SCIENCE CHINA
Information Sciences

• RESEARCH PAPER •

March 2017, Vol. 60 032105:1–032105:20
doi: 10.1007/s11432-016-0089-7

Feature based problem hardness understanding for requirements engineering

Zhilei REN[1,2], He JIANG[1,2*], Jifeng XUAN[3], Shuwei ZHANG[1,2] & Zhongxuan LUO[1,2]

[1]*Key Laboratory for Ubiquitous Network and Service Software of Liaoning Province, Dalian University of Technology, Dalian* 116621*, China;*
[2]*School of Software, Dalian University of Technology, Dalian* 116621*, China;*
[3]*State Key Laboratory of Software Engineering, Wuhan University, Wuhan* 430072*, China*

Received May 22, 2016; accepted July 8, 2016; published online December 6, 2016

Abstract Heuristics and metaheuristics have achieved great accomplishments in various fields, and the investigation of the relationship between these algorithms and the problem hardness has been a hot topic in the research field. Related research work has contributed much to the understanding of the underlying mechanisms of the algorithms for problem solving. However, most existing studies consider traditional combinatorial problems as their case studies. In this study, taking the Next Release Problem (NRP) from the requirements engineering as a case study, we investigate the relationship between software engineering problem instances and heuristics. We employ an evolutionary algorithm to evolve NRP instances, which are uniquely hard or easy for the target heuristic (Greedy Randomized Adaptive Search Procedure and Randomized Hill Climbing in this paper). Then, we use a feature-based method to estimate the hardness of the evolved instances, with respect to the target heuristic. Experimental results demonstrate that, evolutionary algorithm can be used to evolve NRP instances that are uniquely hard or easy to solve. Moreover, the features enable the estimation of the target heuristics' performance.

Keywords problem hardness, next release problem, computational intelligence, requirements engineering, evolution algorithm

Citation Ren Z L, Jiang H, Xuan J F, et al. Feature based problem hardness understanding for requirements engineering. Sci China Inf Sci, 2017, 60(3): 032105, doi: 10.1007/s11432-016-0089-7

1 Introduction

Heuristics and metaheuristics have been successfully applied to problem solving in various problem domains [1–7], due to their effectiveness and efficiency. Furthermore, aside from problem solving, understanding the behavior of heuristic algorithms, and learning about the strength and weakness of these algorithms have always been hot topics in the research field [8, 9]. To discover the suitability of applying these techniques for problem solving, recent years have witnessed a rapid growing research interest that intends to discover the relationship between algorithm performance and instance-specific features [10–12]. The features represent the measurable attributes of the instances generated by a computational extraction process, which describe the characteristics of the instances [8], such as the distribution indicators of

* Corresponding author (email: jianghe@dlut.edu.cn)

the constraints, correlation coefficient between variables, etc. The research work in this category shares a similar paradigm, i.e., first an evolutionary algorithm is employed to evolve a set of problem instances that are uniquely hard or easy to solve. Then, approaches such as meta-learning [13] and algorithm selection [14] are usually adopted, to mine the potential factors that cause the difference algorithm performance. Within this mining process, the suitable choice of the features are essential to the success of the learning stage.

However, despite the promising achievements these approaches have accomplished, these approaches still face several limitations. For example, most of these approaches consider traditional combinatorial problems as their case studies, such as the traveling salesman problem [10, 12, 15] and the graph coloring problem [11]. The reason for this phenomenon might be that, the formulations and the encoding schemes for these traditional problems are relatively simple. Meanwhile, since these traditional problems have been extensively studied, some features could be directly obtained from the existing literature, to describe the characteristics of the problem.

In this study, we take the Next Release Problem (NRP) as a case study, to analyze the relationship between heuristic algorithms' performance and the combinatorial problem instances. Unlike the traditional problems, the NRP originates directly from the software engineering domain [16–18], and plays an important role in the requirements engineering field [19]. Solving the NRP effectively and efficiently is crucial to the software engineering life cycle. Hence, analyzing the underlying mechanisms of the algorithms for the NRP is both of great importance and challenging. On the one hand, the insights into such problem might be generalized to other problem domains, in that there are problems that have similar formulations or encoding schemes as the NRP [20, 21]. On the other hand, understanding the relationship between heuristics and the problem instances is challenging from the following perspectives.

First, understanding the strength and the weakness of a given heuristic is a difficult yet important issue. For example, given the problem formulations and the target heuristic, obtaining instances that are hard to solve itself remain a hard problem[1)] [12]. In the existing literature, most study relies on benchmark instances or benchmark function suites [22–25]. If we could acquire the knowledge about the upper and lower bounds of the target heuristic's performance (e.g., the best case and the worst case optimality gap), we could better understand the behavior of the heuristic.

Second, given problem instances, it is challenging to investigate how to assess the suitability of applying the target heuristic for problem solving. If we could estimate the performance of the target heuristic over the instances accurately, better decisions could be made on the algorithm selection task. This issue is particularly important in the software engineering field, because in the software life cycle, the solution quality achieved by heuristics may have a direct impact on the revenue of stakeholders. Consequently, it would be profitable to analyze the problem instances that are uniquely hard or easy to solve, and examine the possibility of predicting the target heuristics' performance estimation.

Following the existing work, we propose a feature-based research framework, which is illustrated in Figure 1. The framework comprises two stages, which correspond to the instance evolution and the learning process, respectively. In the first stage, we use an evolutionary algorithm to evolve a set of NRP instances that are hard/easy to solve, respectively, with respect to the target heuristic. Then in the second stage, we analyze the instances obtained from the previous stage, and try to learn about the strengths and the weaknesses of the target heuristic in terms of the structure mined from the evolved instances. More specifically, we extract a set of features from the evolved instances, to capture the characteristics of the NRP instances. In particular, this study is unique in that we extract the features not only from the instance variables and the constraints, but also from the relaxed problem of the NRP. With these proposed features, we intend to investigate the possibility of target heuristics' performance over these instances. Besides, since directly evolving instances against sophisticated metaheuristics such as the Approximate Backbone Multilevel Algorithm (ABMA) [16] may be too time consuming, we try evolving instances against simple heuristics instead, such as Greedy Randomized Adaptive Search Procedure (GRASP) [19] and Randomly Hill Climbing (RHC) [26, 27], which are basic components of the aforementioned

1) In this study, we measure the hardness of a problem instance for the target heuristic by calculating the optimality gap that the heuristic could achieve. Larger optimality gap implies harder instances (see Eq. (6)).

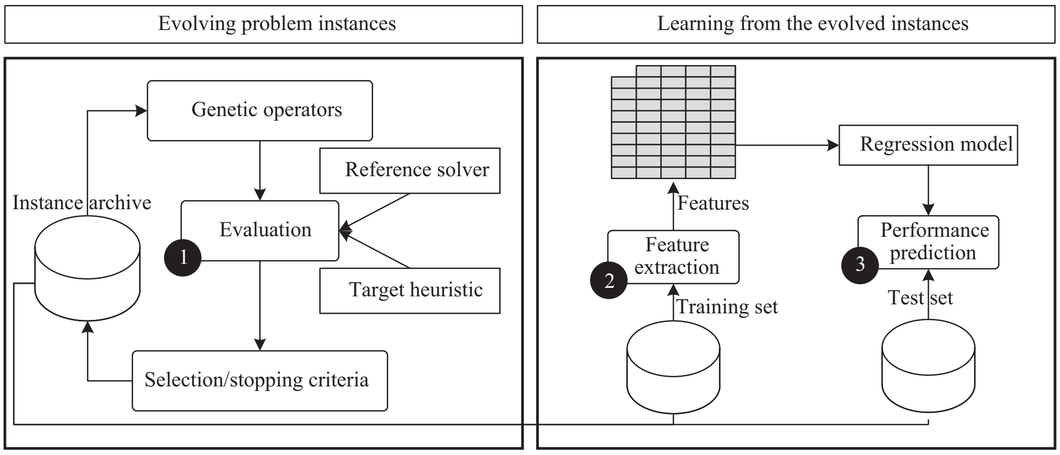


Figure 1 Illustration of the research framework.

metaheuristics, and examine whether these evolved instances pose challenges to the metaheuristics. By this design, we intend to achieve the desirable instances more efficiently. Then, within the learning stage, analysis is conducted to examine what features have major influences over the problem instance hardness. With this framework, we concentrate on the following research questions (RQs):

RQ1: Is an evolutionary algorithm able to evolve NRP instances that are hard/easy to solve, respectively, with respect to the target heuristic?

RQ2: Do the features of the evolved NRP instances with different hardness exhibit different characteristics?

RQ3: Is it possible to predict the performance of the target heuristic, with the help of the proposed features?

The remainder of this paper is organized as follows. In Section 2, we present the related work of both evolving hard instances and the NRP. In Section 3, we propose the simple evolutionary algorithm to evolve the NRP instances. In Section 4, the features capturing the characteristics of the NRP instances are discussed in detail. The experimental results are given and analyzed in Section 5. The threats to validity are discussed in Section 6. Finally, the conclusion and the future work are presented in Section 7.

2 Related work

2.1 Evolving hard problem instances

Recently, there have been a series of studies that focus on the investigation, which intend to measure the features of a problem instance, given a specific heuristic. By performing such task, researchers try to gain insights into understanding the nature of the target heuristic, and discover what features of a problem instance make the target heuristic perform poorly or well. For example, empirical studies focusing on learning from evolve problem instances have been conducted over several problem domains, such as the constraint satisfaction problem [28], the quadratic knapsack problem [29], the graph coloring problem [11], the timetabling problem [30], and the traveling salesman problem [10, 12, 15].

The commonality of these existing methodologies is that, they all intend to discover the underlying relationship between the algorithm performance and the problem-specific features. These features are also known as meta-data in the machine learning community [8], which are used to depict the distribution of the problem instances. Based on the features, approaches such as regression analysis could be applied, to predict the hardness of problem instances, with respect to certain target heuristics. In turn, regression models could be helpful in making decisions about solving the problem with the most suitable algorithm.

