Étude de la dynamique des communautés de twitters en relation avec l'utilisation des hashtags

Hadrien Croubois

14/06/2012 - 27/07/2012

Sous la direction de Mr Bertrand Jouve
Professeur Université de Lyon, Lyon 2 - ERIC/IXXI
Directeur Adjoint Scientifique InSHS-CNRS

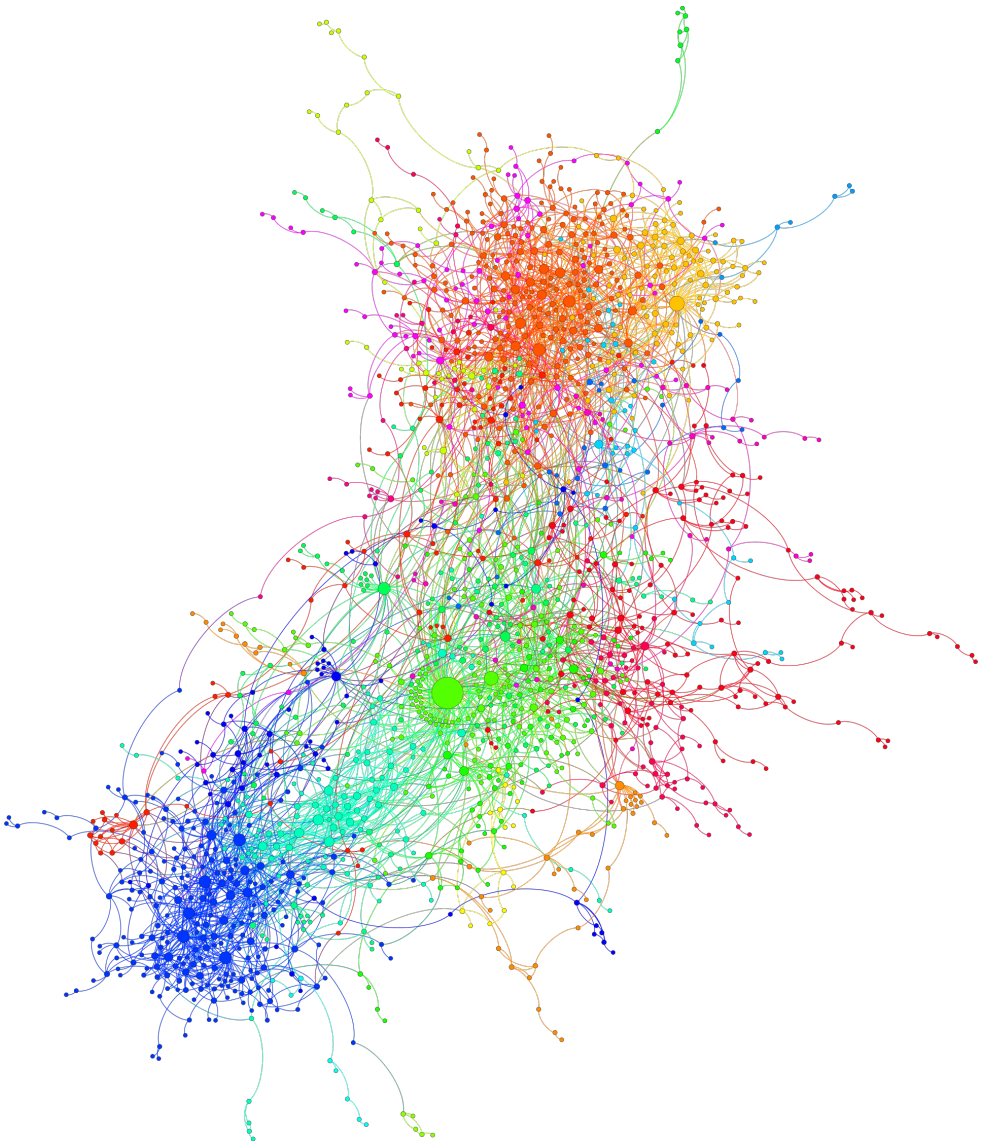

FIGURE 1 – Graphe des auteurs de tweets en France, sur une période de 2h40 le 22/07/2012

hadrien.croubois@ens-lyon.fr

Table des matières

 hadrien.croubois@ens-lyon.fr

Partie 1

Introduction

Twitter[1] est un réseau social de microblogage crée en 2006 par Jack Dorsey. Au fil des années les courts messages de 140 caractères que sont les tweets[2] on su séduire plus de 500 millions d'utilisateurs à travers le monde, parmi lesquels de nombreuses personnalités.

Un mot précédé du signe « # » est appelé *hashtag*. Les hashtags ont pour rôle de décrire le sujet du message. Les utilisateurs ayant la possibilité d'afficher tous les tweets comportant un même hashtag, ils permettent d'identifier de véritables flux de donnés autour d'un seul et même sujet.

Outre le fait d'envoyer des messages au monde entier, les utilisateurs ont la possibilité de suivre leurs amis, répondre – *reply* – à leur message, ou re-poster – *retweet* – ces derniers afin d'augmenter leurs visibilité. Ce sont ces interactions entre utilisateurs qui permettent de voir l'émergence de communautés au sein des utilisateurs de Twitter.

Très vite l'ampleur du réseau et son utilisation privilégiée dans la diffusion d'informations a suscité de nombreuses interrogations sur la topologie d'un tel réseau – réseau vivant au même titre d'internet – et sur les processus de diffusion de l'information, interrogations renforcées par l'utilisation de Twitter comme moyen privilégié de communication entre manifestants lors des révolutions arabes.

Partie 2

Travaux antérieurs

2.1

Étude du réseau et des échanges

Bon nombre des précédentes études concernant le réseau Twitter s'intéressaient à la structure du réseau. Dès 2007, on commence à étudier les caractéristiques topologiques de ce réseau[FT07]. Les multiples fonctionnalités offertes par Twitter (suivie, reponse, retweet) impliquent différentes relations d'adjacences et de distances qu'il faut mettre en relation avec la portée de diffusion des informations, notamment en terme de retweet[FP12].

