Solving the next release problem by means of the fuzzy logic inference system with respect to the competitive market

Hamidreza Alrezaamiri , Ali Ebrahimnejad & Homayun Motameni

To cite this article: Hamidreza Alrezaamiri , Ali Ebrahimnejad & Homayun Motameni (2020) Solving the next release problem by means of the fuzzy logic inference system with respect to the competitive market, Journal of Experimental & Theoretical Artificial Intelligence, 32:6, 959-976, DOI: 10.1080/0952813X.2019.1704440

To link to this article: https://doi.org/10.1080/0952813X.2019.1704440

Published online: 21 Dec 2019.

JOURNAL OF EXPERIMENTAL & THEORETICAL ARTIFICIAL INTELLIGENCE
2020, VOL. 32, NO. 6, 959–976
https://doi.org/10.1080/0952813X.2019.1704440

ARTICLE

Solving the next release problem by means of the fuzzy logic inference system with respect to the competitive market

Hamidreza Alrezaamiri[a], Ali Ebrahimnejad[b] and Homayun Motameni[c]

[a]Department of Computer Engineering, Babol Branch, Islamic Azad University, Babol, Iran; [b]Department of Mathematics, Qaemshahr Branch, Islamic Azad University, Qaemshahr, Iran; [c]Department of Computer Engineering, Sari Branch, Islamic Azad University, Sari, Iran

ABSTRACT
A number of software programms are developed in several releases. Before developing any new release, a set of requirements is suggested for inclusion in the release. Having multiple constraints, it is impossible to develop all the requirements proposed in the next release. The presence of competing companies, replication of product ideas, shortening of the development time and lack of project funding will reduce the cost of developing a release. Developer teams should select a subset of the proposed requirements for development that would provide their clients with the highest amount of satisfaction despite the deadline limitations or cost constraints. The existence of conflicting goals and other constraints makes this choice very complicated. In this paper, an algorithm is introduced which is based on a fuzzy inference system to determine the suitability of each requirement for development in the next release. The proposed algorithm, rather than the developer team, takes the responsibility to select the optimal subset of requirements for the development of the next release. Experimental results of the proposed algorithm are then compared with the results of the genetic algorithm. The subset selected by the proposed algorithm provides much more satisfaction than the genetic algorithm.

ARTICLE HISTORY
Received 25 November 2018
Accepted 22 November 2019

KEYWORDS
Software requirements; Next release problem; Fuzzy Logic; Fuzzy Inference System; Competitive Market

Introduction

Today, software programms are required for management, monitoring and control in many industries. The complexity of the software structure has made software production methods more difficult. One of the methods of generating large software is the incremental development method. In the incremental development method, the software product is developed in several releases (Odzaly, Greer, & Stewart, 2017). In each release, a set of requirements is proposed for development. Due to various problems, it is almost impossible to develop all the proposed requirements in the current release. Software companies take several goals into consideration simultaneously prior to the production of each product. Examples of such goals are: an increase in client satisfaction, a reduction in production costs and a shorter delivery time of the next release. The presence of multiple software companies has greatly increased competition in product development in such a way that in many cases, new ideas and features of newly-presented software are copied by other companies. General and partial copying of the features of a software product has become a concern for manufacturing companies. This concern grows as the product is developed iteratively and in

CONTACT Ali Ebrahimnejad aemarzoun@gmail.com; a.ebrahimnejad@qaemiau.ac.ir Department of Mathematics, Qaemshahr Branch, Islamic Azad University, Qaemshahr, Iran

several releases. For this reason, manufacturing companies try to shorten the distance between the deliveries of successive releases in order to be ahead in the competition with other competitors. On the other hand, client satisfaction rate and software production costs are also important. Selecting an optimal subset of the proposed requirements which can provide clients with the most satisfaction in the shortest time and at the lowest cost has become a major challenge to software companies. The existence of several conflicting goals and constraints has made this choice very difficult (Mougouei & Powers, 2017). For example, the two goals, namely reducing the time and increasing satisfaction, are contradictory. An extreme reduction in production time reduces quality and satisfaction. Other constraints on this choice include interactions between requirements and setting the deadlines. The interactions between requirements are of different types. For example, two requirements may conflict with each other, or it also may not be possible for both of them to be developed in one release. Yet another type of interaction is when two requirements must necessarily be developed simultaneously. The deadline is also usually enforced by software companies themselves. For example, the next release should be available on the market for the next 9 months in all circumstances. Or, for example, the cost of producing a release should not exceed 75% of the considered budget.

The problem of selecting an optimal subset of requirements for development in the next release is known as the Next Release Problem (NRP) (Bagnall, Rayward-Smith, & Whittley, 2001). The research motivation is to introduce an intelligent method to select an optimal subset of requirements giving the highest level of clients' satisfaction in the shortest time and at the lowest cost, while satisfying the constraints of the problem. The challenge of this research is the existence of three conflicting goals and two kinds of constraints leading to a problem belonging to the category of NP-hard problems. The purpose of solving this problem is to help the developer team with the decision-making process. The Fuzzy theory is a suitable method for solving ambiguous and uncertain problems. This theory has so far solved many engineering problems that previously had uncertainties and conflicting goals (Alrezaamiri, Ebrahimnejad, & Motameni, 2019; Ebrahimnejad, Karimnejad, & Alrezaamiri, 2015; Ebrahimnejad, Tavana, & Alrezaamiri, 2016; Shirmohammadi & Hadadi, 2019, 2017).

If the subset of optimal requirements for development in the next release is selected by the members of the developer team itself without using any tools, these choices will certainly be subject to the style of the choices and human error. On the other hand, bad or non-optimal choices will surely lead to clients' dissatisfaction and a reduction in product quality (Ralph, 2018). In this paper, we present an algorithm that uses the fuzzy inference system to determine the suitability of each requirement for development in the next release. The proposed algorithm will then make greedy choices based on the suitability of each requirement and according to the constraints of the problem. Due to the use of fuzzy logic, the requirements selected by the proposed algorithm will lack human errors. The main contributions of this study are summarised as follows:

- This research has been conducted with a consideration of the competition scenario in the market and attention to reducing the risk of copying ideas of the software product.
- In addition to cost and customer satisfaction objectives, this research has also considered the objective of development time.
- In contrast to the existing techniques, the proposed algorithm uses a fuzzy inference system for solving a three-objective problem.
- In contrast to the existing techniques, utilising the proposed algorithm could reduce human errors in selection of optimal subsets.

The rest of the paper is organised as follows. Section 2 reviews the related literature. Section 3 explains the next release problem. Section 4 presents the proposed method. Section 5 examines the results of the experiments, and finally, Section 6 concludes the paper.

Literature review

So far, several algorithms and methods have been introduced to solve the next release problem. A full classification of these methods is given in (Hudaib, Masadeh, Qasem, & Alzaqebah, 2018; Pitangueira, Maciel, & Barros, 2015). Based on a simple grouping, most of the methods introduced for solving this problem fall into two general categories. The first category includes optimisation methods that use linear programming approaches, as well as heuristic and metaheuristic algorithms. The second category includes methods that rank the requirements and then select the best ranks. This category has used methods such as the Quality Function Deployment (QFD), Analytical Hierarchy Process (AHP) and Fuzzy Logic.

