UNIVERSITY OF GOTHENBURG

Conference Networking Recommendations based on Past Online Activity

Bachelor of Science Thesis in the Software Engineering and Management Programme

Shan Huang
Petre Mihail Anton

University of Gothenburg
Chalmers University of Technology
Department of Computer Science and Engineering
Gothenburg, Sweden, May 2012

Conference Networking Recommendations based on Past Online Activity

Shan Huang
Petre Mihail Anton

Examiner: Helena Holmström Olsson

University of Gothenburg
Chalmers University of Technology
Department of Computer Science and Engineering
SE-412 96 Gothenburg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
Gothenburg, Sweden May 2012

Conference Networking Recommendations based on Past Online Activity

Shan Huang
Software Engineering and Management
Dept. of Computer Science
Chalmers University and University of Gothenburg
Gothenburg, Sweden
shan.huang@me.com

Petre Mihail Anton
Software Engineering and Management
Dept. of Computer Science
Chalmers University and University of Gothenburg
Gothenburg, Sweden
pemi.anton@gmail.com

ABSTRACT

Although many people attend conferences for the potential networking opportunities, the majority of the networking that happens is random and unarranged. This is a problem realized by the company Shpare AB, who provided the research topic and the task to build a back-end for a web application that handles gathering data from social networking websites with the purpose of generating networking recommendations for conference attendees. This paper documents the technological and conceptual components, with a highlight on the recommendation algorithm, of the back-end system. Findings demonstrate that such a system cannot only be built, but has potential to be enhanced and even mutated for solving problems in other fields.

General terms

social recommendation systems, social network analysis, social networking websites, data-mining

Keywords

recommendation, social networking, analysis

1. INTRODUCTION

During the past several decades, people have spent significant amounts of time and money on attending conferences. Financially, predictions show that the global market for conferencing services is expected to be worth more than £1.2 billion a year by 2013 [1]. Presentations have educational value, but the bigger motivation for people to attend conferences comes from the social aspect, to meet and connect with interesting people. In practice, most of the social networking at conferences is happening randomly.

The value social networking brings to conference has been well recognized. However, services that implement the social aspects for conferences focus on increasing exposure and do not directly benefit the conference attendees. Software that helps people create new social connections exists, yet for other purposes. A common example is dating websites, which use data manually inputted by the users, as opposed to making use of existing data from social networking websites. Moreover, current research does not indicate strategies applicable for generating networking recommendations.

Shpare AB is a company with the goal to solve this problem of social networking at conferences. Their solution is to use the conference attendees' existing information from their social networking websites to generate networking recommendations, through a web application. Shpare has developed an initial mockup that however was missing the core algorithmic parts. There was a front-end for handling user sign up and a page for presenting matches, but a database for storing parsed data from social networking websites did not exist, neither algorithms for the match-making to take place.

This research attempts to create and present a back-end system for software-generated valuable networking recommendations for the attendees of a conference. Valuable networking recommendations are seen as opportunities of establishing a contact with a person who can bring value to oneself, for example by spreading knowledge or making use of their existing social networks. The process of match-making is to be based on the analysis of social network-

ing websites. Therefore, the research question addressed in this paper is as follows:

How can networking recommendations for conference attendees be inferred from existing information on past online activity on social networks?

The paper details the research and construction of the back-end of Shpare's web application. The back-end development consists of a system that can parse social networking websites, store the parsed data in a graph database, and an algorithm to calculate networking recommendations. The technology involved in Shpare's back-end can be largely categorized into two groups: using social networking websites for conferences, and generating matches based on user profiles.

An agile proof of concept method was used for the research. The total development time was two months, where three iterations were completed and validated. The first iteration focused on literature review and proof of technology [2]. It concluded with a prototype with restricted functionality and performance, technically validated and integrated with the front-end. The following two iterations involved developing, testing, refining and validating the system. At the end of the two months, the system had been deployed to production and used at a conference of about 1 000 attendees. The development period concluded with a functional system and a features road map for Shpare's future uses. From the beginning, the authors were aware of their bias and confidence towards positive results from the research. Still, it was unexpected to observe the relatively low degree of complexity for an algorithm to return satisfactory results, and in the numerous opportunities for future work.

The rest of the paper presents the study's theoretical framework in section 2, research methods in section 3, and research results in section 4. Section 5 consists of a discussion of the development and research experience, while also covering related work as well as future work. Finally the conclusions are presented in section 6.

2. THEORETICAL FRAMEWORK

This section describes the core concepts this research study revolves around. These concepts are match-making, data mining, and social networking website. Sections 2.1 to 2.3 introduce each concept and describe their significance and how they are related to the research.

2.1 Match-making

Traditionally, match-making signifies the involvement an agent who selects two subjects from a pool based on their characteristics. For example, this is common in arranged marriages, where characteristics such as age, religion, and social status are important factors in determining whether two people are potentially suitable for each other.

With the services from dating websites, people no longer have to rely on a traditional match-maker. But the principle is fundamentally the same, a dating website usually involves the functionality for people to input personal information, which is then used as criteria to calculate the compatibility to others. Other types of match-making websites such as movie recommendations also follow this general model.

2.2 Data mining

Data mining is the process or practice of examining large collections of data in order to generate new information, typically using specialized computer software. The purpose of data mining is to extract certain patterns and knowledge from a given set of data. During the process of data mining, several tasks usually take place:

- Detecting anomalies. If a piece of data is not within the target range, then it should be discarded or at least put aside. For example when someone is searching for cooking books in a library, other type of books such as novels are anomalies.
- Sorting and storing data. When a relevant piece of data is found and identified, it should be categorized and stored in a way that is convenient for later usage.
- Summarizing data. When possible, aggregate data so that it can accurately present a compact understanding of its underlying data.

2.3 Social networking website

The meaning of social networking has slightly evolved from its inception. A social network, in its general sense, is a group of people who are socially connected to one another. Consequently, social networking is the use or establishment of social networks or connections. The extensive use the social networking concept has made it shift its meaning, usually depicting a social networking service. A social networking website is: „a web-based service that allow individuals to construct a public or semi-public profile within a bounded system, articulate a list of other users with whom they share a connection, and view and traverse their list of connections and those made by others within the system" [3].

The idea of social networking services emerged soon after the creation of Internet. BBS (bulletin board services) and IRC (internet relay chat) provided people with an Internet connection a virtual platform to socially interact with others. Around the turn of the millennium, a second generation of social networking websites spawned with the idea of user-profile centric platforms. In addition to the existing messaging exchange services, the user-profile allows people to conveniently share personal information, including their pictures and details about their daily activities, with others who use the same social networking website. One of the first social websites is almost two decades old [4]. In addition, social networks has long been a studied domain: the „Social Networks" journal [5] has been publishing quarterly issues for the past 33 years [6].

