GRASP-based hybrid search to solve the multi-objective requirements selection problem*

Víctor Pérez-Piqueras[0000−0002−2305−5755], Pablo Bermejo López[0000−0001−7595−910X], and José A. Gámez[0000−0003−1188−1117]

Department of Computing Systems, Intelligent Systems and Data Mining Laboratory (I3A), Universidad de Castilla-La Mancha, Albacete, 02071, Spain
{victor.perezpiqueras,pablo.bermejo,jose.gamez}@uclm.es

Abstract. One of the most important and recurring issues that the development of a software product faces is the requirements selection problem. Addressing this issue is especially crucial if agile methodologies are used. The requirements selection problem, also called Next Release Problem (NRP), seeks to choose a subset of requirements which will be implemented in the next increment of the product. They must maximize clients satisfaction and minimize the cost or effort of implementation. This is a combinatorial optimization problem studied in the area of Search-Based Software Engineering. In this work, the performance of a basic genetic algorithm and a widely used multi-objective genetic algorithm (NSGA-II) have been compared against a multi-objective version of a randomized greedy algorithm (GRASP). The results obtained show that, while NSGA-II is frequently used to solve this problem, faster algorithms, such as GRASP, can return solutions of similar or even better quality using the proper configurations and search techniques. The repository with the code and analysis used in this study is made available to those interested via GitHub.

Keywords: grasp · multi-objective optimization · next release problem · requirements selection · search-based software engineering.

1 Introduction

Software systems are increasing in functionality and complexity over time. This implies that new software projects are potentially more complicated to manage and complete successfully. One of the problematics that can heavily affect the outcome of a project is the planning of a release. In a software project, the product to be delivered is defined by a set of software requirements. These requirements are offered to a group of clients, who will give feedback on which requirements are more important to them. Then, a set of requirements is planned for the release. Selecting the requirements that better fit client interests starts getting complicated when development capacity has to be taken into account.

* This work has been partially funded by the Regional Government (JCCM) and ERDF funds through the project SBPLY/17/180501/000493.

Furthermore, requirements can have dependencies between them. This problem, named requirements selection problem, is very complex and does not have a unique and optimal solution. Two objectives coexist: maximizing the satisfaction of the clients and minimizing the effort of the software developers. Therefore, solutions can range from sets of few requirements with minimal effort and satisfaction, to sets of plenty of requirements, which will imply high client satisfaction but at the cost of a high effort.

Thus, this planning step is critical, especially when applying incremental software development methodologies due to the need to solve the requirements selection problem multiple times, at each iteration. Thus, this problem is candidate to be automated by means of optimization methods. Previous works have studied the applicability of different search techniques, giving preference to evolutionary algorithms, mainly. In our study, we present an algorithm based on the Greedy Randomized Adaptive Search Procedure (GRASP, [7]). We have explored new procedures that allow to improve GRASP performance in the requirements selection problem beyond that of previous studies. The experimentation that we carried out shows that the GRASP metaheuristic can obtain similar results as those of the evolutionary approaches, but reducing drastically its computational cost.

The rest of the paper is structured as follows. In Section 2, a summary of previous works and procedures they applied is made. Section 3 describes our algorithm proposal and defines the solution encoding along with the most important methods and techniques. Then, in Section 4, the evaluation setup is described, along with the algorithms, datasets and methodology used. Section 5 presents and discusses the results of the experimentation. Finally, Section 6 summarizes the conclusions of this study and introduces potential new lines of work for the future.