A key challenge that these approaches face is that it has to be guaranteed that the training instances from which to learn have sufficient diversity [9]. Contrarily, if all the training instances are close to each other in the feature space, or the target heuristic performs similarly over these instances, helpful knowledge would not be possible to be learnt. Hence, what kind of instances and what features should be employed in the learning process play an essential role.

To cope with this challenge, evolutionary algorithms are adopted to evolve the training instances. More specifically, a population of individuals are maintained, with each individual represented by a problem instance. The fitness of each individual is evaluated by certain algorithm performance related measurements, such as approximation ratio [10] and execution time [12]. Along the evolutionary search process, new individuals are produced by applying genetic operators over the individuals, and selection is used to guide the search towards the evolutionary objective, which could be either hard or easy.

Finally, after sufficient problem instances have been evolved, interesting features could be extracted from these instances to capture the distribution of the problem instances, so that the target heuristic's strength and weakness could be understood. For example, meta-learning models could be built, to investigate the correlations between the problem-specific features and the instance hardness [8, 31–36]. Furthermore, with the extracted features, it is possible to construct regression models. Given an instance, the performance of certain heuristic could be predicted, without executing the algorithm.

However, as mentioned in Section 1, most existing studies focus on classical problems whose formulations and encoding schemes are relatively simple. In this study, the case study we consider is the NRP, which has more complex formulations. Since the NRP could be encoded as the combination of numeric vectors and graph based dependencies. The ideas in this study might be able to transfer to other problem that have similar encoding schemes.

2.2 The next release problem

As an important problem from software engineering, the NRP intends to maximize the customer profits from a set of dependent requirements bounded by a predefined budget. With the help of the NRP, a requirements engineer can make a decision for software requirements to balance the company profit and the customers' profit. As a combinatorial optimization problem, the NRP has been proved to be NP-hard even if the customer requirements are independent. In the most extreme case, i.e., when all the requirements are independent, the NRP could be reduced to the knapsack problem [19].

In a software project, let R be the set of all the candidate requirements. The size of R is $|R| = m$. Each requirement $r_j \in R$ $(1 \leqslant j \leqslant m)$ corresponds to a nonnegative cost $c_j \in C$. A directed acyclic graph $G = (R, E)$ indicates the dependencies between these requirements, with R denoting the set of vertices and E specifying the set of arcs. In the dependency graph G, an arc $(r', r) \in E$ means that the requirement r relies on r', i.e., if r is to be realized in the next release, r' must also be realized, to satisfy the dependency. We call the requirement r the child requirement of r'. parents(r) is denoted as the set of requirements that can reach r in the graph, which is defined as parents$(r) = \{r' \in R | (r', r'') \in E, r'' \in \text{parents}(r)\}$. All requirements of parents(r) have to be realized to ensure that r is implemented.

Let S be the customer set related to the requirements R with $|S| = n$. Each customer $s_i \in S$, requests a subset of requirements $R_i \subseteq R$. Let $w_i \in W$ be the revenue for the customer s_i. Let parents(R_i) be $\bigcup_{r \in R_i} \text{parents}(r)$. Given a customer s_i, let the set of requirements requested by s_i be $\hat{R}_i = R_i \bigcup \text{parents}(R_i)$. Hence, a customer s_i can be satisfied by the software release planning if and only if all the requirements in R_i are realized during the next release. Let the cost of satisfying the customer s_i be $\text{cost}(\hat{R}_i) = \sum_{r_j \in \hat{R}_i} c_j$. The NRP is formulated as follows. Given variables $\{y_1, \ldots, y_n\}$ (customer indicator, associated with S) and $\{x_1, \ldots, x_m\}$ (requirement indicator, associated with R), as well as the corresponding revenues $\{w_1, \ldots, w_n\}$ and cost $\{c_1, \ldots, c_m\}$, the NRP aims to select a subset of customers to maximize the corresponding revenues. Without loss of generality, in this study we assume that the customers are sorted in descending order with respect to their revenue-to-cost ratios $(\frac{w_1}{\text{cost}(\hat{R}_1)} > \frac{w_2}{\text{cost}(\hat{R}_2)} > \cdots > \frac{w_n}{\text{cost}(\hat{R}_n)})$. The objective of the NRP is to maximize the following formulations:

$$\sum_{i=1}^{n} w_i y_i, \tag{1}$$

subject to

$$\sum_{i=1}^{m} c_i x_i \leqslant B, \tag{2}$$

$$x_i \geqslant x_j, \quad \forall (r_i, r_j) \in E, \tag{3}$$

$$x_i \geqslant y_j, \quad \forall i, j \text{ where } r_i \text{ is required by customer } j, \tag{4}$$

$$x_i, y_j \in \{0,1\}, \quad 1 \leqslant i \leqslant m,\ 1 \leqslant j \leqslant n. \tag{5}$$

In the problem formulations, constraint (2) indicates that the total budget is bounded ($B \in Z^+$ is predefined). Constraint (3) means that the dependencies between software components have to be satisfied. Constraint (4) illustrates the relationships between the customers and the software components. Constraint (5) ensures that all the variables are integers. Since the NRP was addressed in the literature, there have been various algorithms that are proposed to tackle the problem. For example, Bagnall et al. [19] develop several algorithms for solving the NRP, such as greedy search, local search, and Greedy Randomized Adaptive Search Procedure (GRASP). Ngo-The and Ruhe [37] propose a two-phase algorithm, which features the combination of mathematical programming and genetic programming. He et al. [26] address a hybrid ant based model to tackle the problem, in which a Randomized Hill Climbing (RHC) heuristics is embedded in an ant colony optimization algorithm. Xuan et al. present the backbone-based algorithm in [16], an effective framework in which other heuristics could be embedded as components. From the problem formulation, we could observe that each NRP instance comprises both graph variables and numeric variables. Hence, we could extract features from both the graph and the distributions of the numeric vectors. Furthermore, the findings of this study could be potentially inspiring for other problem domains that have graph based or vector based formulations, such as release planning [20], software upgradability [21], etc. For example, the software upgradability problem intends to find the most compact subset of software packages, so that the upgrade requirements and the dependencies among software packages are satisfied. Since the dependencies could be formulated as a graph, which is similar to the constraints of the NRP, the ideas in this study may be generalized to the software upgradability problem as well.

3 Evolutionary algorithm to generate NRP instances

In this section, we present the design and the implementation issues of the evolutionary algorithm we use to evolve NRP instances, which are hard or easy to solve, respectively.

The pseudo code of the proposed Evolutionary algorithm to generate NRP Instance (indicated by Evol-NI) is presented in Algorithm 1. Evol-NI follows the paradigm of most evolutionary algorithms, i.e., the algorithm maintains a population of individuals, which undergo generations of evolution. However, in this study, each individual is represented by an NRP instance. More specifically, we intend to evolve a population of NRP instances. Hence, the variables of each individual involve the values of the revenues and the costs, and/or the dependency structure of the instances. By altering the values of the instance variables, one NRP instance is transfered to another.

Similar with the existing evolutionary algorithms, Evol-NI consists of two phases, i.e., the initialization phase and the main loop. During the initialization phase (lines 2–6), Evol-NI generates a population of size random NRP instances, with the input instance as seed. For each instance, the values of the revenues and the costs are randomly assigned, and the range of these values are set with respect to [16]. The dependency graphs of each individual is inherited from the input seed instance. The reason for this design is that, the structural information (e.g., the dependency graph) could be mined from software repositories [16], but the revenues and the costs are hard to determine. By this algorithm scheme, we would be able to analyze the impact of changing the revenues and the costs. Then, Evol-NI undergoes the main loop of evolution (lines 7–19). At each iteration, new NRP instances are produced by applying genetic operators (crossover and mutation) to the individuals of the current population. For each individual, another individual is randomly selected as the mating parent, and the single point crossover is applied over both the revenues

Algorithm 1 EVOL-NI

```
Require: seed instance $\pi$, evolution goal goal, maximum iteration iter, population size size, mutation rate $\mu$, target heuristic
    $h$, reference solver $S$;
Ensure: evolved instance $\pi^*$;
    {Initialization phase};
 1: Initialize an empty instance population pop $\leftarrow \emptyset$;
 2: for $i \leftarrow 1$ to size do
 3:     $\pi' \leftarrow \pi$;
 4:     Randomly reassign the revenues and the costs of $\pi'$;
 5:     pop $\leftarrow$ pop $\bigcup\{\pi'\}$;
 6: end for
    {Main loop};
 7: while iter $> 0$ do
 8:     iter $\leftarrow$ iter $- 1$;
 9:     pop$'$ $\leftarrow \emptyset$;
10:     for each $\pi' \in$ pop do
11:         Randomly select another instance $\pi'' \in$ pop;
12:         child $\leftarrow$ Crossover-NI($\pi', \pi''$);
13:         goal$'$ $\leftarrow$ Mutate-Numeric-NI(goal);
14:         goal$''$ $\leftarrow$ Mutate-Structural-NI(goal$'$);
15:         Evaluate(goal$''$, $h$, $S$);
16:         pop$'$ $\leftarrow$ pop$'$ $\bigcup\{$goal$''\}$;
17:     end for
18:     pop $\leftarrow$ Select(goal, pop $\bigcup$ pop$'$);
19: end while
20: return the best instance of pop, with respect to the goal;
```

and the cost vectors, to obtain the child individual (line 12). Then, perturbation is conducted over the child individual to make the evolutionary process more exploratory, using the mutation operators (lines 13 and 14).

In this study, the crossover and the mutation operators are adopted, to make the search within the instance space more diverse. More specifically, the crossover takes two input instances as the parent, and generates an offspring instance that combines the information of the two parents. The operator is applied over the vectors w and c, which follows the single point crossover paradigm. The pseudo code of the crossover is illustrated in Algorithm 2. Besides the crossover operator, we propose two mutation operators. For the first operator (indicated by Mutate-Numeric-NI), we intend to transfer one NRP instance to another by modifying the values of the numeric variables. The variables we are interested in the revenues, the costs, and the budget ratio. By perturbing these variables, we are able to obtain new offspring instances without changing the dependency structure of the parent instances. The pseudo code of Mutate-Numeric-NI is presented in Algorithm 3. For the second operator (denoted by Mutate-Structural-NI), we obtain offspring instances by altering the dependency structure of the parent instances. To be more specific, we realize the mutation functionality by randomly adding or removing edges between requirements, with a small probability μ. As mentioned, since the dependency graphs are inherited from the benchmark instances, we add a constraint to the Mutate-Structural-NI operator, so that the maximum number of edge does not differ too much from the seed instance (line 6 of Algorithm 4). The reasons we adopt this design are two-fold. On the one hand, in the existing NRP studies, the dependency graph could be extracted from software repositories. Meanwhile, the costs and the revenues are hard to measure, hence are randomly assigned [16, 19]. On the other hand, we choose to perturb the graph to simulate the evolution of the software structure. However, if we do not restrict the structural perturbation, the evolution may lead to instances that are highly different from realistic instances. The pseudo code of the genetic operators are presented in Algorithms 2–4, respectively.