The essential feature of social networking websites, that makes recommendations possible, is the *persistent online activity history*. The need for this features arises from the fact that people provide biased presentations of themselves [7]. With the help of the activity history, that tracks all the changes one goes through via social websites, it becomes easier to obtain a more detailed and precise description of a person. The activity history can take many forms, e.g. as the timeline for Facebook, or the feed for Twitter. The persistence of the data is needed because the larger the span of the activity history, the more information can be understood about a person. Accordingly, the data volume only increases over time, which justifies why social networking websites require large data center solutions [8]. At the moment of this writing, Facebook has almost 850 million users [9]. Another example of the amount of

data social activities can generate is that social websites for conferences gathered information on more than 9 000 [10], respectively 18 000 [11] conferences in 2011.

The social networking websites involved in this research are Facebook, LinkedIn, and Twitter, which are not only some of the biggest social networking websites but are also at the top of most-trafficked websites overall, with Alexa ranks of 2 [12], 12 [13] and 8 [14] at the time of research. Facebook [15] is a profile-based general purpose social networking website where its users can share basic as well as more personal information with friends and the public. LinkedIn [16] is the leading website for business networking. Twitter [17] is a single purpose website where users can only post one type of information, a message that is 140 characters or less. All of these three websites allow users to make connections to other users of the service.

3. RESEARCH METHODS

The following sections motivate the choice of proof of concept, as research strategy (Sect. 3.1), present the research site (Sect. 3.2), and outline the activities that as part of the proposed method (Sect. 3.3).

3.1 Proof of concept as a strategy

The uniqueness of the research question necessitated a research method that would enable innovating based on existing technology and research, and support rapid development. As with any innovation, the set of requirements is known to be pivoting. Also, the time frame in which the prototype would be developed requested by Shpare AB was 2 months. The solution was to follow an agile proof of concept [18] strategy; having incremental iterations enabled the prototype to be confirmed at several stages, with different levels of functionality. For higher efficiency, test driven development [19, 20] was adopted, because it provides the opportunity of not needing to provide documentation with the code, of managing development tasks easily, and of ensuring the system is functional at any given time.

3.2 Research site

After a couple face-to-face meetings with the people behind Shpare AB, the research and development were done via telecommuting, where contact was maintained via e-mail, instant messaging and VoIP. Code sharing was done through a git repository [21, 22], hosted on Bitbucket [23]. Deployment and testing with production server was handled via SSH [24].

3.3 Research cycles

The process is illustrated in Figure 1, and described in detail in the sections below. The time span of the iterations does not necessarily reflect the man-hours spent, as the degree of pressure varied in each stage.

3.3.1 Iteration 1

The first iteration, which lasted approximately 3 weeks, focused on understanding the task, literature and technical research, and creating a simple prototype to kickstart the development.

This iteration started with a meeting between the author's and their mentor from Shpare. The mentor made an introduction of the structure of the entire system and what is required from the back-end. As the meeting concluded, the authors had gained an understanding of Shpare's main use case.

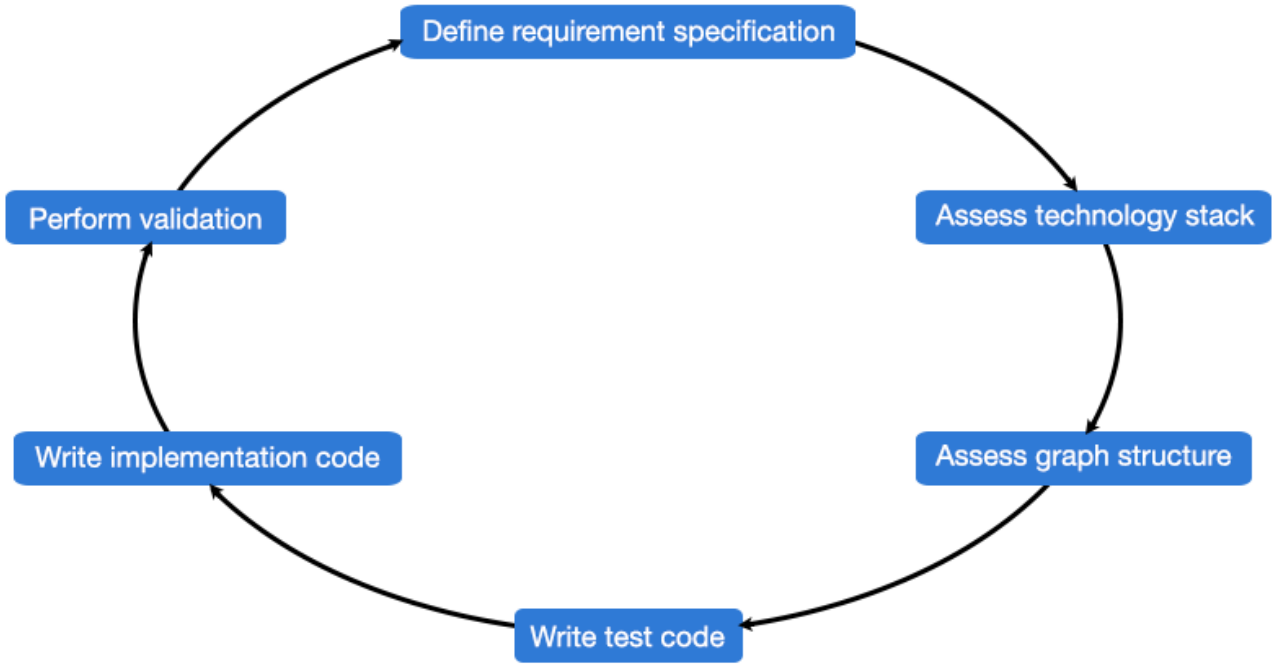

Figure 1. Research cycles

The authors began the research by conducting a literature and technical review to gain the necessary understanding before development can start. The literature review helped the authors learn about match-making, data mining, and social networking websites (as presented in Section 2. Theoretical framework). This process not only provided insight into important concepts related to Shpare, but also established an important foundation to have before the technical review. On the technical review side, the main objective of this first iteration was to decide on a technology stack to use for development, which should consist of a graph database, a programming language, and a web development framework with a matching web server.

Once the technology stack is defined, the authors could start putting together an initial prototype. This prototype is an abstract version of the finished product. The development process, which is repeated throughout the development period, is represented as shown in the following diagram.