In 2001, Bagnall et al first solved the NRP problem (Bagnall et al., 2001). In this paper, authors first used the linear programming method to find the exact solution to the problem. Then, they used methods based on local search algorithms. In NRP problems with a low number of requirements, the linear programming method finds the exact solution to the problem at the right time. However, if the problem is large, the linear programming method cannot solve the problem in a reasonable time span. Other methods introduced in this paper failed to obtain high-quality solutions due to the weakness in their search capabilities. Veerapen, Ochoa, Harman, and Burke (2015) used the integer linear programming method in two problems, namely single-objective and multi-objective problems. They were able to find the exact solution to the problem in the cases of the single-objective and small multi-objective problems. However, their proposed method did not have a proper runtime in the case of large multi-objective problems. Mougouei (2016) proposed a linear programming model for selecting requirements according to the interactions between them. In this paper, they also presented a technique for modelling the interactions between requirements by means of a graph. NRP is inherently a multi-objective problem. However, in some papers, the authors have turned the problem into a single-objective problem by giving weight to each objective. Greer and Ruhe (2004) introduced a new method and used a genetic algorithm to determine the optimal subset of requirements. Araújo, Paixao, Yeltsin, Dantas, and Souza (2017) introduced an architecture based on genetic algorithm and machine learning to solve this problem. To select the optimal subset of requirements, Jiang, Zhang, Xuan, Ren, and Hu, (2010), combined the Ant Colony Optimisation algorithm (ACO) with the Hill Climbing algorithm leading to an increase in the quality of the problem solutions. Masadeh et al. (Masadeh, Alzaqebah, Hudaib, & Rahman, 2018) used the Grey Wolf Optimisation (GWO) algorithm to select the best proposed requirements. The Ants Colony System (ACS) algorithm has been used by Del Sagrado, Del Aguila, and Orellana (2015) for solving the multi-objective problem. In that paper, three types of interactions between requirements were defined and considered in the implementation. The Teaching-Learning-Based Optimisation (TLBO) algorithm was used to solve the next release problem by Chaves-González, Perez-Toledano, and Navasa (2015a). In that paper, three types of interactions between requirements were also considered. Chaves-González et al. (Chaves-González & Pérez-Toledano, 2015; Chaves-González, Perez-Toledano, & Navasa, 2015b) also used a Multi-Objective Artificial Bee Colony Optimisation (MOABC) algorithm and developed Differential Evolution with the Pareto Tournament (DEPT) algorithm to solve this problem. Xuan, Jiang, Ren, and Luo (2012) proposed a Backbone-based Multilevel Algorithm (BMA) to address the large scale NRP. In contrast to direct solving approaches, BMA employs multilevel reductions to downgrade the problem scale and multilevel refinements to construct the final optimal set of customers. In both reductions and refinements, the backbone is built to fix the common part of the optimal customers. Paixao and Souza (2015) presented a robust model, where the uncertainties related to the requirement's importance were modelled in a discrete way using the concept of scenarios (Paixao & Souza, 2015). Ferreira et al (Ferreira, Araújo, Neto, & de Souza, 2016) proposed an interactive model for the Next Release Problem using Ant Colony Optimisation, where the user can define which requirements he would like to include or exclude in the next release.

Karlsson has used two methods, AHP and QFD to select and prioritise software requirements. The requirements were classified and prioritised in the AHP and QFD methods, respectively. In projects with a high number of requirements, these two methods are not suitable due to their long runtime (Karlsson, 1996). Sadiq and Jain (2014) applied fuzzy preference relations for requirements' prioritisation in goal-based requirements in the elicitation process. This method was developed by combining weighted relationships and Fuzzy AHP (FAHP). The AHP method based on the cost-value prioritisation technique was used in several industrial projects by Chopra et al. (Chopra, Gupta, & Chauhan, 2016). In that paper, they also prioritised both the functional and non-functional requirements. Ramzan, Jaffar, and Shahid (2011) introduced the concept of the requirement value for prioritising requirements using fuzzy logic. The fuzzy inference system was used to prioritise requirements and overcome the uncertainty of the problem by Alrashoud and Alrashoud and Abhari (2015). They had considered three criteria, namely the importance of each requirement, risk and effort for the decision-making process. Alrashoud and Abhari (Alrashoud & Abhari, 2017) considered three criteria, namely stakeholders' satisfaction, risk and availability of resources in planning for the next release. Authors used the Adaptive Network-based Fuzzy Inference System (ANFIS) for solving the next release problem. In our most recent work, we formulated the NRP problem for the first time as a fuzzy multi-objective optimisation problem (Alrezaamiri et al., 2019). We used an artificial chemical reaction optimisation algorithm to solve this problem. In the implementation stage, we applied five interactions between requirements as one of the constraints of the problem for the first time. The results and diagrams of the proposed algorithm showed very reliable solutions. The results of related works in terms of utilised techniques, performance metrics, advantages, and disadvantages have been given in Table 1.

Formulating the next release problem

In this section, we will formulate the next release problem. In the NRP problem, n requirements are proposed for development in the next release. This set is represented by R $= \{r_1 \ldots r_n\}$. Each requirement has a cost and an estimated time for development. The cost and development time of each requirement are estimated by the developer team. The cost of each requirement is defined in the set E $= \{e_1 \ldots e_m\}$and the development time is also defined in the set T $= \{t_1 \ldots t_n\}$. The set R is proposed by the m client denoted by the set $C = \{c_1 \ldots c_m\}$. Each client has an importance level for the software company. Also, the clients' importance levels are defined with the W $= \{w_1 \ldots w_m\}$ set. It is assumed that each client in C gives a value to each requirement in R. The value that a requirement r_j has for a particular client c_i is given by an amount $v_{ij} > 0$. A matrix of $m \times n$ holds all the importance values v_{ij}. The total satisfaction $b_j$of a given requirement r_j is calculated as the weighted sum of its values for all the clients considered, and can be expressed as indicated in formula (1). The set of the total satisfaction calculated in that way is denoted by $B = \{b_1, b_2, \ldots, b_n\}$(Chaves-González et al., 2015b).

$$b_j = \sum_{i=1}^{m} w_i * v_{ij} \tag{1}$$

Thus, for each requirement r_i, a cost c_i, a development time t_i and a satisfaction amount b_i are considered. The objective of the problem is to select a subset of requirements that provide the highest satisfaction for clients with the least cost and in the shortest time of development. In addition to these three objectives, there are two constraint categories commonly used for this problem. The first category of constraints is the threshold amount of the development cost or the threshold amount of the development time of release, which is considered by the software company. The cost of developing a release is equal to the sum of the cost of each developed requirement. Despite this constraint, a number of requirements will be selected whose total cost of development does not exceed the threshold value. The constraint of the threshold amount of the development time is also applied when the software company sets a deadline (of six months, for example) for the developer team to finish the release process depending on the market

Table 1. Results of different approaches.

