Identifying Significant Components of Complex Software for Improving Reliability: Using Invocation Relationships and Component Characteristics

Lixing XUE, Zhan ZHANG and Decheng ZUO

School of Computer Science and Technology, Harbin Institute of Technology, Harbin 150001, China

zz@ftcl.hit.edu.cn

Abstract. The scales of software systems are getting larger and larger, which may make the system reliability become low. To improve their reliability, a set of important components that are able to strongly influence the system reliability are usually determined first. However, the existing determining approaches only consider the components which are often called by other components as significant ones, but overlook the components which often call others also have a strong effect on the system. Besides, the approaches all require component invocation probabilities which cannot be obtained easily in large-scale systems. To attack the problems, we propose a novel approach for identifying the significant components in complex systems. This approach includes two component ranking algorithms, which take into account not only the components that are frequently invoked, but also the components which often invoke others. The two algorithms, which do not require component invocation probabilities, can either make significant component ranking only based on the component invocation relationships or consider not only the component invocation relationships but also the component characteristics to achieve results. The significant components are selected according to the two ranking results. Extensive experiments are provided to evaluate the approach and draw comparisons with existing methods.

1. Introduction

Nowadays, various forms of computers are everywhere in our daily life -- desktop computers or laptops we use, smart phones, electronic control units in cars and embedded systems in intelligent electrical appliances, and this offers tremendous gains in convenience. However, any failure of these computers may bring great impact to people, leading the reliability to be a particular concern. As an indispensable part of a computer, software is becoming large scale and rather complex. Unfortunately, a large quantity of components and their complicated invocation relationships incur a huge threat to the system reliability and cause system failures easily. To find out an efficient method for improving the reliability of complex software has been an urgent and practical problem.

Improving software reliability is not a thoroughly new topic. There are four main methods in traditional software reliability engineering: fault prevention, fault removal, fault tolerance, and fault forecasting [1]. However, due to the massive number of components in a large-scale software system, any of these methods can be applied to all components. It will incur too much cost and it is not efficient. In fact, there is no need to do so, because the 80-20 rule also acts in software reliability. Microsoft pointed out that 80% of failures and crashes in Windows and Office can be ascribed to 20%
of the most reported bugs [2]. Therefore, identifying the significant components that impact the system reliability much more than others, is the first job to optimize the reliability of complex software.

For small-scale software, performing sensibility analysis based on a Markov model can be employed to locate the significant components that impact the system reliability much more than others [3-5]. But utilizing this classic approach to a large-scale system will lead to the state-space-explosion problem. Researchers have attempted to attack this issue from other perspectives [6-10]. FTCloud [6], which is one of the ideas, performs component ranking by invoking frequencies and then find out a set of important components which are able to impact the system reliability much more than other components. FTCloud2 [7], a refined work of FTCloud, adds attention on the component characteristics, namely components are divided into noncritical components or critical ones. However, there are still two issues in these component ranking approaches. On one hand, they take into account only components that are frequently invoked by other components, but ignore components that frequently invoke other components. On the other hand, the approach requires stable usage profiles (i.e. component invocation probabilities), which need to be recorded during a long period and cannot be obtained when building new complex software.

To address the problems, we put forward a new component ranking approach for identifying significant components in complex software. The approach utilizes the invocation relationships as well as the component characteristics to performing component ranking twice. One performs the ranking by the impact of components that are frequently called, and the other does component ranking by the effect of components that often invoke others. According to the two ranking results, a set of the significant components that can greatly influence the system reliability are determined. Compared to the existing approaches, the proposed method which does not require component invocation probabilities takes into account the component characteristics and is able to achieve better results. This method also can be used to locate the most significant components when designing the architecture.

The rest of the paper is organized as follows. Structures of software systems are modelled in Section 2. Section 3 details the two component ranking algorithms and discusses how to determine significant components. Massive experiments are provided to evaluate our approach within Section 4. At last, we conclude the paper.

2. Component Invocation Graph

The structure of a software system, i.e. the component invocation relationships, can be modelled by a directed graph $G=\langle C, E \rangle$, where an element $c_i \in C$ denotes a component in the software, and a directed link $e_{ij} \in E$ from a node $c_i$ to another node $c_j$ represents a component invocation relationship, i.e. $c_i$ invokes $c_j$.