After initial prototype was developed, it underwent technical validation that included integration testing against the front-end with mock data. Both the authors and their mentor from Shpare performed the validation.

3.3.2 Iteration 2

The second iteration, which spanned over 4 weeks, was dedicated to expanding the initial prototype to a fully functioning beta version [25].

Before further development could be continued, the authors did a second round of literature and technical reviews focusing on what type of information to target when data-mining the social networking websites. The literature review continued from iteration 1 and consisted of reading papers and online articles that are relevant to data-mining social networks. The technical review consisted of studying and testing the public developer APIs from Facebook, LinkedIn, and Twitter. A need for creating background jobs appeared when working with the social network's APIs. The purpose of background jobs are to distribute time consuming requests, such as parsing the social network APIs, to other processes so the web server that communicates with Shpare's front-end does not get clogged up. The technology stack was then expanded with a tool that handles background jobs.

Development then resumed by first redefining the graph structure and requirement specifications to cover the need for a functioning beta version of the back-end. Once the requirement specifications are written, the authors developed the rest of the system in the test driven development fashion as described in iteration 1.

Iteration 2 concluded with a technical validation of the beta system. The system was first tested with local machines and then deployed to and tested on the production server. A significant performance issue, with benchmarks that are unacceptable in a real world situation, arose during testing with the production server. This lead to some modifications to the technology stack, which will be explained in the following section 3.3.3.

3.3.3 Iteration 3

Iteration 3 lasted about 1 week and consisted of rewriting the entire system with a modified technology stack to meet the needs of the production server.

This iteration was allocated with only 1 week because it was originally intended for only fixing small problems and perfecting the system. However, a much more alarming problem showed up during technical validation at the end of the second iteration. The system was expected to be ready for a conference of more than 1 000 people at the end of the week. Given the limited remaining time, the authors decided to resolve the problem by rewriting the system with a modified technology stack. The rewrite was based on the previously established data structure and requirement specifications from iteration 2 and thus only involved rewriting the test and implementation code. The new back-end was completed towards the end of the week and was again deployed to and tested on the production server. As expected, the new system's performance was acceptable and it was staged into production just in time for the conference.

In conclusion, of the efforts in research and development, the authors' mentor from Shpare attended the conference and helped the authors conduct a qualitative evaluation by interviewing the conference attendees. In addition to the completed product, a set of artifacts are documented: a list of requirement specifications, a table consisting the selection process and resulting technology stack, a visual representation of the graph structure, and a system architecture diagram.

4. RESULTS

4.1 Use case scenario

The following use case scenario illustrates the process from creating a new conference to users getting back their matches. It served as a foundation for the rest of the research and development.

- Shpare receives a list of conference attendees from the conference's organizer. For each attendee in the list, an email address is a required field; optional fields include name, twitter username, etc.
- An admin imports and stores the conference attendees on its front-end database.
- An admin populates Shpare's back-end for the conference.
- An admin sends out email invitations including an account activation link to each conference attendee.
- A user (conference attendee) activates her/his account with the activation link from the invitation email.
- A user defines 'is' and 'wants' tags.
- A user logs on to Shpare's website and connects her/his social network accounts (Facebook, LinkedIn, and Twitter).
- A user visits Shpare's website's „People” page, which shows a list of matches that are loaded from the back-end, in real time.

4.2 Requirement specification

4.2.1 Iteration 1

The following list of requirements are derived solely based on the use case scenario from 4.1. The link between the front-end and the back-end is done via an application programming interface (API), based on POST and GET [26] requests.

Functional requirements

Front-end and back-end APIs

R1. An authenticated request between the front-end and back-end should be possible.

R2. A POST request for creating a user should be possible.

R3. A POST request for updating a user's properties should be possible.

R4. A POST request for tagging a user with a conference should be possible.

R5. A POST request for adding user created tags should be possible.

R6. A POST request for creating a friendship through Shpare should be possible.

R7. A POST request for making a batch insert of above POST requests should be possible.

R8. A POST request for removing a user's tags should be possible.

R9. A POST request for removing a user should be possible.

User

R10. Creating a new user should insert a new user node into the database.

R11. Managing user properties should update the user node's properties in the database.

R12. Retrieving a user's tags should obtain all tags connected to the user node from the database.

R13. Removing a user should delete the user node and all of its edges from the database.

Tag

R14. Adding a tag to a user should create a new tag node if it does not already exist, and connect the tag to the user node in the database.

R15. Batch insertion of adding tags should be possible.

R16. Removing a tag from a user should remove the connection between the tag node and the user node, and the tag node should be deleted if it is not connected to any other user nodes.

4.2.2 Iteration 2

The following list of requirements are derived after the graph structure and algorithm are established.

Functional requirements

Front-end and back-end APIs

R17. A GET request for retrieving a user's matches should be possible.

R18. A GET request for retrieving tag's connected to a user should be possible.

User

R19. Merging two user nodes should combine the properties and edges of two nodes in the database.

Social

R20. Requesting friends from the Facebook API should be possible.

R21. Requesting likes from the Facebook API should be possible.

R22. Requesting connections from the LinkedIn API should be possible.

R23. Requesting job positions from the LinkedIn API should be possible.

R24. Requesting skills from the LinkedIn API should be possible.

R25. Requesting groups from the LinkedIn API should be possible.

R26. Requesting followers from the Twitter API should be possible.

R27. Requesting followings from the Twitter API should be possible.

R28. Requesting mentions from the Twitter API should be possible.

R29. Requesting hashtags from the Twitter API should be possible.

Algorithm

R30. Calculating networking recommendations for a user should return a list of user nodes, ranked by descending scoring, from the database.

Non-functional requirements

R31. The front-end must receive the list of matches within maximum 3 seconds.

R32. The data stored on the back-end must be anonymous.

4.3 System architecture diagram

The system architecture diagram represents the back-end components, and how they interact with each other as well as the rest of the system including the front-end and social network websites' APIs.

