

A Hybridized Approach for Prioritizing Software Requirements Based on K-Means and Evolutionary Algorithms

Philip Achimugu and Ali Selamat

Abstract One of the major challenges facing requirements prioritization techniques is accuracy. The issue here is lack of robust algorithms capable of avoiding a mismatch between ranked requirements and stakeholder's linguistic ratings. This problem has led many software developers in building systems that eventually fall short of user's requirements. In this chapter, we propose a new approach for prioritizing software requirements that reflect high correlations between the prioritized requirements and stakeholders' linguistic valuations. Specifically, we develop a hybridized algorithm which uses preference weights of requirements obtained from the stakeholder's linguistic ratings. Our approach was validated with a dataset known as RALIC which comprises of requirements with relative weights of stakeholders.

Keywords Software · Requirements · Prioritization · Stakeholders · RALIC

1 Introduction

Currently, most software development organizations are confronted with the challenge of implementing ultra-large scale system especially in the advent of big data. This has led to the specification of large number of software requirements during the elicitation life cycle. These requirements contain useful information that will satisfy the need of the users or project stakeholders. Unfortunately, due to some crucial challenges involved in developing robust systems such as inadequate skilled programmers, limited delivery time and budget among others; requirements

P. Achimugu · A. Selamat (✉)

Department of Software Engineering, Faculty of Computing,
Universiti Teknologi Malaysia, Faculty of Computing,
Skudai 81310, Johor, Malaysia
e-mail: aselamat@utm.my

P. Achimugu
e-mail: check4philo@gmail.com

prioritization has become a viable technique for ranking requirements in order to plan for software release phases. During prioritization, requirements are classified according to their degrees of importance with the help of relative or preference weights from project stakeholders.

Data mining is one of the most effective and powerful techniques that can be used to classify information or data objects to enhance informed decision making. Clustering is a major data mining task that has to do with the process of finding groups or clusters or classes in a set of observations such that those belonging to the same group are similar, while those belonging to different groups are distinct, according to some criteria of distance or likeness. Cluster analysis is considered to be unsupervised learning because it can find and recognise patterns and trends in a large amount of data without any supervision or previously obtained information such as class labels. There are many algorithms that have been proposed to perform clustering.

Generally, clustering algorithms can be classified under two major categories [12]: hierarchical algorithms and partitional algorithms. Hierarchical clustering algorithms have the capacity of recursively identifying clusters either in an agglomerative (bottom-up) mode or in a divisive (top-down) mode. An agglomerative method is initiated with data objects in a separate cluster, which decisively merge the most similar pairs until termination criteria are satisfied. Divisive methods on the other hand deals with data objects in one cluster and iteratively divide each cluster into smaller clusters, until termination criteria are met. However, a partitional clustering algorithm concurrently identifies all the clusters without forming a hierarchical structure. A well-known type of partitional clustering algorithms is the centre-based clustering method, and the most popular and widely used algorithm from this class of algorithms is known as K-means algorithm. K-means is relatively easy to implement and effective most of the time [7, 13]. However, the performance of k-means depends on the initial state of centroids which is likely to converge to the local optima rather than global optima. The k-means algorithm tries to minimise the intra-cluster variance, but it does not ensure that the result has a global minimum variance [14].

In recent years, many heuristic techniques have been proposed to overcome this problem. Some of which include: a simulated annealing algorithm for the clustering problem [28]; a tabu search based clustering algorithm [2, 29]; a genetic algorithm based clustering approach [23]; a particle swarm optimization based clustering technique [19] and a genetic k-means based clustering algorithm [21] among others. A major reported limitation of K-means algorithm has to do with the fact that, the number of clusters must be known prior to the utilization of the algorithm because it is required as input before the algorithm can run. Nonetheless, this limitation has been addressed by some couple of authors [3, 8, 9, 26].

In this research, we propose the application of a hybridized algorithm based on evolutionary and k-means algorithms on cluster analysis during requirements prioritization. The performance of the proposed approach has been tested on standard and real datasets known as RALIC. The rest of the paper is organised as follows: Sect. 2 provides a brief background on requirement prioritization. Section 3 describes k-means and evolutionary algorithms. In Sect. 4, we present our proposed

algorithm for solving requirement prioritization problem using evolutionary algorithm and k-means algorithm. Experimental results are discussed in Sect. 5. Finally, Sect. 6 presents the conclusions of this research with ideas for future work.

2 Software Requirements Prioritization

During requirement elicitation, there are more prospective requirements specified for implementation by relevant stakeholders with limited time and resources. Therefore, a meticulously selected set of requirements must be considered for implementation and planning for software releases with respect to available resources. This process is referred to as requirements prioritization. It is considered to be a complex multi-criteria decision making process [25].

There are so many advantages of prioritizing requirements before architecture design or coding. Prioritization aids the implementation of a software system with preferential requirements of stakeholders [1, 32]. Also, the challenges associated with software development such as limited resources, inadequate budget, insufficient skilled programmers among others makes requirements prioritization really important [15]. It can help in planning software releases since not all the elicited requirements can be implemented in a single release due to some of these challenges [5, 16]. It also enhances budget control and scheduling. Therefore, determining which, among a pool of requirements to be implemented first and the order of implementation is necessary to avoid breach of contract or agreement during software development. Furthermore, software products that are developed based on prioritized requirements can be expected to have a lower probability of being rejected. To prioritize requirements, stakeholders will have to compare them in order to determine their relative importance through a weight scale which is eventually used to compute the ranks [20]. These comparisons become complex with increase in the number of requirements [18].

Software system's acceptability level is mostly determined by how well the developed system has met or satisfied the specified requirements. Hence, eliciting and prioritizing the appropriate requirements and scheduling right releases with the correct functionalities are critical success factors for building formidable software systems. In other words, when vague or imprecise requirements are implemented, the resulting system will fall short of user's or stakeholder's expectations. Many software development projects have enormous prospective requirements that may be practically impossible to deliver within the expected time frame and budget [31]. It therefore becomes highly necessary to source for appropriate measures for planning and rating requirements in an efficient way.

