An Interactive Survey Application for Validating Social Network Analysis Techniques

by Mitchell Joblin and Wolfgang Mauerer

Abstract Social network analysis is extremely well supported by the R community and is routinely used for studying the relationships between people engaged in collaborative activities. While there has been rapid development of new approaches and metrics in this field, the challenging question of validity (how well insights derived from social networks agree with reality) is often difficult to address. We propose the use of several R packages to generate interactive surveys that are specifically well suited for validating social network analyses. Using our web-based survey application, we were able to validate the results of applying community-detection algorithms to infer the organizational structure of software developers contributing to open-source projects.

Introduction

Social network analysis (SNA) is an increasingly popular approach to study the relationships between individuals engaged in collaborative activities (Ahn et al., 2007; Mislove et al., 2007; Kumar et al., 2010), and numerous high quality R packages support the thriving SNA community (e.g., **igraph**, **sna**, **graph**, **twitteR**, **Rfacebook**, etc.; Csardi and Nepusz 2006; Butts 2014; Gentleman et al.; Gentry 2015; Barbera and Piccirilli 2015). What is often not clear is the validity of SNA approaches that propose new metrics or apply existing metrics to a new source of data. In the literature, researchers have questioned and criticized studies using SNA because it is unclear if the results are reflective of reality (Donath and Boyd, 2004; Wilson et al., 2009). We developed a web-based survey application for conducting interactive surveys that specifically addresses the unique needs of the SNA community and successfully deployed the application to study the collaborative relationship between software developers in open-source projects and to validate the usage of unsupervised machine learning algorithms to infer the developers' organizational structure (Joblin et al., 2015).

In social network analysis, the relationships between individuals are formalized as a graph where nodes represent people and the edges between nodes represent a particularly interesting connection. For example, Twitter data can be used to construct a retweet network where an edge between individuals exists if one individual has retweeted another individual's tweet. The particular heuristic used to establish an edge between individuals is chosen based on the desired concept to study. For example, a retweet may indicate endorsement of the message being tweeted. From this, one could conclude that users with many retweets of their content are regarded as an influential person within that local group of people. One of the primary challenges with this style of analysis is validating whether the assumptions about the relationship heuristic are correct. It may, for instance, not be clear whether a retweet always indicates a positive sentiment. Alternatively, retweets could also stem from controversial topics and may not be ubiquitously regarded as supportive of the original tweet's message.

While SNA is not primarily concerned with constructing social networks, but rather to analyze the network's properties, the network construction heuristic influences the validity of the subsequent analysis. In general, the goal of SNA is to identify interesting features of a social network that capture an abstract quality of social relationships. For example, finding important or highly influential actors in a network is one of the most well-researched areas of SNA where one considers the local or global network topology to identify individuals that are exceptionally well-connected to other actors. The notion of centrality, and many other network metrics, is a duality where one definition is a mathematical formalization based on the network topology and the other definition is an abstract social concept such as influence. The challenge we wish to address with our survey application is to validate the claim that the mathematical formalizations provided by the field of SNA are congruent with the abstract social concept we wish to identify.

Our survey application is designed to address the following three concerns that are fundamental to the validity of SNA:

- **Network Construction Heuristic**
  Example: Do edges in the network accurately represent relationships between actors in reality?
- **Network Structural Property**
  Example: Does the community structure of a network accurately identify subgraphs of individ-

uals with common goals or interests in reality?

- **Network Metric**
  Example: Do centrality metrics accurately represent the level of influence of an actor in reality?

All source code that implements our survey application is available at a supplementary site: http://github.com/mitchell-joblin/SNA_survey_framework.

Challenges

Developing and deploying a survey for SNA purposes involves a set of unique challenges and requirements that are not currently satisfied by existing survey templates and tools. We now introduce the set of requirements we identified and specifically addressed with our survey application.

