# Estimation of Distribution Algorithms Applied to the Next Release Problem

Víctor Pérez-Piqueras(✉), Pablo Bermejo López, and José A. Gámez

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

**Abstract.** The Next Release Problem (NRP) is a combinatorial optimization problem that aims to find a subset of software requirements to be delivered in the next software release, which maximize the satisfaction of a list of clients and minimize the effort required by developers to implement them. Previous studies have applied various metaheuristics and procedures, being many of them evolutionary algorithms. However, no Estimation of Distribution Algorithms (EDA) have been applied to the NRP. This subfamily of evolutionary algorithms, based on probability modelling, have been proved to obtain good results in problems where genetic algorithms struggle. In this paper we adapted two EDAs to tackle the multi-objective NRP, and compared them against widely used genetic algorithms. Results showed that EDA approaches have the potential to generate solutions of similar or even better quality than those of genetic algorithms in the most crowded areas of the Pareto front, while keeping a shorter execution time.

## 1 Introduction

Successfully managing software releases is one of the major challenges in Software Engineering. As the product goal grows and project scope gets expanded, the difficulty of delivering to clients what is needed increases substantially. Clients interests are usually defined in terms of software requirements, and these software requirements are built based on clients interests. Thus, when there is more than one client, their concerns are usually different or even opposed. Furthermore, complexity of requirements has to be taken into account, in order to not surpass development capacity at each iteration of the development cycle. Finally, the problem can get even more complex if the possible dependencies between requirements are taken into account.

This problem, named NRP, pursues finding a set of requirements for a release that satisfy clients as much as possible and optimize development efforts. This is a NP-hard problem which does not have a unique and optimal solution, and is usually solved manually by experts judgement. Given that solving the NRP is critical for a software project success, and that it has to be solved iteratively every time a release is planned, it is an interesting candidate to be automated


P. García Bringas et al. (Eds.): SOCO 2022, LNNS 531, pp. 98–108, 2023.
https://doi.org/10.1007/978-3-031-18050-7_10

by means of optimization methods. In previous works, different search techniques have been proposed to tackle the NRP. Most of them are based on metaheuristic techniques, being evolutionary algorithms the ones that showed best performance. In this paper, we introduce the possibility of applying Estimation of Distribution Algorithms (EDA) to the NRP. As a first approach, we have applied two univariate EDAs, and compared them against widely used genetic algorithms (GA). Experimentation results show that EDAs can be successfully applied to the NRP and, while evolved populations are not covering all search space areas, they can overcome GAs solutions in specific sections of the space.

The rest of the paper is structured as follows. In Sect. 2, a summary of previous works and applied procedures is made. Section 3 introduces EDA and describes our two proposed algorithms and solution encoding. Then, in Sect. 4, the evaluation setup is described, listing the algorithms, datasets and methodology used. Section 5 presents and discusses the results of the experimentation. Finally, Sect. 6 summarizes the conclusions of this study and introduces potential lines of research.

## 2 Next Release Problem

### 2.1 Related Work

The solution of the NRP is one of the applications of the field of Search-Based Software Engineering (SBSE), where Software Engineering related problems are tackled by means of search-based optimization algorithms.

The NRP was firstly formulated by Bagnall et al. [2]. In its definition, a subset of requirements has to be selected, having as goal meeting the clients needs, minimizing development effort and maximizing clients satisfaction. They applied a variety of metaheuristics techniques, such as simulated annealing, hill climbing and GRASP, but combining the objectives of the problem into a single-objective function.

Other works started formulating the NRP as a multi-objective optimization (MOO) problem, being the first one the proposal of Zhang et al. [15]. This new formulation, Multi-Objective Next Release Problem (MONRP) was based on Pareto dominance [4] and is formally defined in Sect. 2.2. In their proposal, they tackled each objective separately, exploring the non-dominated solutions (NDS). Finkelstein et al. [7] also applied multi-objective optimization considering different measures of fairness. All these studies applied evolutionary algorithms, such as ParetoGA and NSGA-II [5] to solve the MONRP.