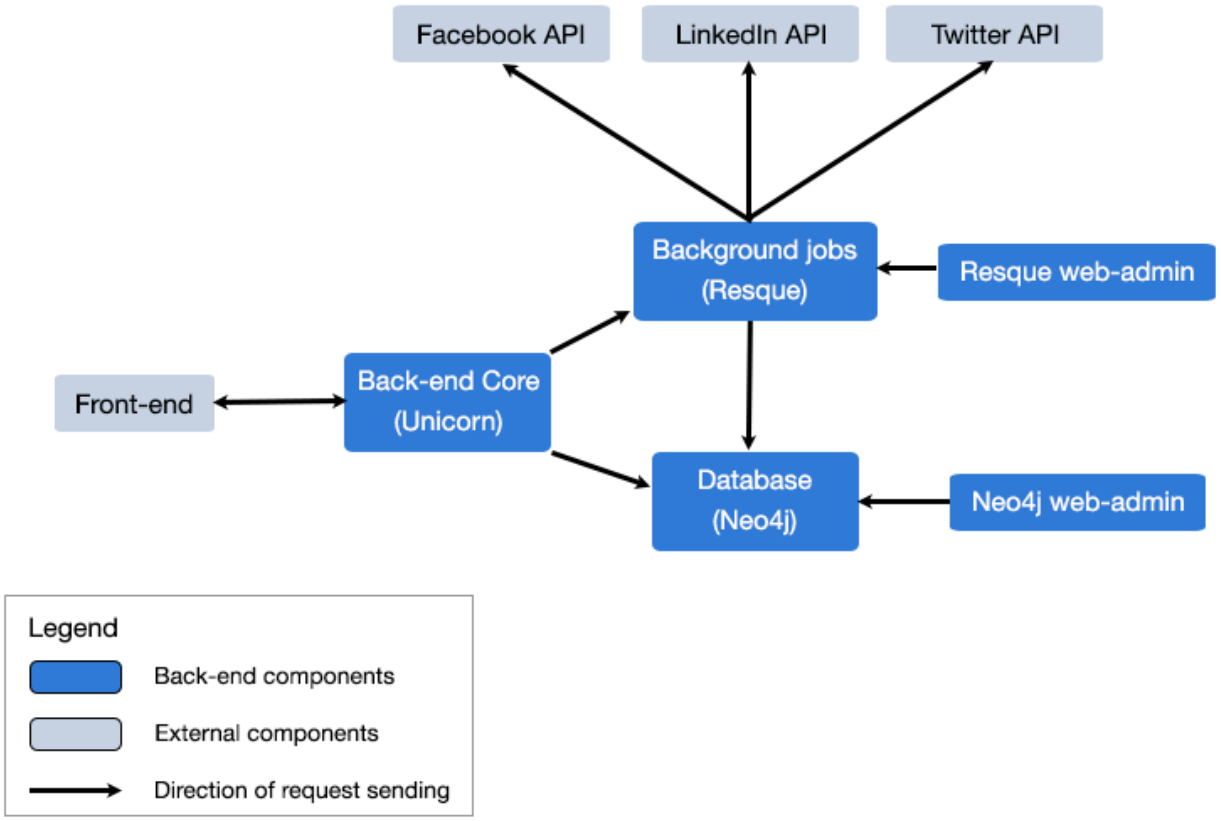

Figure 2. The architecture of the back-end

4.4 Technology stack

The technology stack consists of a graph database, a programming language, a web development framework and a web server. Deciding on what technology to use for each component in the stack depends on three main factors: compatibility with other components in the stack, compatibility with the front-end, and performance. Table 1 includes the chosen components for the stack, the motivation for the selection, and the most important alternatives that were considered.

Solution	Motivation	Alternatives
Graph database: Neo4j	Important factors that were taken into consideration for this decision are: license (is it open source?), performance (will it be able to handle the required complexity and responsiveness?), ease of use (does it have good documentation and good compatibility with different programming languages?). Neo4j [27] although not exceedingly impressive in any factor, is good enough in all areas considered and was ultimately chosen. Another deciding factor was that Neo4j comes with visualization functionality that, according to social networking visualization concepts, enables higher working memory capacity [28].	FlockDB [29]is an open source graph database that is used by Twitter. Although it appears to be more superior to Neo4j in terms of performance and scalability, its documentation was almost non-existent at the time of research. TinkerPop Blueprints [30] on the other hand is a mature and well documented stack of graph related products. The downside though is that it consists of 5 child projects which make up the stack and thus the learning curve was much higher than what is required by Neo4j.
Programming language: Ruby	Shpare's front-end is written in Ruby on Rails [31, 32], so choosing Ruby [33, 34] for the back-end provides great compatibility with it. Neo4j, which is itself written in Java has two good Ruby implementation in neo4j.rb [35] (a JRuby version	Java [38, 39] would have provided better performance with Neo4j. But the language is more verbose in comparison to Ruby. As the authors are more experienced in web development with Ruby, the learning curve would have been higher with Java. JRuby [40, 41] was not only strongly considered but was actually

	of Neo4j, that is Ruby running on top of the Java Virtual Machine [36]) and Neography [37] (a Ruby wrapper that works with Neo4j via its REST API). Neography eventually turned out to be the chosen Neo4j implementation, and Ruby in turn the programming language that was used.	used for most of the development. The authors had initially written a few scripts to test the performances of the two Ruby implementations during the first iteration, and as expected the JRuby version, neo4j.rb gave better benchmarks, thus JRuby was chosen. At a later moment, the authors encountered the serious performance issue on the production server, which led to the change to Ruby.
Web development framework: Sinatra	Sinatra [42, 43] is a DSL (Domain Specific Language) for writing web applications in Ruby. It is a small framework that allows a developer to quickly achieve a specific functionality with minimal effort. Unlike most web frameworks, Sinatra does not have any default project structure and does not rely on generators, which makes it easy to understand exactly what happens in all parts of the project. These factors made Sinatra an ideal choice as a web framework for Shpare's back-end.	Ruby on Rails is the most popular Ruby web framework and has great documentation and a vast community behind it. However it has a default project structure that included too many things that are not needed for this system. Grape [44] is a Ruby web framework that was written for the sole purpose of creating APIs, which provides good compatibility with requirements of the system. But at the time of research, Grape was still a relatively new project and the authors encountered bugs which provided enough reason to abandon it as an alternative.
Web server: Unicorn	Unicorn [45] is a fast HTTP server for Rack [46] applications (Sinatra is Rack based). It is extremely easy to set up and manage. Concurrency can also be easily controlled by starting up other Unicorn processes.	Other web servers for Rack applications exist, such as Mongrel [47], Passenger [48], and Thin [49]. However Unicorn was such an ideal solution compared to the alternatives that the others did not receive much consideration. Note: During much of the development, JRuby instead of Ruby was used, and its web server was Mizuno [50]. Mizuno is a Jetty-powered server that was built for JRuby/Rack applications. Like Unicorn, Mizuno is easy to set up and manage.
Background job: Resque	The two main advantages of Resque [51] are: - Once it is set up, it is very easy to use. - It has a web admin app built in for monitoring processes.	Delayed_job [52] is easier to set up compared to Resque. However Neo4j is not an officially supported back-end database, and although it is potentially possible to configure Delayed_job to work with Neo4j, the time required for that would make it lose its ease to set up advantage over Resque.