Requirement 1: Ease of large scale deployment and collection of responses

Modern social networks can range in size from hundreds to millions of nodes and the survey delivery mechanism should be designed to handle deployment under large scale conditions. A web solution enables the survey participant to easily login to the web interface and submit their responses without the need to download or install any software to complete the survey. Any challenges experienced while participating in a survey create a barrier to completion and likely contribute to lower return-rates and quality responses. In additional to scalability and ease of use, a web solution also allows for aggregation of responses into a common database for later analysis.

Requirement 2: Interactivity

Performing a survey for SNA purposes will often involve the need to display a labeled graph to the survey participants. Research in graph layout and visualization is continually advancing; however, the optimal visualization and layout parameters are dependent on network properties in a non-trivial way. The network size, edge density, graph type (e.g., directed, undirected, weighted, unweighted, one-mode, and two-mode) all influence the optimal visualization. Readability of the graph is necessary for quality responses. To ensure graph details were not obscured by problems such as overlapping nodes or edges, the survey participant should be able to influence a set of visual parameters so that all necessary details of the graph are observable. The adjustable parameters also allow the visually impaired to participate more effectively.

Requirement 3: Dynamic survey content generation

We determined that certain elements of the survey needed to be generated dynamically so that each survey participant would be shown information that was relevant to their particular position in the network. We identified each survey participant through a login process and then computed relevant data such as the subgraph community they were found in and the set of people which we expected to be influential to them. We found this to be a particularly powerful and interesting aspect of our survey because the responses often provided insights about the network that would not have been obvious if we had not shown the relevant network data and instead only asked general questions.

Requirement 4: Integration with existing R infrastructure

One of our primary concerns with developing the survey was the expenditure of effort to prepare our existing SNA analysis pipeline for use in the survey instrument. A substantial amount of support for SNA already exists in R (e.g., **igraph**, **sna**, **graph** etc.), therefore it is highly desirable to seamlessly and effortlessly integrate existing R infrastructure into the survey application. By taking advantage of the Shiny R web application framework, we could avoid a substantial amount of effort to adapt existing R infrastructure to another language or platform for the survey deployment.

Requirement 5: Visually appealing and professional aesthetic

In a preliminary analysis of options for survey platforms, we realized that many of the existing tools did not support a visually appealing or professional aesthetic. We felt that an unprofessional appearance would compromise the seriousness and credibility of the organization hosting the survey and deter survey participation. Perhaps potential survey candidates would perceive the survey poorly

and think that the organization would mishandle the collected data for unethical reasons or via poor execution such that the results would be useless and answering the questions would be futile.

Alternative survey tools

A number of survey tools are available online such as SurveyMonkey (www.surveymonkey.com) and LimeSurvey (www.limesurvey.org), but we found that these tools were not capable of satisfying the requirements for validating SNA techniques. In a canonical survey, a number of predetermined questions are presented to the survey participants and the responses are predetermined categories or free text fields. In the case of predominantly static and predetermined survey content, the features offered by the above tools are more than adequate and customizable. The inadequacies of these tools stem from the lack of features for interfacing with R infrastructure and supporting interactive survey content. Both SurveyMonkey and LimeSurvey have convenient import features that provide a mechanism to display precomputed survey content. Prior to developing our own application, we considered precomputing the survey content for all the possible survey participants and then using the import feature. The problem with this approach was that we did not have a-priori knowledge of the required content and computing all possible variations of the survey content would be incredibly wasteful. When conducting a survey, one typically expects roughly a 10% response rate so computing the necessary data for all potential participants would be roughly 90% waste. This consideration is especially important for researchers working with big data, where there may be potentially millions of survey participants. Furthermore, using this approach would not allow the survey participant to configure any visualization parameters. We found the reactive programming model provided by **shiny** (Chang et al., 2015) to be far more powerful for creating interactive surveys compared to those provided by the alternative survey tools. The added benefit of using **shiny** is that any visualization generated by an R script can be easily converted into a dynamic survey element with just a few lines of code. In contrast, managing a set of precomputed visualization requires potentially vast quantities of storage space and a schema for uniquely identifying the images to be displayed correctly in the survey.