After all the child individuals have been produced, they are evaluated based on the performance of certain algorithm. In this study, the target heuristics we are interested in include GRASP [19] and RHC [26], due to their simplicity and the ability to balance the effectiveness and the efficiency [19]. More

Algorithm 2 Crossover-NI

```
Require: parent instance $\pi_1$, parent instance $\pi_2$;
Ensure: child instance $\pi_3$;
     {Inherit the dependency graph, the budget bound, etc, from the parent};
 1: $\pi_3 \leftarrow \pi_1$;
     {Randomly select the crossover points};
 2: $\text{crossoverpoint}_w \leftarrow \lfloor \mathbf{random}(1, n) \rfloor$;
 3: $\text{crossoverpoint}_c \leftarrow \lfloor \mathbf{random}(1, m) \rfloor$;
     {Conduct the crossover};
 4: for $i \leftarrow 1$ to $\text{crossoverpoint}_w$ do
 5:     Assign the value of $w_i$ with respect to $\pi_1$;
 6: end for
 7: for $i \leftarrow \text{crossoverpoint}_w + 1$ to $n$ do
 8:     Assign the value of $w_i$ with respect to $\pi_2$;
 9: end for
10: for $i \leftarrow 1$ to $\text{crossoverpoint}_c$ do
11:     Assign the value of $c_i$ with respect to $\pi_1$;
12: end for
13: for $i \leftarrow \text{crossoverpoint}_c + 1$ to $m$ do
14:     Assign the value of $c_i$ with respect to $\pi_2$;
15: end for
16: return $\pi_3$;
```

Algorithm 3 Mutate-Numeric-NI

```
Require: input instance $\pi$, mutation rate $\mu$;
Ensure: output instance $\pi'$;
     {Inherit the dependency graph, the budget bound, etc, from the input instance};
 1: $\pi' \leftarrow \pi$;
     {Perturb the values of $w$ and $c$, with respect to the mutation rate};
 2: for each element $w_i$ of the revenue vector of $\pi'$ do
 3:     if random(0, 1) $< \mu$ then
 4:         Randomly reassign the value of $w_i$;
 5:     end if
 6: end for
 7: for each element $c_i$ of the cost vector of $\pi'$ do
 8:     if random(0, 1) $< \mu$ then
 9:         Randomly reassign the value of $c_i$;
10:     end if
11: end for
12: return $\pi'$;
```

specifically, we adopt the optimality gap as the performance measure, which is defined as

$$\text{gap} = \frac{C_{\text{opt}} - C_{\text{avg}}}{C_{\text{opt}}}, \tag{6}$$

where C_{opt} and C_{avg} denote the objective values of the optimal solution, as well as the average objective value of the solutions achieved from the 30 independent executions of the target heuristic (see Algorithm 5). Given two NRP instances, the one over which GRASP obtains higher optimality gap is considered to be harder. To obtain the optimal solution quality, we employ Gurobi, which is among the state-of-the-art mathematical programming solvers2), as the reference solver.

At the end of each iteration, the selection operator is used to select the survival individuals for the next iteration. According to the goal of the evolution (easy or hard), the individuals are sorted in ascending or descending order of the optimality gap the target heuristic could achieve, and the first half of the population survive to the next generation. Then the evolution continues, until the maximum number of iteration is reached. The summary of the proposed algorithm is presented in Table 1.

2) http://www.gurobi.com.

Algorithm 4 Mutate-Structural-NI

```
Require: input instance $\pi$, mutation rate $\mu$, maximum number of edge $\rho$;
Ensure: output instance $\pi'$;
     {Inherit the dependency graph, the budget bound, etc, from the input instance};
 1: $\pi' \leftarrow \pi$;
     {Randomly add or remove dependencies between requirements};
 2: for each pair of requirements $r_i \in R$ and $r_j \in R$, $i < j$ do
 3:     if random(0, 1) $< \mu$ then
 4:         if $(r_i, r_j) \in E$ then
 5:             $E \leftarrow E \backslash (r_i, r_j)$;
 6:         else if $|E| < \rho$ then
 7:             $E \leftarrow E \bigcup (r_i, r_j)$;
 8:         end if
 9:     end if
10: end for
11: return $\pi'$;
```

Algorithm 5 Evaluate

```
Require: instance $\pi$, target heuristic $h$, reference solver $S$;
Ensure: objective value;
 1: $C_{\text{avg}} \leftarrow$ Average objective values of applying $h$ over $\pi$;
 2: $C_{\text{opt}} \leftarrow$ Objective value of applying $S$ over $\pi$;
 3: return $\frac{C_{\text{opt}} - C_{\text{avg}}}{C_{\text{opt}}}$;
```

Table 1 Summary of Evol-NI

Component	Value	Component	Value
Target heuristics	GRASP and RHC	Mutation rate	Parameter, see Section 5
Population size	Parameter, see Section 5	Selection type	Truncation selection
Crossover type	Single point crossover	Fitness evaluation	Optimality gap
Mutation types	Numeric and structural	Evolution goal	Hard/easy

4 Capturing the NRP characteristics using features

In this section, we discuss the features that may lead to hard and easy NRP instances. In this study, we consider three sets of features that may influence the instance hardness.

The first set of features intend to capture the characteristics of the NRP instances' dependency graph, in that software systems could be treated as networks of components connected by certain dependence relationships [38]. As mentioned in Section 2, for each NRP instance, the dependency relationship between software components could be abstracted as a directed graph. Hence, features could be extracted from the graph. In this study, the graph-based features include (1) the average node betweenness, (2) the standard deviation of the node betweenness, (3) and (4) the average in/out degree, (5) and (6) the standard deviation of in/out degree.

The second set of features try to characterize the distribution of the instance parameters, e.g., the distribution of the revenues and the costs. The features in this set include (7) the budget bound, (8) the correlation between each customer's revenue and the corresponding cost, (9)–(11) the average revenue/cost/revenue-to-cost ratio, (12)–(14) the coefficient of variation of the revenues/costs/revenue-to-cost ratio.

The final set of features is extracted from the relaxed version of the NRP. The motivation of these features is inspired by the core concept from the knapsack problem literature [39, 40]. For the knapsack problem, the concept of the core problem is based on the difference between the optimal solutions and the permutation of all the variables, which are sorted according to their profit-to-weight ratio. The larger the difference is, the larger the core problem will be, which might subsequently imply that the corresponding knapsack instance is more difficult. In contrast, small core problems usually imply easy knapsack instances. However, since the optimal solutions could not be obtained in advance, the optimal solution to the relaxed knapsack problem is usually employed as a compromise. More detailed information about the concept could be found in [41].

In the existing literature, it has been demonstrated that the NRP is closely related to the knapsack problem. For example, the NRP is reducible to the knapsack problem by omitting the dependencies between the customers and the requirements [19]. Also, both the problems have similar solution representations, i.e., the solutions could be indicated by a binary vectors. In this study, we intend to extend the idea of the core concept to the NRP domain. Hence, for the rest of this section, we shall introduce these features in detail. As mentioned, in this study, we assume the customers are sorted according to the revenue-to-cost ratio. Given an optimal solution $\{y_1^*, \ldots, y_n^*\}$ to the NRP, we define the following auxiliary variables: $j_1 = \min\{1 \leqslant j \leqslant n | y_j^* = 0\}, j_2 = \max\{1 \leqslant j \leqslant n | y_j^* = 1\}, j_{\min} = \min\{j_1, j_2\}, j_{\max} = \max\{j_1, j_2\}$. Intuitively, the smaller $j_{\max} - j_{\min}$ is, the more similar the sorted customer list is to the optimal solution[3)]. In the most extreme case, if $j_1 = j_2 + 1$ holds, a simple sort to the customer list could lead to the optimal solution. Furthermore, due to the intrinsic hardness of the NRP, we consider the relaxed version of the NRP in [19], i.e., we replace constraint (5) with the following constraint:

$$x_i \in [0,1], \quad y_j \in [0,1], \quad 1 \leqslant i \leqslant m, \quad 1 \leqslant j \leqslant n. \tag{7}$$