De nombreuses personnalités publiques possédant un compte Twitter, aussi bien dans les médias que dans le spectacle ou la politique, ont été à l'origine d'étude sur la nature des échanges entre le grand public et ces différentes communautés[WHMW11], ainsi que sur la quantification de l'influence des auteurs de tweets[BHWM]

2.2

Évolution du réseau liés aux conversations

Lors de l'étude de la dynamique des graphes, la comparaison entre l'augmentation du nombre d'amis ou la fusion de communautés et le flux d'informations par réponse/retweet, la communauté était le plus souvent isolée par l'emploi d'une hashtag. « Conversation Practices and Network Structure in Twitter »[RM12] étudie ainsi l'évolution des liens et des échanges entre les téléspectateurs d'une même émission de télé-réalité. Un tel isolement de la communauté ne permet cependant pas d'étudier les dynamiques de diffusion des hashtags.

Partie 3

Mon travail

3.1

Objectifs

Contrairement aux études précédentes qui délimitaient l'univers d'étude par l'emploi d'un hashtag, l'idée était de pouvoir isoler les hastags employés et ainsi étudier leur diffusion et les éventuelles similitudes entre diffusion des hashtags et fusion de communautés.

L'univers d'étude (la sous-partie du réseau étudier) se devait d'être paramétrable et un graphe dynamique devait être généré à partir du flux de tweets et d'informations complémentaires obtenues en temps réel.

Dès le début j'ai décidé de laisser l'étude du graphe en tant que telle (spatialisation, calcul de grandeurs caractéristiques...) à *gephi*[3], un logiciel libre reconnu aussi bien pour sa facilité d'utilisation que pour ses performances. Ainsi mon rôle était donc d'ouvrir des connexions vers différentes API afin d'obtenir un graphe exportable sous gephi.

1. Twitter. `http://twitter.com/`
2. « gazouillis » en français
3. Gephi, an open source graph visualization software. `http://gephi.org/`

hadrien.croubois@ens-lyon.fr

GET `https://api.twitter.com/1/friends/ids.json?cursor=-1&screen_name=twitterapi`


```
{
    "previous_cursor": 0,
    "ids": [
        143206502,
        143201767,
        777925
    ],
    "previous_cursor_str": "0",
    "next_cursor": 0,
    "next_cursor_str": "0"
}
```


FIGURE 2 – Résultat de requête faite à l'API twitter

3.1.1 Flux de données

Les données brutes sont obtenues via différents fournisseurs.

Les tweets sont obtenus en temps réel via *datasift*[4], un site proposant récupérer tweets et autres messages sur différents réseaux sociaux selon des filtres établis par l'utilisateur. Ces données sont payantes, mais un quota de 10 $ est donné à l'ouverture d'un compte.

Une fois le filtre paramétré un flux est disponible qui permet de récupérer, au format *JSON*[5], les derniers tweets répondant aux critères du filtre. Dans les faits, le filtre est décrit au moyen d'une syntaxe appelée *CSDL*[6], description qui est envoyée à datasift pour être transformé en signature de filtre (aussi appelé *Hash*). Dès lors, tout utilisateur peut récupérer les derniers messages correspondant au filtre en passant en paramètre ses identifiants et la signature du filtre. À cela, l'utilisateur peut aussi ajouter l'identifiant du dernier tweet reçu afin de n'obtenir que les nouveaux tweets.

En plus du tweet à proprement parler, datasift fournit bon nombre d'informations complémentaires telle que des renseignements sur l'auteur ou des données géographiques.

La figure 3 (page 5) donne un exemple type de tweet obtenu via datasift tel que décrit dans la documentation en ligne de l'API[7]s

D'autres informations complémentaires concernant la topologie du réseau sont disponibles via l'API Twitter[8]. De toutes les données accessibles, celles qui nous intéressent sont les requêtes *GET followers/ids* et *GET friends/ids* qui permettent d'obtenir la liste des identifiants des followers (personnes qui suivent) et des amis (personnes suivies) au format JSON. La figure 2 donne un exemple de données obtenues

La liste reçue contient jusqu'à 5000 entrées, si tous les résultats n'ont pas été renvoyés, un curseur non nul sera envoyé. Ce curseur permet, via une nouvelle requête, de récupérer les éléments suivants. Ainsi en répétant les requêtes tant que le curseur est différent de 0 on peut récupérer l'intégralité des résultats.

L'API twitter a cependant un défaut, le nombre de requêtes est limité à 150 par heure et par IP pour les connexions non identifiés. Ce quota peut être repoussé à 350 par heure et par compte pour les requêtes identifiées. La solution pour obtenir un grand nombre de données et donc d'utiliser plusieurs comptes en bouclant, une erreur étant renvoyée si le quota de requête est dépassé.

L'identification se fait via le protocole OAuth[9] utilisé notamment par les API gmail et facebook, ce qui nécessite l'utilisation d'algorithme de cryptage[10].

3.1.2 Différents liens de parenté

Afin de bien définir la notion de communauté désirée, il est important de caractériser les liens de parentés (arêtes du graphe) entre auteurs de tweets (les nœuds du graphe). De nombreuses caractérisations des liens de parenté peuvent être utilisées et leur comparaison a fait l'objet de nombreuses études.

Afin d'avoir le graphe le plus complet possible, j'ai préféré considérer comme témoin d'un lien de parenté tous les échanges ou possibilités d'échange d'information entre auteurs de tweets. On pourra ainsi produire des arêtes dirigées

4. Datasift, data on demand. `http://datasift.com/`
5. JavaScript Object Notation : `http://www.json.org/`
6. Curated Stream Definition Language. `http://dev.datasift.com/csdl`
7. `http://dev.datasift.com/docs/targets/twitter/tweet-output-format`
8. Twitter Developpers. `https://dev.twitter.com/`
9. OAuth Community Site. `http://oauth.net/`
10. SHA-256. `http://fr.wikipedia.org/wiki/SHA-256`

