 2011.

[4] J. Leskovec, J. Kleinberg, and C. Faloutsos. Graphs over time: densification laws, shrinking diameters and possible explanations. In *Proceedings of ACM SIGKDD*, KDD '05, pages 177–187. ACM, 2005.