Many requirements prioritization techniques exist in the literature. All of these techniques utilize a ranking process to prioritize candidate requirements. The ranking process is usually executed by assigning weights across requirement based on pre-defined criteria, such as value of the requirement perceived by relevant stakeholders or the cost of implementing each requirement. From the literature;

analytic hierarchy process (AHP) is the most prominently used technique. However, this technique suffers bad scalability. This is due to the fact that, AHP executes ranking by considering the criteria that are defined through an assessment of the relative priorities between pairs of requirements. This becomes impracticable as the number of requirements increases. It also does not support requirements evolution or rank reversals but provide efficient or reliable results [4, 17]. Also, most techniques suffer from rank reversals. This term refers to the inability of a technique to update rank status of ordered requirements whenever a requirement is added or deleted from the list. Prominent techniques that suffer from this limitation are case base ranking [25]; interactive genetic algorithm prioritization technique [31]; Binary search tree [17]; cost value approach [17] and EVOLVE [11]. Furthermore, existing techniques are prone to computational errors [27] probably due to lack of robust algorithms. [17] conducted some researches where certain prioritization techniques were empirically evaluated. From their research, they reported that, most of the prioritization techniques apart from AHP and bubble sorts produce unreliable or misleading results while AHP and bubble sorts were also time consuming. The authors submitted that; techniques like hierarchy AHP, spanning tree, binary search tree, priority groups produce unreliable results and are difficult to implement. [4] were also of the opinion that, techniques like requirement triage, value intelligent prioritization and fuzzy logic based techniques are also error prone due to their reliance on experts and are time consuming too. Planning game has a better variance of numerical computation but suffer from rank reversals problem. Wieger's method and requirement triage are relatively acceptable and adoptable by practitioners but these techniques do not support rank updates in the event of requirements evolution as well. The value of a requirement is expressed as its relative importance with respect to the other requirements in the set.

In summary, the limitations of existing prioritization techniques can be described as follows:

- 2.1.1 Scalability: Techniques like AHP, pairwise comparisons and bubblesort suffer from scalability problems because, requirements are compared based on possible pairs causing $n(n-1)/2$ comparisons [17]. For example, when the number of requirements is doubled in a list, other techniques will only require double the effort or time for prioritization while AHP, pairwise comparisons and bubblesort techniques will require four times the effort or time. This is bad scalability.
- 2.1.2 Computational complexity: Most of the existing prioritization techniques are actually time consuming in the real world [4, 17]. Furthermore, [1] executed a comprehensive experimental evaluation of five different prioritization techniques namely; AHP, binary search tree, planning game, \$100 (cumulative voting) and a new method which combines planning game and AHP (PgcAHP), to determine their ease of use, accuracy and scalability. The author went as far as determining the average time taken to prioritize 13 requirements across 14 stakeholders with these techniques. At the end of the experiment; it was observed that, planning game was the fastest while AHP

was the slowest. Planning game prioritized 13 requirements in about 2.5 min while AHP prioritized the same number of requirements in about 10.5 min. In other words, planning game technique took only 11.5 s to compute the priority scores of one requirement across 14 stakeholders while AHP consumed 48.5 s to accomplish the same task due to pair comparisons.

- 2.1.3 Rank updates: This is defined as ‘anytime’ prioritization [25]. It has to do with the ability of a technique to automatically update ranks anytime a requirement is included or excluded from the list. This situation has to do with requirements evolution. Therefore, existing prioritization techniques are incapable of updating or reflecting rank status whenever a requirement is introduced or deleted from the rank list. Therefore, it does not support iterative updates. This is very critical because, decision making and selection processes cannot survive without iterations. Therefore, a good and reliable prioritization technique will be one that supports rank updates. This limitation seems to cut across most existing techniques.
- 2.1.4 Communication among stakeholders: Most prioritization techniques do not support communication among stakeholders. One of the most recent works in requirement prioritization research reported communication among stakeholders as part of the limitations of their technique [25]. This can lead to the generation of vague results. Communication has to do with the ability of all relevant stakeholders to fully understand the meaning and essence of each requirement before prioritization commences.
- 2.1.5 Requirements dependencies: This is a crucial attribute that determines the reliability of prioritized requirements. These are requirements that depend on another to function. Requirements that are mutually dependent can eventually be merged as one; since without one, the other cannot be implemented. Prioritizing such requirements may lead to erroneous or redundant results. This attribute is rarely discussed among prioritization research authors. However, dependencies can be detected by mapping the pre and post conditions from the whole set of requirements, based on the contents of each requirement [24]. Therefore, a good prioritization technique should cater or take requirements dependences into cognizance before initiating the process.
- 2.1.6 Error proneness: Existing prioritization techniques are also prone to errors [27]. This could be due to the fact that, the rules governing the requirements prioritization processes in the existing techniques are not robust enough. This has also led to the generation of unreliable prioritization results because; such results do not reflect the true ranking of requirements from stakeholder’s point of view or assessment after the ranking process. Therefore robust algorithms are required to generate reliable prioritization results.

3 K-Means and Evolutionary Algorithms

K-Means algorithm is one of the most popular types of unsupervised clustering algorithm [6, 10] which is usually used in data mining, pattern recognition and other related researches. It is aimed at minimizing cluster performance indexes, square-error and error criteria. The concept of this algorithm borders on the identification of *K* clusters that satisfies certain criteria. Usually, to demonstrate the applicability of *K*-means algorithm, some data objects are chosen to represent the initial cluster focal points and secondly, the rest of the data objects are assembled to their focal points based on the criteria of minimum distance which will eventually lead to the computation of the initial clusters. However, if these clusters are unreasonable, it is easily modified by re-computing each cluster's focal point. This process is repeatedly iterated until reasonable clusters are obtained. In the context of requirement prioritization, the numbers of clusters are likened to the number of requirements while the data objects are likened to the attributes describing the expected functionalities of a particular requirement. Therefore, *K*-Means algorithm is initiated with some random or heuristic-based centroids for the desired clusters and then assigns every data object to the closest centroid. After that, the *k*-means algorithm iteratively refines the current centroids to reach the (near) optimal ones by calculating the mean value of data objects within their respective clusters. The algorithm will terminate when any one of the specified termination criteria is met (i.e., a predetermined maximum number of iterations is reached, a (near) optimal solution is found or the maximum search time is reached).