Shiny web application framework

Shiny is a web application framework for R that allows one to easily transform their existing R code into an interactive web application. By using Shiny, we were able to quickly implement a web-based survey instrument without the need to significantly alter our existing R infrastructure for social network analysis. To build a Shiny web application two main components need to be implemented, a 'server.R' file which constructs the R objects to be displayed by the application and a 'ui.R' file to control layout and appearance. Alternatively, one can choose to implement an interface using HTML and CSS to achieve greater flexibility and customization. An example survey question taken from our application is provided in Figure 1. In the following subsections, we introduce the basic elements for implementing the example question including the creation of interactive visualizations using the **shiny** and **igraph** packages.

Example server R script

In our particular implementation we stored the user data in a relational database, as will likely be common in many applications. Below we illustrate the implementation to retrieve a specific person's ego network in the form of an edge list data frame from a MySQL database. The **igraph** package for network analysis is then used to construct the graph object from the edge list data frame and then finally plot it. Using this basic template, one can insert their own network analysis algorithms for a specific purpose. The reactive mechanism for retrieving the interactive user input from the UI is also illustrated for the vertex size and vertex label visualization parameters. In the next section, we will see how the UI is implemented for these particular visualization parameters.

Shiny uses a very powerful reactive programming paradigm to couple the client and server elements to support interactivity. Using this model, *reactive values* represent values that can change over time, and *reactive expressions* represent operations that depend on the use of reactive values. The reactive expressions track the state of reactive values so whenever an update occurs, the dependent reactive expressions are re-executed. The reactive programming concept also supports the important separation of computationally intensive processes from the interactive elements to prevent lag in the user interface. We provide an example server script in Figure 2 to illustrate a basic example for generating interactive survey content using the reactive programming model.