Given the optimal solution $\{\hat{y}_1, \ldots, \hat{y}_n\}$ to the relaxed NRP, we define $\hat{j}_1$, $\hat{j}_2$, $\hat{j}_{\min}$, $\hat{j}_{\max}$ as follows: $\hat{j}_1 = \min\{1 \leqslant j \leqslant n | \hat{y}_j = 0\}, \hat{j}_2 = \max\{1 \leqslant j \leqslant n | \hat{y}_j > 0\}, \hat{j}_{\min} = \min\{\hat{j}_1, \hat{j}_2\}, \hat{j}_{\max} = \max\{\hat{j}_1, \hat{j}_2\}$. With the auxiliary variables $\hat{j}_{\min}$ and $\hat{j}_{\max}$, the customer set could be partitioned into three subsets (denoted as subset1, subset2, and subset3 correspond to those customers with index ranging from $[1, \hat{j}_{\min})$, $[\hat{j}_{\min}, \hat{j}_{\max}]$, and $(\hat{j}_{\max}, n]$, respectively). We introduce the following features: (15)–(17) relative scales of the three subsets, (18)–(23) the rescaled sum of revenues and sum of the costs over the three customer subsets as features. Besides, (24) the upper bound obtained from the optimal solution to the relaxed NRP instance is also used as a feature. Among these features, features (15)–(23) are defined as follows:

$$\text{scale.subset1} = \frac{\hat{j}_{\min}}{n}, \tag{8}$$

$$\text{scale.subset2} = \frac{\hat{j}_{\max} - \hat{j}_{\min}}{n}, \tag{9}$$

$$\text{scale.subset3} = \frac{n - \hat{j}_{\max}}{n}, \tag{10}$$

$$\text{sum.revenue.subset1} = \frac{\sum_{1 \leqslant i < \hat{j}_{\min}} w_i}{\sum_{1 \leqslant i \leqslant n} w_i}, \tag{11}$$

$$\text{sum.revenue.subset2} = \frac{\sum_{\hat{j}_{\min} \leqslant i \leqslant \hat{j}_{\max}} w_i}{\sum_{1 \leqslant i \leqslant n} w_i}, \tag{12}$$

$$\text{sum.revenue.subset3} = \frac{\sum_{\hat{j}_{\max} < i \leqslant n} w_i}{\sum_{1 \leqslant i \leqslant n} w_i}, \tag{13}$$

$$\text{sum.cost.subset1} = \frac{\sum_{1 \leqslant i < \hat{j}_{\min}} \text{cost}(\hat{R}_i)}{\sum_{1 \leqslant i \leqslant m} c_i}, \tag{14}$$

$$\text{sum.cost.subset2} = \frac{\sum_{\hat{j}_{\min} \leqslant i \leqslant \hat{j}_{\max}} \text{cost}(\hat{R}_i)}{\sum_{1 \leqslant i \leqslant m} c_i}, \tag{15}$$

$$\text{sum.cost.subset3} = \frac{\sum_{\hat{j}_{\max} < i \leqslant n} \text{cost}(\hat{R}_i)}{\sum_{1 \leqslant i \leqslant m} c_i}. \tag{16}$$

In summary, in this section, we propose 24 features that may have influences on the hardness of the evolved instances, which are illustrated in Table 2. In the following section, we would proceed to examine the influences of these features.

3) If there are multiple optimal solutions, the one which minimizes $j_{\max} - j_{\min}$ is adopted.

Table 2 Summary of the proposed features

Feature set	ID	Feature name	Description
1	1	avg.node.between	Average node betweenness of the dependency graph
	2	sd.node.between	Standard deviation of the node betweenness of the dependency graph
	3	avg.in.degree	Average in degree of the dependency graph
	4	avg.out.degree	Average out degree of the dependency graph
	5	sd.in.degree	Standard deviation of the in degree of the dependency graph
	6	sd.out.degree	Standard deviation of the out degree of the dependency graph
2	7	budget.bound	Total budget bound
	8	cor.revenue.cost	Correlation between each customer's revenue and cost
	9	avg.revenue	Average revenue of the customers
	10	avg.cost	Average cost of the customers
	11	avg.ratio	Average revenue-to-cost ratio of the customers
	12	cov.revenue	Coefficient of variation of the revenues
	13	cov.cost	Coefficient of variation of the costs
	14	cov.ratio	Coefficient of variation of the revenue-to-cost ratios
3	15	scale.subset1	Relative scale of subset 1, see Eq. (8)
	16	scale.subset2	Relative scale of subset 2, see Eq. (9)
	17	scale.subset3	Relative scale of subset 3, see Eq. (10)
	18	sum.revenue.subset1	Rescaled sum of revenues over subset 1, see Eq. (11)
	19	sum.revenue.subset2	Rescaled sum of revenues over subset 2, see Eq. (12)
	20	sum.revenue.subset3	Rescaled sum of revenues over subset 3, see Eq. (13)
	21	sum.cost.subset1	Rescaled sum of costs over subset 1, see Eq. (14)
	22	sum.cost.subset2	Rescaled sum of costs over subset 2, see Eq. (15)
	23	sum.cost.subset3	Rescaled sum of costs over subset 3, see Eq. (16)
	24	upper.bound	The upper bound obtained from the optimal solution to the relaxed NRP instance

5 Experimental results

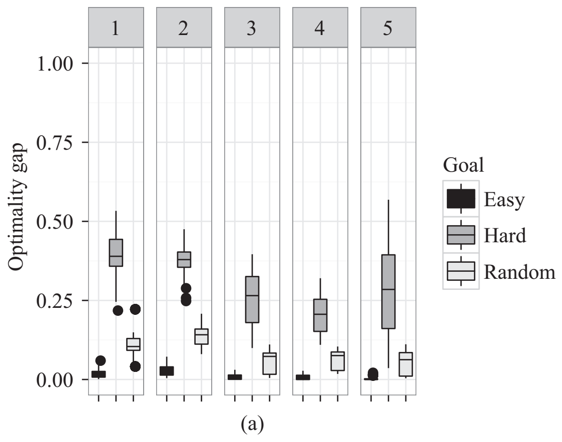
In this section, we intend to systematically investigate the RQs raised in Section 1. Through investigating the RQs, we would like to examine the possibility of obtaining instances that achieve sufficient diversity in the objective space and the feature space (RQ1 and RQ2), as well as the possibility of estimating the performance of heuristics (RQ3). In particular, for each experiment, we consider two target heuristics, i.e., GRASP and RHC, against which we evolve the instances. The reason we do not consider more complicated heuristics such as the Approximate Backbone Multilevel Algorithm (ABMA) within the evolution process is that ABMA has a relatively high computational complexity compared with the two target heuristics in this study. Meanwhile, this is also a common paradigm in the existing instance evolving studies [10, 12]. However, after the evolution process, we are interested in evaluating the performance of ABMA variants (in which GRASP and RHC are embedded) over the evolved instances, to examine whether these instances pose challenge to ABMA.

5.1 Preliminaries

All the experiments are conducted on 5 Intel i5 3.2-GHz PCs with 4 GB memory running GNU/Linux with kernel 3.16, and each PC executes single Evol-NI process. The algorithms are implemented in C++, and the predictive models are realized in R. More specifically, for the predictive models, we consider 3 regression models, i.e., the Ridge Regression (RR) [42], the Multivariate Adaptive Regression Splines (MARS) [43] and the Random Forests (RF) [44]. The reasons we adopt these models are two-fold. First, all the 3 models have been widely used in the field of algorithm performance estimation, and have been demonstrated to be effective in predicting heuristics' performance [10, 12, 45]. Second, the 3 models provide the interface to rank the features with respect to their relative importance. In particular, MARS and RF support built-in feature selection mechanisms [43, 44]. For RR, we adopt the implementation

Table 3 Summary of parameter configurations

Target heuristic	Population size		Mutation rate	
	Easy	Hard	Easy	Hard
GRASP	10	10	5	15
RHC	0.06	0.04	0.02	0.05



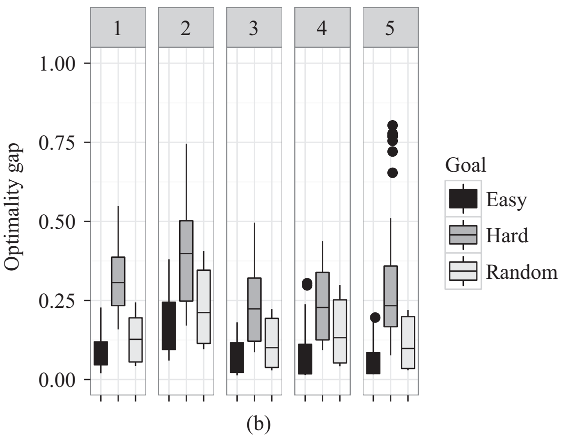


Figure 2 Performance of GRASP and RHC over evolved and random instances. (a) GRASP; (b) RHC.

of [46], which is able to adaptively select penalty parameters for features.

For the seed instances for Evol-NI, we use the benchmark instances from [19] (NRP1–NRP5, with budget ratio selected from $\{0.3,\ 0.5,\ 0.7\}$). For these benchmark instances, the values of both the revenues and the costs are randomly assigned [19]. To evolve hard NRP instances, we execute Evol-NI over each seed instance for 30 independent runs. The easy NRP instances are generated in a similar way. By this design, we obtain 450 hard NRP instances and 450 easy NRP instances for each target heuristic, respectively. As a comparison, we also generate 900 random instances. For these instances, the dependency graphs are the same as the seed instances, while the revenues and the costs follow the same distributions of the seed instances.

Besides, as described in Section 3, there are two main parameters in Evol-NI, i.e., the population size and the mutation rate. For the other parameters, such as the number of objective value evaluations, we observe that Evol-NI is not very sensitive, as long as the number is not too small. Hence, for each combination of target heuristic and evolution goal, the maximum number of objective value evaluations is set with 2000. Then, we employ an off-the-shelf tuning tool irace [47] to automatically tune the two parameters. In particular, the candidates of the two parameter are $\{5, 10, 15, 20\}$ for the population size, and $[0.01, 1]$ for the mutation rate, respectively. We adopt the default setup of irace, and obtain the recommended parameter values, which are presented in Table 3.