hadrien.croubois@ens-lyon.fr

```
{
    "interaction": {
        "source": "TweetDeck",
        "author": {
            "username": "stewarttownsend",
            "name": "Stewart Townsend",
            "id": 14065694,
            "avatar": "http://a2.twimg.com/profile_images/1302306721/twitterpic_normal.jpg",
            "link": "http://twitter.com/stewarttownsend"
        },
        "type": "twitter",
        "link": "http://twitter.com/stewarttownsend/statuses/136447843652214784",
        "created_at": "Tue, 15 Nov 2011 14:17:55 +0000",
        "content": "Morning San Francisco - 36 hours and counting.. #datasift",
        "id": "1e10f949c51aab80e074df944f5e8e46"
    },
    "twitter": {
        "user": {
            "name": "Stewart Townsend",
            "url": "http://www.stewarttownsend.com",
            "description": "Developer Relations at Datasift (www.datasift.com)  - Car racing
                petrol head, all things social lover, co-founder of www.flowerytweetup.com",
            "location": "iPhone: 53.852402,-2.220047",
            "statuses_count": 28247,
            "followers_count": 3094,
            "friends_count": 510,
            "screen_name": "stewarttownsend",
            "lang": "en",
            "time_zone": "London",
            "listed_count": 221,
            "id": 14065694,
            "id_str": "14065694",
            "geo_enabled": true
        },
        "id": "136447843652214784",
        "text": "Morning San Francisco - 36 hours and counting.. #datasift",
        "source": "<a href=\"http://www.tweetdeck.com\" rel=\"nofollow\">TweetDeck</a>",
        "created_at": "Tue, 15 Nov 2011 14:17:55 +0000"
    },
    "demographic": {
        "gender": "male"
    },
    "language": {
        "tag": "en"
    },
    "salience": {
        "content": {
            "sentiment": 3
        }
    }
}
```


FIGURE 3 – Exemple type de tweet obtenu via datasift

hadrien.croubois@ens-lyon.fr

vers l'origine de l'information

- *Message à un membre :* arête de l'auteur de la réponse vers sont interlocuteur, car la réponse – même si elle n'est pas lue – témoigne d'un flux d'information en premier lieu. En effet, il arrive que certaines personnes répondent à une publication sans que la réponse ne soit lue, c'est par exemple le cas de tout message commentant un tweet émis par une personnalité publique. Dans ce dernier cas, la réponse n'est que représentative de la réception du premier message.
- *Retweet d'un message :* arête du retweeteur vers le retweeté
- *Relation Followers/Friends :* arête du suivant vers la personne suivie

Ce sont ces trois différents liens de parenté entre auteurs de tweets qui caractérisent les communautés étudiées.

3.1.3 Dynamique en temps réel

Afin d'étudier l'évolution des communautés, il est nécessaire de donner un caractère dynamique au graphe. Celui-ci est généralement obtenu en générant le graphe à plusieurs instants distincts. Cette méthode est cependant limitée par la quantification du temps entre plusieurs mise à jour du graphe. Afin de refléter plus efficacement la réalité de l'évolution continue du graphe, mon approche a été différente.

Les nœuds du graphe, c'est-à-dire les auteurs de tweet, apparaissent quand ils écrivent un tweet pour la première fois. L'API datasift, qui nous fournit ces tweets, nous donne par ailleurs une date/heure de publication que nous pouvons utiliser comme repère temporel.

De la même manière, les arêtes entre nœuds préexistants peuvent être liées à la publication d'une réponse ou d'un retweet. Dans ce cas là, la date de publication de la réponse ou du retweet, fournie par datasift, peut être utilisée comme date d'apparition de l'arête.

Reste les arêtes liées à des liens followers/friends. Dans la mesure où ce sont principalement ces arêtes qui définissent les communautés et où il n'existe pas de flux indiquant l'évolution de ces données, il faut régulièrement réactualiser, l'ensemble de ces données en interrogeant, l'API twitter pour l'ensemble des nœuds de notre graphe. C'est cette discrétisation du temps qui est la principale limite à l'exploitation d'un graphe à la dynamique entièrement continue.

3.1.4 Hashtags

Contrairement aux études précédentes qui suivaient l'évolution d'une communauté autour d'un unique hashtag, l'objectif est ici d'observer l'utilisation et la diffusion de hashtags au sein du réseau de tweeteurs.

Un hashtag peut alors être considéré comme une hyper-arête dans le graphe des auteurs de tweets, reliant tous ceux qui en ont fait l'emploi dans leurs publications. Malheureusement, gephi ne gère pas les hyper-arêtes dans sont format actuel (*GEXF 1.2draft*[WM12]).

La solution employée est donc d'utiliser le format *liststring*. Ce format permet d'ajouter à chaque noeud la liste des hashtags employés au cours de l'acquisition des données. Par ailleurs le format GEXF 1.2draft actuellement utilisé par gephi demande que l'intégralité des hashtags utilisés soit décrite au début du fichier.

Il est important de noter que le format liststring décrit dans les spécifications du format GEXF 1.2draft n'est pas encore exploitable par gephi. La rétro-compatibilité des futures versions devrait cependant assurer que toutes les données acquises dès aujourd'hui seront exploitables quand les algorithmes disponibles dans gephi auront été rendus compatibles avec l'utilisation d'un tel format de données.

Les hashtags n'étant par ailleurs pas fournis directement par datasift il est nécessaire de les parser manuellement dès la réception des tweets.

3.2 Présentation du logiciel

Le programme réalisé au cours de ce stage, nommé *QtTwitGraph*, présente trois onglets (figure 4 page 4).

- Le premier onglet contient les réglages généraux du programme tels que la durée d'acquisition, les fréquences de rafraichissement des noeuds et des arêtes, les paramètres du flux de tweet ou l'emplacement où enregistrer le fichier produit.
- Le second onglet contient les identifiants des différents comptes datasifts utilisés pour obtenir les tweets en temps réel.
- Enfin, le troisième onglet contient les identifiants des comptes twitter utilisés pour obtenir les données des followers/friends. Lors de l'ajout d'un nouveau compte, QtTwitGraph se charge d'ouvrir le navigateur par défaut afin d'autoriser l'accès au compte twitter (figure 6 page 9). Le code (*pin*) renvoyé par twitter permet ensuite à l'application de récupérer les identifiants uniques (figure 7 page 9).

Le lancement de l'acquisition des données provoque l'ouverture d'une nouvelle fenêtre qui affiche plusieurs informations quant à l'état du programme (figure 5 page 7). On peut notamment suivre le nombre de tweets obtenus, le nombre d'auteurs de tweets distincts et le nombre d'arêtes entre eux.