Inversely, the evolutionary algorithm (EA) is best illustrated with principles of differential evolution algorithms, by considering requirements as individuals. After generating the initial solution, its requirements are stored into a pool, which forms the initial population in the tournament. Thereafter, the requirements are categorized into various classes containing its respective attributes. The weights of attributes between two requirements are randomly selected for computation until the entire weights are exhausted. Meaning, weights of attributes from each requirement are mated for crossover. For instance, assuming we have two requirements *X* and *Y* with respective attributes as $\langle x_1, \dots, x_n \rangle$ and $\langle y_1, \dots, y_n \rangle$. These two attributes are considered as prospective couple or parent. So, for each couple, we randomly extract crossover point, for instance 2 for the first and 5 for the second. The final step is to apply random mutation. These processes are performed from the first to the last requirement. Once a pair (combination) of requirement is selected, one out of the four local search operators is applied randomly based on stakeholder's weights. Finally, offspring (requirements) generated by these crossover operators are mutated according to the stakeholder's weights. Mutation is specifically achieved by selecting randomly one out of two operators according to the weights distribution. Selecting each possible pair of requirement is based on a random order. Mating and mutation operators are repeatedly applied for a certain number of generations, and finally a feasible solution is constructed using the requirements in the pool. To guarantee a feasible solution, recombination and mutation operators of

EA are not allowed to lose any customer i.e. the offspring must contain the same number of customers as the parent, otherwise parent are stored into the requirement pool instead of offspring.

4 Proposed Approach

Our approach is based on the relative weights provided by project stakeholders of a software development project. The evolutionary algorithm works as a hyper-heuristics which assigns different coefficient values to the relative scores obtained from the pool of functions included in the system. At the beginning of the process, all functions (or metrics) evenly contribute to the calculation of the relative cumulative values of a specific requirement. The system evolves so that the requirements which provide the most valued weights across the relevant stakeholders have the highest coefficients. The differential evolution (DE) algorithm [30] was chosen among other candidates because, after a preliminary study, we conclude that DE obtained very competitive results for the problem under consideration. The reason lies in how the algorithm makes the solutions evolve. Our system can be considered as a hyper-heuristics which uses differential evolution to assign each requirement with a specific coefficient. These values show the relative importance of each requirement. Differential evolution performs the search of local optima by making small additions and subtractions between the members of its population. This feature is capable of solving rank reversals problem during requirement prioritization since the algorithm works with the weights provided by the stakeholders. In fact, the stakeholder is defined as an array of floating point values, s , where $s(x)$ is the coefficient which modifies the result provided by the learning function fx . We consider a finite set of collection of requirements $X = \{R_{11}, R_{12} \dots R_{1k}\}$ that has to be ranked against $Y = \{R_{21}, R_{22} \dots R_{2k}\}$. Our approach consist of set of input $R_{11}, R_{12}, \dots, R_{1k}$, associated with their respective weights w_1, w_2, \dots, w_k that represents stakeholders' preferences and a fitness value function required to calculate the similarity weights across requirements. The requirement $(R_{11}, R_{21}, \dots, R_{nk})$ represent input data that are ranked using the fitness function on similarity scores and stored in the database. In this approach, we assume that, the stakeholder's preferences are expressed as weights, which are values between 5 and 1. These weights are provided by the stakeholders. In this research, one of our objectives is to automatically rank stakeholder's preferences using EA based learning. The designed algorithm will compute the ranks of requirements based on a training data set. To rank the preferential weights of requirements across relevant stakeholders, there is need to identify the following: a ranking technique for the best output and a measure of quality that aids the evaluation of the proposed algorithm.

In practical application of the learning process, $X = (r_1, r_2, \dots, r_n)$; $Y = (r'_1, r'_2, \dots, r'_n)$ probably represents two requirements with their respective attributes that are to be ranked. For each requirement, attributes are not necessarily mutually

independent. In order to drive the synthetic utility values, we first exploited the factor analysis technique to extract the attributes that possess common functionalities. This caters for requirement dependencies challenges during the prioritization process. The attributes with the same functionalities are considered to be mutually dependent. Therefore, before relative weights are assigned to the requirements by relevant stakeholders, attention should be paid to requirement dependencies issues in order to avoid redundant results. However, when requirements evolve, it becomes necessary to add or delete from a set. The algorithm should also be able to detect this situation and update rank status of ordered requirements instantly. This is known as rank reversals. It is formally expressed as: (1) failure of the type $1 \rightarrow 5$ or $5 \rightarrow 1$; (2) failures of the type $1 \rightarrow \phi$ or $5 \rightarrow \phi$ (where ϕ = the null string) (called *deletions*); and (3) failures of the type $\phi \rightarrow 1$ or $\phi \rightarrow 5$ (called *insertions*). A weight metric w , on two requirement (X, Y) is defined as the smallest number of edit operations (deletions, insertions and updates) to enhance the prioritization process. Three types of rank updates operations on $X \rightarrow Y$ are defined as: a *change* operation ($X \neq \phi$ and $Y \neq \phi$), a *delete* operation ($Y = \phi$) and an *insert* operation ($X = \phi$). The weights of all the requirements can be computed by a *weight function* w . An arbitrary weight function w is obtained by computing all the assigned non-negative real number $w(X, Y)$ on each requirement. On the other hand, there is mutual independence between attributes, and the measurement is an additive case, so we can utilize the additive aggregate method to conduct the synthetic utility values for all the attributes in the entire requirements. As we can see in Algorithm 1, differential evolution starts with the generation of random population (line 1) through the assignment of a random coefficient to each attribute of the individual (requirement). The population consists of a certain number of solutions (this is, a parameter to be configured). Each individual (requirement) is represented by a vector of weighting factors provided by the stakeholders. After the generation of the population, the fitness of each individual is assigned to each solution using the Pearson correlation. This correlation, $\text{corr}(X, Y)$, is calculated with the scores provided by stakeholders for every pair of requirement of the RALIC dataset [22].

The closer the value of correlation is to any of the extreme values, the stronger is the correlation between the requirements and the higher is the accuracy of the prioritized results. On the contrary, if the result tends toward 0, it means that the requirements are somewhat uncorrelated which gives an impression of poor quality prioritized solution.