Although EDA approaches have been applied to SBSE problems, none has been used to tackle the NRP, to the authors' knowledge. From the most recent reviews, in Ramírez et al. [12] only an EDA application to software testing [13] is referenced; and in Gupta et al. [9] and Alba et al. [1] EDA approaches are not mentioned or matched to any solution of the NRP. Thus, we find it of interest to develop new EDA-based algorithms to be applied to the NRP.

Comparison among study proposals has been generally carried out by analyzing the Pareto fronts and execution time returned by the algorithms. However, later works started using a set of numerical metrics that evaluate different features of the Pareto fronts. Most studies compare the metrics obtained from their proposals against those obtained by other algorithms commonly applied to MOO, analyzing the performance and the advantages and weaknesses of using each algorithm to solve the MONRP. Based on this, we provide a similar comparison framework for our proposal.

### 2.2 Multi-objective Next Release Problem

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 NDS, using multi-objective optimization (MOO).

In MOO, there is no unique and optimal solution, but a Pareto front of NDS [4]. The Pareto front is a vector or set of configuration values for the decision variables that satisfies the problem constraints and optimizes the objective functions. Thus, the Pareto front contains a set of solutions that are not dominated by any other. Given a solution vector $x = [x_1, x_2, \ldots, x_j]$ where $j$ is the number of problem objectives, it dominates a solution vector $y = [y_1, y_2, \ldots, y_j]$ if and only if $y$ is not better than $x$ for any objective $i = 1, 2, \ldots, j$. In addition, there must exist at least one objective $x_i$ that is better than the respective $y_i$ of $y$. Conversely, two solutions are non-dominated as long as neither of them dominates the other.

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 NDS set. 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 $r_i$ [10]. 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, that is: $s_j = \sum_{i=1}^{m} w_i \times v_{ij}$. 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.

The MONRP objectives are the following:

$$Maximize\ S(X) = \sum_{j \in X} s_j \quad Minimize\ E(X) = \sum_{j \in X} e_j \tag{1}$$

In addition, requirements in vector $X$ might have to satisfy the constraints of the problem. These constraints are related to the interactions between requirements and to the total effort of the development.

## 3 Proposal: Univariate EDAs for the MONRP

EDAs are evolutionary algorithms based on probabilistic modelling and were designed as an alternative to genetic algorithms (GAs). As GAs, EDAs are population-based algorithms, however instead of relying upon the goodness of genetic operators, EDAs apply a more normative approach, which consists of learning a probability distribution from a set of promising individuals of the current population and sampling the estimated distribution in order to obtain the next population [11]. As no crossover nor mutation operator is needed, the number of hyperparameters decreases, thus simplifying their configuration. The complexity of an EDA is related with the degree of explicit interrelations (dependencies) it allows. Thus, in univariate EDAs no explicit dependency is allowed and interrelations are implicitly catched by the evaluation function (as in GAs), but when multivariate EDAs are used, the dependencies among the variables are explicitly modeled.

Formally, univariate EDAs assume that the $n$-dimensional joint probability distribution (JPD) factorizes like a product of $n$ univariate and independent probability distributions, that is $p_l(x) = \prod_{i=1}^{n} p_l(x_i)$. Thus framework fits well in our goal for this study as we are tackling the MONRP without dependencies between requirements, therefore we propose to adapt two univariate EDAs to work in the domain of this multi-objective problem: UMDA and PBIL. Before describing each algorithm, let us to consider some common issues: an individual is represented by a vector of booleans of length $n$, where each value indicates the inclusion or not of a requirement of the set $R$; both the satisfaction and effort of each requirement are scaled using a min-max normalization; since we only consider univariate EDA, we have not modeled cost restrictions nor interactions between requirements (as in related works [6,7,15]); finally, as we are tackling the MONRP, algorithms must return only NDS, so at each iteration of the execution algorithms update a NDS set with the new generated individuals.