Ref-Aut-year	Method	Metric	Advantage	Disadvantage
Bagnall et al. (2001)	Linear programming & Local search	Max profit Min Cost	Problem well formulated	Long runtime
Ralph (2018)	Liner Programming	HV, NDS	Finds exact solution	Long runtime
Pitangueira et al. (2015)	New Method	Overall Value	Introduced a new modal (GORS)	Cost dependencies were not considered in the model
Hudaib et al. (2018)	New Method	Combination of benefit and penalty	Introduced a powerful method (Evolve)	Uncertainty was not considered in method
(Veerapen et al., 2015)	New Method (IGA)	Similarity Degree Similarity Factor Price of Preference	Introduced a new Architecture	Architecture just considered one stakeholder subjectively
(Chaves-González et al., 2015b)	New Method (PRFGORE)	Ranking Values	Linguistic variables used in the method	Interaction between requirements not considered
(Alrashoud & Abhari, 2017)	New Method (BMA)	Ratio CBS Ratio OC in the CB	Solves the large scale NRP	Only one kind of requirement interaction considered
(Hsieh, Hsu, & Lin, 2018)	New Model (ROF)	Reduction Factor	Considered uncertainty in the NRP	Used of weak evolutionary algorithms to evaluate the model
(Xiang, Yu, Lapierre, Zhang, & Zhang, 2018)	New Model (ACO)	Effectiveness Reduction Factor	Considered interactive algorithm	Interaction between requirements not considered
Mougouei (2016)	Meta Heuristic (ACO+Hill Climbing)	Max profit Runtime	Introduced a simple method	Just a single objective considered
(Greer & Ruhe, 2004)	Meta Heuristic (GWO)	Ratio cost/value runtime	Short runtime	The paper did poor evaluation
Araújo et al. (2017)	Meta Heuristic (ACS)	NDS, HV, Spread, Spacing, Runtime	Paper did good evaluations	Uncertainty not considered in method
(Jiang et al., 2010)	Meta Heuristic (TLBO)	NDS, HV, Spread, Runtime	Introduced a strong method	Uncertainty not considered in method
(Masadeh et al., 2018)	Meta Heuristic (MOABC)	NDS, HV, Spread, Runtime	Introduced an interesting method	Uncertainty not considered in method
(Del Sagrado et al., 2015)	Meta Heuristic (DEPT)	NDS, HV, Spread, Runtime	Introduced a simple and attractive method	Uncertainty not considered in method
(Alrashoud & Abhari, 2015)	Meta Heuristic (ACRO)	NDS, HV, Spread	Five interactions between requirements were considered	Requirement features were just the kind of triangular fuzzy numbers
(Chaves-González et al., 2015a)	AHP, QFD	Pair-wise comparison	Introduced the first approach to solve the NRP	Very long runtime
(Chaves-González & Pérez-Toledano, 2015)	AHP	Pair-wise comparison	Functional & non-functional requirements considered	The paper did poor evaluation
(Chaves-González & Pérez-Toledano, 2015)	Fuzzy logic	Requirement value	Introduced a multi-level value based technique	Interaction between requirements not considered
(Paixao & Souza, 2015)	Fuzzy Inference System (FIS)	Rank	For the first time, three objectives considered	Just two interactions between requirements considered
(Ferreira et al., 2016)	Adaptive Network-based Fuzzy Inference System	Rank	Used historical data and the learned process of ANFIS	Dependency between requirements not considered

and competitors. Despite this constraint, a number of requirements will be selected whose total amount of development time is less than the specified deadline.

The second category of constraints is the interactions between the requirements. There are different types of interactions and dependencies between requirements which manifest themselves as constraints in the problem. In paper (Del Sagrado et al., 2015), the interaction between the requirements was classified into four broad categories.

-*Implication*. $r_i \Rightarrow r_j$ If the requirement r_i is not developed, the requirement r_j cannot be developed either.

-*Combination*. $r_i \oplus r_j$. A requirement r_i cannot be chosen separately from a requirement r_j.

-*Exclusion*. $r_i \otimes r_j$. A requirement r_i cannot be chosen together with a requirement r_j.-*Modification*. The development of the requirement implies that some other requirements change their satisfaction or implementation effort.

Despite these objectives and constraints, determining priority between requirements is a very complicated task. We will use the fuzzy inference system to prioritise the requirements in these conditions.

The proposed method

As explained in the previous sections, selecting an optimal subset of the proposed requirements has become an important challenge for developer teams. The existence of conflicting goals has placed this problem in the NP-hard problems category. Considering several opposing objectives, as well as several constraints and problem enlargements, most of the solutions presented earlier are ineffective or difficult. For example, linear programming methods for large and complex problems will have a very long runtime. Single-objective metaheuristic algorithms are faced with the challenge of weighting each objective. In addition, some objectives must be maximised and some others must be minimised. The output of multi-objective metaheuristic algorithms is a set of solutions. The developer team must select one of them again as the final solution. These choices and weights are always accompanied by human errors. Our proposed algorithm uses a fuzzy inference system to solve this problem and then makes greedy choices based on constraints. The Fuzzy Inference System (FIS) has great ability in helping solve complex decision-making problems. The fuzzy inference system is established based on the if-then rules as a result of which it is possible to obtain the relation between a number of input and output variables. Therefore, FIS can be used as a prediction model for situations where input and output data are highly uncertain (Hsieh et al., 2018; Pourjavad & Shahin, 2018; Xiang et al., 2018).

To make a fuzzy inference, a basic fuzzy system is determined based on the observational data. Then, the Premise and Consequent parts are fuzzified using fuzzy membership functions. In the following, different parts of the Premise of each of the rules are combined after which the effect of each rule on the final output of the system is determined. Finally, the Consequent parts of rules are combined in the form of a fuzzy set to obtain the final output of a system. By using defuzzification methods, the fuzzy output of the system becomes a definite number. In this paper, we use the Mamdani implication method which uses the maximum-minimum method for combining fuzzy rules. Accordingly, to determine the suitability of each requirement, we use three input fuzzy variables and one fuzzy output variable for development in the next release. For each of the requirement characteristics, namely the development time, the development cost and the satisfaction rate, an input fuzzy variable is defined. Also, for each of the input fuzzy variables, we consider three linguistic variables, namely low, medium, and high. The only fuzzy output variable of the fuzzy inference system is the degree of suitability of each requirement which has five linguistic variables: very low, low, medium, high and very high. Triangular and trapezoidal membership functions have been used to display linguistic variables. The triangular and trapezoidal membership functions used in the fuzzy inference system can be seen in formulas (2) and (3), respectively. Table 2 shows 27 rules for the fuzzy inference engine.

Table 2. Fuzzy if – then rules.

	Time	Cost	Satisfaction	Suitability		Time	Cost	Satisfaction	Suitability
1	Low	Low	High	Very High	15	Medium	Medium	Low	Low
2	Low	Low	Medium	Very High	16	Medium	High	High	Medium
3	Low	Low	Low	High	17	Medium	High	Medium	Low
4	Low	Medium	High	Very High	18	Medium	High	Low	Very Low
5	Low	Medium	Medium	High	19	High	Low	High	High
6	Low	Medium	Low	Medium	20	High	Low	Medium	Medium
7	Low	High	High	High	21	High	Low	Low	Very Low
8	Low	High	Medium	Medium	22	High	Medium	High	Medium
9	Low	High	Low	Low	23	High	Medium	Medium	Low
10	Medium	Low	High	Very High	24	High	Medium	Low	Very Low
11	Medium	Low	Medium	High	25	High	High	High	Low
12	Medium	Low	Low	Medium	26	High	High	Medium	Very Low
13	Medium	Medium	High	High	27	High	High	Low	Very Low
14	Medium	Medium	Medium	Medium					

$$\mu(x) = \begin{cases} 0 & x \leq a \\ \frac{x-a}{b-a} & a < x \leq b \\ \frac{c-x}{c-b} & b \leq x < c \\ 0 & c \leq x \end{cases} \tag{2}$$

$$\mu(x) = \begin{cases} 0 & x \leq a \\ \frac{x-a}{b-a} & a < x \leq b \\ 1 & b \leq x \leq c \\ \frac{d-x}{d-c} & c \leq x < d \\ 0 & d \leq x \end{cases} \tag{3}$$

Two prominent rules in this table are:

- If the development time of a requirement is low, and the cost of its development is also low while its satisfaction rate is high, the suitability of this requirement for development is very high.
- If the development time of a requirement is high, and the cost of its development is also high while its satisfaction rate is low, the suitability of this requirement for development is very low.

In algorithm (1), you can see the proposed method. This algorithm is written with a focus on the limit of the time threshold value. This algorithm can also be used with a slight change for problems with cost threshold constraints.

The inputs of the algorithm are the amount of time, cost, and overall satisfaction with each requirement. The fuzzy rules and the threshold value are also received from the input. The outputs of the algorithm, i.e. the development cost, the development time and satisfaction rate are the subset of the selected requirements. The selected requirements are stored in the NextRel set. The capacity variable holds the development time of the selected requirements. The value of this variable is 0 before the requirements are selected. The algorithm first runs the fuzzy inference system. After defuzzification, the suitability level of each developmental requirement for the next release is determined as a number. The algorithm creates a data structure called Req which holds the suitability, cost, time, and overall satisfaction with each requirement. Then, it arranges these structures according to their suitability. In the loop, the requirements are assessed with respect to their suitability in order to examine the constraints. For each requirement, the type-1 constraint (time threshold value) is checked first. If it does not violate the type-1 constraint requirement, the type-2 constraint (interaction between requirements) will be checked. In Section 3, three categories of interaction between requirements were defined. In the proposed algorithm, when facing any requirement r_i that is dependent, we perform one of the following actions in accordance with its interaction type at the time of greedy choices.