2 Requirements selection

2.1 Related work

The requirements selection problem is studied in the Search-Based Software Engineering (SBSE) research field, where Software Engineering related problems are tackled by means of search-based optimization algorithms. The first definition of the requirements selection problem was formulated by Bagnall et al. [1]. In their definition of the Next Release Problem (NRP), a subset of requirements has to be selected, having as goal meeting the clients[1] needs, minimizing development effort and maximizing clients satisfaction. In their work, different metaheuristics algorithms, such as simulated annealing, hill climbing and GRASP algorithms were proposed, but all of them combined the objectives of the problem using an aggregate function. The same procedure of single-objective proposals was followed by Greer and Ruhe [9]. They studied the generation of

[1] Although "stakeholder" is a more appropriate term, "client" will be used to keep coherence with previous works present in the literature.

feasible assignments of requirements to increments, taking into account different resources constraints and stakeholders perspectives. Genetic algorithms (GAs) were the optimization technique selected to solve the NRP. Later, Baker et al. [2] demonstrated that metaheuristics techniques could be applied to real-world NRP outperforming expert judgement, using in their study simmulated annealing and greedy algorithms. The works of del Sagrado et al. [5] applied ACO (Ant Colony Optimization). All of these approaches followed a single-objective formulation of the problem, in which the aggregation of the objectives resulted in a biased search.

It was not until the proposal of Zhang et al. [13] that the NRP was formulated as a multi-objective optimization (MOO) problem. This new formulation, Multi-Objective Next Release Problem (MONRP), formally defined in Section 2.2, was based on Pareto dominance. Their proposal tackled each objective separately, exploring the non-dominated solutions. Finkelstein et al. [8] also applied multi-objective optimization considering different measures of fairness. All these studies applied evolutionary algorithms, such as ParetoGA and NSGA-II [4] to solve the MONRP.

Other works that kept exploring evolutionary algorithms to solve the MONRP are those of Durillo et al. [6]. They proposed two GAs, NSGA-II and MOCell (MultiObjective Cellular genetic algorithm), and an evolutionary procedure, PAES (Pareto Archived Evolution Strategy).

2.2 Multi-objective formulation

As mentioned in the introduction, the NRP requires a combinatorial optimization of two objectives. While some studies alleviate this problem by adding an aggregate (single-objective optimization), others tackle the two objectives by using a Pareto front of non-dominated solutions (MOO). Defining the NRP as a multi-objective optimization problem gives the advantage that a single solution to the problem is not sought, but rather a set of non-dominated solutions. In this way, one solution or another from this set can be chosen according to the conditions, situation and restrictions of the software product development. This new formulation of the problem is known as MONRP.

The MONRP can be defined by a set $R = \{r_1, r_2, \ldots, r_n\}$ of n candidate software requirements, which are suggested by a set $C = \{c_1, c_2, \ldots, c_m\}$ of m clients. In addition, a vector of costs or efforts is defined for the requirements in R, denoted $E = \{e_1, e_2, \ldots, e_n\}$, in which each e_i is associated with a requirement r_i . Each client has an associated weight, which measures its importance. Let $W = \{w_1, w_2, \ldots, w_m\}$ be the set of client weights. Moreover, each client gives an importance value to each requirement, depending on the needs and goals that this has with respect to the software product being developed. Thus, the importance that a requirement r_j has for a client c_i is given by a value v_{ij}, in which a zero value represents that the client c_i does not have any interest in the implementation of the requirement r_j. A $m \times n$ matrix is used to hold all the importance values in v_{ij}. The overall satisfaction provided by a requirement r_j is denoted as $S = \{s_1, s_2, \ldots, s_n\}$ and is measured as a weighted sum of

all importance values for all clients. The MONRP consists of finding a decision vector X, that includes the requirements to be implemented for the next software release. X is a subset of R, which contains the requirements that maximize clients satisfaction and minimize development efforts.

3 Proposal

Evolutionary algorithms have been widely applied to solve the MONRP [3,11,13]. Most of the previous studies are based on algorithms such as NSGA-II, ParetoGA or ACO, usually comparing their performance with other algorithms less suited to the problem, e.g. generic versions of genetic or greedy algorithms. However, evolutionary approaches involve a high computational cost, as the algorithms have to generate a population of solutions and evolve each one of the solutions applying many operators and fitness evaluations. For this reason, in this work we have pursued to design an algorithm that can return a set of solutions of similar quality to those obtained by the evolutionary proposals, but reducing the cost of its computation. In this section, it is presented the multi-objective version of a greedy algorithm, along with the solution encoding used. Then, the most relevant procedures of the algorithm and enhancements are presented.