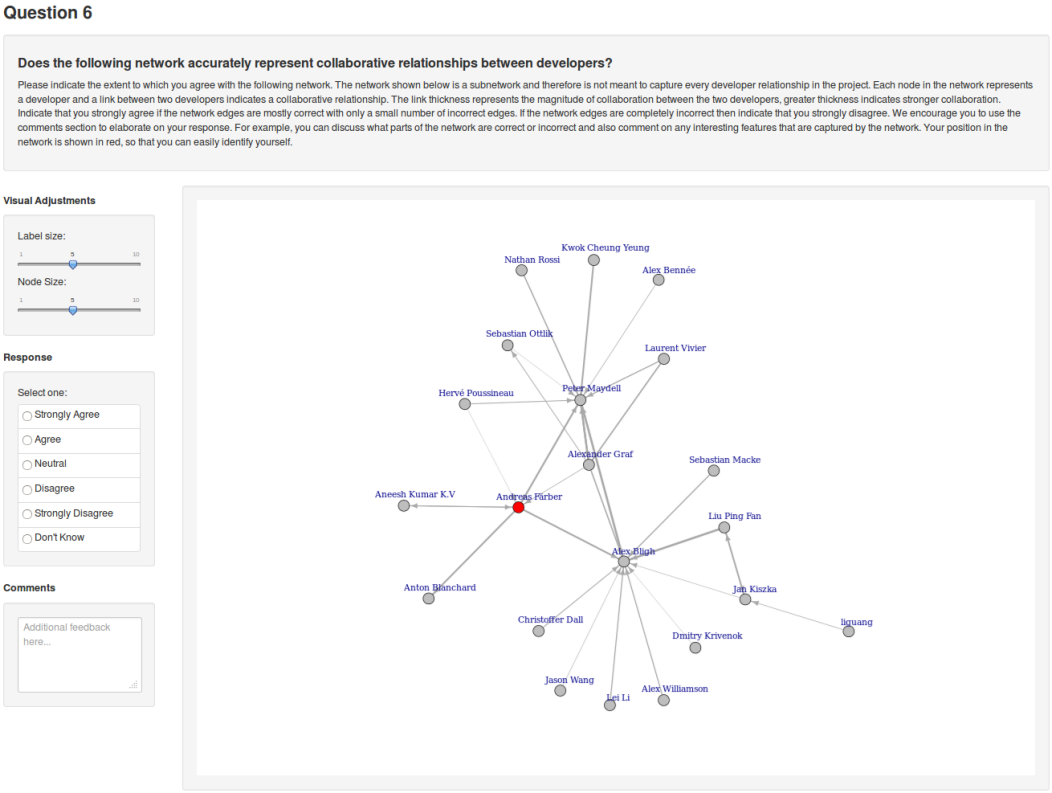

Figure 1: Example survey question.

After first executing a connection to the MySQL database, a reactive expression (`graph.data`) is defined to encapsulate the computationally intensive graph processing algorithms. The reactive expression first retrieves an ID for a specific edge list stored in the database. In example question shown in Figure 1, the ID corresponds to a specific user's community network and in the login phase description we discuss how to identify survey participants using a login process. Next, the edge list corresponding to the ID is retrieved from the database and then any computationally intensive processing is performed. Alternatively, if a database is unavailable, then an archive file or any type of storage format can be loaded into R within this reactive expression.

In the next expression, a reactive endpoint is created. Inside the expression, the reactive values that represent the vertex size (`input$vertex.size`) and label size (`input$label.size`) will cause the reactive endpoint `output$graph` to be evaluated every time an update occurs to one of the visualization parameters. A critical aspect of the implementation is the separation of the computationally intensive SNA algorithms from the visualization parameters. Without the use of reactive conductors (e.g., `graph.data`), the computationally intensive code would be re-evaluated for every update to the visualization parameters and would result in severe lag in the user interface. In the next section, we demonstrate how the binding between the elements in the server script and the user interface is established. More information about the reactive programming model used by shiny server is available in the Shiny Package Documentation.

Example HTML UI

The user interface component of the shiny web application controls the appearance and layout of the survey including all survey questions and response fields. We chose to implement the UI in HTML and CSS using the open-source framework Bootstrap, alternatively one could also implement the UI in R using **shiny**. The advantage to developing the UI in HTML is greater customizability of the look and feel, but the features provided by **shiny** are quite sufficient for most purposes and doesn't require HTML knowledge. With Bootstrap, we were able to achieve a professional aesthetic and maximum flexibility using only the basic features offered by the framework. An example survey question is shown in Figure 3 demonstrating the UI implementation in HTML to display the survey question text, an **igraph** network object with user configurable visualization parameters, and a multi-category response section. This example illustrates all the basic elements of a survey question and all the questions in our example application follow a similar format. Beginning at the top of Figure 3, the

```
library(shiny)
library(igraph)
library(RMySQL)

shinyServer(function(input, output, session, clientData) {
  # Create MySQL connection object
  con <- dbConnect(MySQL(),
                   user = 'USERNAME',
                   password = 'PASSWORD',
                   host = 'HOST',
                   dbname = 'DBNAME')

  # Query database for an edge list and perform SNA
  # and return a reactive conductor
  graph.data <- reactive({
                  # Get unique graph ID from database
                  graph.id <- get.graph.id(con)

                  # Query MySQL database for a specific network
                  edge.list <- query.graph.edges(con, graph.id)

                  # Insert any computationally intensive code for processing
                  # the graph here to avoid being recomputed
                  # for every visualization update})

  # Generate reactive endpoint
  output$graph <- renderplot({
                    edge.df <- graph.data()

                    # Get input from UI for graph label and node size
                    # from reactive sources
                    vertex.size <- input$vertex.size
                    label.size  <- input$label.size

                    # Create igraph graph object and plot
                    g <- graph.data.frame(edge.df)
                    plot(g, vertex.size = vertex.size,
                         vertex.label.cex = label.size)})
}
```

