Automatically Exploiting Implicit Design Knowledge When Solving the Class Responsibility Assignment Problem

Yongrui Xu
State Key Lab of Software Engineering, School of Computer Science, Wuhan University, China
xuyongrui@whu.edu.cn

Peng Liang*
State Key Lab of Software Engineering, School of Computer Science, Wuhan University, China
liangp@whu.edu.cn

Muhammad Ali Babar
School of Computer Science, The University of Adelaide, Australia
ali.babar@adelaide.edu.au

Abstract—Assigning responsibilities to classes is not only vital during initial software analysis/design phases in object-oriented analysis and design (OOAD), but also during maintenance and evolution phases, when new responsibilities have to be assigned to classes or existing responsibilities have to be changed. Class Responsibility Assignment (CRA) is one of the most complex tasks in OOAD as it heavily relies on designers’ judgment and implicit design knowledge (DK) of design problems. Since CRA is highly dependent on the successful use of implicit DK, (semi-)automated approaches that help designers to assign responsibilities to classes should make implicit DK explicit and exploit the DK effectively. In this paper, we propose a learning based approach for the Class Responsibility Assignment (CRA) problem. A learning mechanism is introduced into Genetic Algorithm (GA) to extract the implicit DK about which responsibilities have a high probability to be assigned to the same class, and then the extracted DK is employed automatically to improve the design quality of the generated solutions. The proposed approach has been evaluated through an experimental study with three cases. By comparing the solutions obtained from the proposed approach and the existing approaches, the proposed approach can significantly improve the design quality of the generated solutions to the CRA problem, and the generated solutions by the proposed approach are more likely to be accepted by developers from the practical aspects.

Keywords—Search-Based Software Engineering; Class Responsibility Assignment; Object-Oriented Analysis and Design

I. INTRODUCTION

In object-oriented analysis and design (OOAD), assigning responsibilities to classes is a vital task, which usually has a significant impact on the overall design of an application [1]. This task is vital not only during initial analysis/design phases for object-oriented software development, but also maintenance and evolution phases, when new responsibilities have to be assigned to new classes or existing responsibilities have to be moved to other classes. The main goal of Class Responsibility Assignment (CRA) is to find an optimal assignment of responsibilities which are presented in terms of methods and attributes to classes [2][3]. It is challenging for designers, especially novice designers, to perform this task without appropriate (semi-) automated support as it “tends to be a challenging skill to master with many ‘degrees of freedom’ or alternatives” [1]. Several (semi-) automated approaches have been developed to support CRA, providing different levels of automated support to novice (as well as experienced) designers to achieve good responsibilities assignment [2][3][4][5][6].

Most (semi-) automated approaches treat the CRA problem as an optimization problem because there is a large space of candidate solutions for a design problem [2][4][5][6]. Simons and his colleagues [5] analyzed the cardinality of the search space for the CRA problem and concluded that “the number of possible designs for even small class designs is beyond human comprehension”. They gave an example to show an exponential growth of the search space with respect to the number of responsibilities that “even for eight attributes and eight methods (i.e., 16 responsibilities) allocated into five classes results in a search space of cardinality 132,300,000”. Clarke and colleagues outlined four characteristics of the problems that may benefit from a search-based approach [7], and Bowman and colleagues explained why the CRA problem exhibits such characteristics in [2]. It indicates that the CRA problem is well-suited to be considered as a search problem and addressed by the Search-Based Software Engineering (SBSE) techniques [8][9].

SBSE seeks to reformulate software engineering (SE) problems as search-based optimization problems, in which optimal or near-optimal solutions are sought by a search algorithm in a search space of candidate solutions, guided by a fitness function that distinguishes between better and worse solutions [8]. An SBSE approach to the CRA problem employs a search algorithm (e.g., genetic algorithm (GA)) to find optimized solutions, and harnesses a fitness function based on design properties (e.g., coupling, cohesion, and complexity) to evaluate the design quality (i.e., modifiability in this work) of candidate solutions [2][4].

However, the essential difficulty is how to exploit implicit design knowledge (DK) when addressing the CRA problem because each design problem contains complex and implicit DK which should be analyzed thoroughly by designers [10]. If (semi-) automated approaches generate solutions without considering exploiting the specific DK with regard to design problem instances, the design quality of the generated solutions may not be satisfied, and these solutions are less likely to be accepted by developers from the practical aspects.