Table 1. The technology stack of the back-end

4.5 Graph database

Firstly, all the relevant data types that describe users were gathered. Shpare's front-end provides the functionality to manually input tags. These tags can be one of two categories, either a tag describing who the user is, or a tag describing what the user wants. After studying the public APIs from Facebook, LinkedIn, and Twitter, the relevant types of information to target were selected. The result of this is displayed in Table 2.

Website	Data
Front-end	is, wants
Facebook	Friends, Likes
LinkedIn	Connections, Jobs, Skills, Groups
Twitter	Followers, Followings, Hash tags, Mentions

Table 2. Data used in the graph database

Secondly, a graph database management system was chosen to store the data to be used by the algorithm. Graph databases use graph structures with nodes, edges, and properties to represent and store data [53]. It has already been established that graph theory is useful in solving social networking problems [54]. In addition, a graph DBMS (database management system) perform better than normal DBMS by several order of magnitude [55, 56].

Thirdly, the database structure was designed to follow the three-mode network model [54] of users, conferences and tags. This structure can be seen as the result of combining two two-mode network models, one with users and conferences, and the other one with users and tags, both having the same user base. In contrast with the one-mode model, this structure creates the possibility of generating matches by identifying common tag nodes between pairs of user nodes for a certain conference. On the long term, performing statistics on the tag nodes can reveal information useful on how these tags should be handled, e.g. cases where they are similar. Similarly, performing statistics on conference nodes can reveal information useful for conference organizers, e.g. conferences with the same user base. One might argue that conference nodes introduce a great number of edges, equal to the number of conference participants. However, without these nodes the algorithm might return as networking recommendations users who are not participating to the current conference. Popular tag nodes can easily have over 1 000 edges, whereas only big conferences can generate so many edges.

The final structure of the graph database contains three types of nodes: user, tag and conference. Besides the user to conference edges, users are to tag nodes by is, want, Facebook likes, LinkedIn jobs, LinkedIn skills, LinkedIn groups, Twitter hashtags edges and

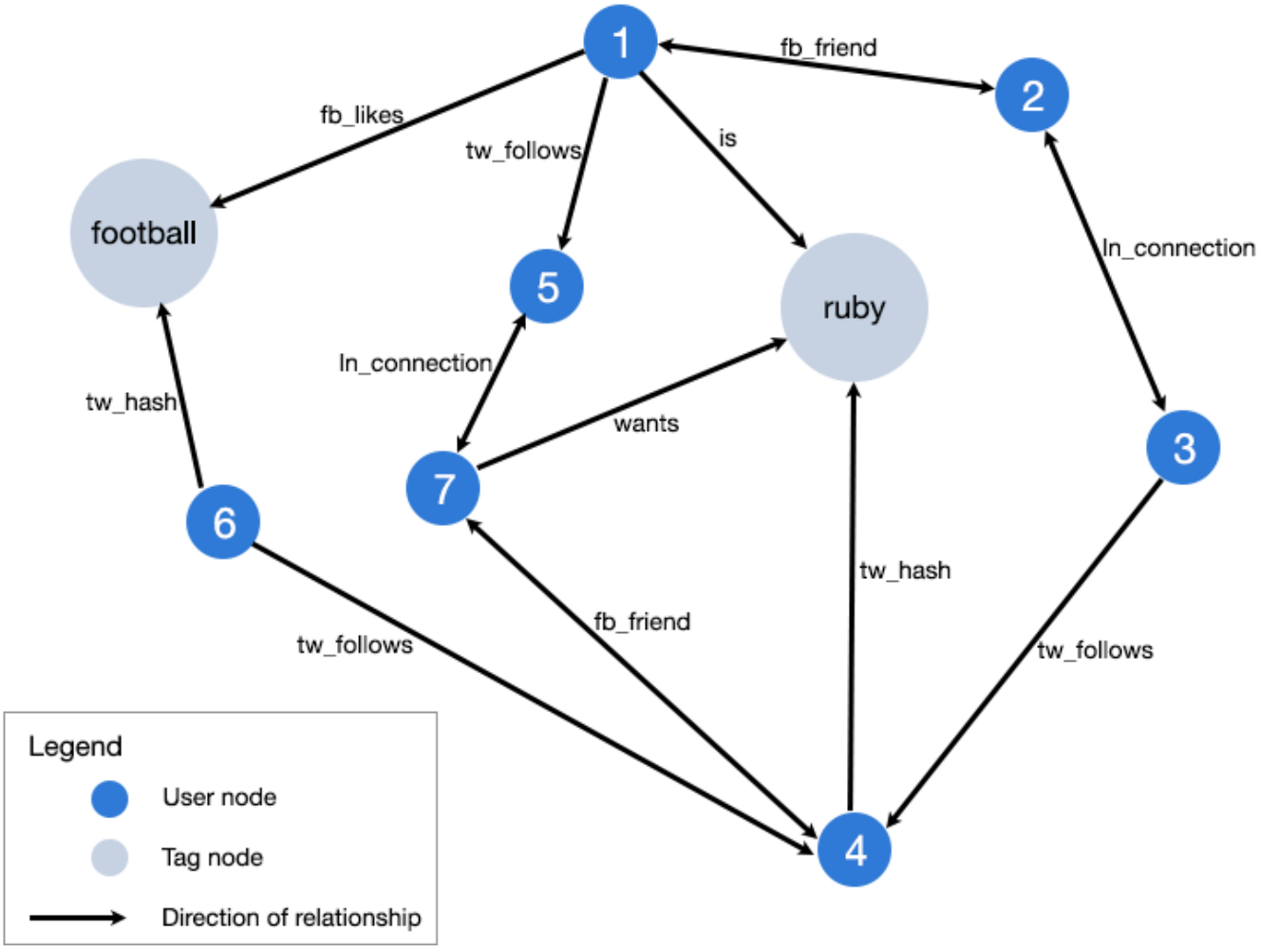

Figure 3. An example of the structure of the graph database

to other users by Facebook friends, LinkedIn connections, Twitter followers, and Twitter followings edges.

Figure 3 shows an example of this structure; for clarity, the conference node is omitted, as all the conference edges would clutter the diagram. The users are represented by numbers, and only the front-end can convert actual users to numbers and vice versa; this way, anonymity is preserved during the matching process.