3.1 GPPR: a GRASP algorithm with Pareto front and Path Relinking

GRASP is a multi-start method designed to solve hard combinatorial optimization problems, such as the MONRP. It has been used in its simplest version [11,3] to solve the requirements selection problem.

The basic actions of a canonical GRASP procedure consist of generating solutions iteratively in two phases: a greedy randomized construction and an improvement by local search (see Sections 3.3 and 3.4). These two phases have to be implemented specifically depending on the problem at hand.

We have designed a variant of the GRASP procedure with the goal of solving the MONRP in a hybrid manner, that is, applying both single-objective and multi-objective search methods. The algorithm, named GRASP algorithm with Pareto front and Path Relinking (GPPR), executes a fixed number of iterations, generating at each iteration a set of solutions, instead of only one solution per iteration (which is a different approach from the canonical GRASP that generates one solution per iteration, but in the end works identically). Additionally, we have extended the procedure, updating the Pareto front with the new solutions found after each iteration, and adding a post-improvement procedure known as Path Relinking (see Section 3.5), that will enhance the quality of the Pareto front found by exploring trajectories that lead to new non-dominated solutions. The pseudocode of GPPR is shown in Algorithm 1.

Each one of the operators included in the pseudocode is described in detail in Sections 3.3, 3.4 and 3.5, respectively. As explained previously, GPPR is an algorithm that applies a hybrid approach. It maintains and updates at each

Algorithm 1 GPPR pseudocode

```
procedure GPPR(maxIterations)
    nds ← ∅                                   ▷ empty set of non-dominated solutions
    for i = 0 to maxIterations do
        solutions ← constructSolutions()
        solutions ← localSearch(solutions)
        solutions ← pathRelinking(solutions, nds)
        nds ← updateNDS(solutions)
    end for
    return nds
end procedure
```

iteration a set of non-dominated solutions, returned in the form of a Pareto front at the end of the search. However, it uses an aggregate of the two problem objectives in some phases of the execution (depending on the methods chosen for each phase).

3.2 Solution encoding

Each candidate solution in GPPR is represented by a vector of booleans of length n. Each value of the vector indicates the inclusion or not of a requirement of the set R (see Section 2.2). The satisfaction and effort of each requirement are scaled using a min-max normalization. Each solution is evaluated by means of a *singleScore* value that mixes the scaled satisfaction and effort of the set X of selected requirements in the solution. In this version of the GPPR, we did not model cost restrictions nor interactions between requirements.

3.3 Construction

In this phase a number of solutions are constructed. Their generation can be either randomized or stochastic. We have designed two methods for the construction phase:

- **Uniform.** First, the number x of selected requirements is randomly chosen. Then, x requirements are selected randomly, having each requirement r of the set R of length n a probability $\frac{1}{n}$ of being selected. This construction method works as a random selection of requirements.
- **Stochastic.** The probability of each requirement being selected is proportional to its *singleScore*.

3.4 Local search

This phase is executed after the construction of an initial set of solutions, and it aims to find solutions in the neighbourhood that enhance the former ones. Since GPPR aims to generate solutions fast, it performs a ranking-based forward

search, in which it tries to find and return a neighbour that is better than the initial one. This search method tends to fall into local optima, but it can be corrected increasing the number of executions or applying extra operators after this phase (see Subsection 3.5).

3.5 Path Relinking

One of the adverse characteristics of GRASP is its lack of memory structures. Iterations in GRASP are independent and do not use previous observations. Path Relinking (PR) is a possible solution to address this issue. PR was originally proposed as a way to explore trajectories between elite solutions. In the problem being tackled, elite solutions are the non-dominated solutions. Using one or more elite solutions, trajectories that lead to other elite solutions in the search space are explored, in order to find better solutions. PR applied to GRASP was introduced by Laguna and Martí [10]. It has been used as an intensification scheme, in which the generated solutions of an iteration are relinked to one or more elite solutions, creating a post-optimization phase.

The PR method can be applied either after each iteration, involving a higher computational cost; or at the end of the execution, relinking only the final elite solutions, reducing the effectiveness of this method but speeding up the execution.