Algorithm 1: A pseudocode of the proposed method.

```
1- input threshold, t, e, b, fuzzy rules.
2-output NextRel, cost, satisfaction, capacity.
– – – – – – – – – – – – – – – – – – – – – – – – – – – – – – – – – – – – – – – – –
3-initialise capacity = 0, cost = 0, satisfaction = 0, NextRel = empty;
4-for I = 1: n
5-suitability(i) = FIS (t, e, b)          /* calculate suitability requirement i using fuzzy if-then rules given in table 1. */
6-end for
7- build structure Req and sort them based score suitability;
8-for I = 1: n                                                                  /* for each requirement */
9-    if (t.Req(i) + capacity < threshold)                                      /* investigate constraint type 1 */

10-      if the requirement Req(i) do not have an interaction                   /* investigate constraint type 2 */
11-        NextRel = NextRel ∪ Req(i)                          /* Req(i) selected for development in the next release.*/
12-        capacity = capacity + t.Req(i);   /* amount of time developing the requirements that have been selected so far */
13-        cost = cost + e.Req(i);
14-        Satisfaction = Satisfaction + b.Req(i);

15-    else                                        /* If the requirement Req(i) has an interaction with other requirements*/
16-          if ($Req_i \oplus Req_j$)
17-            If (t.Req(i) + t.Req(j) + capacity < threshold)                  /* both requirements select together */
18-                  NextRel = NextRel ∪ Req(i) ∪ Req(j);
19-                  capacity = capacity +t.Req(i) + t.Req(j);
20-                  cost = cost + e.Req(i) + e.Req(j);
21-                  satisfaction = satisfaction + b.Req(i) + b.Req(j);
22-            end if
23-            else if ($Req_j \rightarrow Req_i$)
24-            if(Req(i) be selected before)
25-                   NextRel = NextRel ∪ Req(i);
26-                   capacity = capacity + t.Req(i);
27-                   cost = cost + e.Req(i);
28-                   satisfaction = satisfaction + b.Req(i);
29-            end if
30-        elseif ($Req_i \otimes Req_j$)
31-            if (Req(j) don not be selected before)
32-                   NextRel = NextRel ∪ Req(i);
33-                   capacity = capacity + t.Req(i);
34-                   cost = cost + e.Req(i);
35-                   satisfaction = satisfaction + b.Req(i);
36-            end if
37-      end if
38-   end if
39-   end if
40- end for
```

If the requirement r_i has an interaction *combination* with a requirement r_j, both requirements are selected for development in the next release in case the total development time of both requirements, do not violate the first constraint. Otherwise none will be selected.

If the requirement r_i has an interaction *Implication* with a requirement r_j, the r_i requirement cannot be selected if the requirement r_j has not already been developed. Otherwise, if r_j has already been developed and r_i does not violate the first constraint, the requirement r_i will be selected for development in the next release.

If the requirement r_i has an interaction *Exclusion* with a requirement r_j, requirement r_i is selected if the requirement r_j is not already selected and does not violate the first constraint requirement r_i.

Note that the time complexity of the proposed algorithm is $O(NM^2)$for FIS and $O(N)$ for greedy choices. Here N is the number of requirements and M presents the length of intervals created by the membership functions. Since M is a coefficient, the total time complexity of the proposed algorithm will be $O(N)$.

Although many researchers solved the next release problem as a NP-complete problem using evolutionary algorithms, here we solve this problem based on a different point of view, and by the help of fuzzy logic, to decide which requirements should develop in the next release. However, as argued by Alrashoud and Abhari (Alrashoud & Abhari, 2015) using the fuzzy inference system for solving three-objective problems leads to better results in contrast to evolutionary algorithms.

Experiments

In this section, we will first introduce the datasets used in the experiments. Then, we present the results of the experiments on the proposed method and compare them.

Test methods and datasets used

All tests were carried out on a system by the specifications of: CPU core i7 1.6GHz, Ram 4GB and OS win 10 64bit using the MATLAB R2016a. To better investigate the proposed method, a small dataset and a large dummy dataset were used. Table 3 shows the first dataset. This dataset contains 24 requirements and 5 clients. Satisfaction rate for each requirement has been surveyed by clients. The score that clients gave to each requirement was in the range of 1 to 5 where 1 is the lowest score and 5 is the highest. Giving a score of 1 to a requirement means that the client has the least interest in developing this requirement in the next release. The time required for developing each requirement is determined by the developer team according to the strength and personnel number of each team (Paredes-Valverde, Del Pilar Salas-Zárate, Colomo-Palacios, Gómez-Berbís, & Valencia-García, 2018). The cost of developing each requirement is determined by the developer team.

The second dataset is larger and more complex than the first. Table 4 presents this data which consists of 80 requirements and 5 clients. The interaction between requirements has also been presented in this dataset. In this dataset, there are 4 interactions of *Implication* type, 4 interactions of *combination* type and 4 interactions of *Exclusion* type. The existence of these interactions as the type-2 constraint makes solving the problem more complicated.

In the first and second datasets, clients have different levels of importance for the software company. This level of importance is used to determine the final satisfaction level of each requirement in Formula (1). For example, the final satisfaction level of r_1 in dataset 1 is equal to 23 according to Formulation (1):

$$b_1 = \sum_{i=1}^{m} w_i * v_{i1} = 2*2+1*1+4*2+3*2+4*1 = 23$$

Table 5 shows the importance level of clients in each dataset.

Results and analysis

In this subsection, we will evaluate the proposed algorithm on two datasets. Figure 1 shows the membership function diagram of the linguistic variables of each of the input and output variables

Table 3. Dataset 1.

	r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8	r_9	r_{10}	r_{11}	r_{12}	r_{13}	r_{14}	r_{15}	r_{16}	r_{17}	r_{18}	r_{19}	r_{20}	r_{21}	r_{22}	r_{23}	r_{24}
Client$_1$	2	3	1	4	3	4	2	4	3	2	4	4	4	2	5	5	2	4	2	4	3	2	3	1
Client$_2$	1	2	3	3	2	5	4	2	3	3	4	4	3	1	3	4	1	2	4	2	1	2	3	2
Client$_3$	2	3	4	4	2	4	5	1	4	2	4	3	3	3	5	3	3	3	3	5	4	5	4	2
Client$_4$	2	4	3	5	3	4	3	5	2	5	3	2	3	2	2	3	5	2	3	4	1	1	3	1
Client$_5$	1	2	2	2	2	5	4	3	3	4	2	5	4	3	4	2	4	2	4	3	2	3	4	1
Cost	5	8	3	10	4	9	8	4	5	5	6	7	3	2	8	8	3	4	6	7	9	7	10	1
Time	6	5	4	8	6	7	8	3	6	3	4	8	2	1	7	10	9	6	5	8	6	4	3	2

Table 4. Dataset 2.