**Algorithm 1.** MONRP-UMDA pseudocode

```
procedure MONRP-UMDA(maxGenerations)
    nds ← ∅                                   ▷ empty set of non-dominated solutions
    P ← generateRandomPopulation()
    for i = 0 to maxGenerations do
        individuals ← selectIndividuals(P)
        probModel ← learnProbModel(individuals)
        P ← sampleNewPopulation(probModel)
        evaluate(P)
        nds ← updateNDS(P, nds)
    end for
    return nds
end procedure
```

### 3.1 MONRP-UMDA

In Univariate Marginal Distribution Algorithm (UMDA) [11, Chapter 3] the JPD is factorized as the product of marginal distributions: $p_l(x;\theta^l) = \prod_{i=1}^{n} p_l(x_i;\theta_i^l)$. The MONRP-UMDA (Algorithm 1) starts creating a random population and, at each generation, it selects the non-dominated individuals of the population, learns the probability model $p_l$ from them, and samples a new population using $p_l$. Finally, new individuals are evaluated and the global NDS set updated by adding the new non-dominated individuals found. After execution, it returns the NDS set.

### 3.2 MONRP-PBIL

Probability Based Incremental Learning (PBIL) [11, Chapter 3] combines the mechanisms of a generational GA with simple competitive learning. Differently to UMDA, in which populations are transformed into a probability model whose only purpose is to sample new populations, PBIL algorithm attempts to create a probability model which can be considered a prototype for high evaluation vectors for the function space being explored. In a manner similar to the training of a competitive learning network, the values in the probability model are gradually shifted towards representing those in high evaluation vectors.

MONRP-PBIL (Algorithm 2) is quite similar to MONRP-UMDA, except in the updating of the probabilistic model. First, instead of selecting always the best individual found so far, at each iteration we randomly sampled an individual from the NDS set. Then, we update the probability model (vector) in two steps:

(1) $V_i = V_i \cdot (1.0 - LR) + bestIndividual_i \cdot LR$, being $i$ the $i^{th}$ gene position, and $LR$ the learning rate hyperparameter, ranging from 0 to 1.
(2) If $(Prob_{rand} < Prob_{mut})$: $V_i = V_i \cdot (1.0 - MS) + r \cdot MS$, $r$ being a random number $\in \{0, 1\}$ and $MS$ the mutation shift hyperparameter, ranged from 0 to 1.

**Algorithm 2.** MONRP-PBIL pseudocode

```
procedure MONRP-PBIL(maxGenerations, LR, Prob_mut, MS)
    nds ← ∅                                  ▷ empty set of non-dominated solutions
    probModel ← initProbModel()              ▷ set all vector initial values to 0.5
    for i = 0 to maxGenerations do
        P ← sampleNewPopulation(probModel)
        evaluate(P)
        bestInd ← selectBestIndividual(P)
        probModel ← updateProbModel(probModel, bestInd, LR, Prob_mut, MS)
        nds ← updateNDS(P, nds)
    end for
    return nds
end procedure
```

## 4 Experimental Evaluation

In this section, we present the experimental method used in the evaluation. Then, we describe other algorithm approaches used to be compared against our proposal, along with the datasets used to evaluate the algorithms. The source code for the algorithms, implemented in Python 3.8.8, along with the experimentation setup and datasets used is available at the following repository: https://github.com/uclm-simd/monrp/tree/soco22.

### 4.1 Algorithms

Our experimentation framework includes the following algorithms to compare performance and effectiveness of the two multi-objective univariate EDA versions:

- **Random search.** This is a baseline algorithm, expected to be outperformed by all algorithms. The procedure generates as many random solutions as the maximum number of evaluation functions specified. Then, it returns the NDS set.
- **Single-Objective GA.** It combines the two objective functions of the MONRP into a single objective. Then, updates the NDS set with new individuals after each generation.
- **NSGA-II.** The Non-dominated Sorting Genetic Algorithm-II [5] is a state-of-the-art multi-objective GA. It uses elitism and ranks each individual based on the level of non-dominance.