4.6 Algorithm

In order to generate the networking recommendations, the software algorithm performs several steps. The first 6 of these steps are done in 3 database queries, followed by a scoring mechanism. At Shpare's request, the details of the scoring portion of the algorithm are with kept.

1. The node which represents the target user is identified.

2. The node which represents the target conference is identified.

3. All the users who participate the target conference are identified, except the target user.

4. From all participants to the target conference, only the ones who haven't established a direct social networking connection with the target user are kept.

5. The match quality of users is determined by calculating the scores based on the individual tags connecting the filtered participants to the target user.

6. The final list of users is sorted based on the resulting scores.

7. Only the top scoring 20 matches are returned.

According to the graph figure, which gives an example of a simplistic graph, the algorithm would function in the following way for user 4:

1. User 4 is identified.

2. The conference node is omitted from the diagram for the sake of clarity. In this example it is assumed that all user nodes are connected to the same target conference.

3. Users 1, 2, 3, 5, 6, 7 are identified.

4. Users 1, 2, 5 are kept.

5, 6. The scoring algorithm is applied to to the target user and the filtered users from step 4.

7. Values 1, 2, 5 are returned. There is no difference in score between 2 and 5 so their order is irrelevant.

4.7 Qualitative evaluation

Inconclusive. The feedback obtained during the conference from the interviews were not suitable to be used for analysis. In retrospect, this makes sense because of the following reasons.

1. People who attend the same conference usually have at least one thing and likely several things in common. This makes it difficult to generate bad matches. The general attitude of conference attendees is that they seemed to be more excited about the idea of getting networking recommendations rather than the actual people recommended to them. These factors lead to biased responses.

2. A proper assessment of the quality of the recommendations takes much longer time. For a connection made through Shpare, it could require a follow up investigation in a few months, or even a few years time for a full evaluation of the match.

5. DISCUSSIONS

5.1 Lessons learnt

The authors strived to design an algorithm that would generate matches that would help people in networking at conferences. Given the vast potential of reusing social networking websites' data for this purpose, we have only achieved a core foundation. This solution is yet simple to deploy and use, and produces accurate results, which makes it a good basis for further development. At the end of iteration 3, the beta product went through rigorous testing during a live conference, where the authors' mentor from Shpare as well as the conference attendees and deemed it successful.

5.1.1 Frequent testing with real data and environment

The biggest lesson learnt to test more frequently with real data on the production server. The performance issue encountered at the end of iteration 2 was of extreme severity, especially considering that the system was expected to be fully functional within a week. The specific problem was due to the conversion of a Java object to a JRuby object, which took minutes instead of milliseconds. This problem was previously undiscovered due to the difference in size of the graph database with mock data compared to the graph database with real data. With mock data there were around 3 000 nodes in the database, where as with real data there were around 50 000 nodes. The conversion problem directly caused other problems such as the JVM running out of memory, which froze the back-end core.

To solve this problem the authors had two options, attempt to optimize the technology stack and production server or rewrite the entire back-end with neography instead of neo4j.rb. After attempting the first option for a few days and getting nowhere, the authors decided to move to the second option. Neography uses pure Ruby and interacts with the Neo4j database through the Neo4j REST api. This means that there is never any direct conversion of Java objects to Ruby objects.

The authors spent a combined ~200 hours in the final week in order to produce a working system. Needless to say, this problem could have been catastrophic. To avoid such panic and stress in the future, the development process should be enhanced with frequent testing on the production server using real data.

5.1.2 Test the examples provided by the documentation

When working with implementation of the algorithm, the authors had initially followed a query style from Neo4j's official documentation [57]. The specific query style however proved to be not scalable and would get exponentially slower as the size of the database increases. To solve this problem, the authors had to break down the initial query into three smaller queries that are chained together. It is important to point out that the three smaller queries did not alter the abstract algorithm, which means that the Neo4j's official documentation was misleading. Thus, the lesson learnt here is to not be over confident with the referenced documentation and encourage the practice of frequent technical evaluation.

5.1.3 Test driven development

As for the rest of the development process, the authors are quite satisfied with the experience especially using test driven development practice. Requirement specifications provided a one-to-one matching to writing test code, which provided clear instructions for the ensuing implementation code. Running the test suites frequently also points out any previously passed tests which could later fail due to adding new features or changing previous code. Test driven development allowed the authors to work efficiently and confidentially, which was important considering the amount of development required in the given time frame.

5.2 Related Work

Researchers from IBM and University of Minnesota have analyzed the performance of 4 different recommendations algorithms [58]. Their study, performed on the social networking website Beehive within IBM, contains a survey of 500 users and a field study of 3 000 users to benchmark Content matching, Content-plus-Link, Friend-of-Friend and SONAR algorithms for friend recommendation. The Content matching algorithm finds users with similar content; Content-plus-Link builds on top of the content matching algorithm, taking into account social link information obtained from the social network structure. The friend-of-friend algorithm computes matches based solely on social network links, and SONAR „aggregates social relationship information from different public data sources within IBM”. Based on the survey results on the recommendations of unknown people rated „good”, the order of the algorithms is Content matching, Content-plus-Link, Friend-of-Friend and SONAR, with Content matching having the highest score. Peculiarly, the score of „not good” recommendations follows the same order. It is interesting to see that for known matches, the benchmark reveals a mirroring order of performance, both for „good” and „not good” recommendations. The same pattern, where the SONAR algorithm leads, emerges also from the controlled field study, based on „good” matches, and the number of resulting connections. However, the order was again reversed for the percentage of „introductions”.

Therefore, the aforementioned study shows that the Content matching algorithm is the highest rating when the focus is having the chance of being introduced to unknown people, which is the scena-

rio of business networking. Besides the slightly lower success ratio, the time required for computing Content-plus-Link was 25% longer in the tests the authors performed for Shpare.

Another IBM study on the same 4 algorithms has shown that the Content matching algorithm generates the „highest increase in betweenness centrality for a connecting user, i.e. the new connections gained by the user placed the participant in a role of connecting users that were previously disconnected" [59]. The same study has also analyzed the activity levels of the resulting connections of the 4 algorithms, out of which the Content matching algorithm has generated the most active connections over a period of 12 months.

StreetSpark [60] is a mobile app that creates an „interest graph" based on your social networking activity. Here are a few examples of such activity: being fan of the same page on Facebook, following the same people on Twitter, checking-in to the same places on Foursquare, or liking the same music on Pandora. Even if their algorithm seems to be based on similar data input to Shpare's, StreetSpark scores friends-of-friends connection. This indicates that their focus is on a different market than business networking. StreetSpark's solution is supposed to be location-aware and real-time, but without more information it is hard to say if there is a precompiled list of recommendations out of which a few are selected based on geographic location or if this is is recompiled at location changes.