Figure 2: Server R script for example survey question.

survey question is specified inside a Bootstrap "well" element. Next, the visualization parameters for the network are specified as sliders as is shown in Figure 1. Moving further downward, the binding between the graph object provided by the 'server.R' script and the UI is made. Lastly, the response fields "agree" and "disagree" are specified as HTML form inputs. This basic example question can easily be extended to suit the needs of a wide variety of survey questions. For example, one can easily rewrite the question, change the visualization parameters, or introduce different response categories (e.g., five level Likert item). From a technical standpoint, Figure 3 clearly demonstrates how the input elements for dynamically altering the graph visualization are implemented and how the graph plot generated by the 'server.R' script is integrated into the UI.

The binding between the elements of the 'server.R' script and the UI are achieved through the variable identifiers. One can see that the HTML id tags match the corresponding variable identifiers in the 'server.R' script. In this case, the `vertex.size` visualization parameter is displayed as a slider. For categorical or other discrete parameters, drop-down menus can be used instead of a slider when needed. The appearance of the example question rendered in a browser is shown in Figure 1 and includes the graph, the basic visualization adjustments, the categorical response input, and an additional text input for comments.

The above serves as a starting point to construct more elaborate applications and easily integrate

```
<h3>Question 1</h3>
<--Survey question text-->
<div class="well">
  <h4>Does the following network accurately represent collaborative relationships?</h4>
</div>

<--Display network visualization-->
<div class="row-fluid">
  <div class="span2">
    <!--Insert all user configurable graph visualization parameters here-->
    <h5>Visual Adjustments</h5>
    <div class="well">
      <!-- Slider to change vertex size parameter, input id
               matches variable name in server.R-->
      <label class="control-label" for="vertex.size">Vertex Size:</label>
      <input class="jslider"
      data-format="#,##0.#####" data-from="1" data-locale="us" data-round="false"
      data-skin="plastic" data-smooth="false" data-step="1" data-to="10" id="vertex.size"
      name="vertex.size" type="slider" value="5">
    </div>

    <div class="span10">
      <div class="well">
        <!--Insert graph plot, id matches output variable identifier in server.R-->
        <div class="shiny-plot-output" id="graph" style=
        "width: 100%; height: 800px; margin-left:0px; margin-right:0px;
        margin-bottom:0px; margin-top:0px">
        </div>
      </div>
    </div>

    <!--Specify response fields-->
    <div>
      <h5>Response</h5>
      <div class="well">
        <div>
          <label>Select one:</label>
          <table class="table table-condensed table-bordered" style=
          "background-color:white;margin-bottom:0px">
            <tbody>
              <tr>
                <td><label class="radio"><input name="q1a" type="radio" value="agree">
                Agree</label></td>
              </tr>
              <tr>
                <td><label class="radio"><input name="q1a" type="radio" value="disagree">
                Disagree</label></td>
              </tr>
            </tbody>
          </table>
        </div>
      </div>
    </div>
  </div>
</div>
```

Figure 3: HTML UI for example survey question.