	r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8	r_9	r_{10}	r_{11}	r_{12}	r_{13}	r_{14}	r_{15}	r_{16}	r_{17}	r_{18}	r_{19}	r_{20}
Client$_1$	2	3	1	1	2	3	3	4	1	3	4	4	3	2	3	2	2	3	4	2
Client$_2$	4	2	3	1	1	2	4	2	2	1	2	3	3	2	1	2	3	3	3	1
Client$_3$	1	3	2	2	3	3	2	3	4	1	3	3	4	1	3	1	2	2	3	3
Client$_4$	3	2	2	0	2	1	3	2	3	2	3	2	1	3	4	2	2	1	3	0
Client$_5$	1	2	3	1	3	2	2	3	3	3	2	2	3	3	2	1	1	4	3	3
Cost	14	19	15	8	17	15	9	11	12	15	14	12	14	20	9	3	10	8	14	4
Time	8	17	14	9	16	12	8	13	9	15	12	7	13	19	7	5	4	9	16	6
	r_{21}	r_{22}	r_{23}	r_{24}	r_{25}	r_{26}	r_{27}	r_{28}	r_{29}	r_{30}	r_{31}	r_{32}	r_{33}	r_{34}	r_{35}	r_{36}	r_{37}	r_{38}	r_{39}	r_{40}
Client$_1$	3	1	4	1	3	3	3	3	4	2	2	3	2	1	2	2	1	3	3	2
Client$_2$	3	3	3	2	1	5	2	2	3	3	1	3	5	2	4	4	3	2	3	1
Client$_3$	2	2	2	3	2	3	3	4	3	3	3	2	4	1	2	3	1	3	1	2
Client$_4$	3	1	2	3	3	3	4	4	3	4	2	2	2	2	3	2	2	3	5	1
Client$_5$	1	2	3	2	3	4	2	3	2	3	2	4	3	3	2	2	3	5	5	1
Cost	5	10	6	7	6	15	9	19	13	12	5	2	17	6	4	14	9	13	18	4
Time	2	7	11	8	10	16	13	17	14	12	2	6	18	10	9	12	9	17	20	4
	r_{41}	r_{42}	r_{43}	r_{44}	r_{45}	r_{46}	r_{47}	r_{48}	r_{49}	r_{50}	r_{51}	r_{52}	r_{53}	r_{54}	r_{55}	r_{56}	r_{57}	r_{58}	r_{59}	r_{60}
Client$_1$	2	2	3	1	1	1	2	2	3	3	3	3	4	3	2	1	3	1	3	2
Client$_2$	4	3	1	1	3	2	5	2	4	3	2	2	1	2	1	2	3	2	1	1
Client$_3$	1	3	2	3	3	3	3	3	3	2	3	1	2	3	1	2	2	1	2	3
Client$_4$	3	4	3	3	4	4	1	1	3	5	2	1	3	2	1	3	4	4	2	3
Client$_5$	3	1	1	2	1	2	3	3	2	2	1	3	3	3	1	1	2	1	1	2
Cost	7	16	14	20	13	8	16	12	9	12	5	3	17	8	1	3	14	14	17	8
Time	11	17	14	20	7	9	12	13	14	13	5	3	18	9	1	4	10	13	19	8
	r_{61}	r_{62}	r_{63}	r_{64}	r_{65}	r_{66}	r_{67}	r_{68}	r_{69}	r_{70}	r_{71}	r_{72}	r_{73}	r_{74}	r_{75}	r_{76}	r_{77}	r_{78}	r_{79}	r_{80}
Client$_1$	2	2	3	5	4	3	4	3	2	1	1	3	2	3	1	1	2	3	3	1
Client$_2$	1	3	2	3	5	2	4	2	3	2	5	2	1	3	2	4	3	3	1	2
Client$_3$	2	1	2	3	3	1	3	3	3	1	3	1	3	4	2	2	3	4	1	1
Client$_4$	2	2	3	3	3	3	2	3	2	1	2	3	3	2	2	2	2	3	3	1
Client$_5$	4	1	3	4	1	3	2	1	4	2	2	2	4	2	2	3	2	3	2	3
Cost	10	16	15	18	13	15	9	8	20	1	6	9	3	15	4	17	11	19	7	14
Time	16	15	14	8	12	11	9	5	20	3	7	14	4	13	1	16	11	18	7	15
Interactions		$r_{41} \Rightarrow r_{27}$				$r_{13} \Rightarrow r_{28}$			$r_3 \Rightarrow r_{29}$			$r_{34} \Rightarrow r_{65}$			$r_7 \oplus r_{66}$			$r_{14} \oplus r_{37}$		
		$r_{24} \oplus r_{38}$				$r_{16} \oplus r_{39}$			$r_{21} \otimes r_{22}$			$r_{32} \otimes r_{33}$			$r_{46} \otimes r_{47}$			$r_{65} \otimes r_{66}$		

Table 5. Clients' Level of importance.

Clients weights	Client$_1$	Client$_2$	Client$_3$	Client$_4$	Client$_5$
Dataset 1	2	1	4	3	4
Dataset 2	3	2	1	5	1

used to solve dataset 1. The results of the evaluation of the proposed algorithm on dataset 1 are given in Tables 6 and 7. This dataset has no constraints on interactions between requirements and is used as a not-so-complex dataset. Table 6 considered the five cost threshold amount constraint and Table 7 considered the four-time threshold amount constraints. We used the genetic algorithm for comparisons (Greer & Ruhe, 2004). So far, no research in this area has been carried out considering the three objectives of satisfaction, time, and cost, and there has been no similar method for comparison. The genetic algorithm in paper (Greer & Ruhe, 2004) planned the set of requirements for several releases. Here, our attention is just on planning for the next release. Since the genetic algorithm is a randomisation-based one, an average of 20 independent runs is presented in the tables where the results of the proposed algorithm are always the same for each particular defuzzification.

Table 6 considers five constraints on the cost threshold value. For example, the 80% cost threshold value constraint on this dataset equals 113.6 units. That is, the developer team can use up to a maximum of 80% of the overall project budget. The overall project budget is equal to the cost of

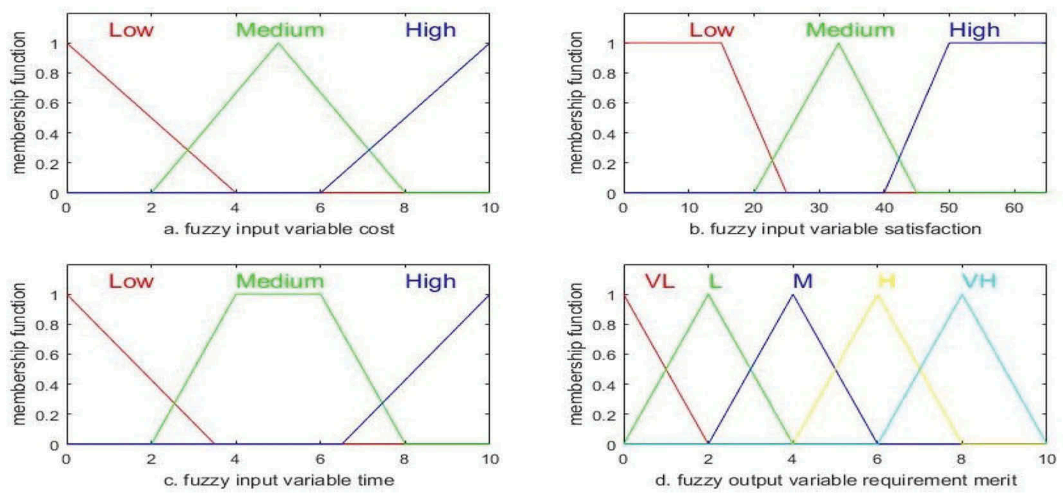

Figure 1. The membership function diagrams of the linguistic of fuzzy data.

Table 6. Constraint cost threshold.