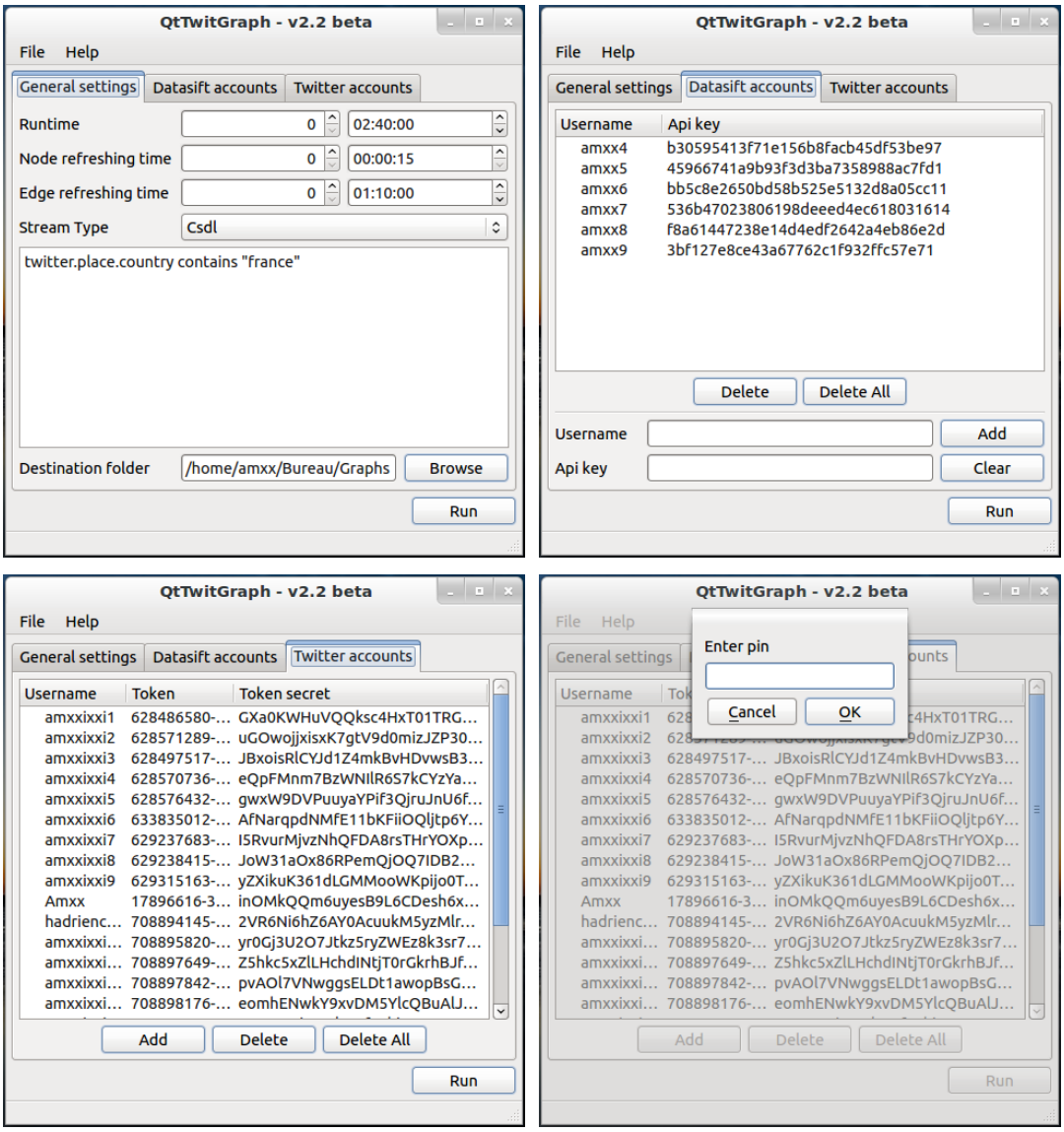

FIGURE 4 – Vue générale de QtTwitGraph

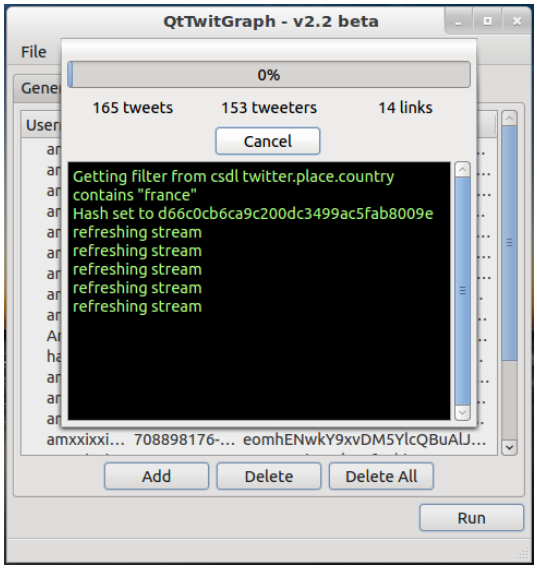

FIGURE 5 – Fenêtre d'exécution de QtTwitGraph

hadrien.croubois@ens-lyon.fr

Lors de la fermeture du programme (ou à la demande de l'utilisateur soit via le menu soit via le raccourci clavier *Ctrl+S*) tous les paramètres et identifiants de comptes sont enregistrés.

L'annulation de l'acquisition produira un fichier contenant le graphe des données acquises pendant tout le temps déjà écoulé.

3.3 Exportation vers Gephi

L'exportation du graphe vers gephi se fait en utilisant le format GEXF 1.2draft[WM12] tel que décrit par gephi [11]. Ce format, qui évolue avec gephi, permet de décrire simplement des graphes complexes.

Ce format utilise la structure XML, ce qui en simplifie largement la lecture et l'écriture. De plus, il est important de noter que ce format est supporté par de nombreuses librairies dans des langages aussi variés que C++, R, Java, Python, Perl et JavaScript.

La figure 8 (page 10) retrace la structure d'un fichier produit. On notera trois parties :

- La description des attributs personnalisés (et la liste des hashtags utilisés pour l'attribut « hashatags »),
- La liste de nœuds, chacun décrit avec l'ensemble de ses attributs
- La liste des arêtes

3.4 Choix d'implementation

3.4.1 Introduction a Qt

Un framework multiplateforme C++ *Qt* [12] est un framework orienté objet écrit en C++. Utilisé par de très nombreux développeurs et par de grandes entreprises comme Boeing, Google, Adobe et même la Nasa, il permet de développer rapidement des applications graphiques multithreadées.

Le choix de Qt a été pour moi une évidence aussi bien pour sa simplicité d'utilisation que pour ses nombreux modules et son caractère multiplateforme. Les nombreuses possibilités offertes par Qt donnent par ailleurs de nombreuses possibilités d'évolution tel que la traduction facile du programme.

Timer & Threads Parmi les nombreux ajouts de Qt au C++, les facilités de multithreading et la présence de signaux sont probablement celles qui m'ont été le plus utiles.