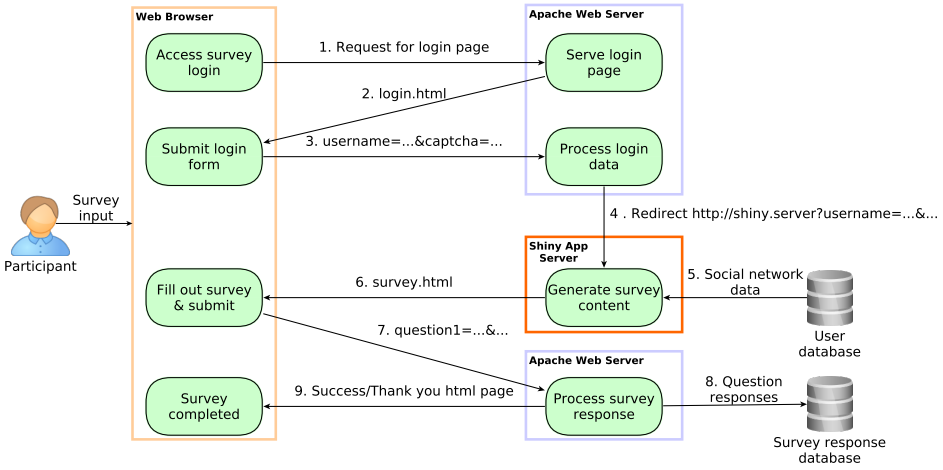

Figure 4: Event sequence and data flow for survey execution.

specific SNA results. For example, one could extend the above code to show only the vertex induced subgraph of nodes with a particularly high centrality or show only the relationships that exist between actors during specific temporal periods using sliders to select the time range of an evolving network.

Survey execution process

We now introduce the execution of the survey application, the data flow, and the general architectural elements used to accomplish the goals of each phase. The survey is broken up into three main phases discussed in detail below, namely the login phase, the survey completion phase, and the response collection phase. The login phase is used to identify the survey participant so that the appropriate visual elements can be generated for this specific user. Next, the shiny application server queries the database storing user data to generate the appropriate survey content for the specific user. Finally, the survey responses are collected and subsequently stored in a relational database so that further analysis can be easily performed. Figure 4 illustrates the sequence of events and the data exchanged between the main architectural elements. An example survey can be found at the following site: `http://rfhinf067.hs-regensburg.de/survey/?p.id=1&c.id=1`.

To design a survey, all that is necessary is to modify the shiny server R script, the PHP script for processing the responses, and the database schema that stores the survey responses. However, our application provides several generic question types and a database schema that should cover most use cases.

Login phase

The login process primarily serves to acquire the necessary user information to generate the appropriate survey content, in most cases only a user name or email address is sufficient. We also include a Captcha authentication to avoid problems associated with automated attempts to access the survey. The login page is hosted on a standard Apache web server and uses basic JavaScript to generate a URL that contains the user information as URL parameters. The login phase is shown as step 1 through 4 in Figure 4. For example, user Jane Doe accesses the login page and enters the email address j.doe@email.com, then the URL `shiny.server?username=j.doe@email.com` is generated and the user is automatically redirected to the generated shiny application after the login form has been submitted and the Captcha has been authenticated. The URL parameters allow you to create a custom survey for each individual survey participant. For example, you may wish to display the participant's ego network and ask specific question about the network's structure or authenticity with regard to capturing real-world relationships.

The login step does not contain security features that would prevent someone from logging into the survey using someone elses email. If security is a primary concern, an alternative solution is to append a unique string of characters as a URL parameter and forgo the login process entirely by instead sending an e-mail to each survey participant containing the unique link. For example, instead of generating the URL shown above from the login process, the participant would receive a link of the form `shiny.server?username=jvnoqk91m`. Then in the shiny server application, the unique string can be matched to the user to create the appropriate survey content.

```
graph <- reactive({
                  ## Parse URL for parameters into key value pairs
                  parseQueryString(clientData$url_search)})

                  ## Retrieve username parameter
                  username <- as.character(query()['username'])

                  ## Query database for subgraph containing the person
                  query.database(username)})
```

Figure 5: Example reactive expression for retrieving URL parameters.

Survey completion phase

The events of this phase are illustrated in steps 5 and 6 of Figure 4. In the survey completion phase, the user is presented with the survey content that is automatically generated based on the individual's login data. This feature allows the questions, response fields, and figures to be altered to present information that is most relevant to the particular survey participant. For example, in our survey we asked participants about whether our community detection algorithm accurately identified meaningful communities and what commonalities were responsible for producing the community. We used the login parameters to select the specific community that the survey participant was found in and displayed the specific subgraph. The URL parameters are retrieved using the code shown in Figure 5 and is located in the 'server.R' script. The example code can be extended to use any value to identify the survey participant. In the login phase description, we discuss a more secure way of performing the login by using a unique string instead of the users name and e-mail address. This code example can easily be adapted to retrieve the unique string from the URL parameters to implement the more secure login approach.