Defuzzy		Benefit	Cost	Time	constraint
FIS	sos	935	126	113	Max 90% cost = 127.8
	som	954	125	115	
	sol	944	125	117	
	centroid	935	126	113	
	Bisector	954	125	115	
GA	-	917	127	110.8	
FIS	sos	845	108	100	Max 80% cost = 113.6
	som	851	107	99	
	sol	854	107	104	
	centroid	851	107	99	
	Bisector	851	107	99	
GA	-	851.3	110	98.2	
FIS	sos	811	99	94	Max 70% cost = 99.4
	som	811	99	94	
	sol	811	99	94	
	centroid	811	99	94	
	Bisector	811	99	94	
GA	-	793.6	98.4	87.2	
FIS	sos	716	85	82	Max 60% cost = 85.2
	som	720	82	87	
	sol	725	85	85	
	centroid	707	85	78	
	Bisector	707	85	78	
GA	-	706.5	84.6	76.7	
FIS	sos	595	70	59	Max 50% cost = 71
	som	582	68	64	
	sol	609	68	71	
	centroid	618	70	66	
	Bisector	611	70	67	
GA	-	616.3	70.8	70.1	

developing all the proposed requirements. In this experiment, the goal is to select a subset of requirements so that the total cost of their developing does not exceed the cost threshold value. It should also meet the following two criteria; firstly, the highest clients satisfaction and, secondly, the minimum development time.

Table 7. Constraint time threshold.

Defuzzy		Benefit	Cost	Time	constraint
FIS	som	935	126	113	Max 4 month = 120 days
	mom	954	125	115	
	lom	944	125	117	
	centroid	935	126	113	
	Bisector	954	125	115	
GA	-	932.25	122.11	118.23	
FIS	som	788	94	88	Max 3 month = 90 days
	mom	720	82	87	
	lom	775	95	88	
	centroid	788	94	88	
	Bisector	788	94	88	
GA	-	787.1	97.13	88.4	
FIS	som	554	66	56	Max 2 month = 60 days
	mom	527	60	57	
	lom	522	57	57	
	centroid	568	63	58	
	Bisector	568	63	58	
GA	-	566.4	73.71	58.3	
FIS	som	331	38	26	Max 1 month = 30 days
	mom	319	31	27	
	lom	319	31	27	
	centroid	322	32	26	
	Bisector	319	31	27	
GA	-	350.2	41.33	28.6	

The proposed algorithm was evaluated with five different defuzzifiers. Table 6 shows the best results of each test with a dark background. The results show that the type of defuzzy function affects the output of the proposed method. In other words, changing the defuzzy function may change the selected subset. The proper choice of the defuzzy function depends on the decision-maker's view. In all tests, the proposed algorithm has more benefits than the genetic algorithm. In this multi-objective problem, upper bounds have been chosen for cost and time goals leading to considering these goals as the constraints. So the benefit goal is in the first priority.

In Table 7, the time threshold value constraint was considered; that is, for example, the developer team had to deliver the next release of the software for up to the next three months according to its number of members. This table shows that the genetic algorithm has had better choices within a one-month time limit. However, as the deadline for development extended, the proposed algorithm has had better choices.

Tables 8 and 9 show the results of experiments on the second dataset. The number of requirements in this dataset is very high, and there is a constraint of interaction between requirements. In Table 8, five tests were performed with the constraint of a different cost threshold value. In all of these tests, the proposed algorithm yielded more satisfaction than the genetic one. For example, within 90% cost limit, the proposed algorithm with the *sol* defuzzifier could achieve 9% more satisfaction than the GA algorithm, while the cost and time values are approximately equal. Also within 60% cost limit, the proposed algorithm could achieve 10% more satisfaction than the GA algorithm. With decreasing the cost threshold, the satisfaction of the proposed algorithm was also better than the genetic one.

In Figure 2, runtimes of two algorithms in three cost constraints 50%, 70% and 90% are shown. This comparison is based on dataset 2. In all constraints, the proposed algorithm has less runtime. When a cost constraint is increased (from 50% to 90%), the proposed algorithm can choose more requirements, then its runtime increases. However, the GA runtime decreases in this situation since GA chromosomes, after any operation, should be validations. Each invalid chromosome should be repaired to avoid transgression from the considered constraints in the problem. The more tightened the cost constraint is, the more the chromosomes need repair. Repairs cause the difference in runtime of GA.

In Table 9, four tests are considered with different threshold times. For example, one of the constraints is the 730-day time limit for the next release. This dataset can be a real example of

Table 8. Constraint cost threshold.

Defuzzy		Benefit	Cost	Time	constraint
FIS	sos	2358	791	785	Max 90% cost = 795.6
	som	2329	787	782	
	sol	2367	789	788	
	centroid	2346	785	782	
	Bisector	2346	785	782	
GA	-	2169.8	787.2	785.9	
FIS	sos	2193	700	693	Max 80% cost = 707.2
	som	2194	707	701	
	sol	2211	700	704	
	centroid	2180	702	691	
	Bisector	2175	700	692	
GA	-	2024.2	699.5	702.1	
FIS	sos	2010	611	611	Max 70% cost = 618.8
	som	2006	617	608	
	sol	2026	612	611	
	centroid	1996	617	606	
	Bisector	1991	617	611	
GA	-	1835.6	610.7	606.1	
FIS	sos	1833	528	523	Max 60% cost = 530.4
	som	1833	528	523	
	sol	1824	529	529	
	centroid	1831	525	524	
	Bisector	1831	525	524	
GA	-	1659.4	522.6	522.6	
FIS	sos	1617	439	446	Max 50% cost = 442.0
	som	1570	441	442	
	sol	1544	439	445	
	centroid	1653	440	447	
	Bisector	1653	440	447	
GA	-	1477.4	436.2	444.8	

Table 9. Constraint time threshold.

Defuzzy		Benefit	Cost	Time	constraint
FIS	som	2240	725	721	Max 2 years = 730 days
	Mom	2240	725	721	
	Lom	2256	726	723	
	centroid	2231	728	719	
	Bisector	2226	726	720	
GA	-	2094.6	727.2	727.8	
FIS	som	1859	546	539	Max 1.5 Years = 547 days
	mom	1866	543	537	
	lom	1892	551	542	
	centroid	1862	542	542	
	Bisector	1862	542	542	
GA	-	1715.8	536.4	540.4	
FIS	som	1362	343	356	Max 1 year = 365 days
	mom	1310	353	358	
	lom	1284	366	357	
	centroid	1389	343	354	
	Bisector	1389	343	354	
GA	-	1252.8	353.1	355.7	
FIS	som	839	190	179	Max 6 month = 183 days
	mom	776	189	178	
	lom	778	200	183	
	centroid	796	159	183	
	Bisector	796	159	183	
GA	-	737.3	182.4	178.2	

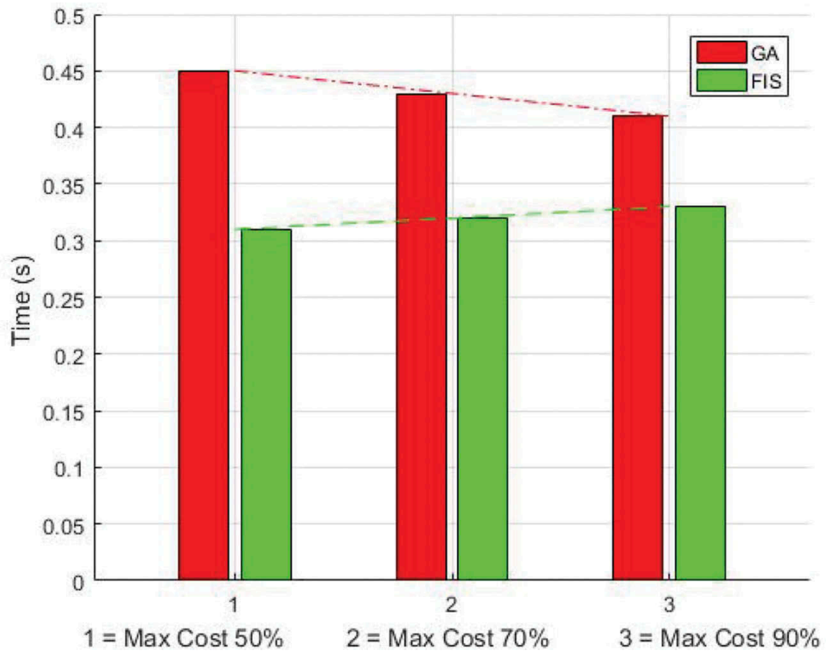

Figure 2. Comparison of runtimes in the presence of the cost constraints.

large projects such as the introduction of the next service pack of an operating system. In these tests, the proposed algorithm has had better choices than the genetic one. For example, within the 730-day time limit, the proposed algorithm with the *lom* defuzzifier could choose a subset of requirements which could yield a satisfaction rate equal to 2256 in 723 days at a 726-unit cost, while the genetic algorithm yields a lower satisfaction rate than the proposed one in 728 days on average at a cost of about 727 units. Another example of Table 9, within the 1-year time limit, the proposed algorithm with the *centroid* and *bisector* defuzzifier could choose a subset of requirements which could yield a satisfaction rate equal to 1389 in 354 days at a 343-unit cost. Whereas the genetic algorithm achieves a satisfaction rate 1252.8, in 356 days on average at a cost of about 353 units.