Each component invocation graph containing $n$ components is corresponding with an $n \times n$ adjacent matrix $A = \left[a_{ij}\right]_{n \times n}$, where each entry $a_{ij}$ indicates whether there is a directed link from node $c_i$ to node $c_j$. If the directed link exists, the value of $a_{ij}$ is set to be 1, otherwise it is 0. This can be formulized as:

$$a_{ij} = \begin{cases} 1 & \text{if } e_{ij} \in E \\ 0 & \text{else} \end{cases} \quad (1)$$

A directed link ending at node $c_i$ is called an incoming link of $c_i$ and the number of incoming links of $c_i$ is named its indegree $\deg^+(c_i)$. $\deg^+(c_i)$ is able to be calculated by:

$$\deg^+(c_i) = \sum_{k=1}^{n} a_{ki} \quad (2)$$
A directed link starting from node $c_i$ is named an outgoing link of $c_i$ and the number of outgoing links of $c_i$ is called its outdegree $\text{deg}^-(c_i)$. $\text{deg}^-(c_i)$ can be computed by:

$$\text{deg}^-(c_i) = \sum_{k=1}^{n} a_{ik}$$  \hspace{1cm} (3)

For a component $c_i$, the number of its parent nodes is $\text{deg}^+(c_i)$, indicating that a total of $\text{deg}^+(c_i)$ component will invoke it, while the number of its children nodes is $\text{deg}^-(c_i)$, showing that it will invoke a total of $\text{deg}^-(c_i)$ components in the software system.

3. Ranking and Significant Component Determining

Section 3.1 details one of the algorithms and Section 3.2 introduces the other one. Then determination of significant components which are able to greatly impact on the system reliability is discussed in Section 3.3.

3.1. Component Ranking Inspired by PageRank

In a component invocation graph, some components may have larger indegrees than other components. We intuitively feel that these components may be frequently invoked by others and therefore they are able to influence the global reliability of this software much more than other components [3]. In terms of the principle of PageRank [11], an algorithm is put forwarded to score the significance of the software components.

Considering a system which has $n$ components, each component $c_i$, $i = 1, 2, L, n$, with $n \in \mathbb{N}$ has a corresponding a significance value $VP(c_i)$ in the range of $[0,1]$. $VP(c_i)$ can be calculated by:

$$VP(c_i) = (1 - \alpha) f(c_i) + \alpha \sum_{c_j \in E} \frac{VP(c_j)}{\text{deg}^-(c_j)}$$  \hspace{1cm} (4)

where $f(c_i)$ is the basic significance value of $c_i$ itself, while $\sum_{c_j \in E} \frac{VP(c_j)}{\text{deg}^-(c_j)}$ is the significance value derived from other components which are invoked by $c_i$. The parameter $\alpha$ ($0 \leq \alpha \leq 1$) is used to balance the derived value and the basic value, which is usually set as 0.85 [6, 7]. The basic significance value $f(c_i)$ is defined as

$$f(c_i) = \begin{cases} 
\beta & \text{if } c_i \text{ is a critical component} \\
\frac{1 - \beta}{|NC|} & \text{if } c_i \text{ is a noncritical component}
\end{cases} \hspace{1cm} (5)
$$

In Formula (5), $|C|$ and $|NC|$ denote the quantity of critical components and noncritical components, respectively. The parameter $\beta$ ($\frac{|C|}{n} \leq \beta \leq 1$) balances the weight of the two category of components. If $\beta = \frac{|C|}{n}$, this approach does not consider the component characteristics at all and the ranking result is only determined by invocation relationships.

Formula (4) can be written in vector form:
where the matrix $T = [t_{ij}]_{n \times n}$ is defined by:

$$t_{ij} = \begin{cases} \frac{1}{\text{deg}(c_i)} & \text{if } e_{ij} \in E \\ 0 & \text{else} \end{cases}$$

Formula (6) can be solved via calculating the eigenvector with eigenvalue 1.

This algorithm ranks the components only based on the component invocation relationships if the critical components cannot be determined. Once they are provided, the algorithm is able to achieve more accurate ranking results. The algorithm outputs the significance values of all the components and a component that has a larger value has the power impacting the global reliability much more than the one with a small value.