Les objets de type *QObject* et *QWidget* peuvent en effet émettre des signaux (*signals*), lesquels peuvent déclencher l'exécution de méthodes (*slots*). Le multithreading inhérent à Qt permet ainsi de lancer l'exécution de plusieurs objets gérant les différentes fonctionnalités du programme (connexion asynchrone, gestion du flux, récupération des données complémentaires...) et permet à tous ces objets de communiquer entre eux.

Des *Timers* sont là pour émettre des signaux à intervalle régulier et ainsi demander la mise à jour régulière des données.

3.4.2 Utilisation des Api

Les différentes API utilisées demandent des identifiants afin d'authentification les requêtes qui leur sont faites. Ces identifiants, obtenus par la fenêtre standard (onglets 2 & 3, figure 4 page 7) sont enregistrés dans des structures de données ad hoc. Disposant d'un caractère booléen indiquant si les identifiants sont réutilisables (comme dans le cas de l'API Twitter où les comptes ont un quota par heure, mais sont réutilisables au bout d'un certain temps), ces structures contiennent des méthodes fournissant les identifiants du compte à utiliser ou gérant l'expiration des comptes (quotas atteints).

Ces structures de données ont par ailleurs la possibilité d'envoyer des signaux à la fenêtre principale pour provoquer l'effacement des comptes non réutilisables expirés (compte datasift vide)

Api Datasift La récupération des données datasift se fait via une méthode appelée régulièrement par un timer. L'appel de cette dernière provoque, si aucune demande de mise à jour n'est en cours, une requête asynchrone des nouveaux tweets relatifs au filtre.

Si le compte datasift utilisé est vide, un message d'erreur est renvoyé par datasift qui déclenche le passage au compte suivant et le renouvellement de la requête. Le compte est supprimé des comptes utilisés et un message est affiché parmi les informations d'exécution.

Une fois la requête terminée, les tweets sont parcourus. Pour chaque tweet un descripteur de l'émetteur (noeud du graphe) est envoyé à la structure gérant les noeuds. Si ce noeud est nouveau, la structure est juste référencée, sinon

11. GEXF File Format. `http://gexf.net/format/`
12. prononcer comme l'Anglais *cute* (/kjut/))

FIGURE 6 – Autorisation d'accès à l'API Twitter - OAuth

FIGURE 7 – PIN de confirmation d'autorisation - OAuth

hadrien.croubois@ens-lyon.fr

```
<?xml version="1.0" encoding="UTF-8"?>
<gexf version="1.2" xmlns="http://www.gexf.net/1.2draft">
    <meta lastmodifieddate="2012-07-22">
        <creator>H.Croubois</creator>
        <description>Produced using QtTwitGraph</description>
    </meta>
    <graph timeformat="date" mode="dynamic" defaultedgetype="directed">
        <attributes class="node">
            <attribute title="Name" type="string" id="name"/>
            <attribute title="Screen Name" type="string" id="screen_name"/>
            <attribute title="Tweet Number" type="int" id="tweet_number"/>
            <attribute title="Hashtags" type="liststring" id="hashtags">
                <options>#summertime|#Gainsbourg|#LR|#VieillesCharrues|#LT|(...)</options>
            </attribute>
        </attributes>
        <nodes>
            <node id="758033" start="2012-07-22T13:45:49" label="martinlittle">
                <attvalues>
                    <attvalue for="name" value="Martin Little"/>
                    <attvalue for="screen_name" value="martinlittle"/>
                    <attvalue for="tweet_number" value="1"/>
                    <attvalue for="hashtags" value="#Paris|#tourdefrance"/>
                </attvalues>
            </node>
            <node id="2763841" start="2012-07-22T13:37:13" label="pjournel">
                <attvalues>
                    <attvalue for="name" value="Pierre Journel"/>
                    <attvalue for="screen_name" value="pjournel"/>
                    <attvalue for="tweet_number" value="4"/>
                    <attvalue for="hashtags" value=""/>
                </attvalues>
            </node>
        (...)
        </nodes>
        <edges>
            <edge target="610993" id="0" source="50613" start="2012-07-22T14:27:16"/>
            <edge target="5086231" id="1" source="50613" start="2012-07-22T14:27:17"/>
            <edge target="6574682" id="2" source="50613" start="2012-07-22T14:27:18"/>
            <edge target="10461152" id="3" source="50613" start="2012-07-22T14:27:22"/>
            <edge target="14400518" id="4" source="50613" start="2012-07-22T14:27:26"/>
            <edge target="14630211" id="5" source="50613" start="2012-07-22T14:08:31"/>
            <edge target="15277602" id="6" source="50613" start="2012-07-22T14:27:30"/>
        (...)
        </edges>
    </graph>
</gexf>
```

FIGURE 8 – Structure des données exportées vers gephi

les données relatives à ce tweet (hashtags) sont ajoutées au descriptif de l'émetteur et le nombre de tweets écrits est incrémenté avant de supprimer la structure inutile.

Dans le cas où cet auteur est nouveau, une requête est par ailleurs ajoutée à la liste d'attente du gestionnaire de l'API Twitter (voir ci-dessous) afin de générer la liste des arêtes relatives à ce noeud.

Si le tweet est un retweet ou une réponse, une structure d'arête est envoyée au graphe qui l'enregistre si elle est nouvelle et la supprime dans le cas contraire.

Toutes les communications entre structures de données se font par l'intermédiaire de signaux.

Le filtre datasift est paramétré avant la première requête soit directement via le hash du flux, soit via une requête à l'API datasift pour obtenir un hash à partir de la description CSDL (voir la figure 5).

Api Twitter La récupération de données via l'API Twitter demande une identification OAuth. L'obtention des identifiants uniques grâce à la méthode « Three-Legged » a déjà été traitée lors de l'ajout de compte (cf. 3.2 Présentation du logiciel).

Une méthode se charge dès lors d'utiliser le compte twitter courant pour obtenir la liste de ses amis et transmettre des arêtes au gestionnaire de graphe qui les enregistrera les nouvelles arêtes si elles sont relatives à deux nœuds du graphe (c.-à-d. si le membre suivi est un nœud du graphe déjà identifié). Si une requête ne suffit pas (rappelons que les réponses sont limitées à 5000 identifiants), une nouvelle requête sera produite avec le nouveau curseur.