* Corresponding author
This work is funded by the NSFC under Grant No. 61472286.
In manual design approaches, Domain-Driven Design (DDD) is one of the successful approaches that help designers tackling complexity when analyzing and designing software systems [11]. One important reason that DDD works for OOAD is that DDD emphasizes making implicit (domain-) concepts explicit, which manually extracts and employs implicit (domain-) design knowledge for the design problems [11]. Consequently, there should be a similar mechanism introduced in the (semi-) automated search-based approaches to extract implicit DK and then employ the extracted DK to improve the design quality of candidate solutions for the CRA problem, which is the main contribution of this work.

To be more specific, we propose a learning mechanism based on DK emergence to support automatically extracting and employing implicit DK for the CRA problem. The widely used GA is extended with a learning operator to implement the learning mechanism. An association rule mining algorithm from data mining is employed in the learning operator to extract the implicit DK about which responsibilities have a high probability to be assigned to the same class for design problems. The extracted DK can be subsequently employed to improve the design quality of the generated solutions.

We introduced the idea of leveraging learning mechanism for CRA in a position paper [12]. This paper reports fully developed approach for exploiting implicit DK when solving the CRA problem and the experiments executed to evaluate our approach. Note that our work focuses exclusively on assigning responsibilities to domain classes [11]. The output of the proposed approach is often referred to as analysis or domain models. Domain models are solution-independent and platform-independent descriptions of a problem domain produced in the early analysis phase of OOAD [1].

The rest of this paper is organized as follows. Section II introduces the background of this work. Section III discusses the concept DK used in this work, and presents the basic idea of the proposed approach and explains how to exploit implicit DK in the approach for solving the CRA problem. Section IV describes the research questions with the evaluation plan of the proposed approach for the CRA problem. Section V presents the evaluation results and the analysis based on the results. We discuss the related work in Section VI and conclude this work with future directions in Section VII.

II. BACKGROUND

In this section, we first introduce the CRA problem (Section II.A). We then introduce Genetic Algorithm (GA) [13], which is the most widely used search algorithm in SBSE research (Section II.B).

A. CRA Problem

The CRA problem is encountered in the early phases of OOAD and has a great impact on the overall design of software [1]. The CRA problem is defined as “deciding where responsibilities, under the form of class operations (as well as the attributes they manipulate), belong and how objects should interact (by using those operations)” [2]. A formal definition to describe the CRA problem is given below:

Consider \( S = (C, Res, Dep) \) as being the search space of design solutions for a design problem, in which \( Res \) is the set of \( R \) responsibilities for the design problem and \( Dep \) is the set of dependencies between responsibilities in \( Res \). The goal of the CRA problem is to assign all the responsibilities in \( Res \) into \( C \) classes \( \{ \text{Class1, Class2, ..., ClassC} \} \) to maximize the cohesion, while minimizing the coupling and complexity of a solution in \( S \). For each \( \text{Class} \), the set of responsibilities it contains is defined as \( Rs(\text{Class}) \). The relationships between classes and their containing responsibilities should satisfy the following constraints:

1) \( Rs(\text{Class}_i) \neq \emptyset, i = 1, 2, ..., C \)
2) \( \bigcup_{i=1}^C Rs(\text{Class}_i) = Res \)
3) \( Rs(\text{Class}_i) \cap Rs(\text{Class}_j) = \emptyset \),
   \( i, j = 1, 2, ..., C \) and \( i \neq j \)
4) \( 1 \leq C \leq R \)

Since the search space \( S \) is defined by all possible allocations of the responsibilities (i.e., methods and attributes) to arbitrary number of classes, it leads to a very large search space even for a small system [5]. As mentioned before, the CRA problem is well suited for being treated as an optimization problem, and applying a search algorithm (e.g., GA) in finding optimized solutions.

B. Genetic Algorithm

In the field of artificial intelligence, GA is a widely used search heuristic that mimics the process of natural selection. This heuristic is routinely used to generate solutions for optimization and search problems with techniques inspired by natural evolution, such as inheritance, mutation, selection, and crossover [13].

In GA, each candidate solution is represented as a vector of components referred to as individuals or chromosomes. The set of candidate solutions currently under consideration is referred to as the current population, with each successive population considered referred to as a generation [13]. For each population, certain individuals are selected with selection operator to go forward to the stages of crossover and mutation with crossover and mutation operators. Then individuals with the best fitness are reinserted into the next generation. When the termination condition satisfies, the optimal candidate solutions are acquired from the individuals in the population [13][14].

III. APPROACH

In this section, we first describe DK emergence for CRA (Section III.A), which is the theoretical basis of the proposed approach. To support the DK emergence, we then extend GA with a learning operator that introduces an extra learning mechanism for exploiting implicit DK (Section III.B). The concrete implementation of the learning operator in the extended GA for CRA can be found in Section III.C.

