2017 International Conference on Applied Mechanics and Mechanical Automation (AMMA 2017)
ISBN: 978-1-60595-471-4

Digraph-based Requirement Prioritization for Comparisons Reduction

Wan-yu CHEN*, Hong-hui CHEN, Tao CHEN and Fei CAI

Science and Technology on Information Systems Engineering Laboratory,
National University of Defense Technology, Changsha, China

*Corresponding author

Keywords: Requirement prioritization, Requirement directed graph, Comparison reduction, Dominance test.

Abstract. The increasing size of requirements is challenging to requirement prioritization in a large scale software engineering projects. Most existing requirement prioritization methods have high accuracy but suffer a problem of scalability. To overcome the aforementioned shortcomings, this paper proposes a prioritization approach based on Requirement digraph (R-digraph). Via a transformation from requirements to nodes and from prioritization relationships to edges, we propose one algorithm to build the R-digraph with less human interactions, and another algorithm is designed to perform the dominance test and to output the final rank of requirements. A comprehensive experiment is conducted to evaluate the performance of our proposal. The results show that our approach achieves reliable performance in terms of less human interaction than the state-of-the-art.

Introduction

In the development of a commercial software system, an increasing demand for prioritizing requirement candidates is observed [8]. In the requirement prioritization process, the human-participate comparison exhausts the majority of execution time and workload. Hence, decreasing the human-participate comparison becomes a chief way of alleviating the burden of requirements prioritization. Most requirement prioritization methods conduct a pair-wise comparison, like analytic hierarchy process (AHP), hierarchy AHP and bubblesort [2, 7, 8], to rank the requirements. Such methods introduce many redundant human comparisons which can lead to a relatively accurate result [3, 5, 9].

Hence, in this paper, we propose a prioritization approach, i.e., R-digraph, based on a newly-built requirement digraph. Our method aims to decrease the human-participate comparisons and thus to improve the applicability of requirement prioritization. In a requirement digraph, the nodes denote the requirements and the directed edges denote the priority. We use an example to indicate the priority relationship between two requirements, i.e., $R_1 \geq R_2$ means R_1 is preferred than R_2. In our approach, the initial graph is built with two randomly selected connected-nodes from the requirement set and an edge between them. Then we take one node from the rest requirements and add the corresponding edges based on a human-engaged comparison at one time until all requirements are included. Such an incremental approach has the following research questions to be answered.

- **How to Build a Required Digraph with Less Comparisons?**

Once a node is newly added when incrementally building the requirement digraph, we aim to get as much prioritization information as possible. Thus, we argue that the node to be compared should be distinctively selected so as to provide more information than others. More information a comparison can provide results in less comparisons needed and better applicability of the corresponding approach. Intuitively, nodes with larger degree are often associated with more priority information and thus eliminate more uncertainty of requirement priority. Such intuition can be explained by the information entropy which can be calculated for each node in a digraph. As a consequence, a heuristic algorithm is proposed to select nodes for comparing with newly added nodes.

- **When Can We Output the Final Ranked List?**

When there are comparability between every two nodes according to the relation transitivity of a digraph, we can output the final ranked list. In other words, once all requirements are included in the digraph as nodes, we should examine whether each pair of two nodes has been assigned a priority or preference relationship. To solve this problem, we propose a dominance test algorithm to find pairs of nodes where the preference has not been assigned.

The contributions of the paper are as follows: (1) we propose a new requirement prioritization approach R-digraph. By adding nodes incrementally to a digraph, a ranked list of requirements can be generated. To the best of our knowledge, we are the first to use graph model in the field of requirement prioritization; (2) we conduct a comprehensive experiment and the results show that our proposed R-digraph method can significantly reduce the number of comparisons, which means that our proposal can be injected into some practical applications.

The R-digraph Prioritization Method

The Definition of Requirement Digraph

Definition: *Requirement digraph*: If we transform the requirements into the nodes in a directed graph, and the edges indicates the priority relations between requirements with directions from lower to the prior, we call the directed graph a requirement digraph.

We use $G=(V,E)$ to identify a requirement digraph. V denotes the set of nodes in G while E denotes the set of edges in G. The degree d_v of a node v is the total number of the edges connected to it. The number of edges started from node v is d_v^{out} while the number of edges ended at node v is d_v^{in}.

The Prioritization Process

The prioritization process of our method mainly includes two steps: first is building the initial requirement digraph for N requirements. And the second is the dominance test process to complete the digraph and output the final rank list.

Step 1: Digraph Building: In the digraph building process, every time we insert a new node with edges indicating the prioritization relationship, we try to choose the node to compare with which can reduce the total comparisons as much as possible. Thus, we design an algorithm *Comparenode(G)* to achieve it.