Si la requête n'a pas pu être exécutée, car le compte utilisé atteint son quota limite, un message d'erreur est retourné ce qui déclenche le passage au compte suivant si le compte courant est le même que celui identifié comme expiré. La requête est ré-exécutée avec les nouveaux identifiants.

Cependant, le nombre de requêtes émises est potentiellement très important. En effet les requêtes étant asynchrones, la ré-actualisation du graphe implique l'émission d'une requête par noeud. Ces requêtes, pouvant dépasser le millier, pose problème, les structures de connexion et les threads qui les accompagnent étant une charge importante pour la machine ce qui peut faire planter le framework.

Les requêtes sont donc empilées dans une liste qui permet de gérer le nombre de thread exécutant ces requêtes. Ces theads, dont le nombre est défini en fonction du nombre de cœurs disponibles, gèrent chacun une requête et attendent la fin de celle-ci avant de transmettre le résultat au graph et d'exécuter la requête suivante.

Cela évite l'augmentation inconsidérée de l'utilisation de la mémoire, qui pourrait mener à des fuites.

À la fin du programme, il peut y avoir un temps de latence avant l'écriture du graphe, temps correspondant à l'exécution des dernières requêtes. Cependant, une fois l'exécution du programme terminée (temps d'acquisition terminé) ou annulée, aucune requête ne pourras être ajoutée à la liste d'attente, qu'il s'agisse de re-exécuter une requête transmise avec les identifiants d'un compte expiré ou d'obtenir la fin d'une requête partielle.

3.4.3 Bibliothèques utilisées

En plus des nombreux modules que contient Qt, j'ai eu besoin de certaines librairies pour prendre en charge des fonctionnalités non gérées.

QJson [13] est une librairie qui gère le format d'échange JSON. Elle contient ainsi un parseur et un sérialiseur permettant la conversion d'objets JSON vers des objets de type QVariant (QVariantMap & QVariantList)

QOAuth [14] est une librairie qui permet, sous Qt, de gérer les communications utilisant le protocole OAuth. Faisant appel à la librairie QCA (Qt Cryptographic Architecture), elle implémente les procédures d'obtention des jetons de connexion ainsi que la signature des requêtes authentifiées

3.4.4 Structures de données

Les structures de données gérant le graphe reprennent la structure des fichiers GEXF, lesquels contiennent la liste des nœuds et la liste des arêtes.

Sont donc utilisés comme structure contenant les nœuds et les arêtes des objets de type « QMap » comparable à des tables de hachage dont les clefs sont respectivement les identifiants uniques des nœuds et les paires source/destination des arêtes. Les entrées des tables en questions sont, quant à elle, des pointeurs vers des structures ad hoc contenant tous les attributs des nœuds/arêtes en question.

Ainsi il est simple de vérifier l'existence d'un nœud ou d'une arête en cherchant la clef correspondante dans la table de hachage. La modification se fait simplement en intervenant sur les attributs de l'objet pointé.

13. QJson. `http://qjson.sourceforge.net`
14. QOAuth. `https://github.com/ayoy/qoauth`

3.5 Difficultés rencontrées

Au cours de mon stage, j'ai dû faire face à de nombreuses difficultés.

Une première version de mon programme, développé sans l'utilisation du framework Qt (décrit dans la section 3.4.1, page 8), m'a permis d'établir les principales structures de données, mais l'utilisation de librairies de multithreading bas niveau (comparé a celles offertes par Qt) a rendu extrêmement compliqué la gestion des interactions entre différents gestionnaires d'API ainsi que la gestion des messages d'erreur.

Ces difficultés ainsi que la volonté de produire une véritable interface graphique m'ont poussé à utiliser la framework Qt. Cependant, la première version produite souffrait de montreuses fuites de mémoire liées à une mauvaise hiérarchisation des objets graphiques (dont l'impact mémoire est largement plus important que les structures gérant les APIs et le graphe). Ce problème a été réglé grâce à l'utilisation du créateur de fenêtre présent dans QtCreator et par l'implémentation des structures de calcul comme membre des fenêtres correspondantes.

On notera cependant que Qt souffre intrinsèquement de faibles fuites de mémoire au niveau de l'architecture principale de l'application. Ces fuites étant très contenues et n'évoluant pas au cours du temps, elles ne posent cependant pas de véritable problème.

L'API Twitter m'a aussi posé beaucoup de problèmes pour ce qui est de la procédure d'authentification. N'étant pas habitué à la gestion des en-têtes de paquets et à la signature des données, l'absence de documentation détaillée pour la librairie m'a obligé à lire de nombreux codes sources souvent non commentés. Il est important d'ajouter que l'API Twitter fonctionne de telle manière que toute requête authentifiée avec des identifiants non valides est exécutée comme une requête non authentifiée sans qu'aucun message n'indique de la mauvaise identification. Il m'a ainsi été très difficile de comprendre que j'utilisais le nom d'utilisateur à la place du mot de passe, les réponses me faisant longtemps penser que la procédure d'authentification utilisée était incorrecte.

Enfin s'est posé le problème du grand nombre de threads ouverts lors de la mise à jour des arêtes. L'inexistence de requêtes synchrones sous Qt peut en effet se révéler fourbe, car la séparation de l'émission de la requête du traitement de la réponse oblige à l'utilisation d'un sémaphore comptabilisant les ressources disponibles. Or, le multithreading de Qt étant partiellement virtuel, les sémaphores peuvent parfois bloquer plusieurs threads. Cela m'a fait comprendre que je ne pouvais pas me fier au multithreading implémenté par Qt dès lors que j'intervenais personnellement sur l'exécution de ces threads. La solution est venue avec le type QTreadPool qui s'est avéré une excellente réponse dès que j'avais réussi à rendre mes requêtes synchrones et à les implémenter au sein d'objets héritant de la propriété QRunnable. La création des requêtes synchrones a alors demandé que le lancement de la requête et l'envoi de l'objet répondu au graphe soient faits à la suite. Pour cela, il m'a suffi d'insérer entre les deux étapes une boucle vide stoppée à la réception du signal de changement de statut de la requête.

Partie 4 **Conclusion**

4.1 Résultats

L'exploitation des graphes obtenus ne peut se faire que dans la cadre d'une étude sociologique sur les comportements de tweeteurs. Cela demande, en plus de connaissance en sociologie que je n'ai pas, des acquisitions de longue durée avec pour critère des filtres habilement choisis.

Par ailleurs, et tant que gephi ne gérera pas les attributs de type *liststring* tel que décrit dans la spécification du format GEXF 1.2draft, l'étude des listings de hashtags est extrêmement difficile.