Algorithm 1: Pseudo-code for the DE/K-means algorithm

1. generateRandom centroids from K clusters (population)
2. assign weights of each attribute to the cluster with the closest centroid
3. update the centroids by calculating the mean values of objects within clusters:
Fitness (population)
4. while (stop condition not reached) do
5. for (each individual of the population)
6. selectIndividuals (xTarget, xBest, xInd1, xInd2)
7. xMut diffMutation (xBest, F, xInd1, xInd2)

8. $xTrial$ binCrossOver ($xTarget$, $xMut$, CrossProb)
9. calculateFitness ($xTrial$)
10. updateIndividual ($xTarget$, $xTrial$)
11. endfor
12. endwhile
13. return bestIndividual (population)

From Algorithm 1, the main loop begins after evaluating the whole population (line 2). It is important to note that differential evolution and k-means algorithms are iterative in nature. This becomes very necessary when uninterrupted generations attempt to obtain an optimal solution and terminates when the maximum number of generations is reached (line 4). We initiated the process by selecting four parameters (line 6). $xTarget$ and $xBest$ are the parameters being processed. The former stands for the weights provided by project stakeholders and the latter stands for the prioritized weights for two randomly chosen requirements denoted as $xInd1$ and $xInd2$ respectively. Next, mutation is performed (line 7) according to the expression: $xMut \ xBest + F (xInd1 _ xInd2)$ in order to determine sum of the cumulative relative weights of requirements across the total number of stakeholders involved in the software development project. This task execute in twofold: (diffMutation 1 and 2). The first phase has to do with the calculation of $xDiff$ across project stakeholders with the help of expression: $xDiff \ F (xInd1 _ xInd2)$. $xDiff$ represents the mutation to be applied to compute the relative weights of requirements in order to calculate the best solution. Subsequently, the modification of each attribute of $xBest$ in line with the mutation indicated in $xDiff$ give rise to $xMut$. At the end of the mutation process, $xTarget$ and $xMut$ individuals are intersected (line 8) using binary crossover with the concept of crossover probability, crossProb. Then, the obtained individual, $xTrial$, is evaluated to determine its accuracy (line 8) which is compared against $xTarget$. The evaluation procedures of the prioritization process encompass the establishment of fitness function, disclosure of agreement and disagreement indexes, confirmation of credibility degree, and the ranking of attributes/requirements. These data are represented by weights reflecting the subjective judgment of stakeholders. The best individual is saved in the $xTarget$ position (line 10). This process is iterated for each individual in the population (line 5) especially when the stoppage criteria are not met (line 4). However, in the context of this research, the stoppage criteria are certain number of generations which is also set during the configuration of the parameters. At the end of the process, the best individuals (most valued requirements) are returned as the final results of the proposed system (line 13). It is important to note here that results have been obtained after a complete experimental process using RALIC dataset.

Based on the above description, the proposed algorithm is built on three main steps. In the first step, EA-KM applies k-means algorithm on the selected dataset and tries to produce near optimal centroids for desired clusters. In the second step, the proposed approach produces an initial population of solutions, which will be applied by the EA algorithm in the third step. The generation of an initial population is carried out in several different ways. These could be through candidate

solutions generated by the output of the k-means algorithm or randomly generated. The process generates a high-quality initial population, which will be used in the next step by the EA algorithm. Finally, in the third step, EA will be employed for determining an optimal solution for the clustering-based prioritization problem. To represent candidate solutions in the proposed algorithm, we used one-dimensional array to encode the centroids of the desired clusters. The length of the arrays is equal to $d * k$, where d is the dimensionality of the dataset under consideration or the number of features that each data object has and k is the number of clusters.

The optimal value of the fitness function $J(w, \mathbf{x}, \mathbf{c})$ is determined by the following prioritization equations:

$$\mathbf{x}(w) = f(w, \mathbf{x}, \mathbf{c}), \quad \mathbf{x}(0) = \mathbf{x}_0(\mathbf{c}(0)) \quad (1)$$

And the set of constraints

$$\begin{cases} g_i(\mathbf{x}, \mathbf{c}) = 0 & \text{for } j = 1, \dots, E \\ g_j(\mathbf{x}, \mathbf{c}) \geq 0 & \text{for } j = E + 1, \dots, S \end{cases} \quad (2)$$

where, $\mathbf{x} \in R^m$ is the requirement set described by the function $f \in R^m$; $\mathbf{c} \in R^n$ is the criteria used to determine the relative weights of requirements. The optimal value of $J(w, \mathbf{x}, \mathbf{c})$ is achieved by varying the criteria $c_i(w)$, $i = \overline{1, n}$ within the boundaries specified by (1) and (2). All functions here are to be regarded as discrete, obtained through the relative weights w_1, w_2, \dots, w_n provided by the stakeholders. During prioritization, the state of the requirement at weight w is conventionally described by the number of stakeholders $N(s)$. Therefore, the prioritization process in this case is one-dimensional ($m = 1$), and the Eq (2) becomes:

$$\frac{\Delta N(s)}{\Delta w} = f(N) - \kappa(c)c(w)N, \quad N(0) = N_0 \quad (3)$$

Where $f(N)$ is a real-valued function which models the increase in the number of requirements; $c(w)$ is the weight of requirements based on pre-defined criteria; $\kappa(c)$ is a quality representing the efficacy of the ranked weights. The rank criteria $c(w)$ in (3), is the only variable directly controllable by the stakeholders. Therefore, the problem of requirements prioritization can be regarded as the problem of planning for software releases based on the relative weights of requirements. The optimal weights of requirements are in the form of a discrete ordered program with N requirements given at weights w_1, w_2, \dots, w_n . Each requirement is assessed by s stakeholders characterized by their defined criteria C_{ij} , $i = \overline{1, n}, j = \overline{1, d}$ in the set. These criteria can be varied within the boundaries specified by the constraints in Eq. (3). The conflicting nature of these constraints and the intention to develop a model-independent approach for prioritizing requirements makes the utilization of computational optimization techniques very viable.

All the experiments were executed under the same environment: an Intel Pentium 2.10 GHz processor and 500 GB RAM. Since we are dealing with a stochastic

algorithm, we have carried out 50 independent runs for each experiment. Results provided in the following subsections are average results of these 50 executions. Arithmetic mean and standard deviation were used to statistically measure the performance of the proposed system.