In Figure 5, the URL parameters containing the user login data are first parsed and stored as key value pairs. The user name is then retrieved and used to query the database containing the user data. Using this information we can generate the specific subgraph for each survey participant. This portion of the script can be edited to query for any user specific data. Alternatively, if your user data is stored in files, you can edit this component to read specific user files.

It was our experience that using auto-completion was very helpful for questions where the response was a user name. Since many of the mentioned names should already be in the database, we were able to take advantage of this to provide auto-completion. This allowed us to maintain consistency of names between survey response and our database. In addition, the name consistency made the processing of the responses much easier and yielded higher quality results. Shiny does not provide any auto-completion feature, however, we were able to use ShinySky to achieve this functionality.

Response collection phase

The final phase is illustrated in Figure 4 in steps 7 through 9. Upon completion and submission of the survey, the responses are sent using HTTP POST to a PHP script running on the Apache server. The PHP script checks for errors in the responses and cleans erroneous or troublesome characters. Once the responses have been cleaned and verified they are then saved to the database. Once the data have been successfully saved to the database, the user is redirected to a "thank you" page to confirm that their submission was successful. You may wish to edit the PHP script and database schema to suit your particular needs since they relate to the specific survey questions. Alternatively, you may simply write the survey responses to a file. Since R has sufficient support for MySQL, we found that saving the responses to a database made the analysis of the responses relatively simple.

Discussion and future work

In our research focused on validating social network analysis techniques for the purpose of identifying the organizational structure of open source developers (Joblin et al., 2015), we found this survey application to be extremely valuable. Our research goal was to determine if developer networks, constructed from operational data stored in the version control system, contain information that is reflective of the real world. More specifically, the survey results indicated that community-detection techniques are effective for decomposing developer networks into communities that reflect important real-world relationships. We were able to show that the developer communities are comprised of a

collection of well coordinated individuals with common goals, interests, and shared mental models. In the software engineering domain, this kind of information is incredibly valuable for identifying where coordination challenges are most likely to arise and negatively impact developer productivity or code quality (Cataldo and Herbsleb, 2013; Pinzger et al., 2008; Nagappan et al., 2008; Meneely et al., 2008), effective coordination are becoming increasingly important. With the support of R packages, applying unsupervised machine learning algorithms to networks is no longer a significant technical challenge, but the challenge of evaluating the results with respect to the ground truth model of reality still requires attention. In particular, without demonstrating that the results of applying SNA have real-world significance and validity, our research contribution would have had only a very limited impact. On this basis, we see that the challenge of validating the information that can be gleaned from social networks is fundamental to developing effective network analysis methodology that is capable of delivering valuable insights. Since the phenomenon of globally distributed collaboration is becoming common practice in many domains, we see this survey application as an important contribution to the general research community. In the future, we fully intend to reuse and improve this web-based survey application with one of our primary goals being to turn the application into an R package that is easier to setup and use.

Conclusion

Through the use of multiple R packages, we were able to successfully develop a powerful survey instrument for validating SNA approaches. Surveys for the validation of SNA techniques demand unique features that are not currently provided by existing tools. Our application provides the fundamental structural elements to conduct a large scale and fully-automated social-network based survey. Furthermore, we provide a fully-functional application that addresses the fundamental threats to validity in the social network analysis domain. Additionally, the basic application provides a substantial foundation to support the development of more elaborate survey instruments.