Les courtes acquisitions effectuées ont cependant confirmé la fonctionnalité du programme et l'utilisation très limitée des performances aussi bien en terme d'utilisation CPU que mémoire. Lors de l'acquisition de 3h de données, l'espace utilisé a en effet toujours été contenu à moins de 20 Mo.

La dynamique des graphes obtenus est par ailleurs parfaitement visible sous gephi.

La figure 9 (page 13) présente différentes visualisations de la plus grosse composante connexe obtenue après 2h40 d'acquisition. Pour les 3 premières vues, la taille des nœuds correspond à leurs degrés respectifs tandis que la couleur des nœuds représente différents degrés de centralité et de modularité calculés par Gephi (du bleu foncé au violet en passant pas le bleu clair, le vert, le jaune, l'orange et le rouge). Pour la dernière vue, couleur et taille des nœuds reflètent le nombre de tweets émis durant la durée d'acquisition.

hadrien.croubois@ens-lyon.fr

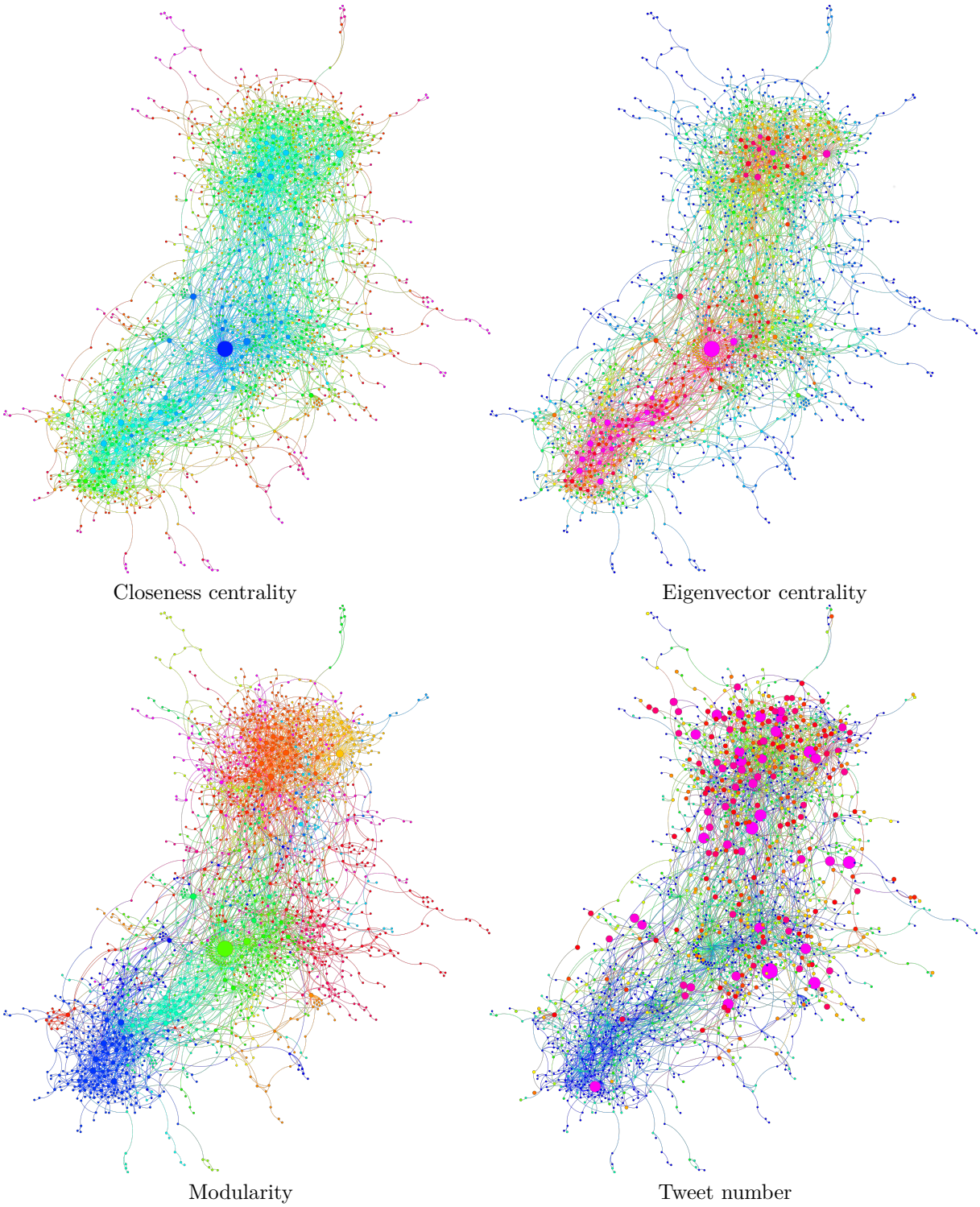

FIGURE 9 – Différentes visualisation du graphe des auteurs de tweets en France, sur une période de 2h40 le 22/07/2012

hadrien.croubois@ens-lyon.fr

4.2 Possibilités d'évolutions

4.2.1 Durée de vie

La dynamique actuelle donne une date d'apparition aux nœuds et aux arêtes, mais pas de date de disparition. Il serait potentiellement utile de donner une durée de vie aux nœuds, et à la remettre à jour à chaque manifestation du nœud de telle manière à ce qu'elle soit rallongée si le nœud est vivant et à rajouter une nouvelle copie du nœud avec une nouvelle date d'apparition.

Les difficultés à l'implémentation d'une telle fonctionnalité sont cependant nombreuses :

- Les arêtes sont identifiées par des couples (source, destination) d'identifiants uniques aux nœuds. Or deux apparitions du même nœud devraient avoir le même identifiant unique ce qui n'est pas gérable par gephi,
- La durée de vie n'aurait de véritable intérêt que si elle est calculée à partir de la centralité du nœud, ce qui oblige à calculer des statistiques sur le graphe lors de l'acquisition et à les mettre à jour en temps réel ce qui risque, de se révéler être un gouffre de performance système, surtout si la donnée ne peut pas être calculée de manière dynamique et qu'elle doit être recalculée pour tout le graphe à chaque instant.

4.2.2 Implémentation des hashtags sous forme d'hyper-arêtes

La gestion des hashtags comme attribut des nœuds ne permet pas de leur donner un caractère dynamique. Cela empêche d'étudier de manière dynamique la diffusion des hashtags et ainsi de suivre leur utilisation au cours du temps pour identifier les nœuds les plus influents.