5 Experimental Results and Discussion

The experiments described in this research considered the likelihood of calculating preference weights of requirements provided by stakeholders so as to compute their ranks. The RALIC dataset was used for validating the proposed approach. The PointP and RateP portions of the dataset were used, which consist of about 262 weighted attributes spread across 10 requirement sets from 76 stakeholders. RALIC stands for replacement access, library and ID card. It was a large-scale software project initiated to replace the existing access control system at University College London [22].

The dataset is available at: <http://www.cs.ucl.ac.uk/staff/S.Lim/phd/dataset.html>. Attributes were ranked based on 5-point scale; ranging from 5 (highest) to 1 (lowest). As a way of pre-processing the dataset, attributes with missing weights were given a rating of zero.

For the experiment, a Gaussian Generator was developed, which computes the mean and standard deviation of given requirement sets. It uses the Box-Muller transform to generate relative values of each cluster based on the inputted stakeholder's weights. The experiment was initiated by specifying a minimum and maximum number of clusters, and a minimum and maximum size for attributes. It then generates a random number of attributes with random mean and variance between the inputted parameters. Finally, it combines all the attributes into one and computes the overall score of attributes across the number of clusters k . The algorithm defined earlier attempts to use these combined weights of attributes in each cluster to rank each requirement. For the k -means algorithm to run, we filled in the variables/observations table which has to do with the two aspect of RALIC dataset that was utilized (PointP and RateP), followed by the specification of clustering criterion (Determinant W) as well as the number of classes. The initial partition was randomly executed and ran 50 times. The iteration completed 500 cycles and the convergence rate was at 0.00001. As an initialization step, the DE algorithm generated a random set of solutions to the problem (a population of genomes). Then it enters a cycle where fitness values for all solutions in a current population are calculated, individuals for mating pool are selected (using the operator of reproduction), and after performing crossover and mutation on genomes in the mating pool, offspring are inserted into a population. In this research, elitism was performed to save the chromosomes of the old solution so that crossover and mutation can re-occur for new solutions. Thus a new generation is obtained and the process begins again. The process stops after the stopping criteria are met, i.e. the "perfect" solution is recognized, or the number of generations has reached its

maximum value. From each generation of individuals, one or few of them, that has the highest fitness values are picked out, and inserted into the result set.

The weights of requirements are computed based on their frequencies and a mean score is obtained to determine the final rank. Figure 1 depicts the fitness function for the mean weights of the dataset across 76 stakeholders. This is achieved by counting the numbers of requirements, where the DE simply add their sums and apportion precise values across requirements to determine their relative importance.

The results displayed in Table 1 shows the summary statistics of 50 experimental runs. For 10 requirements, the total number of attributes was 262 and the size of each cluster varied from 1 to 50 while, the mean and standard deviation of each cluster spanned from 1–30 and 15–30, respectively.

Also, Table 2 shows the results provided by each cluster that represents the 10 requirements during the course of running the algorithm on the data set. It displays the sum of weights, within-class variances, minimum distance to the centroid, average distance to the centroid and maximum distance to the centroids. Table 3 shows the distances between the class centroids for the 10 requirements across the total number of attributes while, Table 4 depict the analysis of each iteration. Analysis of multiple runs of this experiment showed exciting results as well. Using 500 trials, it was discovered that, the algorithm classified requirements correctly,

Fig. 1 Fitness function for mean weights

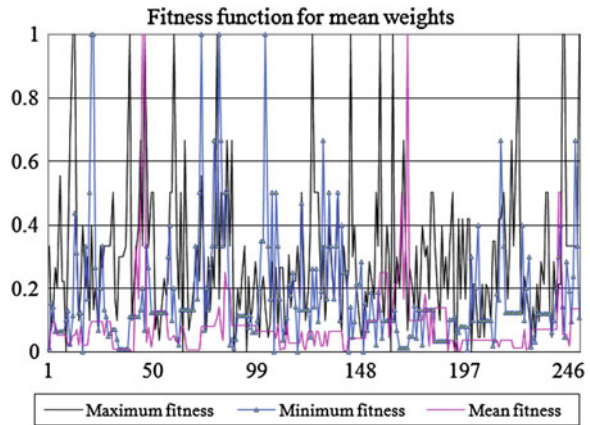


Table 1 Summary statistics

Variables	Obs.	Obs. with missing data	Obs. without missing data	Min	Max	Mean	Std. deviation
Point P	262	0	262	2.083	262	28.793	24.676
Rate P	262	0	262	0.000	262	5.123	15.864

Obs. = Objects

Table 2 Results by class

Class	1	2	3	4	5	6	7	8	9	10
Sum of weights	53	61	31	14	27	29	12	30	8	1
Within-class variance	7.302	8.283	37.897	172.896	2.393	12.699	3.607	1.992	1.190	0.000
Min. distance to centroid	0.532	0.365	0.352	3.769	0.695	0.232	0.663	0.253	0.412	0.000
Ave. distance to centroid	2.518	2.673	5.166	11.814	1.233	2.863	1.491	1.149	0.925	0.000
Max. distance to centroid	5.174	5.646	13.838	15.693	4.412	6.395	4.618	3.175	1.604	0.000

Min. = Minimum, Ave = Average, Max. = Maximum

where the determinants (W) for each variable were computed based on the stakeholder's weights. The sum of weights and variance for each requirement set was also calculated.

The learning process consists of finding the weight vector that allows the choice of requirements. The fitness value of each requirement can be measured on the basis of the weights vectors based on pre-defined criteria used to calculate the actual ranking. The disagreements between ranked requirements must be minimized as much as possible. We can also consider the disagreements on a larger number of top vectors, and the obtained similarity measure which can be used to enhance the agreement index. The fitness value will then be a weighted sum of these two similarity measures.