In this proposal we have decided to apply PR at each iteration as a third phase, after the local search. The pseudocode is described in Algorithm 2. For each one of the solutions found after the local search, this procedure will try to find a path from each solution to a random elite solution from the set of non-dominated solutions (NDS). This path will help the procedure to find intermediate solutions that can possibly be better than the former ones. For this purpose, each solution in the current set of solutions obtained after the local search is assigned an elite solution from the current NDS. Then, it calculates the Hamming distance of these two solutions. Having the distance value, the procedure finds the bits that are different, that is, the requirements included in one solution that are not in the other. Then, it updates the current solution (flips the bit that returns the highest *singleScore* value) and saves the new path solution in a solution path list, decrementing the distance from the current solution to the elite one. When the distance is zero, the best solution found in the path is appended to a set of best solutions (*bestSols* in Algorithm 2) found by the PR procedure. After finding best path solutions for all the initial solutions, the procedure returns the set of former solutions plus the new solutions found.

4 Evaluation setup

In this section, we present the experimental evaluation. We describe competing approaches used to be compared against our proposal, along with the datasets used to evaluate the algorithms. Our algorithms have been implemented in

Algorithm 2 Path Relinking pseudocode

```
procedure PathRelinking(solutions, nds)
    bestSols ← ∅
    for sol in solutions do
        currSol ← sol                          ▷ Create a copy to be modified
        eliteSol ← getRandomSol(nds)
        distance ← countDistance(currSol, eliteSol)
        pathSols ← ∅
        while distance > 0 do
            diffBits ← findDiffBits(currSol, eliteSol)
            currSol ← flipBestBitSingleScore(diffBits)
            pathSols ← savePath(currSol)
            distance ← distance − 1
        end while
        bestSols ← bestSols ∪ findBestSol(pathSols)
    end for
    return solutions ∪ bestSols
end procedure
```

Python 3.8.8. The source code, experimentation setup and datasets are available at the following repository: https://github.com/UCLM-SIMD/MONRP/tree/ola22.

4.1 Algorithms

To properly compare the effectivity and performance of our proposal, besides GPPR we have included in our experiments the following algorithms: Random search, Single-Objective GA and NSGA-II. The ranges of parameters used in the experimentation for each algorithm are described in Section 4.3, along with their descriptions.

4.2 Datasets

We have tested the performance of the algorithms using a variety of datasets from different sources. Datasets P1 [9] and P2 [11] include 5 clients and 20 requirements, and 5 clients and 100 requirements, respectively.

Due to the privacy policies followed by software development companies, there is a lack of datasets to experiment with. For this reason, we have created synthetically a larger dataset (S3) that includes 100 clients and 140 requirements, in order to evaluate the shift in performance of the algorithms.

4.3 Methodology

We tested a set of configurations for each algorithm and dataset. Each configuration was executed 10 times. For the Single-Objective GA and NSGA-II,

populations were given values among $\{20, 30, 40, 100, 200\}$ and number of generations took values $\{100, 200, 300, 500, 1000, 2000\}$. Crossover probabilities range from $\{0.6, 0.8, 0.85, 0.9\}$. Two mutation schemes were used, *flip1bit* and *flipeachbit*, and mutation probabilites from $\{0, 0.05, 0.1, 0.2, 0.5, 0.7, 1\}$. Both algorithms used a binary tournament selection and a one-point crossover scheme. For the replacement scheme, both Single-Objective GA and NSGA-II applied elitism. The total amount of different hyperparameter configurations executed for each GA and each dataset was 1680. Our GPPR algorithm was tested using a number of iterations from $\{20, 40, 60, 80, 100, 200, 500\}$ and a number of solutions per iteration from $\{20, 50, 100, 200, 300, 500\}$. We tested all combinations of construction methods, local search and PR (including configurations with no local search and no PR methods), which resulted in 1008 different hyperparameter configurations executed for each dataset.