Algorithm 1 Comparenode(G)

Require:
Let *G* denote the current directed graph;
Max(G) denotes the set of nodes which have the most degree in the graph *G*;
MinDif(Max(G)) denotes the node *v* that has the minimum $|d_v^{in}\text{-}d_v^{out}|$ in *Max(G)*.

Ensure:
Let *Cnode* is the final node chosen to be compared;

1: **if** *sizeof(Max(A))*==1 **then**
2: *Cnode* ← *Max(A)*;
3: **else if** *sizeof(MinDif(Max(A)))*==1 **then**
4: *Cnode* ← *MinDif (Max(A))*;
5: **else**
6: *Cnode* ← *at random(MinDif (Max(A)))*;
7: **return** Cnode;

When there are more than one node with the most edges, Algorithm *Comparenode(G)* is aimed to choose the node with the least $|d_v^{in}-d_v^{out}|$ in step 4. In order to verify the choosing rule in this function, we give proofs with the concept of information entropy. According to the principle of maximum entropy [4,6], the object with the distribution of maximum entropy can provide the most

information. Intuitively, the node in a digraph with larger entropy contains more information, which can eliminate more uncertainty in prioritization. Entropy can be calculated as the Eq.1 showing:

$$H = -\sum_{i=1}^{n} p_i \log(p_i) \quad (1)$$

Where p_i means the possibility of the occurrence of situation i and n is the amount of situations. In the digraph $G=(V,E)$, E_j means the degree of node j. I_j means the d_j^{in} of node j. O_j means the d_j^{out} of node j. So the entropy of node j is calculated by Eq.2:

$$H(j) = -\sum_{i=1}^{n} p_i \log(p_i) = -\frac{I_j}{E_j}\log(\frac{I_j}{E_j}) - \frac{O_j}{E_j}\log(\frac{O_j}{E_j}) \quad (2)$$

Then we take the derivative of $H(I_j)$ as Eq.3 shows:

$$H'_{I_j} = \frac{1}{E_j} \times \log(\frac{E_j - I_j}{I_j}) \quad (3)$$

Through analyzing the derivative in Eq.3, we find that the smaller value of $|I_j - O_j|$ can lead to bigger value of $H(I_j)$. So we choose the node with smallest $|d_v^{in} - d_v^{out}|$ among the nodes with the biggest degree. We also test this choosing rule in Section 3.

Combining the algorithm of *Comparenode (G)*, we design the algorithm *DGBuilding* in Algorithm 2 to build the initial digraph of N requirements, which is the main algorithm in our first step. The mainly steps are as follows:

(1) Compare R_1 and R_2 and build the initial digraph;

(2) Before inserting a new node, choose the node to compare with algorithm *Comparenode(G)*;

(3) If the new node R_n is prior to the compared node R_c, when R_c is the end node, add edge (R_c, R_n) to digraph G and set R_n as the end node; when R_c is not the end node, choose the node to compare with among the front nodes of R_c and repeat this process until R_n is not prior to R_c. After this repetition, if $R_n \notin V$, add node R_n and edges (R_{c-1}, R_n) and (R_n, R_c) into the digraph G;

(4) If the compared node R_c is prior to the new node R_n, when R_c is the start node, add edge (R_n, R_c) to digraph G and set R_n as the start node; when R_c is not the start node, choose the node to compare with among the back nodes of R_c and repeat this process until R_c is not prior to R_n, After this repetition, if $R_n \notin V$, add node R_n and edges (R_c, R_n) and (R_n, R_{c-1}) into the digraph G.

Algorithm 2 DGBuilding

Require:
Let *R* denotes the set of n requirements; *G* denotes the current digraph;
V denotes the set of nodes in *G*; *E* denotes the set of edges in *G*;
R_n denotes the requirement node waiting to be inserted into *G*;

Ensure:
The digraph G of *R*;

```
for i=0; i<n; i++ do
    R_c ← Comparenode( G );
    while R_n is superior t than R_c and R_n ∉ V do
        if rear( R_c )= =∅ then
            Add (R_c, R_n) to E; Add R_n to V;
            Break;
        else
            R_c ← Comparenode(rear(R_c ))
    if R_n ∉ V then
        Add (R_{c-1}, R_n) and ( R_n, R_c ) to E; Add R_n to V;
    while R_c is prior than R_n and R_n ∉ V do
```

12: **if** $front(R_c) == \varnothing$ **then**
13: Add (R_n, R_c) to E; Add R_n to V;
14: Break;
15: **else**
16: $R_c \leftarrow Comparenode(front(R_c))$;
17: **if** $R_n \notin V$ **then**
18: Add (R_c, R_{n-1}) and (R_n, R_{c-1}) to E; Add R_n to V;
19: **return** G;