5.2 Answers to research questions

Investigation of RQ1: We address RQ1 by illustrating the distribution of the performance measure (optimality gap) of the target heuristics over the two sets of evolved instances and the random instances in Figure 2. In the figure, each boxplot depicts the distribution of the optimality gap achieved by GRASP and RHC over the evolved instances, with respect to the combinations of the seed instances and evolution goals. Companioned with the boxplots, in Table 4, we present the statistics of the results obtained by Evol-NI, over each seed instance. The table is organized as follows. The first two columns specify the information of the seed instances, which consist of the instance indicator (represented as ID) [19] and the budget ratio. Then, in columns 3–14, we present the statistics for the optimality gaps achieved by Evol-NI for GRASP and RHC, respectively. In particular, for the two target heuristics, associated with each evolution goal (easy and hard), we provide the average optimality gap (denoted as gap), the corresponding standard deviation (indicated as SD) with respect to the multiple independent runs, as

Table 4 Average optimality gap achieved by Evol-NI with different evolution goals

Seed instance		Easy instance (GRASP)			Hard instance (GRASP)			Easy instance (RHC)			Hard instance (RHC)		
ID	Ratio	Gap	SD	Time (s)	Gap	SD	Time (s)	Gap	SD	Time (s)	Gap	SD	Time (s)
	0.30	0.16	0.04	72.53	0.41	0.06	202.74	0.03	0.01	93.29	0.41	0.07	196.81
1	0.50	0.09	0.02	51.28	0.30	0.06	185.85	0.02	0.01	87.48	0.40	0.06	332.65
	0.70	0.04	0.01	52.83	0.21	0.04	213.27	0.01	0.00	72.42	0.37	0.06	229.44
	0.30	0.27	0.04	419.09	0.56	0.08	12033.29	0.04	0.01	905.57	0.38	0.03	14566.88
2	0.50	0.16	0.02	2809.66	0.39	0.05	32362.53	0.03	0.01	721.33	0.40	0.03	50536.51
	0.70	0.08	0.02	797.49	0.23	0.03	4918.88	0.01	0.01	4196.27	0.35	0.04	30021.20
	0.30	0.14	0.02	658.42	0.38	0.06	2407.49	0.02	0.01	608.38	0.32	0.04	17216.69
3	0.50	0.07	0.02	690.99	0.23	0.03	6787.79	0.01	0.00	708.24	0.28	0.05	718.03
	0.70	0.02	0.00	725.36	0.11	0.02	759.74	0.00	0.00	741.79	0.16	0.03	721.03
	0.30	0.16	0.06	2261.15	0.36	0.06	10615.78	0.02	0.01	2223.48	0.26	0.04	60097.61
4	0.50	0.07	0.04	3131.61	0.23	0.02	50224.76	0.00	0.01	3123.86	0.21	0.04	61732.66
	0.70	0.02	0.00	2886.16	0.12	0.01	4635.16	0.00	0.00	2842.63	0.14	0.02	4941.40
	0.30	0.10	0.03	1383.88	0.42	0.19	2700.80	0.00	0.00	1781.87	0.40	0.09	2100.34
5	0.50	0.05	0.03	1816.00	0.30	0.13	1438.72	0.00	0.00	1590.77	0.29	0.11	4119.73
	0.70	0.02	0.00	1823.50	0.13	0.05	1859.58	0.00	0.00	1890.91	0.13	0.08	1997.98

well as the average time of the evolution process in seconds (represented by time). From Figure 2 and Table 4, it is obvious that, Evol-NI is able to obtain NRP instances that are uniquely hard or easy to solve with respect to the target heuristic. For example, when we use NRP1 with a budget ratio 0.3 as the seed instance, and employ GRASP as the target heuristic, Evol-NI achieves an average gap of 0.41 (for the hard evolution goal), and 0.16 for the easy evolution goal. When we consider the efficiency perspective of Evol-NI, we observe that the time required by the evolution process varies greatly for different seed instances. For example, taking RHC as the target heuristic, evolving hard instances over NRP4 with a budget ratio 0.5 costs the longest time, i.e., more than 60000 s in average. Contrarily, evolving easy instances over NRP1 with a budget ratio 0.7 consumes only 72.42 s in average.

Furthermore, when we compare the evolved instances and the random instances, we could to some extend confirm the necessity of the hard instances. For instance, from Figure 2, we observe that the performance of GRASP and RHC over the random instances lies in between the hard and the easy instances, which is not hard enough to challenge the target heuristic. Consequently, evaluating the performance of heuristics only with the benchmark and the random instances is insufficient, in that they may not cover the range of the possible instance difficulty properly. Contrarily, the evolved instances could provide a better reference to measure the best and the worst performance of the target heuristics.

Besides, as mentioned in this study, we employ two simple heuristics rather than more sophisticated algorithms, as the target heuristics against which the instances are evolved. The reason for this design is that, the evaluation of each generated instances requires both multiple executions of target heuristics, and the execution of the reference solver. Hence, a question rises naturally that, how will metaheuristics perform over the instances evolved with respect to the embedded heuristics? To investigate this question, we consider two variants of ABMA, which embed GRASP and RHC into ABMA (denoted as ABMA_G and ABMA_R), respectively. The reasons for this framework are two-fold: (1) ABMA is an effective framework, which is able to balance the exploration and exploitation of the search process. (2) Due to the modular design, any heuristics could be embedded in ABMA.

The performances of ABMA_G and ABMA_R are evaluated similarly as in the simple heuristics' cases, i.e., over each instances, we independently execute ABMA variants for 30 independent times, with the same embedded heuristic as the one used in the evolution process. We plot the distribution of ABMA's performance with boxplots in Figure 3. From Figure 3, the following observations could be found. First, the instances that are easy for GRASP/RHC are also easy for GRASP_G/GRASP_R, which is as expected. Second, compared with GRASP and RHC, ABMA_G and ABMA_R tend to achieve lower gap, which is due to ABMA's multilevel mechanism. In particular, we observe that the improvement of ABMA_G over

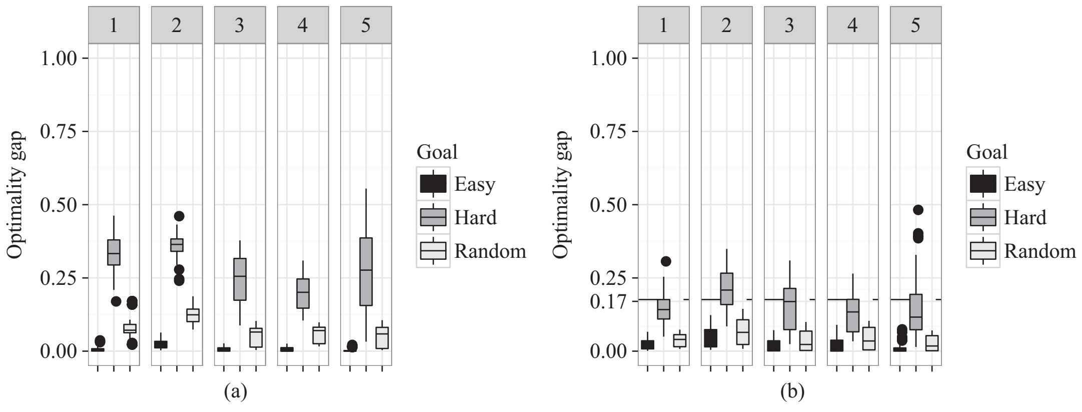


Figure 3 Performance of ABMA variants over evolved and random instances. (a) ABMA$_{\mathrm{G}}$; (b) ABMA$_{\mathrm{R}}$.

GRASP is less significant than that of ABMA$_{\mathrm{R}}$ over RHC. The reason might be that, GRASP is a greedy-like constructive heuristic, while RHC is an iterative hill climbing, which is more suitable to be embedded into other frameworks. However, we could find that the instances that are hard for simple heuristics could still pose challenges to more sophisticated algorithms. For example, in Figure 3(b), for the instances evolved with NRP1 as the seed instance, the third quartile of gap for ABMA$_{\mathrm{R}}$ is up to 0.1762, which indicates that over 25% of evolved instances, the gap achieved by ABMA$_{\mathrm{R}}$ is above or equal to 0.1762. This observation to some extent demonstrates that, even only evolving from simple heuristics, the obtained instances may pose challenge to the more sophisticated metaheuristics.

Answer to RQ1: Based on these observations, we could answer RQ1 positively, i.e., Evol-NI is able to evolve NRP instances that are hard/easy to solve, respectively.

Investigation of RQ2: Within the previous RQ, we have demonstrated that Evol-NI is able to evolve NRP instances that are hard or easy to solve, respectively. Now we intend to examine whether the problem-specific features of evolved instances exhibit different characteristics.

For all the 24 features, we compare the two sets of feature values with Mann-Whitney-Wilcoxon test. We specify the null hypothesis to be that there do not exist significant differences between the values of the feature for the hard and easy instances. Considering all the features proposed, the detailed results of the hypothesis testing are presented in Table 5. From the table, we observe significant difference over the majority of the features (22 and 21 for GRASP and RHC, respectively, when considering the 95% confidence level), which implies that the hard and easy instances obtained by Evol-NI could achieve good diversity in the feature space. When we consider the 90% confidence level, significant differences could be detected over all the features, except for 2 features for RHC (scale.subset3 and sum.cost.subset3). Hence, these features might be potentially helpful in the subsequent regression tasks.

Then, in Figures 4 and 5, we illustrate the distributions of the typical feature values of the evolved instances, for the two target heuristics[4]. Specifically, Figures 4 and 5 consist of 4 subfigures, which correspond to 4 features (avg.node.between, cov.revenue, scale.subset3, and sum.revenue.subset1). From Figures 4 and 5, we could observe significant differences over 3 out of 4 subfigures, which are evolved with respect to different goals. For example, considering the cov.revenue feature for both GRASP and RHC, the hard instances tend to favor smaller feature value, compared with their easy counterparts.

However, considering the scale.subset3 feature, we observe that the distributions for the hard and the easy instances exhibit similar patterns, i.e., the majority of the feature values are very small. The reason might be as follow. As mentioned in Section 4, the subset $[j_{\max}, n]$ indicates the customers that are associated with small avenue-to-cost ratio, who are not selected by the optimal solution to the relaxed NRP. Meanwhile, since the integer constraint is removed from the relaxed NRP formulations, there could be more non-zero $\hat{y}_j$ values compared with the original NRP, because $\hat{y}_j$ could be fractional values for $1 \leqslant j \leqslant n$. Hence, $\hat{j}_{\max} = \max\{\min\{1 < j < n | \hat{y}_j = 0\}, \max\{1 < j < n | \hat{y}_j > 0\}\}$ tends to be close to n. Consequently, in the third subfigures of Figures 4 and 5, we observe that for both the easy and the hard

4) More detailed version is available at http://oscar-lab.org/people/~zren/evol-ni.