### 3.2. Component Ranking Inspired by TrustRank

Opposite to the condition discussed in Section 3.1, a component invocation graph may have larger outdegrees than other components. These components seem to call others frequently and their failures may have influence on many subsequent components. Therefore, the components should also be considered to be significant. According to the thought of inverse PageRank in TrustRank [12], we propose another method to estimate the significance of the software components.

Assuming that a software system has $n$ components, a significance value $VT(c_i)$ is defined for each component $C_i$, which is in the range of $[0,1]$. $VT(c_i)$ can be obtained by:

$$VT(c_i) = (1-\alpha)f(c_i) + \alpha \sum_{c_j \in k} \frac{VT(c_j)}{\text{deg}^+(c_j)}$$

where $f(c_i)$ is the basic significance value of $c_i$ itself, while $\sum_{c_j \in k} \frac{VT(c_j)}{\text{deg}^+(c_j)}$ is the significance value derived from other components which are invoked by $c_i$. The definitions of $\alpha$ and $f(c_i)$ are the same as and those in Section 3.1.

Formula (8) can be written in vector form:

$$
\begin{bmatrix}
VT(c_1) \\
VT(c_2) \\
\vdots \\
VT(c_n)
\end{bmatrix} = (1-\alpha)
\begin{bmatrix}
f(c_1) \\
f(c_2) \\
\vdots \\
f(c_n)
\end{bmatrix} + \alpha
\begin{bmatrix}
VT(c_1) \\
VT(c_2) \\
\vdots \\
VT(c_n)
\end{bmatrix}
\begin{bmatrix}
\text{M} \\
\text{M} \\
\vdots \\
\text{M}
\end{bmatrix}
$$

where the matrix $U = [u_{ij}]_{n \times n}$ is defined by:

$$u_{ij} = \begin{cases} \frac{1}{\text{deg}^+(c_i)} & \text{if } e_{ij} \in E \\ 0 & \text{else} \end{cases}$$

In this way, it can be solved by calculating the eigenvector with eigenvalue 1.

Similarly, this algorithm is able to rank the components only using the component invocation relationships as well as to achieve more accurate ranking results when adding the information of component characteristics. The algorithm also gives the significance values of all the components as the output and a component that has a larger value has more influence on the system reliability.


3.3. Significant Components Determining

On the basis of the above two algorithms, two series of significance values of all the components in the software are able to be obtained. One focuses on the components which are frequently invoked and the other pays attention on the components that frequently calls others. The two ranking results are both effective and valuable.

Therefore, Top-\(\frac{K}{2}\) \((0 < K \leq 100)\) components are respectively selected from the two ranking results and a total of \(K\%\) most significant components are determined.

4. Experiments

A lot of experiments are provided in this section in order to validate the proposed approach, estimate the impact of the parameters and draw performance comparisons between our approach and FTCloud / FTCloud2.

4.1. Experimental Setup

To validate the effectiveness and efficiency of the approach proposed in this paper, we compare seven approaches, which are as follows:

- RandSel. We randomly choose \(K\) percent components and then improve their reliability by applying fault-tolerance strategies.
- OurID1. Top-\(K\) percent most important components are identified by the proposed method, which only utilizes the invocation relationships. Then their reliability is improved.
- OurID2. Top-\(K\) percent most important components are identified by the proposed method, which adds the consideration on the component characteristics. Then their reliability is improved.
- FTCloud. Top-\(K\) percent most important components are by FTCloud. Then their reliability is improved.
- FTCloud2. Top-\(K\) percent most important components are by FTCloud2. Then their reliability is improved.

A graph whose degree distribution exhibit a power law is called a scale-free graph. A lot of previous works show that not only many networks but also the program structures, such as MySQL and Linux Kernel, appear to be approximate scale-free properties [13, 14]. Similarly to [6, 7], we also apply Pajek [15], which is network analysis software, to randomly draw scale-free directed graphs as component invocation graphs for the experimental studies and draw performance comparisons. In the section, three graphs which have 500, 1000 and 2000 nodes respectively are generated.

In is section, \(inf\) denotes the internal failure probability of the components, including two initial settings -- 0.05 and 0.1. For fair comparisons, we suppose that the internal failure probability \((inf)\) can be reduced by 30 percent after improving the reliability.

In OurID1, OurID2, FTCloud and FTCloud2, the parameter \(\alpha\) is used to balance the weight of the derived significance and the basic significance the component has itself. Similar works [6, 7] have proved that 0.85 is an optimal value. Thus, in this section, we also set the parameter \(\alpha\) to be 0.85.