The SNA community is rapidly growing with many new and exciting techniques frequently being proposed, however, the real power of SNA lies within its accurate representation of the real-world. In order to achieve congruency between reality and the insights gleaned through SNA, appropriate validation tools are a necessary requirement. In this paper, we presented an example application that significantly lowers the barrier to entry for conducting social-network based surveys. By using this survey application to validate SNA techniques, one is able to more easily test the reliability and validity of SNA approaches and increase the scientific rigor of the field.

Bibliography

Y.-Y. Ahn, S. Han, H. Kwak, S. Moon, and H. Jeong. Analysis of topological characteristics of huge online social networking services. In *Proceedings of the International Conference on World Wide Web*, pages 835–844. ACM, 2007. [p149]

P. Barbera and M. Piccirilli. *Rfacebook: Access to Facebook API via R*, 2015. URL http://CRAN.R-project.org/package=Rfacebook. R package version 0.6. [p149]

C. T. Butts. *sna: Tools for Social Network Analysis*, 2014. URL http://CRAN.R-project.org/package=sna. R package version 2.3-2. [p149]

M. Cataldo and J. D. Herbsleb. Coordination breakdowns and their impact on development productivity and software failures. *IEEE Transactions on Software Engineering*, 39(3):343–360, 2013. [p157]

W. Chang, J. Cheng, J. Allaire, Y. Xie, and J. McPherson. *shiny: Web Application Framework for R*, 2015. URL http://CRAN.R-project.org/package=shiny. R package version 0.12.2. [p151]

G. Csardi and T. Nepusz. The igraph software package for complex network research. *InterJournal*, Complex Systems:1695, 2006. URL http://igraph.org. [p149]

J. Donath and D. Boyd. Public displays of connection. *BT Technology Journal*, 22(4):71–82, 2004. [p149]

R. Gentleman, E. Whalen, W. Huber, and S. Falcon. *graph: A package to handle graph data structures*. R package version 1.44.1. [p149]

J. Gentry. *twitteR: R Based Twitter Client*, 2015. URL http://CRAN.R-project.org/package=twitteR. R package version 1.1.9. [p149]

M. Joblin, W. Mauerer, S. Apel, J. Siegmund, and D. Riehle. From developer networks to verified communities: A fine-grained approach. In *Proceedings of the International Conference on Software Engineering*, 2015. [p149, 156]

R. Kumar, J. Novak, and A. Tomkins. Structure and evolution of online social networks. In *Link Mining: Models, Algorithms, and Applications*, pages 337–357. Springer, 2010. [p149]

A. Meneely, L. Williams, W. Snipes, and J. Osborne. Predicting failures with developer networks and social network analysis. In *Proceedings of the Foundations of Software Engineering*, pages 13–23. ACM, 2008. [p157]

A. Mislove, M. Marcon, K. P. Gummadi, P. Druschel, and B. Bhattacharjee. Measurement and analysis of online social networks. In *Proceedings of the Conference on Internet Measurement*, pages 29–42. ACM, 2007. [p149]

N. Nagappan, B. Murphy, and V. Basili. The influence of organizational structure on software quality: An empirical case study. In *Proceedings of the International Conference on Software Engineering*, pages 521–530. ACM, 2008. [p157]

M. Pinzger, N. Nagappan, and B. Murphy. Can developer-module networks predict failures? In *Proceedings of the International Symposium on Foundations of Software Engineering*, pages 2–12. ACM, 2008. [p157]

C. Wilson, B. Boe, A. Sala, K. P. Puttaswamy, and B. Y. Zhao. User interactions in social networks and their implications. In *Proceedings of the European Conference on Computer Systems*, pages 205–218. ACM, 2009. [p149]

Mitchell Joblin
Siemens AG
Wladimirstraße 3
91058 Erlangen
Germany
mitchell.joblin.ext@siemens.com

Wolfgang Mauerer
Siemens AG
Technical University of Applied Science Regensburg
Universitätsstraße 31
93058 Regensburg
Germany
wolfgang.mauerer@oth-regensburg.de