Table 5 Comparison results between feature values of easy/hard instances using the Mann-Whitney-Wilcoxon test

Feature	p-value (GRASP)	p-value (RHC)	Feature	p-value (GRASP)	p-value (RHC)
(1) avg.node.between	< 0.0001	< 0.0001	(13) cov.cost	< 0.0001	< 0.0001
(2) sd.node.between	< 0.0001	< 0.0001	(14) cov.ratio	< 0.0001	< 0.0001
(3) avg.in.degree	< 0.0001	< 0.0001	(15) scale.subset1	< 0.0001	< 0.0001
(4) avg.out.degree	< 0.0001	< 0.0001	(16) scale.subset2	< 0.0001	< 0.0001
(5) sd.in.degree	< 0.0001	< 0.0001	(17) scale.subset3	0.0569	0.1548
(6) sd.out.degree	0.0016	0.0558	(18) sum.revenue.subset1	< 0.0001	< 0.0001
(7) budget.bound	0.0188	< 0.0001	(19) sum.revenue.subset2	< 0.0001	< 0.0001
(8) cor.revenue.cost	< 0.0001	< 0.0001	(20) sum.revenue.subset3	0.0007	< 0.0001
(9) avg.revenue	< 0.0001	< 0.0001	(21) sum.cost.subset1	< 0.0001	< 0.0001
(10) avg.cost	< 0.0001	< 0.0001	(22) sum.cost.subset2	< 0.0001	< 0.0001
(11) avg.ratio	< 0.0001	< 0.0001	(23) sum.cost.subset3	0.0627	0.1995
(12) cov.revenue	< 0.0001	< 0.0001	(24) upper.bound	0.0002	0.0021

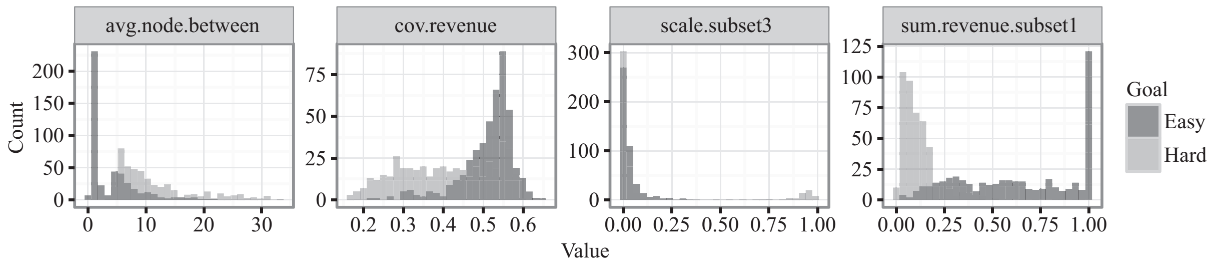


Figure 4 Histograms for typical feature values, targeting GRASP.

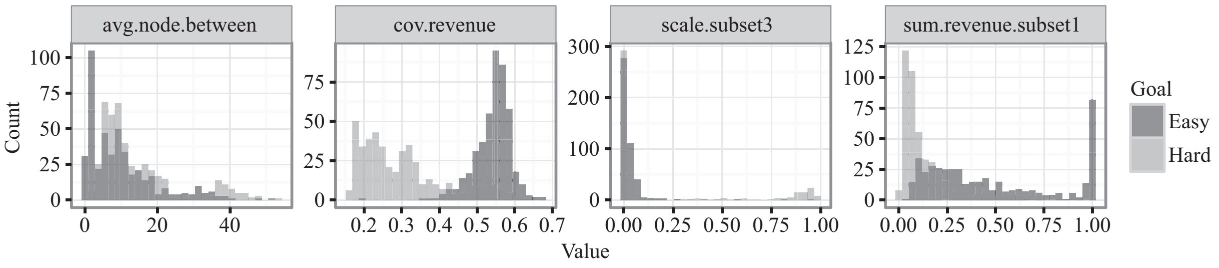


Figure 5 Histograms for typical feature values, targeting RHC.

instances, the feature values of scale.subset3 tend to be small.

Answer to RQ2: We could not give a positive answer to RQ2, i.e., not all the features exhibit different characteristics. However, we observe significant difference over the majority of the features, when we compare the evolved hard NRP instances and their easy counterparts. Furthermore, for the typical features, the evolution trends for the hard and easy instances exhibit significantly different patterns, which to some extend demonstrates the diversification ability of Evol-NI.

Investigation of RQ3: After evaluating the evolved instances from the objective space (RQ1) and the feature space (RQ2), we are interested in the possibility of estimating the solution quality (measured by the optimality gap in Eq. (6)) the target heuristics could achieve, with the proposed features. To achieve this, we employ 3 regression models, i.e., RR, MARS, and RF. We adopt the ten-fold cross-validation to evaluate the performance of the 3 regression models. For the data set, we consider both the evolved instances and the random instances. The reason for incorporating the random instances is that, these instances are of intermediate hardness, which might provide better diversity in the objective space (see Figure 2). Figure 6 presents the ten-fold cross-validation results of the predictive models. In Figure 6, the prediction results for each predictive model are associated with a subfigure, in which the x-axis and the y-axis indicate the optimality gap achieved by the target heuristics and the predicted optimality gap

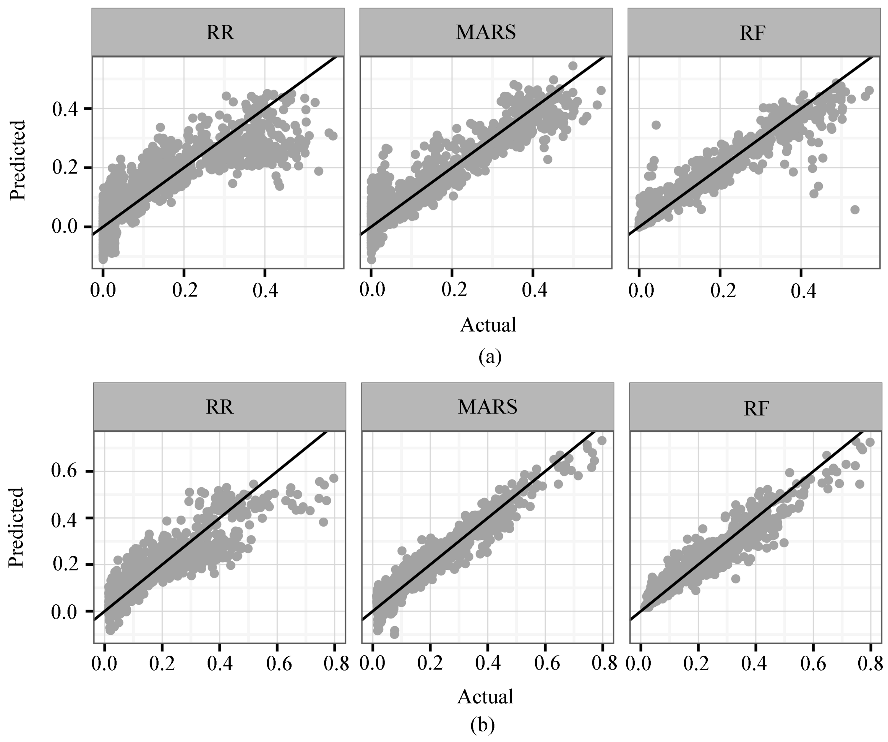


Figure 6 Ten-fold prediction results over test instances, for GRASP and RHC. (a) GRASP; (b) RHC.

over the test instances, respectively. Each point indicates the results over one or more instances. The reference line $y = x$ is also plotted in Figure 6. Hence, the closer a point is to the reference line, the more precise the prediction will be. For example, in Figure 6, we observe that for GRASP, MARS is able to achieve the best accuracy, followed by RF. Furthermore, the average Root Mean Squared Error (RMSE) obtained by the 3 models are 0.0057 (RR), 0.0030 (MARS), and 0.0033 (RF). Similarly, for GRASP, the RMSE achieved are 0.0064 (RR), 0.0041 (MARS), and 0.0031 (RF), respectively.

Then, we examine the possibility of predicting the performance of the two ABMA variants. The experiment setup is similar as in the experiment for GRASP and RHC, i.e., we employ RR, MARS, and RF, and adopt ten-fold cross-validation to evaluate these models. Also, the instances include the evolved instance, as well as the random instances. In Figure 7, we present the ten-fold cross-validation results for the two variants of ABMA. Figure 7 is organized similarly as Figure 6. From Figure 7, we can see that all the trained models are able to predict the performance of the ABMA variants properly. For $\mathrm{ABMA_G}$, the average ten-fold RMSE for the three models are 0.0033 (RR), 0.0024 (MARS), and 0.0020 (RF). For $\mathrm{ABMA_R}$, the average ten-fold RMSE values are 0.0061 (RR), 0.0041 (MARS), and 0.0030 (RF).

To gain more insights into the relationships between heuristic performance and the problem-specific features, we present the importance values of the features for different regression models in Table 6. In the figure, the importance indicators for the 3 models are the loess R^2 statistics of each features (for RR) [48], the change in the Residual Sums of Squares (RSS) as features are added (for MARS) [49], and the total decrease in node impurities of each feature (for RF) [50], respectively. For each predictive model, the larger the indicator value is, the more important the corresponding feature will be. For instance, for the RF model, the relative importance of each feature is captured by the total decrease in node impurities (NodeImpurity) of each feature, which is measured by calculating how much RSS increases when that feature is randomly permuted [44]. Hence, higher indicator values imply higher feature importance. From the table, we observe that the features extracted from the relaxed NRP instances (such as sum.revenue.subset1 and scale.subset1) are among the top of the list under most scenarios, which to some extend implies the usefulness of these features. As an example, when using RF to predict the performance of GRASP, the sum.revenue.subset1 feature is the most importance feature,

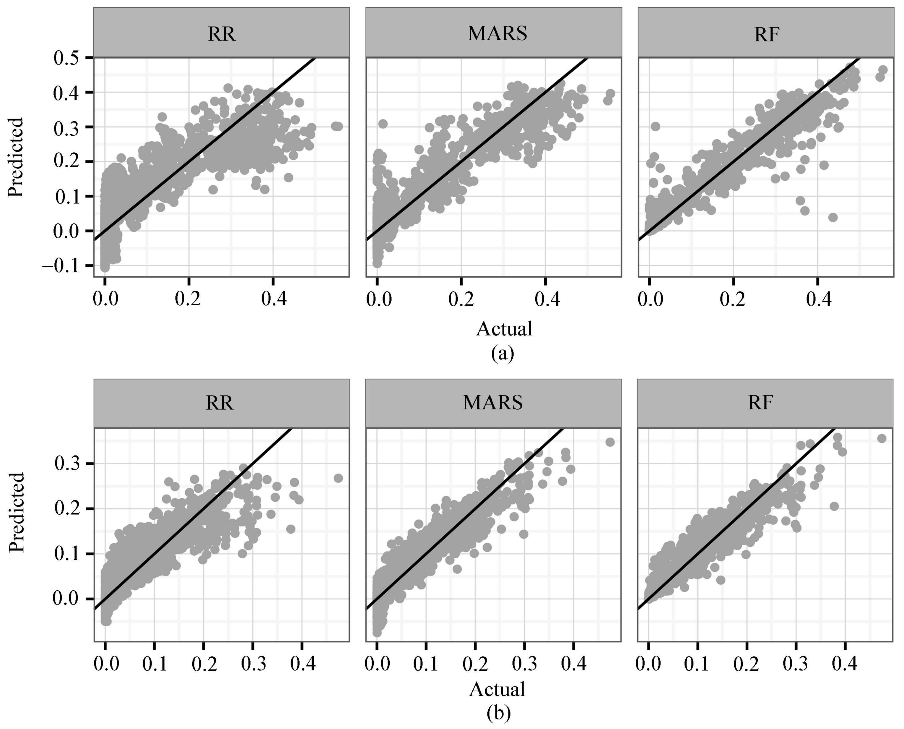


Figure 7 Ten-fold prediction results over test instances, for ABMA_G and ABMA_R. (a) ABMA_G; (b) ABMA_R.