Definition 5.1 Let X be a measurable requirement that is endowed with attributes of σ -functionalities, where N is all subsets of X . A learning process g defined on the measurable space (X, N) is a set function $g : N \rightarrow [0, 1]$ which satisfies the following properties:

$$g(\phi) = 0, g(X) = 1 \quad (4)$$

But for requirements X, Y ; the learning process equation will be:

$$X, N \subseteq Y \in N \rightarrow [0, 1] \quad (5)$$

From the above definition, X, Y, N, g are said to be the parameters used to measure or determine the relative weights of requirement. This process is monotonic. Consequently, the monotonicity condition is obtained as:

$$g(X \cup Y) \geq \max\{g(X), g(Y)\} \text{ and } g(X \cap Y) \leq \min\{g(X), g(Y)\} \quad (6)$$

Table 3 Distances between class centroids

Class	1	2	3	4	5	6	7	8	9	10
1	0	9.702	35.486	68.298	7.053	22.502	3.929	12.849	10.292	441.143
2	9.702	0	45.180	77.989	16.745	32.193	12.148	22.536	19.985	446.808
3	35.486	45.180	0	32.812	28.436	12.987	33.529	22.644	25.196	422.664
4	68.298	77.989	32.812	0	61.247	45.797	66.249	55.453	58.008	407.820
5	7.053	16.745	28.436	61.247	0	15.450	5.847	5.796	3.241	437.356
6	22.502	32.193	12.987	45.797	15.450	0	20.614	9.657	12.212	429.318
7	3.929	12.148	33.529	66.249	5.847	20.614	0	11.135	8.780	441.935
8	12.849	22.536	22.644	55.453	5.796	9.657	11.135	0	2.568	434.397
9	10.292	19.985	25.196	58.008	3.241	12.212	8.780	2.568	0	435.539
10	441.143	446.808	422.664	407.820	437.356	429.318	441.935	434.397	435.539	0

Table 4 Statistics for each iteration

Iteration	Within-class variance	Trace (W)	Determinant (W)	Wilks'Lambda
0	1,128.978	289,018.282	3.00114E + 12	0.899
1	74.074	18,962.929	106503842.6	0.000
2	29.394	7,524.991	33285074.33	0.000
3	20.151	5,158.594	22195537.74	0.000

In the case where $g(X \cup Y) \geq \max\{g(X), g(Y)\}$, the learning function g attempts to determine the total number of requirements being prioritized and if $g(X \cap Y) \leq \min\{g(X), g(Y)\}$, the learning function attempts to compute the relative weights of requirements provided by the relevant stakeholders.

Definition 5.2 Let $h = \sum_{i=1}^n X_i \cdot 1_{X_i}$ be a simple function, where 1_{X_i} is the attribute function of the requirements $X_i \in N, i = 1, \dots, n$; X_i are pairwise disjoint, but if $M(X_i)$ is the measure of the weights between all the attributes contained in X_i , then the integral of h is given as:

$$\int h.dM = \sum_{i=1}^n M(X_i) \cdot x_i \tag{7}$$

Definition 5.3 Let X, Y, N, g be the measure of weights between two requirements, the integral of weights measure $g : N \rightarrow [0, 1]$ with respect to a simple function h is defined by:

$$\int h(r)g(r) = \vee(h(r_i) \wedge g(X_i)) = \max \min\{r'_i, g(Y_i)\} \tag{8}$$

Where $h(r_i)$ is a linear combination of an attribute function $1r_i$ such that $X_1 \subset Y_1 \subset \dots \subset X_n \subset Y_n$ and $X_n = \{r|h(r) \geq Y_n\}$.

Definition 5.4 Let X, N, g be a measure space. The integral of a measure of weights by the learning process $g : N \rightarrow [0, 1]$ with respect to a simple function h is defined by

$$\int h(r).dg \cong \sum [h(r_i) - h(r_{i-1})].g(X_i) \tag{9}$$

Similarly, if Y, N, g is a measure space; the integral of the measure of the weights with respect to a simple function h will be:

$$\int h(r').dg \cong \sum [h(r'_i) - h(r'_{i-1})].g(Y_i) \tag{10}$$

However, if g measures the relative weights of requirements, defined on a power set $P(x)$ and satisfies the definition 5.1 as above; the following attribute is evident:

$$\forall X, Y \in P(x), X \cap Y = \phi \Rightarrow g_2(X \cup Y) = g_2(X) + g_2(Y) + \lambda g_2(X)g_2(Y) \quad (11)$$

For $0 \leq \lambda \leq \infty$

In practical application of the learning process, the number of cluster which represents the number of requirements must be determined first. The attributes that describes each requirement are known as the data elements that are to be ranked. Therefore, before relative weights are assigned to requirements by stakeholders, attention should be paid to requirement dependencies issues in order to avoid redundant results. Prioritizing software requirements is actually determined by relative perceptions which will inform the relative scores provided by the stakeholders to initiate the ranking process.

Prioritizing requirements is an important activity in software development [31, 32]. When customer expectations are high, delivery time is short and resources are limited, the proposed software must be able to provide the desired functionality as early as possible. Many projects are challenged with the fact that, not all the requirements can be implemented because of limited time and resource constraints. This means that, it has to be decided which of the requirements can be removed for the next release. Information about priorities is needed, not just to ignore the least important requirements but also to help the project manager resolve conflicts, plan for staged deliveries, and make the necessary trade-offs. Software system's acceptability level is frequently determined by how well the developed system has met or satisfied the specified user's or stakeholder's requirements. Hence, eliciting and prioritizing the appropriate requirements and scheduling right releases with the correct functionalities are essential ingredients for developing good quality software systems. The matlab function used for k-means clustering which is `idx = kmeans(data, k)`, that partitions the points in the n-by-p data matrix `data` into `k` clusters was employed. This iterative partitioning minimizes the sum, over all clusters, of the within-cluster sums of point-to-cluster-centroid distances. Rows of data corresponds to attributes while columns corresponds to requirements. K-means returns an n-by-1 vector `idx` containing the cluster indices of each attribute which is utilized in calculating the fitness function to determine the ranks.

Further analysis was performed using a two-way analysis of variance (ANOVA). On the overall dataset, we found significant correlations between the ranked requirements. The results of ANOVA produced significant effect on the Rate P and Rank P with minimized disagreement rates (p-value = 0.088 and 0.083 respectively).

The requirements are randomly generated as population, while the fitness value is calculated which gave rise to the best and mean fitnesses of the requirement generations that were subjected to a stoappge criteria during the iteration processes. The best fitness stopping criteria option was chosen during the simulation process.