Friend recommendation algorithms have been implemented by popular social websites. Myspace [61] launched Friend Recommendation, dubbed „People you may know" on November 15, 2008, and their algorithm generated recommendations for their 140 million users in 40 to 50 minutes on a 16 CPU machine [62]. In comparison, Shpare's algorithm produces recommendations for one thousand users in about 30 minutes, on a relatively basic dual-core machine. Although the details of the algorithms were not made public, the fact that a recommendation algorithm can give results so fast shows how much more performance can be improved.

5.3 Future Work

5.3.1 Adding other social networking websites

The development from this research used data from Facebook, LinkedIn, and Twitter, which make a great set to begin with due to their popularity. There are of course plenty of other social networking websites that could be integrated to make the database richer. The authors have made a selection these websites as shown in the listing below.

- Google+ [63] is a social networking website introduced by Google in June, 2011. Despite its 'youth' Google+ has established a user base competitive to Facebook and Twitter. This was made possible by the convenience provided by Google to convert existing Google account to Google+ accounts. Google+ does not provide many new types of information due to its similarity in features to Facebook. However, Google+ would still provide great value due to the size of its user base.
- Foursquare [64] is a social networking website with a niche on location based services. This brings a new helpful type of data that makes it possible to assign higher points to matches who live closer by.
- Github [65] is an online platform where users can host their programming projects. Its social aspect allows users to follow other users and even specific projects. Incorporating Github thus provides good value for tech conferences.

5.3.2 Diverse analysis mechanisms and a self-improving algorithm

The version of the algorithm this research presents has achieved only the easiest of social networking analysis types, and that is static analysis. Given the time frame, it was the optimal choice for proving the concept. The next steps that can be done are semantic analysis and dynamic analysis [66].

In the case of Content matching algorithms, semantic analysis did not yield significantly better results [58]. In fact, another study [67] observes that, with semantic analysis, „similarities between users [who are not friends] are approximately equal, irrespective of the topological distance between them". Above this, the work required to implement semantic analysis on the amount of data harvested by Shpare cannot be neglected. However, semantic analysis has not yet been done, to the authors knowledge, with the scope of understanding the attitude people have in public debates. The results of this analysis is supported by the current structure of the graph database, as it only needs replacing the edge that links a person to a tag.

A suggested approach for implementing dynamic analysis with the benefit of improving the algorithm is to follow if users make contact with each other on the social networking sites after Shpare's recommendations. In time, this would lead to better recommendations. Also, in this scenario adding a feature to allow users to rate each match would be helpful. By understanding the pattern of users' rating, both the algorithm can be enhanced and the users can have the algorithm provide custom tailored matches. Allowing users to rate the matches and semantic analysis pave they way to a dynamic and self-improving system. The algorithm can be written in such a way where it functions as a brain that constantly updates the scoring system depending on the changes in the graph database.

5.3.3 New graph database project

Neo4j was deemed the best graph database project during the technical evaluation. However it was problematical in numerous occasions, most often regarding performance. Neo4j is technically scalable but achieving high performance for a database with large amount of nodes and edges requires a lot of environment configuration and expensive hardware. The authors believe there is a need to start a new graph database project, where good performance should be a built-in feature even for large databases running on moderate hardware. Due to the scope of this paper, not much research has been done on how to build a graph database. Nevertheless the authors believe that using a functional language like Erlang [68], instead of Java which is used by Neo4j, would be a better option. Erlang is concurrent, distributed, and fault-tolerant, all of which are characteristics ideal for the environment of a graph database.

5.3.4 Extended functionality

One of the areas in which the social networking analysis can be used is marketing [69]. Presdo Match [70], a competitor to Shpare, has created functionality through which exhibitors and sponsors can search attendee profiles by matching keywords, thus being able to connect with targeted prospects even before the start of the

event. In the current setup of the system, the same functionality can be achieved by implementing user nodes in the database structure, and filter them out in the matching process.

5.3.5 Recommendation evidence

The quality of any system that generates recommendations is evaluated by the user satisfaction and acceptance. When users are presented evidence along with the recommendations, it was noticed that these quality indicators have higher values [59]. Thus, complementing the front-end so that it displays publicly available data used by the algorithm is a valuable addition.

6. CONCLUSIONS

The research began by applying a proof of concept method with the aim to build a system that can be used to generate networking recommendations for conference attendees based on existing information on past online activity on social networks. The outcome of the research was a functional version of such a system that is not only easy to set up, but also easy to monitor with the help of certain components of the technology stack such as the Neo4j and Resque web-admin interfaces. This system is accompanied by a description of its technology stack, and an overview of its architecture.

The response from the author's mentor confirmed the validity of the system, which was also tested at a conference of more than 1 000 attendees. The result of the study demonstrates the technology and concepts, and shows that such a system is indeed possible to be used in a real world situation.

The idea of providing networking recommendations for conference attendees is a fresh idea from Shpare, as is the approach to combine the concepts of reusing data from social networking websites and match-making algorithms. During the study, the authors began to understand the vast potential of the product. Although a proof of concept product was built and made fully functional, there remains untapped potential in advancing this idea by incorporating semantic analysis and machine learning. With small tweaks of the code base, the system can be applied to more fields such as marketing and various other types of recommendations.

ACKNOWLEDGEMENT

We would like to thank Lars Pareto, who dedicated many hours to provide invaluable guidance and feedback on this paper. We would also like to thank our mentor from Shpare AB, Anders Fredriksson, who presented a unique thesis topic and the opportunity to build a part of an exciting project. Last but not least, we would like to thank the many helpful users of Stackoverflow [71] who shared their experience on similar coding problems and saved us a lot of frustration during the development period.

REFERENCES

All the web references were accessed the 15th of May 2012.