Table 6 Top 5 feature importance values for predictive models

Target heuristic	RR (loess R^2)		MARS (RSS)		RF (NodeImpurity)	
	Feature	Imp	Feature	Imp	Feature	Imp
GRASP	(18) sum.revenue.subset1	0.66	(18) sum.revenue.subset1	24.6	(18) sum.revenue.subset1	9.02
	(21) sum.cost.subset1	0.63	(10) avg.cost	7.7	(21) sum.cost.subset1	4.91
	(15) scale.subset1	0.62	(7) budget.bound	6.4	(7) budget.bound	1.99
	(10) avg.cost	0.46	(1) avg.node.between	4.2	(15) scale.subset1	1.89
	(19) sum.revenue.subset2	0.41	(3) avg.in.degree	3.6	(10) avg.cost	1.45
RHC	(18) sum.revenue.subset1	0.57	(18) sum.revenue.subset1	23.5	(21) sum.cost.subset1	6.59
	(21) sum.cost.subset1	0.55	(7) budget.bound	6.8	(18) sum.revenue.subset1	3.68
	(15) scale.subset1	0.55	(24) upper.bound	6.8	(20) sum.revenue.subset3	2.46
	(23) sum.cost.subset3	0.41	(16) scale.subset2	6.3	(7) budget.bound	1.81
	(17) scale.subset3	0.38	(13) cov.cost	6.2	(24) upper.bound	1.37
ABMA_G	(18) sum.revenue.subset1	0.67	(18) sum.revenue.subset1	20.9	(12) cov.revenue	2.74
	(21) sum.cost.subset1	0.64	(10) avg.cost	5.7	(21) sum.cost.subset1	2.14
	(15) scale.subset1	0.63	(7) budget.bound	4.4	(24) upper.bound	0.83
	(10) avg.cost	0.47	(24) upper.bound	3.2	(7) budget.bound	0.71
	(19) sum.revenue.subset2	0.43	(1) avg.node.between	2.3	(18) sum.revenue.subset1	0.43
ABMA_R	(18) sum.revenue.subset1	0.54	(12) cov.revenue	7.3	(18) sum.revenue.subset1	8.04
	(21) sum.cost.subset1	0.51	(23) sum.cost.subset3	3.3	(21) sum.cost.subset1	3.98
	(15) scale.subset1	0.48	(24) upper.bound	2.3	(7) budget.bound	1.73
	(12) cov.revenue	0.47	(7) budget.bound	1.7	(10) avg.cost	1.61
	(9) avg.revenue	0.42	(10) avg.cost	0.6	(15) scale.subset1	1.47

in that randomly permuting this feature in the RF model will lead to an increase of the RSS by 9.02, which is larger than all the other features.

Answer to RQ3: To this end, RQ3 could also be answered positively. We could predict the performance of the target heuristic based on the problem-specific features properly.

As a brief summary, in this section, we present the experimental results and the discussions, considering

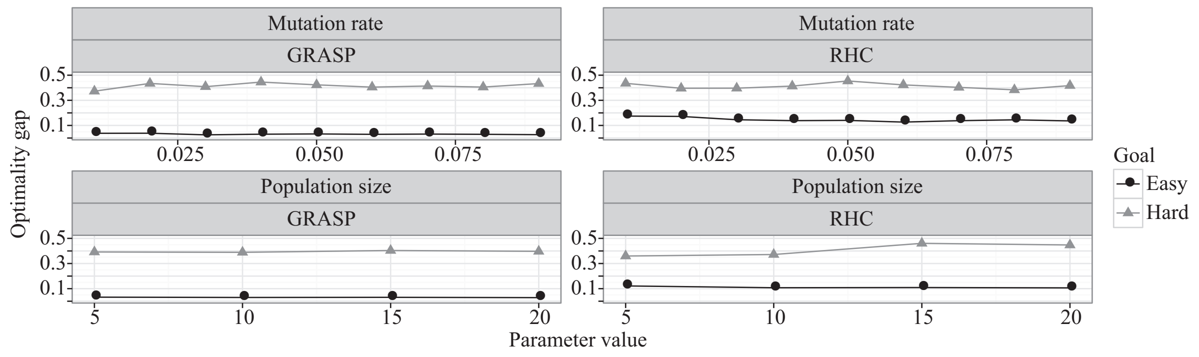


Figure 8 Impact of parameters on Evol-NI.

two heuristics. By investigating the research questions, we confirm the usefulness of the problem-specific features in the NRP domain. These features could be used in estimating the solution quality that target heuristics could reach.

6 Threats to validity

There are several objections a critical reader might raise to the evaluation presented in this study, among which the following three threats deserve special attention.

First, in this study, we apply an off-the-shelf parameter configuration package to conduct the parameter tuning task. Hence, a question that rises naturally is that, is the proposed algorithm sensitive to the parameters? To investigate this issue, we carry out the sensitivity analysis over a representative instance, to examine whether Evol-NI is sensitive to the parameters. In Figure 8, we present the results of the sensitivity analysis. The experiment is conducted over NRP1 with a budget ratio of 0.3 as follows. First, the parameters are assigned with the output of irace. Then, for each parameter, we enumerate several values in its feasible range, meanwhile keep the other parameter assigned with respect to irace. For each set of modified parameter configuration, we execute Evol-NI over the instance for 30 times, and plot the average optimality gap against the parameter value, to investigate Evol-NI's behavior when perturbing the parameter values. Through this experiment scheme, we intend to evaluate the quality, as well as the robustness of the offline tuning process. From Figure 8, we observe that irace could achieve effective parameters that lead to promising results. For example, considering the population size parameter, when evolving hard instances for RHC, Evol-NI favors larger population size, which is consistent with the output of irace. Meanwhile, the proposed evolutionary algorithm is not very sensitive to the parameters. For example, when evolving hard instances for the two target heuristics, the optimality gap is always above 0.3, which to some extend implies the robustness of the proposed algorithm.

Second, one might argue the choice of the hardness measurement. In this study, we evaluate the hardness of a given NRP instance for the target heuristic using the optimality gap achieved by the heuristic. Indeed, there are other options such as runtime of the target heuristic. The reason we adopt optimality gap rather than runtime is that, despite the evaluation overhead, the optimality gap corresponds directly with the decision making task. Consequently, the decision makers may be more interested in this performance metric. To mitigate this threat, inspired by [51], a feasible approach is to extend the evolution goal from single-objective to multi-objective. In particular, instead of considering the hardest/easiest instances, Pareto dominance relationships between two instances could be defined, i.e., one instance is considered to dominate another instance if the target heuristic is more time consuming and achieves larger optimality gap over the other instance. Hence, multi-objective evolutionary algorithms could be applied to obtain instances that are both computationally difficult and time consuming to solve.

Third, following the above issue, the scalability of the instances involved in the experiments also poses a potential threat to validity. In this study, we consider the instances from [19], in which the number of requirements ranges from 100 to 1000. The reason is that during the instance evolution, we have to solve the generated instances to optimality, so that the performance of the target heuristic could be evaluated objectively. If we would investigate the evolution of large scale instances, the computational overhead is intolerable. Meanwhile, in the evolutionary community, the surrogate model [52, 53] is a popular technique to accelerate the evolution process. The idea of the surrogate model is to build an approximate evaluation model to evaluate the individuals of the population, which is usually more efficient than the original problem. Inspired by such techniques, in this study, one possible solution is to employ an approximate model (e.g., relaxing the problem by removing part of the integer constraints) during the beginning iterations of instance evolution, and transfer the model gradually towards the original problem.

7 Conclusion and future work

In this paper, we concentrate on evolving hard and easy software engineering problem instances. Taking the Next Release Problem (NRP) as a case study, this study has contributed to the understanding of problem hardness in Search-Based Software Engineering (SBSE). The contributions of this paper could be summarized as follows. First, given the target heuristic (GRASP and RHC in this study), we develop an evolutionary algorithm Evol-NI to generate hard and easy NRP instances, respectively. Second, we propose a feature-based method to capture the characteristics of software engineering problem instances, which extends the work of understanding combinatorial problem hardness. Third, we investigate the possibility of estimating the performance of the target heuristic.

Despite the promising results the proposed framework achieved in this study, there are several limitations that require much future work. First, in this study, we only consider the relative optimality gap as the hardness measurement. Investigating the behavior of the evolutionary algorithm in multi-objective context is an interesting direction. Possible candidate measurements include absolute optimality gap, runtime, etc. Second, since the target heuristics might be parameterized, an interesting direction is to evolve instances with different combinations of parameters, and train predictive models to adaptively set the parameter values [54]. Third, in this study, the performance comparison between different target heuristics is not investigated. In the future, we would consider extending the study into related directions, such as algorithm selection [8, 33] and population based algorithm portfolio [55]. Finally, in this study, we consider the NRP as the case study, which is a typical problem in the field of software engineering. Meanwhile, since many problems in the software engineering fields could be described as combinatorial optimization problems. Hence, ideas in this study could be easily transferred into new problem domains. For example, since many software engineering problems have similar encoding schemes as the NRP, such as release planning [20], software upgradability [21], etc.