In Section 4.2, OurID1 and RandSel are compared to validate the identification by the proposed method. The impact of Top-\(K\) and component internal failure probability is discussed in Section 4.3 and 4.4 respectively. Performance comparison between OurID1 and FTCloud is drawn within Section 4.5. In this comparison, the component characteristics are not involved. In Section 4.6, OurID2 and FTCloud2 are compared when considering the component characteristics.

4.2. Validity of Identification

To validate the significant components identified by our approach have greater power impacting on the system, RandSel and OurID are compared in each graph respectively with different \(K\) values. Table 1 lists the experimental results of software reliability.
Table 1. Validity of Identification.

| Node Numbers | Component intf=0.05 | Component intf=0.1 |
|--------------|---------------------|--------------------|
| Methods      | Top2%   | Top6%   | Top10%  | Top20% | Top2%   | Top6%   | Top10%  | Top20% |
| 500 RandSel  | 0.7913  | 0.7934  | 0.7988  | 0.8109 | 0.6281  | 0.6312  | 0.6397  | 0.6591 |
| OurID1       | 0.7625  | 0.7634  | 0.7642  | 0.7678 | 0.5870  | 0.5882  | 0.5893  | 0.5947 |
| 1000 RandSel | 0.7625  | 0.7634  | 0.7642  | 0.7678 | 0.5870  | 0.5882  | 0.5893  | 0.5947 |
| OurID1       | 0.7206  | 0.7322  | 0.7390  | 0.7494 | 0.5450  | 0.5681  | 0.5817  | 0.6029 |
| 2000 RandSel | 0.6964  | 0.6984  | 0.6972  | 0.7059 | 0.4988  | 0.5025  | 0.5003  | 0.5164 |
| OurID1       | 0.7206  | 0.7322  | 0.7390  | 0.7494 | 0.5450  | 0.5681  | 0.5817  | 0.6029 |

Table 1 shows that OurID1 provides better reliability performance than RandSel, in all experimental settings. This phenomenon shows that improving the reliability of significant components identified by OurID1 can get higher system reliability than enhancing the reliability of randomly selected components. Obviously, OurID1 is able to find out significant components that have stronger effect on the global reliability.

As the K value grows, the reliability provided by OurID1 increases monotonously, while the reliability provided by RandSel may or may not increase. This result indicates that it is effective to apply OurID1 to locate important components.

With the growths of node number, OurID1 is able to consistently provide better the reliability performance. This observation suggests that OurID1 is capable of effectively identifying significant components in different sizes of software systems.

When the component internal failure probability changes from 0.05 to 0.1, OurID1 can consistently provide higher reliability than RandSel, indicating that OurID1 is free from component internal failure probabilities.

4.3. Impact of K

To demonstrate that the parameter K has no impact on the identification results, we compare RandSel and OurID1 under multiple K values and record the reliability results in Table 2. The node number is 1000.

Table 2. Impact of Top-K% on System Reliability.

| Component intf | Methods | Values of Top-K% |
|----------------|---------|------------------|
|                | Top10%  | Top20% | Top30% | Top40% | Top50% | Top60% | Top80% | Top100% |
| 0.05 RandSel   | 0.7642  | 0.7678  | 0.7809  | 0.7930  | 0.8380  | 0.8602  | 0.8680  | 0.8718  |
| OurID1         | 0.8308  | 0.8438  | 0.8508  | 0.8561  | 0.8604  | 0.8640  | 0.8687  | 0.8718  |
| RandSel        | 0.5947  | 0.6142  | 0.6230  | 0.6951  | 0.7413  | 0.7559  | 0.7627  |          |
| OurID1         | 0.6950  | 0.7159  | 0.7274  | 0.7359  | 0.7430  | 0.7488  | 0.7567  | 0.7627  |

When the K value is set as not less than 80, using OurID1 can obtain better reliability performance than using RandSel regardless of the component internal failure probabilities. When K reaches 100, there is no difference between the two methods, since all the components in either methods are selected. This observation demonstrates that K values do not have impact on OurID1.

4.4. Impact of Component Internal Failure Probability

To show that OurID1 is free from the component internal failure probability (intf) settings, OurID1 and RandSel are compared respectively, under component internal failure probability values ranging.
from 0.01 to 0.1 with a step value of 0.01. In this experiment, the node number is 1000 too, and four K values (2, 6, 10 and 20) are used. The results are depicted Figure 1.