A. Design Knowledge Emergence for CRA

In the field of complexity theory, emergence refers to the arising of novel and coherent structures, patterns, and
properties during the process of self-organization in complex systems [15]. It is a process whereby larger entities, patterns, and regularities arise through interactions among smaller or simpler entities that themselves do not exhibit such properties. Emergent phenomena are conceptualized as occurring on the macro level, in contrast to the micro-level components and processes out of which they arise [15]. Emergence is central in the theories of complex systems.

According to the emergence mechanism, DK should be treated in a different way. We will take an example to explain it. Suppose there are some designers analyze and design for the same design problem simultaneously, and they provide their design solutions (e.g., domain models) independently. Obviously, each pair of these design solutions are different because there are many ‘degrees of freedom’ or alternatives even for a simple design problem, whilst these solutions are produced separately by designers who have different design experience. Despite that these design solutions are different, some partial designs in these solutions may be exactly the same because these partial designs focus on specific sub design problems in fine granularity, which have consensus solutions for most designers. Since a set of design decisions provide the rationale for a concrete (partial) design, if the same partial designs appear repeatedly in different design solutions, the corresponding design decisions of these partial designs emerged from these design solutions. As we know “getting smart from dumb things” in most situations, emerged design decisions should be made explicitly because most designers believe these decisions are appropriate for the design problem.

In the above example, we treat the whole independent design solutions produced by different designers as the design population for the design problem, and each solution is treated as a design individual of the design population. For a design problem, the design population is regarded as a macro level design, which is composed of the individual designs. Since our work focuses exclusively on assigning responsibilities to domain classes, the scope of design decision is limited to responsibility assignment, and the DK in this work can be regarded as which responsibilities have a high probability to be assigned to the same class. If it is difficult to decide which responsibilities should be assigned to the same class in the design individuals, we should turn to the macro level design population to find the implicit appropriate DK, which can help to decide the assignment of responsibilities.

To help understand DK emergence, we may review a general and frequently-used type of DK in OO design, design patterns [16], which emerge from design solutions with similar design contexts of design problems instead of being invented or created by expert designers [16]. In our opinion, a design pattern is discovered from a specific design population, which is composed of many design individuals for design problems with similar design contexts. Using design pattern as an analogy, if expert designers produce design solutions independently for a CRA design problem, appropriate DK for the design problem can also emerge from similar parts of the produced solutions (i.e., design individuals), even the appropriate DK for the design problem is unknown.

However, human resource is expensive and only a few designers work for design problems in software development projects. That is why DK emergence has hardly happened for manual design in practice. CRA can benefit from a GA based automated approach with another characteristic besides those mentioned in [1][7]: GA is population-based and we can readily map the proposed concepts, design individual and design population, into the concept individual and population of GA. Appropriate DK for a specific design problem can emerge from individuals (i.e., design solutions) of a population in GA. However, existing GA should be adapted to explicitly support representing, extracting, and learning DK during the automated search process. A learning based GA (LGA) extended from GA is detailed in the next subsection.

B. Learning based Genetic Algorithm (LGA)

To explicitly support the learning mechanism, which can be used to automatically extract and employ implicit DK for the CRA problem during the automated search process, the widely used GA has been extended with a learning operator. The extended GA is called as a learning based GA (LGA), and the pseudo-code and the high-level description of LGA are shown in Fig. 1.

![Pseudo-code of a Learning-based Genetic Algorithm](image)

![Abstracted high level description of a Learning-based Genetic Algorithm](image)

Fig. 1. The pseudo-code and workflow description of LGA

The procedure of LGA is similar to that of traditional GA. In LGA, the first generation is also made up of randomly selected chromosomes, and certain individuals are selected to go forward to the stages of crossover and mutation. However, different from traditional GA, a learning operator (Step 6 in Fig. 1(b)) should be further applied to the individuals before generating the next generation when the mutation stage (Step 5 in Fig. 1(b)) is completed. In the learning stage, the DK emergence mechanism from individuals, which is introduced in Section III.A, is employed as the primary learning mechanism for the population. Then individuals adapt themselves to produce the next generation based on the emerging DK. Different from the selection, crossover, and mutation operators which are executed in each generation, the LGA only executes the learning operator in every LearningSteps generation to reduce the computational cost of the learning phase in the proposed approach.