L'une des solutions serait d'implémenter les hashtags comme des hyper-arêtes en donnant un attribut de temps (correspondant à la première utilisant d'un hashtag) à l'appartenance du nœud à l'hyperarête correspondante. Cette solution complète ne sera cependant possible que quand les hypergraphes seront gérés par le format GEXF.

4.2.3 Gestion des données

En l'état actuel, toutes les données sont enregistrées en mémoire vive pendant l'exécution du programme, l'écriture du fichier ne se faisant qu'après l'acquisition. Cela implique que toute erreur critique provoquant l'arrêt anticipé du programme, tel qu'une rupture de connexion ou l'expiration de tous les comptes datasift, provoquera la perte des données.

Une solution permettant de réduire les contraintes en mémoires vives serait d'enregistrer les données constituant le graphe dans une base de données.

Outre le travail supplémentaire d'implémentation des échanges de données, cela implique un temps d'accès aux données beaucoup plus important qu'il ne l'est à l'heure actuelle. Bien que la sécurité liée à la possibilité d'accès aux données temporaires même en cas d'interruption prématurée soit indiscutable, la réduction de l'utilisation mémoire peut être relativisée par le fait que l'occupation de l'espace mémoire n'est pas le premier obstacle à l'étude de très gros graphe, les limitations intrinsèques à Gephi étant atteintes bien avant.

4.2.4 Autres évolutions

De nombreuses autres évolutions sont possibles, notamment la traduction du programme (dont l'interface a volontairement été écrite en anglais afin de le rendre utilisable par le plus grand nombre) grâce aux outils de traductions offerts par le framework Qt.

Par ailleurs, le framework Qt étant multiplateforme, il serait intéressant de produire un exécutable Windows. Cela demande tout de même un certain travail, les librairies utilisées n'étant pas toujours disponibles pour Windows. Il faut donc les compiler et importer les *.dll* correspondant dans le projet. Ne connaissant pas bien les procédures d'utilisation des extensions Windows .dll et n'aillant pas de machine pour tester le résultat je ne me suis donc par aventuré dans les processus de compilation croisés hasardeux. Le programme étant par ailleurs plutôt destiné à de longues acquisitions de donnée, j'ai pensé qu'il serrait plus susceptible d'être exécuté sur un serveur fonctionnant sous un système Linux ou Unix.

4.3 Bilan

Je pense avoir beaucoup appris au cours de ce stage et cela aussi bien sur le plan scolaire que personnel.

Ces 6 semaines ont pour moi été l'occasion de monter un projet de développement comme j'en avais rarement eu l'occasion, m'obligeant notamment chercher et apprendre à exploiter de nouveaux outils. J'ai appris à utiliser un framework complet alors que j'avais l'habitude d'applications plus simples n'utilisant que les librairies standards C++. J'ai aussi appris à utiliser différentes API et formats d'échanges afin de rendre mon programme compatible avec d'autres outils.

J'ai par ailleurs eu, à de nombreuses reprises, l'occasion de remettre en cause mes choix, aussi bien d'implémentation que de conception.

Par ailleurs, ce projet a pour moi été l'occasion de mettre en application ma connaissance des multitreading pour la première fois à cette échelle.

Enfin, l'accueil qui m'a été fait à l'IXXI m'a permis de découvrir le quotidien d'un laboratoire de recherche et de me faire une meilleure idée du rôle d'un individu au sein d'une équipe de recherche.

Partie 5

Remerciements

Malgré la courte durée de mon stage et les quelques problèmes logistiques rencontrés, j'ai eu l'opportunité de me faire une idée de ce à quoi ressemble le quotidien d'un chercheur.

Je tenais à remercier tous les chercheurs et administratifs de l'IXXI pour leur accueil.

Je tenais aussi à remercier Mr Bertrand Jouve de m'avoir encadré durant ce stage, et ce malgré ses très nombreuses obligations professionnelles.

Enfin, je tenais à remercier Mr Eric Fleury et Mr Eric Thierry pour leur précieuse aide et leurs conseils éclairés.

hadrien.croubois@ens-lyon.fr

Références

[AlAd12] Enas M Al-lozi and Mutaz M Al-debei. A Framework of Value Exchange and Role Playing in Web 2.0 WebSites. 2012 :1–13, 2012.

[BHWM] Eytan Bakshy, Jake M Hofman, Duncan J Watts, and Winter A Mason. Everyone ' s an Influencer : Quantifying Influence on Twitter Categories and Subject Descriptors.

[FP12] Jorge Fabrega and Pablo Paredes. Three Degrees of Distance on Twitter. 2012.

[FT07] Tim Finin and Belle Tseng. Why We Twitter : Understanding Microblogging. 2007.

[GSG12] Saptarshi Ghosh, Ajitesh Srivastava, and Niloy Ganguly. Effects of a soft cut-off on node-degree in the Twitter social network. *Computer Communications*, 35(7) :784–795, April 2012.

[KsGN11] Funda Kivran-swaine, Priya Govindan, and Mor Naaman. The Impact of Network Structure on Breaking Ties in Online Social Networks : Unfollowing on Twitter. pages 1101–1104, 2011.

[LWCc12] Rui Li, Shengjie Wang, and Kevin Chen-chuan. Multiple Location Profiling for Users and Relationships * from Social Network and Content. 5(11) :1603–1614, 2012.

[RM12] Luca Rossi and Matteo Magnani. Conversation Practices and Network Structure in Twitter *. In *International AAAI Conference on Weblogs and Social Media*, 2012.

[VKH] Duy Q Vu, Shashank Khandelwal, and David R Hunter. The Dynamics of Health Behavior Sentiments on a Large Online Social Network. (814) :1–20.

[WHMW11] Shaomei Wu, Jake M. Hofman, Winter a. Mason, and Duncan J. Watts. Who says what to whom on twitter. *Proceedings of the 20th international conference on World wide web - WWW '11*, page 705, 2011.

[WM12] Gexf Working and Group March. GEXF 1.2draft Primer. pages 1–25, 2012.

[ZCPB] Quanyan Zhu, Andrew Clark, Radha Poovendran, and Tamer Bas. SODEXO : A System Framework for Deployment and Exploitation of Deceptive Honeybots in Social Networks.

Table des figures

hadrien.croubois@ens-lyon.fr