Table 5 Prioritized results

Class	Point P	Rate P
1	17.347	4.604
2	7.652	4.230
3	52.831	4.258
4	85.639	3.714
5	24.396	4.370
6	39.844	4.172
7	19.435	1.276
8	30.188	4.167
9	27.635	4.410
10	26.000	2.620

The requirements generations significantly increased while the mean and best values were obtained for all the requirements which will aid the computation of final ranks for all the requirements. Also, the best, worst and mean scores of requirements were computed. In the context of requirement prioritization, the best scores can stand for the most valued requirements while the worst scores can stand for the requirements that were less ranked by the stakeholders. The mean scores are the scores used to determine the relative importance of requirements across all the stakeholders for the software development project. Therefore, mutation should be performed with respect to the weights of attributes describing each requirement set.

50 independent runs were conducted for each experiment. The intra cluster distances obtained by clustering algorithms on test dataset have been summarized in Table 5. The results contains the sum of weights and within-class variance of the requirements. The sum of weights are considered as the prioritized results. The iterations depicted in Table 6 represents the number of times that the clustering algorithm has calculated the fitness function to reach the (near) optimal solution. The fitness is the average correlation for all the lists of attribute weights. It is dependent on the number of iterations to reach the optimal solution. As seen from Table 6, the proposed algorithm has produced the highest quality solutions in terms of the determinant as well as the initial and final within class variances. Moreover, the standard deviation of solutions found by the proposed algorithm is small, which means that the proposed algorithm could find a near optimal solution in most of the runs. In other words, the results confirm that the proposed algorithm is viable and robust. In terms of the number of function evaluations, the k-means algorithm needs the least number of evaluations compared to other algorithms.

Table 6 Optimization summary

Repetition	Iteration	Initial Within-class variance	Final Within-class variance	Determinant (W)
1	6	1,106.869	23.889	26,745,715.142
2	6	1,136.975	16.750	23,724,606.023
3	6	1,121.779	16.807	23,820,814.376
4	4	1,117.525	24.996	27,447,123.233
5	4	1,125.507	17.803	25,404,643.926
6	4	1,125.913	19.185	27,601,524.028
7	4	1,128.978	18.691	21,068,536.793
8	4	1,117.436	18.454	25,247,590.810
9	3	1,108.735	521.172	195,940,759,324.757
10	3	1,118.845	20.662	26,631342.963
11	3	1,094.139	464.243	176,968,571,045.744
12	3	1,132.570	21.818	28,704,234.905
13	3	1,126.964	524.758	231,605,057,844.517
14	3	1,122.652	21.945	31,405,941.682
15	3	1,123.154	23.738	35,101,366.712
16	3	1,119.050	24.730	36,708,579.807
17	3	1,102.570	361.772	163,961,516,715.797
18	3	1,110.858	526.407	261,348,714,992.157
19	3	1,113.150	463.952	201,111,726,242.012
20	3	1,112.708	21.817	29,895,099.111
21	3	1,115.139	23.943	35,497,900.830
22	3	1,124.640	21.747	27,948,507.938
23	3	1,126.804	25.552	29,231,104.006
24	3	1,133.939	22.136	29,327,321.253
25	3	1,088.150	29.300	40,797,707.187
26	3	1,126.818	518.924	210,390,947,859.052
27	3	1,134.805	25.031	36,453,233.568
28	3	1,126.092	22.147	28,437,781.242
29	3	1,105.032	24.348	35,559,759.396
30	3	1,115.281	466.852	188,849,288,951.012
31	3	1,072.613	36.720	52,595,047.023
32	3	1,117.914	521.401	173,748,277,159.091
33	3	1,113.605	19.234	27,416,942.756
34	3	1,116.027	19.304	25,746,288.325
35	3	1,122.918	20.620	26,253,031.369
36	3	1,118.186	26.385	34,310,522.697
37	3	1,136.200	22.156	29,263,683.373
38	3	1,120.218	25.017	33,223,123.872
39	3	1,124.442	520.499	211,383,774,209.661

(continued)

Table 6 (continued)

Repetition	Iteration	Initial Within-class variance	Final Within-class variance	Determinant (W)
40	3	1,120.536	26.388	35,745,608.991
41	3	1,131.305	26.587	37,270,986.568
42	3	1,125.849	579.831	128,941,063,793.784
43	3	1,100.849	522.269	176,077,098,304.402
44	3	1,121.924	25.182	36,102,923.262
45	3	1,116.445	25.120	36,854,622.383
46	3	1,116.223	522.626	228,664,613,438.719
47	3	1,116.968	36.729	55,171,737.251
48	3	1,127.872	20.857	29,381,923.033
49	3	1,130.837	20.633	27,643,374.936
50	3	1,104.055	587.943	187,806,412,544.351

6 Conclusion

The requirements prioritization process can be considered as a multi-criteria decision making process. It is the act of pair-wisely selecting or ranking requirements based on pre-defined criteria. The aim of this research was to develop an improved prioritization technique based on the limitations of existing ones. The proposed algorithm resolved the issues of scalability, rank reversals and computational complexities. The method utilized in this research consisted of clustering/evolutionary based algorithms. The validation of the proposed approach was executed with relevant dataset while the performance was evaluated using statistical means. The results showed high correlation between the mean weights which finally yielded the prioritized results. The approach described in this research can help software engineers prioritize requirements capable of forecasting the expected behaviour of software under development. The results of the proposed system demonstrate two important properties of requirements prioritization problem; (i) Ability to cater for big data and (ii) ability to effectively update ranks and minimize disagreements between prioritized requirements. The proposed technique was also able to classify ranked requirements by computing the maximum, minimum and mean scores. This will help software engineers determined the most valued and least valued requirements which will aid in the planning for software releases in order to avoid breach of contracts, trusts or agreements. Based on the presented results, it will be appropriate to consider this research as an improvement in the field of computational intelligence. In summary, a hybrid method based on differential evolution and k-means algorithm was used in clustering requirements in order to determine their relative importance. It attempts to exploit the merits of two algorithms simultaneously, where the k-means was used in generating the initial solution and the differential evolution was utilized as an improvement algorithm.