The stop criterion used in other works [13,11,3] is the number of function evaluations, commonly set to 10000. To adapt our experiments to this stop criterion, we restricted the execution of our GAs to: $Pop.\ size \times \#Gens. \leq 10000$; and for the GPPR: $Iterations \times Sols.\ per\ Iteration \leq 10000$.

The GPPR normalizes the satisfaction and effort values, scaling them between 0 and 1. To properly compare its Pareto front solutions against those returned by the GAs, these evolutionary approaches have also used the normalized version of the dataset values. To evaluate the results, we compared the obtained Pareto fronts and a set of quality indicators of the results generated by the algorithms and their efficiency:

- **Hypervolume (HV)**. Denotes the space covered by the set of non-dominated solutions [14]. Pareto fronts with higher HV are preferred.
- **Δ-Spread**. It measures the extent of spread achieved among the obtained solutions [6]. Pareto fronts with lower Δ-Spread are preferred.
- **Spacing**. It measures the uniformity of the distribution of non-dominated solutions [12]. Pareto fronts with greater spacing are preferred.
- **Execution time**. The total time taken by the algorithm to finish its execution. Algorithms with lower execution time are preferred.

Mean values of these metrics have been calculated and compared in a pairwised manner between algorithms using the Wilcoxon rank-sum non-parametric test, which allows to assess whether one of two samples of independent observations tends to have larger values than the other.

5 Results and analysis

5.1 Best configurations

The Single-Objective GA's best hyperparameter configuration includes a population size of 100 individuals, a number of generations of 100 (maximum number to stay under the 10,000 limit) and a $P_c = 0.8$. The mutation operator that showed a better performance was the *flip1bit*. This operator gives a chance of

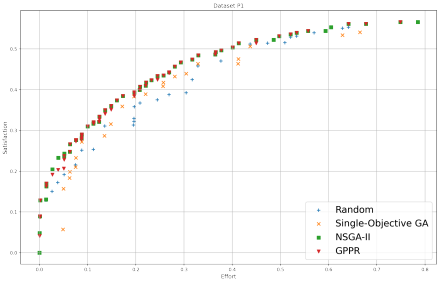

Fig. 1. Pareto front for dataset P1

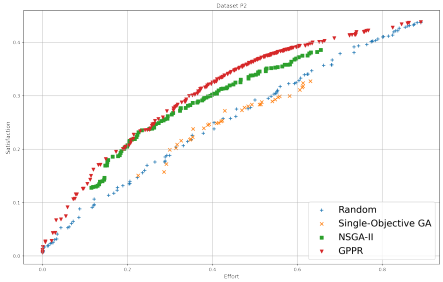

Fig. 2. Pareto front for dataset P2

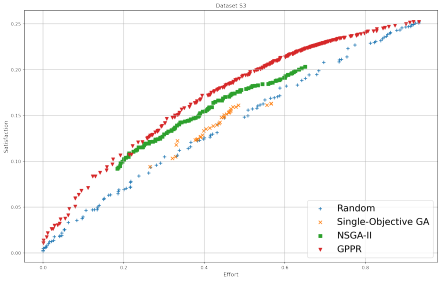

Fig. 3. Pareto front for dataset S3

flipping only one bit of the booleans vector. The best-performing probability is $P_m = 1$, which means that we always mutates one random bit of each individual. That probability is equivalent to using $P_m = \frac{1}{n}$ at gene level, n being the number of genes (scheme used in [6,11]). The best hyperparameter configuration for the NSGA-II used a population size of 100 individuals and 100 generations. The best crossover probability (P_c) was the lowest, 0.6, and the best mutation operator was the *flip1bit*, using a $P_m = 1$. For the GPPR, the ratio between iterations and number of solutions per iteration is less important, as this algorithm does not have memory. Thus, a similar hyperparameter configuration to those of the GAs was used. The construction method that showed a better performance was the *stochastic* one, giving preference to requirements with higher *singleScore*. In all scenarios, hyperparameter configurations with *uniform* construction performed worse.

5.2 Pareto results

Pareto results are shown in Figures 1, 2 and 3. For the sake of space, we omit results of worst-performing hyperparameter configurations.