As shown in Figure 1, employing OurID1 can get higher reliability than using RandSel, no matter what the K value is and how \( intf \) changes. This result suggests that different component internal failure probability settings have no impact on OurID1.

![Figure 1. Impact of Component Internal Failure Probability.](image)

4.5. Performance Comparison without FTCloud

FTCloud requires the invocation relationships as well as the component invocation probabilities while our approach does not need the component invocation probabilities and ranks the components. Furthermore, our approach takes into account both the components which are frequently invoked and the ones which frequently invoke other components, while FTCloud takes into account only the former.

The comparative experiments are conducted on three settings of node numbers (500, 1000 and 2000) respectively with various K values. The initial component internal failure probability is set as 0.1 (\( intf = 0.01 \)). The experiment results of software system reliability are illustrated in Table 3.
Table 3. Performance Comparison with FTCloud.

| Node | Methods | Top2% | Top4% | Top6% | Top10% | Top20% | Top30% | Top40% | Top50% |
|------|---------|-------|-------|-------|--------|--------|--------|--------|--------|
| 500  | OurID1  | 0.6543| 0.6696| 0.6786| 0.7120 | 0.7343 | 0.7516 | 0.7619 | 0.7711 |
|      | FTCloud | 0.6544| 0.6696| 0.6753| 0.6955 | 0.7106 | 0.7218 | 0.7385 | 0.7639 |
| 1000 | OurID1  | 0.6476| 0.6746| 0.6837| 0.6950 | 0.7159 | 0.7274 | 0.7359 | 0.7430 |
|      | FTCloud | 0.6476| 0.6745| 0.6804| 0.6857 | 0.7013 | 0.7084 | 0.7178 | 0.7261 |
| 2000 | OurID1  | 0.5450| 0.5579| 0.5681| 0.5817 | 0.6029 | 0.6047 | 0.6066 | 0.6083 |
|      | FTCloud | 0.5363| 0.5468| 0.5579| 0.5683 | 0.5855 | 0.5902 | 0.5960 | 0.6001 |

When the node number is 500, OurID1 provides better reliability performance than FTCloud under all the Top-K% settings except 2%. When the node number grows to 1000, the reliability performance of OurID1 is better than that of FTCloud under all the Top-K% settings. When the node number reaches 2000, OurID1 consistently outperforms FTCloud. This observation indicates that OurID1 is more efficient than FTCloud in larger software systems.

When the node number is smaller and the Top-K% value is set to be small, that is only a few (may be less than 30) significant components are selected, the performance of OurID1 is a little worse than FTCloud, but the difference is very small. Nevertheless, it is acceptable, because OurID1 requires less information and is able to be applied more widely. FTCloud requires stable component invocation probabilities and can only be used to optimize the systems which have been running after a period of time while OurID1 works only by employing component invocation relationships and can be applied as long as system structure information is provided.

4.6. Performance Comparison with FTCloud2

Compared to FTCloud, FTCloud2 introduces the component characteristics, but still ignores the impact of the components that frequently call others on the system reliability. The method put forward in this paper allows users to rank components taking into account the component characteristics and requires less information than FTCloud2.

The comparisons between our method and FTCloud2 are performed on three settings of node numbers (500, 1000 and 2000) respectively using multifarious K values. The initial component internal failure probability is set as 0.1 ($inf = 0.1$), too. The parameter $\beta$ is set as 0.7 [7]. The experiment results of Table 4 records the experiment results of the system global reliability.

Table 4. Performance Comparison with FTCloud2.

| Node | Methods | Top2% | Top4% | Top6% | Top10% | Top20% | Top30% | Top40% | Top50% |
|------|---------|-------|-------|-------|--------|--------|--------|--------|--------|
| 500  | OurID2  | 0.634 | 0.6494| 0.6582| 0.6917 | 0.7139 | 0.7315 | 0.7411 | 0.7511 |
|      | FTCloud2| 0.6341| 0.6493| 0.6551| 0.675  | 0.6903 | 0.7016 | 0.7181 | 0.7431 |
| 1000 | OurID2  | 0.6341| 0.6611| 0.6703| 0.6818 | 0.7022 | 0.7141 | 0.7227 | 0.7399 |
|      | FTCloud2| 0.634 | 0.6609| 0.6669| 0.6725 | 0.688  | 0.6953 | 0.7048 | 0.713  |
| 2000 | OurID2  | 0.5367| 0.5495| 0.5598| 0.5735 | 0.5947 | 0.5965 | 0.5985 | 0.6003 |
|      | FTCloud2| 0.5279| 0.5385| 0.5497| 0.5601 | 0.5772 | 0.5819 | 0.588  | 0.5921 |