Step 2: Digraph Completing and Outputting Rank List: After Step 1, the initial requirement digraph includes all the requirement nodes. However, there are still some nodes which do not have priorities relation with each other. So we need an algorithm *Dtest* in Algorithm 3 to check the dominance and completing the digraph so that we can get the final rank list. The mainly steps are as follows:

(1) Judge the number of the start nodes in digraph G;

(2) If the start node is only one, put it into the rank list and delete the node and its front edges in digraph G and get new G. Repeat step (1) with the new digraph G;

(3) If there are more than one start node, we put them into a new requirements set and repeat the *DGBuilding* algorithm and *Dtest* algorithm with this new set;

(4) When the start node is also the end node in digraph, put it into the rank list and end this algorithm. The output is the ranked list of N requirements.

After the process with *Dtest* algorithm, the completed requirement digraph can be achieved. And we can get the rank list of N requirements through outputting the start node of requirements digraph one by one continually.

Algorithm 3 Dtest

Require:
Let $zero(C(V,E))$ denotes the set of the nodes $d_v^{in} = 0$;

Ensure:
The rank List of R; $G'(V', E') \leftarrow G(V,E)$;

1: **while** $V' \neq \varnothing$ **do**
2: **if** $sizeof(zero(G'(V', E'))) \neq 1$ **then**
3: $G(E) \leftarrow G(E) \cup DGBuilding(zero(G'(V', E')))$;
4: **if** $sizeof(zero(G'(V', E'))) == 1$ **then**
5: $V' \leftarrow Deletezero(G'(V', E'))$ from V'; $E' \leftarrow Delete\{(R_i, R_j) \mid R_i \in zero(G'(V', E'))\}$ from E'; $G' \leftarrow G'(V', E')$;
6: **while** $V \neq \varnothing$ **do**
7: Insert zero(G(V,E)) into List;
8: $V \leftarrow Deletezero(G(V,E))$ from E; $E \leftarrow Delete\{(R_i, R_j) \mid R_i \in zero(G(V,E))\}$ from E; $G \leftarrow G(V,E)$;
9: **return** List;

Evaluation

In this section we design two parts of the evaluation: first part is verifying the effectiveness of the compared node choosing rule in our approach; the second part is performance evaluation comparing our approach to Bubblesort and Binary tree. The requirements used in this section are based on an industrial requirements data set of the company which we cooperate with.

Rule Verifying

We have proved that if there are more than one node having the most degree when choosing the compared node, we aim to find the node with the smallest $|d_v^{in} - d_v^{out}|$ in Section 2. Here we verify this choosing rule through several experiments further. When there are more than one node with the most degree, we consider the following rules to choose the compared node: (1) *Rule1*: Choose the node randomly; (2) *Rule2*: Choose the node with the largest $|d_v^{in} - d_v^{out}|$; (3) *Rule3*: Choose the node with the smallest $|d_v^{in} - d_v^{out}|$, which we use in our method.

We take those three rules into our R-digraph method and test them with 20 groups of requirements prioritization respectively. Every group contains 50 requirements chosen from the same requirements set randomly. Figure 1 shows the required comparisons with the three rules respectively.

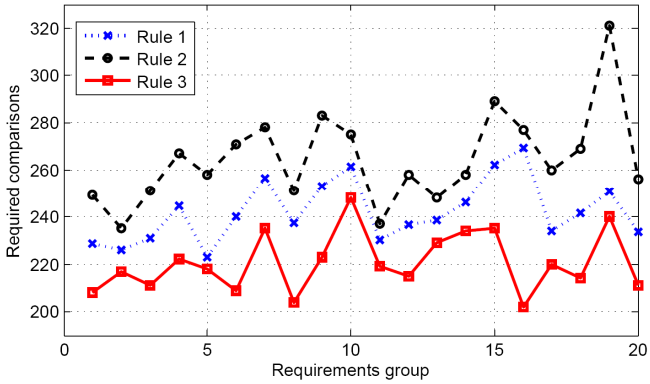

Figure 1. Required comparisons with 20 groups of requirements.

In Figure 1, the x-axis means the group sequence while the y-axis denotes the required comparisons for every group of requirements. The lines from up to down mean the needed comparisons with *Rule2*, *Rule1* and *Rule3* in turn, which confirms the effectiveness of the rule that choosing the node with the smallest $|d_v^{in}-d_v^{out}|$ in our method.

Performance Evaluation

Scalability Techniques like AHP, pairwise comparison and bubblesort suffer from scalability problems because they do comparisons among each pair of requirements [1]. We do experiments to record the amount of man-participate comparisons using our method, binary tree and bubblesort respectively. The amount of requirements increasing from 10 to 700 with every step of 10 requirements. Figure 2 shows the comparison results.