The ranges of parameters used in the experimentation for each algorithm are described in Sect. 4.3, for further detail.

### 4.2 Datasets

Algorithms performance has been tested using two widely used public datasets (P1 and P2), taken from previous NRP works. Due to the lack of datasets with

high number of clients, we decided to create another one (S3) synthetically, to test algorithms in a significantly larger instance. Dataset P1 [8] includes 20 requirements and 5 clients. Dataset P2 [14] includes 100 requirements and 5 clients. Dataset S3 includes 140 requirements and 100 clients. Each dataset contains a set of proposed requirements, defined by a vector of efforts, one effort value for each requirement. Clients are also included, defined by a vector of importances. The priority that each client gives to each requirement is also contained in the dataset, by means of a matrix of values in which each value represents the importance of a requirement for a client.

### 4.3 Methodology

We tested a set of hyperparameter configurations for each algorithm and dataset. Each configuration was executed 10 times.

For the Single-Objective GA and NSGA-II, populations were given values in the range 20 to 200 and number of generations took values among 100 to 2000. Crossover probabilities were assigned values among 0.6 to 0.9 and mutation probabilites in the range from 0 to 1. Both algorithms used a binary tournament selection, a one-point crossover scheme and an elitist replacement scheme. Both EDA approaches used population sizes and number of generations among 50 to 200. UMDA used two replacement schemes: a default one and an elitist replacement. PBIL used learning rates, mutation probabilities and mutation shifts with values between 0.1 to 0.9.

The stop criterion used by other works [3,14,15] is the number of function evaluations, commonly set to 10000. To adapt our experiments to this stop criterion, we restricted the execution of our algorithms to: $Pop.\ size * \#Gens. \leq 10000$.

We normalized datasets satisfaction and effort values, scaling them between 0 and 1. To evaluate the results, we compared the obtained Pareto fronts and a set of metrics. These metrics are quality indicators of the results generated by the algorithms and their efficiency: Hypervolume (HV), $\Delta$-Spread, Spacing and execution time.

Mean values of these metrics have been calculated and compared pairwise between algorithms using a non-parametric statistical test, more specifically, the Mann-Whitney U test with Holm correction for multiple comparison. This is a non-parametric statistical hypothesis test that allows to assess whether one of two samples of independent observations tends to have larger values than the other. All the experiments were run in the same runtime environment. The machines used had 32 Gb RAM, of which only 8 Gb were used, and two 3.00 GHz 4-core Intel Xeon E5450 processors. The operating system used was a CentOS Linux 7 with a 64-bit architecture.

## 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 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 [14]). 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 $P_m = 1$.

Regarding UMDA, best hyperparameter configuration included a population size of 200, 50 generations and an elitist replacement scheme. With the upper limit of individual evaluations set and the datasets used, it seemed to generate better results when setting a higher population size than increasing the number of generations. Regarding the replacement scheme, elitism tends to produce wider Pareto fronts.

With respect to PBIL, a population size and number of generations of 100 was used, with learning and mutation rates of 0.5 and a mutation shift of 0.1. In this case, population size and number of generations did not show a significant difference in performance. However, learning rate and mutation configurations did affect the results. A higher learning rate than 0.5 caused the algorithm to underperform. Mutation worked similarly, enhancing results when increasing the probability up to a limit, and generating worse results with probability values above 0.5. For the mutation shift, high values showed bad performance, indicating that high variations in PBIL probability vector are not suiting this problem.