The runtimes in the presence of the four time constraints have been given in Figure 3. In all constraints, the proposed algorithm has shorter runtime. For example, for the time threshold of one year, the proposed algorithm is 37% faster than the GA.

Remark 1: It is worth noting that the output of the proposed method, similar to the existing methods of solving single objective problems, is a unique optimal solution. In fact, the development team has decided to get a unique optimal solution and not a set of non-dominated solutions according to the proposed approach. For those problems with a unique optimal solution, it is not possible to calculate the hypervolume and Spread (Quality) indicator.

Sensitivity analysis

In this subsection, we investigate the sensitivity analysis on the second dataset for verifying the proposed algorithm. In the case that a cost limit 50% is considered for this data set, the following requirement set is selected for development in the next release:

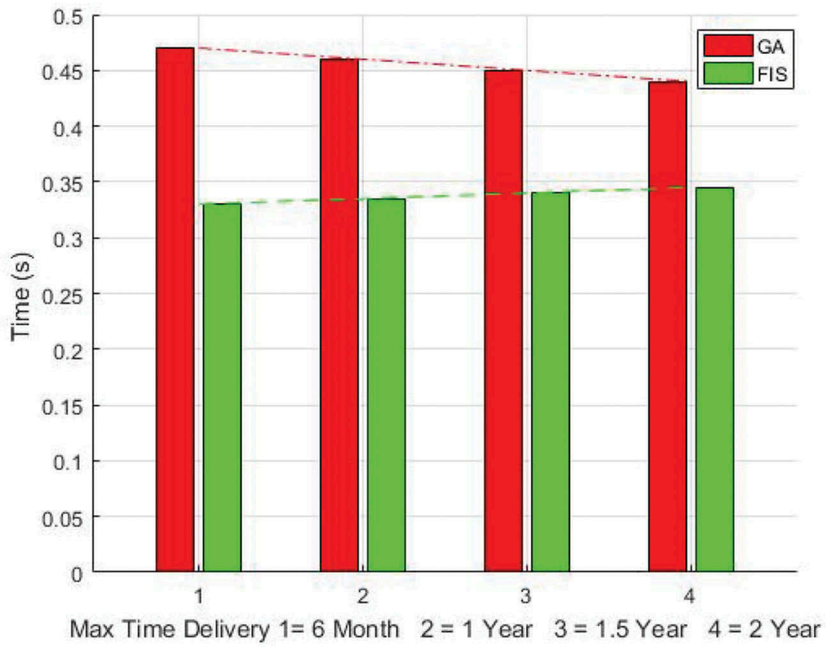

Figure 3. Comparison of runtimes in the presence of time constraints.

$$K_c = \{r_1, r_{11}, r_{13}, r_{14}, r_{16}, r_{17}, r_{18}, r_{20}, r_{21}, r_{23}, r_{24}, r_{25}, r_{27}, r_{29}, r_{30}, r_{31}, r_{32}, r_{35}, r_{36}, r_{37}, r_{38}, r_{40}, r_{41}, r_{43}, \\ r_{45}, r_{46}, r_{49}, r_{50}, r_{51}, r_{52}, r_{55}, r_{56}, r_{57}, r_{58}, r_{60}, r_{64}, r_{65}, r_{68}, r_{70}, r_{71}, r_{72}, r_{73}, r_{75}, r_{77}, r_{79}\}$$

The satisfaction amount of set k_c evaluated by LOM is 1544 units. The development cost of this set is 439 units, and the development time is 445 units. If the cost of the requirement r_1 is increased by 10%, then the requirements r_1 and r_{46} are deleted from the set k_c and the requirements r_9 and r_{12} are added. In this condition, the satisfaction of the new set is 1543 units, the cost of developing will be 441 units, and the development time will be 444 units. If the cost of the requirement r_1 decreases by 10%, then the requirement r_{46} of the set k_c is deleted and the requirement r_9 is added. In this case, the satisfaction of the new set is 1541 units, while the development cost will be 441.6 units, and the development time will be 445 units.

In the case that a time limit 0.5 year is considered for the second data set, the following requirement set is selected for development in the next release:

$$K_T = \{r_1, r_{11}, r_{13}, r_{14}, r_{16}, r_{17}, r_{21}, r_{29}, r_{31}, r_{32}, r_{35}, r_{37}, r_{40}, r_{50}, r_{51}, r_{52}, r_{55}, r_{56}, r_{64}, r_{68}, r_{70}, r_{71}, r_{73}, r_{75}\}$$

The satisfaction amount of K_T evaluated by LOM is 778 units; the development cost is 200 units and the development time is183 units. If development time of the requirement r_1 increases by 10%, then the requirement r_{71} will be deleted from K_T and the requirement r_{20} will be added to K_T. The satisfaction of the new set is 764 units; the development cost is 198 units and the development time is 182.2 units. If development time of the requirement r_1 decreases about 10%, the set K_T will not change, and only the development time of this set will decrease from 183 units to 182.2 units.

In the case that a cost limit of 90% is considered for the second data set, the requirement set Z_c is selected for development in the next release which includes the requirement r12. The Z_c set evaluated based on LOM will have a satisfaction of 2211, a cost of 700 units and a development time of 704 units. If the requirement cost r_{12} increases by 10%, the Z_c set will not change. But if the requirement cost r_{12} decreases by 10%, the requirement r_4 will be deleted from the Z_c set and the requirement r_{47} will be added. The satisfaction of the new set is 2230 units; the cost of development is 706.8 units, and the development time will be 707 units.

Conclusions

In this paper, the NRP problem was examined. Here, three objectives were considered, namely development time, development cost, and satisfaction. The presence of rivals in the market, the reduction of the development time of each release and the cost savings of the software company all impose a reduction of development costs on the company. However, increasing client satisfaction with the product is also an important objective for companies. Despite the three objectives, it is very difficult to select a subset of requirements for development in the next release. In this paper, we introduced an algorithm that uses the fuzzy inference system and greedy choices based on constraints to determine the suitability of each requirement for development in the next release. The purpose of implementing the proposed algorithm is to help in deciding on the selection of an optimal subset of requirements for development in the next release. Instead of the direct choices of the members of the developer team, the use of such algorithms reduces human selection errors. The quality of the proposed algorithm was evaluated on two datasets with different complexities. The first dataset was relatively small, while the second was large and complex. The results of the proposed algorithm were compared with those of the genetic algorithm. The results of the proposed algorithm, with all constraints considered, could provide much more satisfaction compared to the genetic algorithm. The results also showed that in the case of larger problems, the proposed algorithm has a better result in contrast to the genetic algorithm. Moreover, scalability is one of the advantages of the proposed algorithm. However, in the case that the development team allocates wrong cost and time values to the requirement, the total cost of the software or its delivery time will not result according to the predicated programm. Also, allocation of crisp numbers to satisfaction of requirements can be one of the limitations of the proposed method. In fact, during the production of the software, clients' satisfaction from the requirements may be changed, so the allocation of fuzzy numbers instead of crisp numbers can alleviate this problem. In the future, we will attempt to modify the proposed algorithm to overcome these limitations.