Acknowledgements This work was supported in part by National Program on Key Basic Research Project (Grant No. 2013CB035906), New Century Excellent Talents in University (Grant No. NCET-13-0073), National Natural Science Foundation of China (Grant Nos. 61370144, 61403057), and Fundamental Research Funds for the Central Universities (Grant Nos. DUT15TD37, DUT16RC(4)62).

Conflict of interest The authors declare that they have no conflict of interest.

References

1 Gong W, Cai Z, Liang D. Adaptive ranking mutation operator based differential evolution for constrained optimization. IEEE Trans Cybern, 2015, 45: 716–727
2 Gong W, Cai Z. Differential evolution with ranking-based mutation operators. IEEE Trans Cybern, 2013, 43: 2066–2081
3 Luo C, Cai S, Su K, et al. Clause states based configuration checking in local search for satisfiability. IEEE Trans Cybern, 2015, 45: 1014–1027
4 Chen W N, Zhang J. Ant colony optimization for software project scheduling and staffing with an event-based scheduler. IEEE Trans Softw Eng, 2013, 39: 1–17
5 Haeri S, Trajkovic L. Intelligent deflection routing in buffer-less networks. IEEE Trans Cybern, 2015, 45: 316–327
6 Tang K, Yang P, Yao X. Negatively correlated cooperative search. arXiv:1504.04914

7 Jiang H, Xuan J, Ren Z. Approximate backbone based multilevel algorithm for next release problem. In: Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation, Portland, 2010. 1333–1340
8 Smith-Miles K A. Cross-disciplinary perspectives on meta-learning for algorithm selection. ACM Comput Surv, 2008, 41: 6
9 Smith-Miles K A, van Hemert J I. Discovering the suitability of optimisation algorithms by learning from evolved instances. Ann Math Artif Intell, 2011, 61: 87–104
10 Mersmann O, Bischl B, Trautmann H, et al. A novel feature-based approach to characterize algorithm performance for the traveling salesperson problem. Ann Math Artif Intell, 2013, 69: 151–182
11 Smith-Miles K A. Towards insightful algorithm selection for optimisation using meta-learning concepts. In: Prcoeedings of International Joint Conference on Neural Networks, Hong Kong, 2008. 4118–4124
12 van Hemert J I. Evolving combinatorial problem instances that are difficult to solve. Evol Comput, 2006, 14: 433–462
13 Vilalta R, Drissi Y. A perspective view and survey of meta-learning. Artif Intell Rev, 2002, 18: 77–95
14 Oentaryo R J, Handoko S D, Lau H C. Algorithm selection via ranking. In: Proceedings of the 29th AAAI Conference on Artificial Intelligence, Austin Texas, 2015. 1826–1832
15 van Hemert J I. Property analysis of symmetric travelling salesman problem instances acquired through evolution. In: Evolutionary Computation in Combinatorial Optimization. Berlin: Springer, 2005. 122–131
16 Xuan J, Jiang H, Ren Z, et al. Solving the large scale next release problem with a backbone-based multilevel algorithm. IEEE Trans Softw Eng, 2012, 38: 1195–1212
17 Nie L, Jiang H, Ren Z, et al. Query expansion based on crowd knowledge for code search. IEEE Trans Serv Comput, 2016, 9: 771–783
18 Xuan J F, Martinez M, DeMarco F, et al. Nopol: automatic repair of conditional statement bugs in java programs. IEEE Trans Softw Eng, in press. doi: 10.1109/TSE.2016.2560811
19 Bagnall A J, Rayward-Smith V J, Whittley I M. The next release problem. Inf Softw Tech, 2001, 43: 883–890
20 Greer D, Ruhe G. Software release planning: an evolutionary and iterative approach. Inf Softw Tech, 2004, 46: 243–253
21 Ignatiev A, Janota M, Marques-Silva J. Towards efficient optimization in package management systems. In: Proceedings of the 36th International Conference on Software Engineering, Hyderabad, 2014. 745–755
22 Sun J, Zhang Q, Yao X. Meta-heuristic combining prior online and offline information for the quadratic assignment problem. IEEE Trans Cybern, 2014, 44: 429–444
23 Ren Z, Jiang H, Xuan J, et al. New insights into diversification of hyper-heuristics. IEEE Trans Cybern, 2014, 44: 1747–1761
24 Gao W F, Liu S Y, Huang L L. A novel artificial bee colony algorithm based on modified search equation and orthogonal learning. IEEE Trans Cybern, 2013, 43: 1011–1024
25 Ren Z, Zhang A, Wen C, et al. A scatter learning particle swarm optimization algorithm for multimodal problems. IEEE Trans Cybern, 2014, 44: 1127–1140
26 He J, Zhang J Y, Xuan J F, et al. A hybrid ACO algorithm for the next release problem. In: Proceedings of the 2nd International Conference on Software Engineering and Data Mining, Chengdu, 2010. 166–171
27 He J, Ren Z L, Li X C, et al. Transformed search based software engineering: a new paradigm of sbse. In: Search-Based Software Engineering. Berlin: Springer, 2015. 203–218
28 van Hemert J I. Evolving binary constraint satisfaction problem instances that are difficult to solve. In: Proceedings of Congress on Evolutionary Computation, Canberra, 2003. 1267–1273
29 Julstrom B A. Evolving heuristically difficult instances of combinatorial problems. In: Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation, Montreal, 2009. 279–286
30 Smith-Miles K A, Lopes L. Generalising algorithm performance in instance space: a timetabling case study. In: Proceedings of the 5th International Conference on Learning and Intelligent Optimization, Rome, 2011. 524–538
31 Toledo-Suárez C D, Valenzuela-Rendón M, Terashima-Marín H, et al. On the relativity in the assessment of blind optimization algorithms and the problem-algorithm coevolution. In: Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation. New York: ACM, 2007. 1436–1443
32 Hall N G, Posner M E. Performance prediction and preselection for optimization and heuristic solution procedures. Oper Res, 2007, 55: 703–716
33 Leyton-Brown K, Nudelman E, Shoham Y. Empirical hardness models: methodology and a case study on combinatorial auctions. J ACM, 2009, 56: 22
34 Smith-Miles K, Wreford B, Lopes L, et al. Predicting metaheuristic performance on graph coloring problems using data mining. In: Hybrid Metaheuristics. Berlin: Springer, 2013. 417–432
35 Nallaperuma S, Wagner M, Neumann F, et al. A feature-based comparison of local search and the christofides algorithm for the travelling salesperson problem. In: Proceedings of the 12th Workshop on Foundations of Genetic Algorithms XII. New York: ACM, 2013. 147–160
36 Smith-Miles K, Baatar D, Wreford B, et al. Towards objective measures of algorithm performance across instance space. Comput Oper Res, 2014, 45: 12–24
37 Ruhe G, Ngo-The A. Optimized resource allocation for software release planning. IEEE Trans Softw Eng, 2009, 35: 109–123
38 Wen L, Dromey R, Kirk D. Software engineering and scale-free networks. IEEE Trans Syst Man Cybern, 2009, 39: 845–854
39 Pisinger D. Core problems in knapsack algorithms. Oper Res, 1999, 47: 570–575

40 Pisinger D. Where are the hard knapsack problems? Comput Oper Res, 2005, 32: 2271–2284

41 Balas E, Zemel E. An algorithm for large zero-one knapsack problems. Oper Res, 1980, 28: 1130–1154

42 Cule E. Ridge: Ridge Regression With Automatic Selection of The Penalty Parameter, 2014. R package version 2.1-3. https://cran.r-project.org/src/contrib/Archive/ridge/

43 Friedman J H. Multivariate adaptive regression splines1. Ann Stat, 1991, 19: 1–141

44 Breiman L. Random forests. Mach Learn, 2001, 45: 5–32

45 Hutter F, Xu L, Hoos H H, et al. Algorithm runtime prediction: methods & evaluation. Artif Intell, 2014, 206: 79–111

46 Cule E, de Iorio M. Ridge regression in prediction problems: automatic choice of the ridge parameter. Genetic Epidemiol, 2013, 37: 704–714

47 López-Ibánez M, Dubois-Lacoste J, Stützle T, et al. The Irace Package, Iterated Race for Automatic Algorithm Configuration. Technical Report Technical Report TR/IRIDIA/2011-004, IRIDIA, Université Libre de Bruxelles, Belgium, 2011

48 Kuhn M. Caret: Classification and Regression Training, 2012. R package version 5.15-044. https://cran.r-project.org/src/contrib/Archive/caret/

49 Milborrow S. Earth: Multivariate Adaptive Regression Splines, 2015. R package version 4.4.2. https://cran.r-project.org/src/contrib/Archive/earth/

50 Liaw A, Wiener M. Classification and regression by randomforest. R News, 2002, 2: 18–22

51 Jiang H, Sun W, Ren Z, et al. Evolving hard and easy traveling salesman problem instances: a multi-objective approach. In: Simulated Evolution and Learning. Berlin: Springer, 2014. 216–227

52 Sun X, Gong D, Jin Y, et al. A new surrogate-assisted interactive genetic algorithm with weighted semisupervised learning. IEEE Trans Cybern, 2013, 43: 685–698

53 Gong W, Zhou A, Cai Z. A multioperator search strategy based on cheap surrogate models for evolutionary optimization. IEEE Trans Evol Comput, 2015, 19: 746–758

54 Nallaperuma S, Wagner M, Neumann F. Parameter prediction based on features of evolved instances for ant colony optimization and the traveling salesperson problem. In: Parallel Problem Solving from Nature –PPSN XIII. Berlin: Springer, 2014. 100–109

55 Tang K, Peng F, Chen G, et al. Population-based algorithm portfolios with automated constituent algorithms selection. Inf Sci, 2014, 279: 94–104