### 5.2 Pareto Front Results

To analyze Pareto front results, a random execution of the best algorithm configurations is plotted for each dataset (see Fig. 1). 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, the GA's aggregation of objectives biases the search, leaving unexplored areas. NSGA-II 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. Regarding EDAs, for

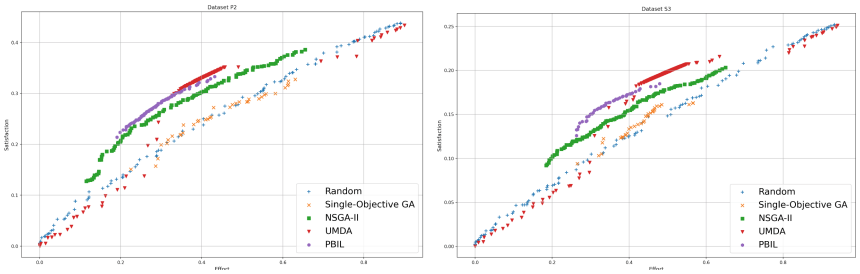


**Fig. 1.** Results from best configuration of each algorithm for datasets P2 (left) and S3 (right).

datasets P2 and S3 both algorithms struggled to generate a wide Pareto front. In the case of UMDA, it tends to group the best solutions in a certain area of the space, presumably where the probability vector drifted. In the remaining areas of the Pareto, only the random initial solutions are found, which were never updated by the algorithm. Regarding PBIL, as it does not generate an initial set of random solutions, but an initial vector, it did not create solutions in other areas of the space apart from the region were the probability vector was focused. It is interesting to highlight that both EDAs are unable to generate wide, high-quality Pareto fronts. Instead, both algorithms focused on a small region of the search space, progressively restricting the area with more consecutive generations, caused by the probability vector values stabilizing at extreme probabilities for 0 or 1 values. Despite this effect, it is worth mentioning that both EDAs have been able to overcome the Pareto front solutions returned by NSGA-II in the most balanced area of the heap.

### 5.3 Metrics Results

The mean values of the metrics obtained for each algorithm and dataset after 10 independent executions have been statistically compared, as explained in Sect. 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, as depicted in Table 1.

For the HV metric, NSGA-II obtained the best possible value for dataset P1, but UMDA generated best results for the other two datasets. This is caused by the initial random solutions generated. PBIL did not obtain a high HV metric for large datasets. For the $\Delta$-Spread metric, best values were achieved by the Single-Objective GA for P1 and P2 datasets, but PBIL reached the lowest (best) $\Delta$-Spread in dataset S3. Regarding Spacing, no EDA approach could surpass NSGA-II values, both algorithms obtaining similar results for each dataset. Finally, regarding execution time, EDA approaches resulted to run very fast, being PBIL the fastest algorithm for all datasets. These results show that EDA can lead to a very efficient and specific search through the Pareto front, being quicker than GAs to generate and evolve solutions, thanks to the simplified

**Table 1.** Average metrics of the best hyperparameter configurations for each algorithm and dataset

| Dataset | Algorithm | HV | Δ-Spread | Spacing | Execution time (s) |
|---|---|---|---|---|---|
| P1 | Single-Objective GA | 0.594↓ | **0.615** | 0.323↓ | 17.967↓ |
| | NSGA-II | **1.000** | 0.963↓ | **0.382** | 180.991↓ |
| | UMDA | 0.878↓ | 0.744↓ | 0.330↓ | 5.494↓ |
| | PBIL | 0.554↓ | 0.692↓ | 0.316↓ | **3.036** |
| P2 | Single-Objective GA | 0.157↓ | **0.637** | 0.128↓ | 82.713↓ |
| | NSGA-II | 0.407↓ | 0.969↓ | **0.245** | 616.415↓ |
| | UMDA | **0.975** | 1.008↓ | 0.111↓ | 21.489↓ |
| | PBIL | 0.138↓ | 0.670 | 0.114↓ | **4.099** |
| S3 | Single-Objective GA | 0.102↓ | 0.720 | 0.105↓ | 125.702↓ |
| | NSGA-II | 0.286↓ | 0.970↓ | **0.206** | 859.928↓ |
| | UMDA | **0.981** | 1.025↓ | 0.103↓ | 23.079↓ |
| | PBIL | 0.089↓ | **0.678** | 0.105↓ | **3.802** |

methods they use, most of them applied only to the probability vector, instead of to the population.