Acknowledgment The authors appreciate the efforts of the Ministry of Science, Technology and Innovation Malaysia (MOSTI) under Vot 4S062 and Universiti Teknologi Malaysia (UTM) for supporting this research.

References

1. Ahl, V.: An experimental comparison of five prioritization methods—investigating ease of use, accuracy and scalability. Master’s thesis, School of Engineering, Blekinge Institute of Technology, Sweden (2005)
2. Al-Sultan, K.S.: A Tabu search approach to the clustering problem. *Pattern Recogn.* **28**(9), 1443–1451 (1995)
3. Aritra, C., Bose, S., Das, S.: Automatic Clustering Based on Invasive Weed Optimization Algorithm: Swarm, Evolutionary, and Memetic Computing, pp. 105–112. Springer, Berlin (2011)
4. Babar, M., Ramzan, M., and Ghayyur, S.: Challenges and future trends in software requirements prioritization. In: *Computer Networks and Information Technology (ICCNIT)*, pp. 319–324, IEEE (2011)
5. Berander, P., Svahnberg, M.: Evaluating two ways of calculating priorities in requirements hierarchies—An experiment on hierarchical cumulative voting. *J. Syst. Softw.* **82**(5), 836–850 (2009)
6. Chang, D., Xian, D., Chang, W.: A genetic algorithm with gene rearrangement for K-means clustering. *Pattern Recogn.* **42**, 1210–1222 (2009)
7. Ching-Yi, C., and Fun, Y.: Particle swarm optimization algorithm and its application to clustering analysis. In *IEEE International Conference on Networking, Sensing and Control* (2004)
8. Das, S., Abraham, A., Konar, A.: Automatic clustering using an improved differential evolution algorithm. *IEEE Trans. Syst. Man Cybern. Part A Syst. Hum.* **38**(1), 218–237 (2008)
9. Das, S., Abraham, A., and Konar, A.: Automatic hard clustering using improved differential evolution algorithm. In: *Studies in Computational Intelligence*, pp. 137–174 (2009)
10. Demsar, J.: Statistical comparison of classifiers over multiple data sets. *J. Mach. Learn. Res.* **7**, 1–30 (2006)
11. Greer, D., Ruhe, G.: Software release planning: an evolutionary and iterative approach. *Inf. Softw. Technol.* **46**(4), 243–253 (2004)
12. Hatamlou, A., Abdullah, S., and Hatamlou, M.: Data clustering using big bang-big crunch algorithm. In: *Communications in Computer and Information Science*, pp. 383–388 (2011)
13. Jain, A.: Data clustering: 50 years beyond K-means. *Pattern Recogn. Lett.* **31**(8), 651–666 (2010)
14. Kao, Y., Zahara, E., Kao, I.: A hybridized approach to data clustering. *Expert Syst. Appl.* **34**(3), 1754–1762 (2008)
15. Karlsson, L., Thelin, T., Regnell, B., Berander, P., Wohlin, C.: Pair-wise comparisons versus planning game partitioning—experiments on requirements prioritization techniques. *Empirical Softw. Eng.* **12**(1), 3–33 (2006)
16. Karlsson, J., Olsson, S., Ryan, K.: Improved practical support for large scale requirements prioritizing. *J. Requirements Eng.* **2**, 51–67 (1997)
17. Karlsson, J., Wohlin, C., Regnell, B.: An evaluation of methods for prioritizing software requirements. *Inf. Softw. Technol.* **39**(14), 939–947 (1998)
18. Kassel, N.W., Malloy, B.A.: An approach to automate requirements elicitation and specification. In: *Proceeding of the 7th IASTED International Conference on Software Engineering and Applications*. Marina Del Rey, CA, USA (2003)
19. Kennedy, J., Eberhart, R.: Particle swarm optimization. In: *Proceedings of IEEE International Conference on Neural Networks* (1995)

20. Kobayashi, M., Maekawa, M.: Need-based requirements change management. In: Proceeding of Eighth Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems, pp. 171–178 (2001)
21. Krishna, K., Narasimha, M.: Genetic K-means algorithm. *IEEE Trans. Syst. Man Cyber. Part B (Cyber.)* **29**(3), 433–439 (1999)
22. Lim, S.L., Finkelstein, A.: takeRare: using social networks and collaborative filtering for large-scale requirements elicitation. *Softw. Eng. IEEE Trans.* **38**(3), 707–735 (2012)
23. Maulik, U., Bandyopadhyay, S.: Genetic algorithm-based clustering technique. *Pattern Recogn.* **33**(9), 1455–1465 (2000)
24. Moisiadis, F.: The fundamentals of prioritizing requirements. In: Proceedings of Systems Engineering Test and Evaluation Conference (SETE 2002) (2002)
25. Perini, A., Susi, A., Avesani, P.: A Machine Learning Approach to Software Requirements Prioritization. *IEEE Trans. Software Eng.* **39**(4), 445–460 (2013)
26. Qin, A.K., Suganthan, P.N.: Kernel neural gas algorithms with application to cluster analysis. In: Proceedings-International Conference on Pattern Recognition (2004)
27. Ramzan, M., Jaffar, A., Shahid, A.: Value based intelligent requirement prioritization (VIRP): expert driven fuzzy logic based prioritization technique. *Int. J. Innovative Comput.* **7**(3), 1017–1038 (2011)
28. Selim, S., Alsultan, K.: A simulated annealing algorithm for the clustering problem. *Pattern Recogn.* **24**(10), 1003–1008 (1991)
29. Sung, C.S., Jin, H.W.: A tabu-search-based heuristic for clustering. *Pattern Recogn.* **33**(5), 849–858 (2000)
30. Storn, R., Price, K.: Differential Evolution—A Simple and Efficient Adaptive Scheme for Global Optimization over Continuous Spaces, TR-95-012. *Int. Comput. Sci. Inst., Berkeley* (1995)
31. Tonella, P., Susi, A., Palma, F.: Interactive requirements prioritization using a genetic algorithm. *Inf. Softw. Technol.* **55**(2013), 173–187 (2012)
32. Thakurta, R.: A framework for prioritization of quality requirements for inclusion in a software project. *Softw. Qual. J.* **21**, 573–597 (2012)