1. BT Conferencing. Available from: http://www.btconferencing.co.uk/about-us/fast-facts.
2. Wikipedia. Available from: http://en.wikipedia.org/wiki/Proof_of_concept#In_software_development.
3. Boyd, D.M. and N.B. Ellison. *Social Network Sites: Definition, History, and Scholarship.* Journal of Computer-Mediated Communication, 2007. **13**(1): p. 210-230.
4. Wikipedia. Available from: http://en.wikipedia.org/wiki/Classmates.com.
5. Elsevier. Available from: http://www.journals.elsevier.com/social-networks/#description.
6. Science Direct. Available from: http://www.sciencedirect.com/science/journal/03788733.
7. Bar-Anan, Y., T.D. Wilson, and R.R. Hassin. *Inaccurate self-knowledge formation as a result of automatic behavior.* Journal of Experimental Social Psychology, 2010. **46**(6): p. 884-894.
8. Kant, K. *Data center evolution: A tutorial on state of the art, issues, and challenges.* Computer Networks, 2009. **53**(17): p. 2939-2965.
9. Socialbakers. Available from: http://www.socialbakers.com/countries/continents.
10. Lanyrd. Available from: http://lanyrd.com/2011.
11. Conference Hound. Available from: http://conferencehound.com/conferences/2011.
12. Alexa. Available from: http://www.alexa.com/siteinfo/facebook.com.
13. Alexa. Available from: http://www.alexa.com/siteinfo/linkedin.com.
14. Alexa. Available from: http://www.alexa.com/siteinfo/twitter.com.
15. Facebook. Available from: http://www.facebook.com.
16. LinkedIn. Available from: http://www.linkedin.com.
17. Twitter. Available from: http://twitter.com.
18. Parker, P.M. *Proof-of-concept: Webster's Timeline History, 1962 - 2007*, 2010: ICON Group International, Inc.
19. Beck, K. *Test-driven development : by example*, 2003, Boston: Addison-Wesley.
20. Chelimsky, D. *The RSpec book : behaviour-driven development with RSpec, Cucumber, and Friends*, 2010, Lewisville, Tex.: Pragmatic.
21. Loeliger, J. *Version Control with Git: Powerful Tools and Techniques for Collaborative Software Development*, 2009: O'Reilly Media, Inc. 336.
22. Git. Available from: http://git-scm.com/.
23. Bitbucket. Available from: www.bitbucket.org.
24. Barrett, D.J., R.E. Silverman, and R.G. Byrnes. *SSH, the Secure Shell: The Definitive Guide*, 2005: O'Reilly Media, Inc.
25. The Linux Documentation Project. Available from: http://www.tldp.org/HOWTO/Software-Proj-Mgmt-HOWTO/users.html#ALPHABETA.
26. HTTP Protocol. Available from: http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html.
27. Neo4j. Available from: http://neo4j.org/.
28. Zhu, B., S. Watts, and H. Chen. *Visualizing social network concepts.* Decis. Support Syst., 2010. **49**(2): p. 151-161.
29. FlockDB. Available from: http://github.com/twitter/flockdb.
30. TinkerPop. Available from: http://www.tinkerpop.com.

31. Ruby on Rails. Available from: http://rubyonrails.org.
32. Hartl, M. *Ruby on Rails 3 tutorial : learn Rails by example*, 2011, Upper Saddle River, NJ: Addison-Wesley.
33. Flanagan, D. and Y. Matsumoto. *The Ruby Programming Language*, 2008: O'Reilly. 444.
34. Ruby. Available from: http://www.ruby-lang.org/en.
35. neo4j.rb. Available from: http://neo4j.rubyforge.org.
36. JVM. Available from: http://docs.oracle.com/javase/specs/jvms/se7/html/jvms-1.html#jvms-1.2.
37. Neography. Available from: http://github.com/maxdemarzi/neography.
38. Flanagan, D. *Java in a Nutshell, Fourth Edition*, 2002: O'Reilly & Associates, Inc. 700.
39. Oracle. Available from: http://www.oracle.com/technetwork/java/index.html.
40. Edelson, J. and H. Liu. *JRuby Cookbook*, 2008: O'Reilly Media, Inc. 222.
41. JRuby. Available from: http://www.jruby.org.
42. Sinatra. Available from: http://www.sinatrarb.com.
43. Harris, A. and K. Haase. *Sinatra : up and running*, 2012, Sebastopol, CA: O'Reilly & Associates, Inc.
44. Grape. Available from: http://github.com/intridea/grape/wiki.
45. Unicorn. Available from: http://unicorn.bogomips.org.
46. Rack. Available from: http://rack.github.com.
47. RubyGems. Available from: http://rubygems.org/gems/mongrel.
48. Passenger. Available from: http://www.modrails.com.
49. Thin. Available from: http://code.macournoyer.com/thin.
50. Mizuno. Available from: http://github.com/matadon/mizuno.
51. Resque. Available from: http://github.com/defunkt/resque.
52. delayed_job. Available from: http://github.com/collectiveidea/delayed_job.
53. Wikipedia. Available from: http://en.wikipedia.org/wiki/Graph_database.
54. Wasserman, S. and K. Faust. *Social network analysis : methods and applications*, 1994, Cambridge; New York: Cambridge University Press.
55. Neo4j. Available from: http://video.neo4j.org/ajgJ/how-to-get-started-with-neo4j-209.
56. Rene Pickhardt. Available from: http://www.rene-pickhardt.de/time-lines-and-news-streams-neo4j-is-377-times-faster-than-mysql.
57. Cypher. Available from: http://docs.neo4j.org/chunked/1.8.M01/query-match.html#match-zero-length-paths.
58. Chen, J., et al. *Make new friends, but keep the old: recommending people on social networking sites*. in *Proceedings of the 27th international conference on Human factors in computing systems*. 2009. Boston, MA, USA: ACM.
59. Daly, E.M., W. Geyer, and D.R. Millen. *The network effects of recommending social connections*. in *Proceedings of the fourth ACM conference on Recommender systems*. 2010. Barcelona, Spain: ACM.
60. StreetSpark. Available from: http://blog.streetspark.com/#about.
61. Myspace. Available from: http://www.myspace.com.
62. Moricz, M., Y. Dosbayev, and M. Berlyant. *PYMK: friend recommendation at myspace*. in *Proceedings of the 2010 international conference on Management of data*. 2010. Indianapolis, Indiana, USA: ACM.
63. Google+. Available from: https://plus.google.com.
64. Foursquare. Available from: https://foursquare.com.
65. Github. Available from: http://github.com.
66. Thovex, C. and F. Trichet. *Semantic social networks analysis.* Social Network Analysis and Mining, 2012: p. 1-15.
67. Bhattacharyya, P., A. Garg, and S. Wu. *Analysis of user keyword similarity in online social networks.* Social Network Analysis and Mining, 2011. **1**(3): p. 143-158.
68. Erlang. Available from: http://www.erlang.org.
69. Bonchi, F., et al. *Social Network Analysis and Mining for Business Applications.* ACM Trans. Intell. Syst. Technol., 2011. **2**(3): p. 1-37.
70. Presdo. Available from: http://match.presdo.com/about.
71. Stackoverflow. Available from: http://stackoverflow.com.