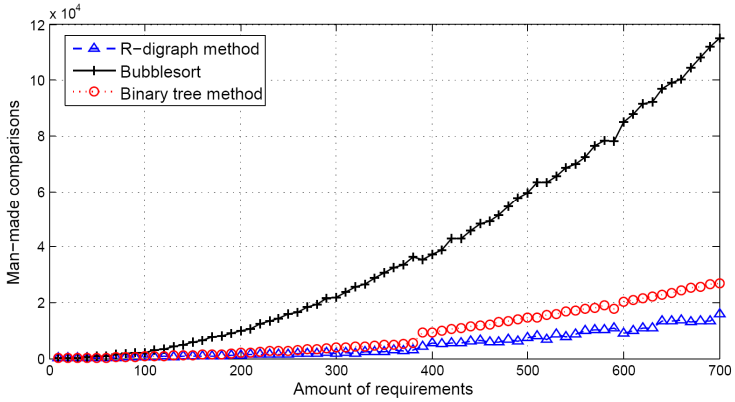

Figure 2. The man-participate comparisons of three methods.

The x-axis means the number of requirements and y-axis denotes the man-participate comparisons needed. From Figure2, we can see that: (1) For a certain amount of requirements, the bubblesort needs the most man-participate comparisons among the three methods while R-digraph method needs the least; (2) With the increase of requirements amount, the needed comparisons with bubblesort increases drastically while the other two methods need less. Compared with binary tree, R-digraph method shows a better performance especially for larger scale requirements. The results illustrates that our method can be used for massive requirements prioritization.

Time Consuming. The amount of man-participate comparisons needed will mainly determine the time of the whole prioritization process. We test the bubblesort, binary tree method and R-digraph with 10 groups of requirements and three stakeholders to do the comparisons. Each group contains 10

sample requirements and we adopt the average time of the three stakeholders for comparisons. Table 1 shows the different execution time of the whole prioritization process with the three methods.

Table 1. Time-consumption for three methods (seconds).

Methods	Median(s)	Mean(s)	Standard deviation
Bubblesort	119	119.68	5.32
Binary tree	91	90.24	4.51
R-diagraph	78	78.21	3.34

From the Table 1, the difference in time (sample mean) among the three methods show that there is less time needed for the prioritization process with R-digraph method. It can be explained by a fact that, compared with the time needed for man-participate comparisons, the execution time needed for the computer calculation can be ignored. To this end, the effective reduction of man-participate comparisons contributes to the least time required for R-digraph method.

Summary

We proposes a requirement prioritization method based on R-digraph, which can achieve the prioritization with less man-participate comparisons. In future we aim to consider the factors of resource constraints in requirements prioritization [10, 11], namely the quantities and species of resource will influence the achievement of requirements.

References

[1] Babar, M.I., Ramzan, M., Ghayyur, S.A.K, Challenges and future trends in software requirements prioritization. In: CNIT'11. pp. 319-324 (2011).

[2] Beg, M.R., Verma, R.P., Joshi, A, Reduction in number of comparisons for requirement prioritization using b-tree. In: Advance Computing Conference. pp. 340-344(2009).

[3] Beg, R., Abbas, Q., Verma, R.P, An approach for requirement prioritization using b-tree. In: International Conference on Emerging Trends in Engineering and Technology. pp. 1216-1221(2008).

[4] Chen, Q., Liang, X., Guo, Z., Entransy theory for the optimization of heat transferca review and update. Int J Heat Mass Tran 63, 65-81(2013).

[5] Ejnioui, A, Software requirement prioritization using fuzzy multi-attribute decision making. In: Open Systems. pp. 1-6(2012).

[6] Frizelle, G., Woodcock, E.C, Measuring complexity as an aid to developing operational strategy. IJOPM 15, 26-39(2013).

[7] Karlsson, J., Ryan, K, A cost-value approach for prioritizing requirements. IEEE Software 14, 67-74 (1997).

[8] Kaur, G., Bawa, S, A survey of requirement prioritization methods. Int.J.Eng.Res. Technol 2, 958-962 (2013).

[9] P.Achimugu, A.Selamat, R., M.N. Mahrin, A systematic literature review of software requirements prioritization research. Information and Software Technology 56, 568-585 (2014).

[10] Pergher, M., Rossi, B, Requirements prioritization in software engineering: A systematic mapping study. In: International Workshop on Empirical Requirements Engineering. pp. 10503-10512 (2013).

[11] Waldmann, B, There's never enough time: Doing requirements under resource constraints, and what requirements engineering can learn from agile development. In: IEEE International Requirements Engineering Conference. pp. 301-305 (2011).