Similarly to the results in Section 4.5, OurID2 outperforms FTCloud2 in all the experimental settings except the node number and the Top-K% are both rather small. OurID2 employs only the invocation relationships to achieve better results than FTCloud2 which utilizes not only the invocation relationships but also the component invocation probabilities.
5. Conclusion & Future Work

We introduce a ranking-based approach for identifying the significant components in complex software. This approach includes two component ranking algorithms proposed from two different perspectives. One regards the components that are often invoked by others as significant components. The other views the components that frequently call others as significant components. The components in software are ranked twice by employing the two algorithms and then the top K percent most important ones are selected in terms of the two ranking results. The effectiveness and efficiency of the approach are validated by a lot of experiments. Compared with FTCloud and FTCloud2, our approach requires less information but is able to provide better performance in most cases, and is able to be employed when designing the system architecture.

The determination of significant components in this approach treats the two ranking results equally, but there may be a better method to select the important components. This will be studied in the future. The future work also includes more experimental analysis of actual software systems and component ranking considering more factors.

References

[1] Lyu M R 1996 Handbook of Software Reliability Engineering (New York: McGraw-Hill)
[2] Rooney P 2002 Microsoft’s CEO: 80-20 Rule Applies to Bugs, Not Just Features ChannelWeb
[3] Cheung R C 1980 A User-Oriented Software Reliability Model IEEE Transactions on Software Engineering SE-6 118-125
[4] Cortellessa V and Grassi V 2007 A Modeling Approach to Analyze the Impact of Error Propagation on Reliability of Component-Based Systems 2007 Int. Symp. on Component-Based Software Engineering (Springer Berlin Heidelberg) 140-156
[5] Pham TT and Xefago D 2012 Reliability Prediction for Component-Based Systems: Incorporating Error Propagation Analysis and Different Execution Models 2012 12th Int Conf. on Quality Software (IEEE) 106-115
[6] Zheng Z, Zhou TC, Lyu MR and King I 2010 FTCloud: A Component Ranking Framework for Fault-Tolerant Cloud Applications 2010 IEEE 21st Int. Symp. on Software Reliability Engineering (IEEE) 398-407
[7] Zheng Z, Zhou T C, Lyu M R and King I 2012 Component Ranking for Fault-Tolerant Cloud Applications IEEE Transactions on Services Computing 5 540-550
[8] Arianyan E, Ahmadi M R and Maleki D 2016 A Novel Taxonomy and Comparison Method for Ranking Cloud Computing Software Products International Journal of Grid and Distribution Computing 9 173-190
[9] Naji-Hasan A H and Shu G O 2014 A Service Profile Based Similarity Framework for Web Service Discovery Telkomnika (Telecommunication Computing Electronics and Control) 12 1079-1087
[10] Lu G, Hai Y and Sun Y 2014 A Reliable Web Services Selection Method for Concurrent Requests Telkomnika (Telecommunication Computing Electronics and Control) 12 1053-1063.
[11] Page L, Brin S, Motwani R and Winograd T 1998 The PageRank Citation Ranking: Bringing Order to the Web Stanford InfoLab Report number:1999-66
[12] Gyongyi Z, Garcia-Molina H and Pedersen J 2004 Combating Web Spam with Trustrank Proc. of the 13th Int. Conf. on Very Large Databases (ACM) 576-587
[13] de Moura A P S, Lai Y C and Motter A E 2003 Signatures of Small-World and Scale-Free Properties in Large Computer Programs Physical Review E 68 017102
[14] Myers C R 2003 Software Systems as Complex Networks: Structure, Function and Evolvability of Software Collaboration Graphs Physical Review E 68 046116
[15] de Nooy W, Mrvar W and Batagelj V 2011 Exploratory Social Network Analysis with Pajek (New York: Cambridge University Press)