The Single-Objective GA shows bad performance, being similar to that of the random procedure. This occurs due to the low number of generations set to keep the maximum number of function evaluations. Configuring a number of generations of one or two magnitude orders higher increases the quality of its

Pareto front. Regarding the Pareto front distribution, this GA's aggregation of objectives biases the search, leaving unexplored areas.

The NSGA-II algorithm generates Pareto fronts of better quality: better solutions and more distributed along the search space. As expected, the crowding operator of the algorithm helps exploring the search space. However, as the dataset size increases, its performance decreases significantly. The reason is the limited number of generations, as this algorithm is expected to perform better in larger datasets when compared against other search methods.

Multi-Objective (MO) local search methods do not worsen the solutions, but do not improve them either, as a GPPR without local search is returning similar Pareto fronts. This implies that local search methods that can only explore the neighbourhood of a solution are not able to improve the solutions. Nevertheless, the PR procedure is capable of finding better solutions. In all cases, algorithms applying this methods returned Pareto fronts of higher quality. Regarding the Pareto distribution, as this procedure starts each iteration randomly, it explores the majority of the search space. The most interesting feature of the GPPR is that, while the dataset size grows, its performance is not demeaned. Therefore, unless the search space is much larger, our GPPR proposal can return a Pareto front of acceptable quality very efficiently, while GAs require higher number of iterations, impplying a less affordable computational cost.

5.3 Metrics results

The mean values of the metrics obtained for each algorithm and dataset after 10 independent runs have been statistically compared, as explained in Section 4.3. Each metric mean value has been compared pair-wise between algorithms, denoting the best value in **bold** and indicating the values that are statistically worse ($P < 0.05$) with a $\downarrow$ symbol (see Table 1).

Regarding the HV metric, our GPPR algorithm has obtained significantly better results than the two GAs in datasets P2 and S3, and worse results for dataset P1, whose search space is very small. These results only denote that the extreme solutions of the Pareto front returned by GPPR cover a larger area than those obtained by the GAs.

The Δ-Spread values show that the Single-Objective GA obtains the lowest values, that is, the best Δ-Spread values. Nevertheless, our GPPR proposal obtains values lower than those of the NSGA-II, outperforming it once again.

The spacing values show that, again, the GPPR outperforms the two GAs in the two larger datasets.Comparing the spacing values of the smallest dataset, P1, it is observed that GPPR spacing values decrease, being close to those obtained by the NSGA-II algorithm. However, in datasets P2 and S3, the GPPR spacing values are significantly greater.

Finally, for the execution time, it is important to consider that comparison between our experiments and those made by other studies is only possible if the same software and hardware requirements are met. Otherwise, only the difference in execution time between algorithms of the same study can be analyzed.

The fastest algorithm is the Single-Objective GA, due to the lightweight methods and few generations that it had executed. The NSGA-II obtained the worst values, because of the additional steps executed at each iteration, and despite implementing a fast-sorting method. When compared against our GPPR proposal, the difference is significant, being the NSGA-II almost ten times slower for the smallest dataset (P1), and fairly slower for larger datasets. It is interesting to highlight that, as dataset size grows, the difference between the NSGA-II and our GPPR proposal decreases. However, MONRP instances are not expected to have a scale large enough that the NSGA-II could outperform our GPPR. Therefore, these values demonstrate that our proposal can be applied satisfactorily to MONRP reducing drastically computational cost.

Table 1. Average metrics of the best configurations for each dataset

Dataset	Algorithm	HV	Δ-Spread	Spacing	Exec. time (s)
P1	Single-Objective GA	0.594↓	**0.615**	0.323↓	**17.967**
	NSGA-II	**1.0**	0.963↓	**0.382**	180.991↓
	GPPR	0.909↓	0.644	0.371↓	18.102
P2	Single-Objective GA	0.157↓	**0.637**	0.128↓	**82.713**
	NSGA-II	0.407↓	0.969↓	0.245↓	616.415↓
	GPPR	**0.973**	0.688↓	**0.300**	250.841↓
S3	Single-Objective GA	0.102↓	0.720	0.105↓	**125.702**
	NSGA-II	0.286↓	0.970↓	0.206↓	859.928↓
	GPPR	**0.977**	**0.711**	**0.293**	488.045↓