## 6 Conclusions and Future Works

In this work, we have presented new algorithm proposals to the MONRP field, introducing the application of EDA techniques. Formerly, this problem has been commonly tackled by means of evolutionary algorithms, mainly GAs. This has been the first time that the EDA family of algorithms has been used to solve MONRP. For this purpose, we have considered the use of univariate EDAs as a first approach. In this paper we have adapted UMDA and PBIL algorithms and compared them against two GAs (Single-Objective GA and NSGA-II) by means of Pareto front results and quality metrics. Our proposals were able to overcome GAs in the most crowded regions of the Pareto front, despite generating low-quality or no solutions in other areas of the search space. Regarding quality metrics, our EDA proposals have been able to get rather good results compared to those of the GAs, showing the best performance in the execution time. Future research will focus on the applicability of multivariate EDA algorithms, aiming to manage the MONRP with modeled dependencies between requirements.

**Acknowledgements.** This work has been partially funded by the Regional Government (JCCM) and ERDF funds through the projects SBPLY/17/180501/000493 and SBPLY/21/180501/000148.

## References


1. Alba, E., Ferrer, J., Villalobos, I.: Metaheuristics and software engineering: past, present, and future. Int. J. Software Eng. Knowl. Eng. **31**(09), 1349–1375 (2021)
2. Bagnall, A.J., Rayward-Smith, V.J., Whittley, I.M.: The next release problem. Inf. Softw. Technol. **43**(14), 883–890 (2001)
3. Chaves-González, J.M., Pérez-Toledano, M.A., Navasa, A.: Software requirement optimization using a multiobjective swarm intelligence evolutionary algorithm. Knowl.-Based Syst. **83**(1), 105–115 (2015)
4. Coello Coello, C.A., Lamont, G.B., Veldhuizen, D.A.V.: Evolutionary Algorithms for Solving Multi-Objective Problems. Springer, Heidelberg (2007). https://doi.org/10.1007/978-0-387-36797-2
5. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans. Evol. Comput. **6**(2), 182–197 (2002)
6. Durillo, J.J., Zhang, Y., Alba, E., Harman, M., Nebro, A.J.: A study of the bi-objective next release problem. Empir. Softw. Eng. **16**(1), 29–60 (2011)
7. 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 Eng. **14**(4), 231–245 (2009)
8. Greer, D., Ruhe, G.: Software release planning: an evolutionary and iterative approach. Inf. Softw. Technol. **46**, 243–253 (2004)
9. Gupta, P., Arora, I., Saha, A.: A review of applications of search based software engineering techniques in last decade. In: 2016 5th International Conference on Reliability, Infocom Technologies and Optimization, ICRITO 2016: Trends and Future Directions, vol. 978, pp. 584–589 (2016)
10. Harman, M., McMinn, P., de Souza, J.T., Yoo, S.: Search based software engineering: techniques, taxonomy, tutorial. In: Meyer, B., Nordio, M. (eds.) Empirical Software Engineering and Verification: International Summer Schools, pp. 1–59. Springer, Berlin Heidelberg (2012). https://doi.org/10.1007/978-3-642-25231-0_1
11. Larrañaga, P., Lozano, J.A.: Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation. Kluwer Academic Publishers, New York (2001)
12. Ramírez, A., Delgado-Pérez, P., Ferrer, J., Romero, J.R., Medina-Bulo, I., Chicano, F.: A systematic literature review of the SBSE research community in Spain. Prog. Artif. Intell. **9**(2), 113–128 (2020)
13. Sagarna, R., Lozano, J.A.: On the performance of estimation of distribution algorithms applied to software testing. Appl. Artif. Intell. **19**(5), 457–489 (2005)
14. del Sagrado, J., del Águila, I.M., Orellana, F.J.: Multi-objective ant colony optimization for requirements selection. Empir. Softw. Eng. **20**(3), 577–610 (2013). https://doi.org/10.1007/s10664-013-9287-3
15. Zhang, Y., Harman, M., Mansouri, A.: The multi-objective next release problem. In: GECCO 2007: Genetic and Evolutionary Computation Conference, pp. 1129–1137 (2007)