Acknowledgments

The authors would like to thank the anonymous reviewers and the associate editor for their insightful comments and suggestions.

Disclosure statement

No potential conflict of interest was reported by the authors.

ORCID

Ali Ebrahimnejad http://orcid.org/0000-0001-6003-6601

References

Alrashoud, M., & Abhari, A. (2015). Perception-based software release planning. *Intelligent Automation & Soft Computing*, *21*(2), 175–195.

Alrashoud, M., & Abhari, A. (2017). Planning for the next software release using adaptive network-based fuzzy inference system. *Intelligent Decision Technologies*, *11*(2), 153–165.

Alrezaamiri, H., Ebrahimnejad, A., & Motameni, H. (2019). Software requirement optimization using a fuzzy artificial chemical reaction optimization algorithm. *Soft Computing*, *23*(20), 9979–9994.

Araújo, A. A., Paixao, M., Yeltsin, I., Dantas, A., & Souza, J. (2017). An architecture based on interactive optimization and machine learning applied to the next release problem. *Automated Software Engineering*, *24*(3), 623–671.

Bagnall, A. J., Rayward-Smith, V. J., & Whittley, I. M. (2001). The next release problem. *Information and Software Technology*, *43*(14), 883–890.

Chaves-González, J. M., & Pérez-Toledano, M. A. (2015). Differential evolution with Pareto tournament for the multi-objective next release problem. *Applied Mathematics and Computation*, *252*, 1–13.

Chaves-González, J. M., Perez-Toledano, M. A., & Navasa, A. (2015a). Teaching learning based optimization with Pareto tournament for the multiobjective software requirements selection. *Engineering Applications of Artificial Intelligence*, *43*, 89–101.

Chaves-González, J. M., Perez-Toledano, M. A., & Navasa, A. (2015b). Software requirement optimization using a multiobjective swarm intelligence evolutionary algorithm. *Knowledge-Based Systems*, *83*, 105–115.

Chopra, R. K., Gupta, V., & Chauhan, D. S. (2016). Experimentation on accuracy of non-functional requirement prioritization approaches for different complexity projects. *Perspectives in Science*, *8*, 79–82.

Del Sagrado, J., Del Aguila, I. M., & Orellana, F. J. (2015). Multi-objective ant colony optimization for requirements selection. *Empirical Software Engineering*, *20*(3), 577–610.

Ebrahimnejad, A., Karimnejad, Z., & Alrezaamiri, H. (2015). Particle swarm optimisation algorithm for solving shortest path problems with mixed fuzzy arc weights. *International Journal of Applied Decision Sciences*, *8*(2), 203–222.

Ebrahimnejad, A., Tavana, M., & Alrezaamiri, H. (2016). A novel artificial bee colony algorithm for shortest path problems with fuzzy arc weights. *Measurement*, *93*, 48–56.

Ferreira, D. N., Araújo, T., Neto, A. A. A. D. B., & de Souza, J. T. (2016). Incorporating user preferences in ant colony optimization for the next release problem. *Applied Soft Computing*, *49*, 1283–1296.

Greer, D., & Ruhe, G. (2004). Software release planning: An evolutionary and iterative approach. *Information and Software Technology*, *46*(4), 243–253.

Hsieh, M. Y., Hsu, Y. C., & Lin, C. T. (2018). Risk assessment in new software development projects at the front end: A fuzzy logic approach. *Journal of Ambient Intelligence and Humanized Computing*, *9*(2), 295–305.

Hudaib, A., Masadeh, R., Qasem, M. H., & Alzaqebah, A. (2018). Requirements Prioritization Techniques Comparison. *Modern Applied Science*, *12*(2), 62.

Jiang, H., Zhang, J., Xuan, J., Ren, Z., & Hu, Y. (2010, June). A hybrid ACO algorithm for the next release problem. In *Software Engineering and Data Mining (SEDM), 2010 2nd International Conference on* (pp. 166–171). IEEE, Chengdu, China. 171.

Karlsson, J. (1996, April). Software requirements prioritizing. In *Requirements Engineering, 1996., Proceedings of the Second International Conference on* (pp. 110–116). IEEE, Colorado Springs, CO, USA.

Masadeh, R., Alzaqebah, A., Hudaib, A., & Rahman, A. A. (2018). Grey Wolf algorithm for requirements prioritization. *Modern Applied Science*, *12*(2), 54.

Mougouei, D. (2016, August). Factoring requirement dependencies in software requirement selection using graphs and integer programming. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering* (pp. 884–887), Singapore, Singapore. ACM.

Mougouei, D., & Powers, D. M. (2017). Modeling and selection of interdependent software requirements using fuzzy graphs. *International Journal of Fuzzy Systems*, *19*(6), 1812–1828.

Odzaly, E. E., Greer, D., & Stewart, D. (2017). Agile risk management using software agents. *Journal of Ambient Intelligence and Humanized Computing*, *9*, 823–841.

Paixao, M., & Souza, J. (2015). A robust optimization approach to the next release problem in the presence of uncertainties. *Journal of Systems and Software*, *103*, 281–295.

Paredes-Valverde, M. A., Del Pilar Salas-Zárate, M., Colomo-Palacios, R., Gómez-Berbís, J. M., & Valencia-García, R. (2018). An ontology-based approach with which to assign human resources to Software projects. *Science of Computer Programming*, *156*, 90–103.

Pitangueira, A. M., Maciel, R. S. P., & Barros, M. (2015). Software requirements selection and prioritization using SBSE approaches: A systematic review and mapping of the literature. *Journal of Systems and Software*, *103*, 267–280.

Pourjavad, E., & Shahin, A. (2018). The application of Mamdani fuzzy inference system in evaluating green supply chain management performance. *International Journal of Fuzzy Systems*, *20*(3), 901–912.

Ralph, P. (2018). The two paradigms of software development research. *Science of Computer Programming*, *156*, 68–89.

Ramzan, M., Jaffar, M. A., & Shahid, A. A. (2011). Value based intelligent requirement prioritization (VIRP): Expert driven fuzzy logic based prioritization technique. *International Journal of Innovative Computing, Information and Control*, *7* (3), 1017-1038.

Sadiq, M., & Jain, S. K. (2014). Applying fuzzy preference relation for requirements prioritization in goal oriented requirements elicitation process. *International Journal of System Assurance Engineering and Management*, *5*(4), 711–723.

Shirmohammadi, H., & Hadadi, F. (2017). Assessment of Drowsy drivers by fuzzy logic approach based on multinomial logistic regression analysis. *IJCSNS*, *17*(4), 298.

Shirmohammadi, H., & Hadadi, F. (2019). Optimizing total delay and average queue length based on the fuzzy logic controller in urban intersections. *International Journal of Supply and Operations Management*, *6*(2), 142–158.
Veerapen, N., Ochoa, G., Harman, M., & Burke, E. K. (2015). An integer linear programming approach to the single and bi-objective next release problem. *Information and Software Technology*, *65*, 1–13.
Xiang, X., Yu, C., Lapierre, L., Zhang, J., & Zhang, Q. (2018). Survey on fuzzy-logic-based guidance and control of marine surface vehicles and underwater vehicles. *International Journal of Fuzzy Systems*, *20*(2), 572–586.
Xuan, J., Jiang, H., Ren, Z., & Luo, Z. (2012). Solving the large scale next release problem with a backbone-based multilevel algorithm. *IEEE Transactions on Software Engineering*, *38*(5), 1195–1212.