6 Conclusions and future work

In this paper, we have studied the applicability of a greedy procedure (GPPR) into a multi-objective problem of the Software Engineering field. The MONRP has been tackled previously using, mainly, evolutionary approaches. Few proposals have used GRASP-based methods, usually applying basic instances of it. Our proposal aimed to design a method capable of generating solutions of similar quality than those of the evolutive approaches, but reducing drastically the computational cost. We have explored different combinations of construction and local search methods, and applied post-construction techniques, such as PR, to improve the solutions found. To evaluate our proposal and compare it against classic methods, we have designed an experimentation framework, in which we have used two real-world datasets and created a new one synthetically, setting a rigorous experiment. The comparison have been carried out using a set of quality metrics and comparing the Pareto fronts, obtaining quite good results and showing that our GPPR proposal can outperform more classical and popular methods, in both performance and Pareto front results. Moreover, the code of the algorithms and experiments has been published to be shared by the scientific community.

In future lines of work, we will explore other approaches to the MONRP. It would also be interesting to try combining our GPPR with a post-optimization phase using an evolutive algorithm capable of enhance former solutions. Additionally, it could be interesting to implement interactions between requirements, which is of interest when projects use long-term planning.

References

1. Bagnall, A.J., Rayward-Smith, V.J., Whittley, I.M.: The next release problem. Information and Software Technology **43**(14), 883–890 (2001)
2. Baker, P., Harman, M., Steinhöfel, K., Skaliotis, A.: Search based approaches to component selection and prioritization for the next release problem. In: 2006 22nd IEEE International Conference on Software Maintenance. pp. 176–185 (2006)
3. Chaves-González, J.M., Pérez-Toledano, M.A., Navasa, A.: Software requirement optimization using a multiobjective swarm intelligence evolutionary algorithm. Knowledge-Based Systems **83**(1), 105–115 (2015)
4. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Transactions on Evolutionary Computation **6**(2), 182–197 (2002)
5. del Sagrado, J., del Águila, I.M., Orellana, F.J.: Ant colony optimization for the next release problem: a comparative study. Proceedings - 2nd International Symposium on Search Based Software Engineering, SSBSE 2010 pp. 67–76 (2010)
6. Durillo, J.J., Zhang, Y., Alba, E., Nebro, A.J.: A study of the multi-objective next release problem. Proceedings - 1st International Symposium on Search Based Software Engineering, SSBSE 2009 (2009)
7. Feo, T., Resende, M.: Greedy Randomized Adaptive Search Procedures. Journal of Global Optimization **6**, 109–133 (1995)
8. Finkelstein, A., Harman, M., Mansouri, S.A., Ren, J., Zhang, Y.: A search based approach to fairness analysis in requirement assignments to aid negotiation, mediation and decision making. Requirements Engineering **14**(4), 231–245 (2009)
9. Greer, D., Ruhe, G.: Software release planning: An evolutionary and iterative approach. Information and Software Technology **46**, 243–253 (2004)
10. Laguna, M., Marti, R.: GRASP and Path Relinking for 2-Layer Straight Line Crossing Minimization. INFORMS Journal on Computing **11**(1), 44–52 (1999)
11. del Sagrado, J., del Águila, I., Orellana, F.J.: Multi-objective ant colony optimization for requirements selection. Empirical Software Engineering **20**, 577–610 (2015)
12. Schott, J.: Fault tolerant design using single and multicriteria genetic algorithm optimization. Ph.D. thesis, Massachusetts Institute of Technology, M.S., USA (1995)
13. Zhang, Y., Harman, M., Mansouri, A.: The multi-objective next release problem. In: GECCO 2007: Genetic and Evolutionary Computation Conference. pp. 1129–1137 (2007)
14. Zitzler, E., Thiele, L.: Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach. IEEE Transactions on Evolutionary Computation **3**(4), 257–271 (1999)