For the CRA problem, integer encoding scheme [17] is adopted in our approach to represent an individual (i.e., a candidate solution). In this scheme, an integer vector of N positions is used, where N is the number of total responsibilities of a design problem. Each position of the
vector corresponds to a particular responsibility, and the value represents the assigned class of that responsibility. The fitness of each individual is calculated by the widely used Modularization Quality (MQ) measure for software designs [18], which is designed to balance the tradeoff between the coupling and cohesion of software designs. The calculation of MQ for an individual that contains $k$ classes is defined in Formula (1). We refer to the internal dependencies of responsibilities in a class $i$ as $Intra_i$, and the dependencies exist between two distinct classes $i$ and $j$ as $Inter_{ij}$, respectively.

$$MQ = \sum_{i=1}^{n} CF_i$$

in which $CF_i = \begin{cases} 0, & \text{if } Intra_i = 0 \\ \frac{2^{2Intra_i + \sum_{j=1}^{n} (Intra_j+Inter_{ij})}}{2^{Intra_i} + \sum_{j=1}^{n} (Intra_j+Inter_{ij})}, & \text{Otherwise} \end{cases}$

C. Solving the CRA Problem with LGA

When the mutation stage in Fig. 1 is completed, individuals are ready for adapting themselves from learning. It is worth noting that the term “learning” used in this work refers to a type of self-motivated learning mechanism of individuals, which is akin to lifelong learning of people for pursuing of knowledge [19]. According to the fitness of each individual, the individuals of a generation are further classified into three groups: preferable individuals group (PG), ordinary individuals group (OG), and inferior individuals group (IG). In each generation, if the fitness of an individual is in the top firstPercent% of this generation, the individual may have good design quality, and it belongs to PG. If the fitness is in the bottom lastPercent%, the individual may have poor design quality, and it belongs to IG. The rest of the individuals with moderate design quality belong to OG. The appropriate values of parameter firstPercent and lastPercent should be determined according to the design problems. For the CRA problem, individuals of IG should apply the emerged DK from individuals of PG to evolve these IG individuals. Since rules provide an appropriate way to encapsulate DK [20], our approach employs an association rule mining algorithm [21] from data mining to extract DK from PG individuals, which helps to implement learning mechanism. Fig. 2 shows the four sub-steps of how to extract the DK according to the relations between responsibilities from PG individuals, and how to apply the extracted DK to IG individuals in order to make adaptive changes for these IG individuals. Note that, the four sub-steps in Fig. 2 implement Step 6 in Fig. 1 (b), and they are named as Step 6.1 to Step 6.4.

1) Construct dataset $D_p$ from PG

A dataset $D_p$ which can be further used to extract DK is needed in our approach, and Step 6.1 in Fig. 2 is used to construct this dataset from PG individuals. The dataset $D_p$ is composed of the records of all PG individuals, and each record contains the responsibilities, which are assigned to the same class in a PG individual. For example, let’s suppose that there are five responsibilities (i.e., $R_1$ to $R_5$) that need to be assigned to classes. As shown in Fig. 3(a), an individual $ind_1$ in PG may assign $R_1$, $R_2$, and $R_3$ to Class 1, and assign other responsibilities ($R_4$ and $R_5$) to Class 2. In this situation, two records exist for $ind_1$: $\{R_1, R_2, R_3\}$ and $\{R_4, R_5\}$. Similarly, suppose that another individual $ind_2$ in PG assigns responsibilities to three classes as shown in Fig. 3(b), and there exist three records for $ind_2$: $\{R_1, R_2\}$, $\{R_3, R_4\}$, and $\{R_5\}$. Hence, the dataset $D_p$ can be generated by combining the records of all individuals in PG. The constructed dataset $D_p$ of the example, which is shown in Fig. 3, is $\{\{R_1, R_2\}, \{R_3, R_4\}, \{R_5\}\}$. For the CRA problem, when $D_p$ contains the DK about how to evolve these IG individuals. Since rules provide an appropriate way to encapsulate DK [20], our approach employs an association rule mining algorithm [21] from data mining to extract the frequent itemsets of responsibilities from $D_p$. In data mining, a frequent itemset contains the elements, which have a high probability to appear together in records. The generated $L_p$ from the Apriori algorithm is composed of all frequent itemsets of $D_p$, and it can be used to highlight general trends in $D_p$. The details of the Apriori algorithm can be found in [21]. In our approach, when Step 6.2 is completed, the DK about which responsibilities have a high probability to be assigned to the same class in PG individuals can emerge and be extracted.

2) Generate frequent itemsets $L_p$

In Step 6.2 of Fig. 2, we use the Apriori algorithm [21], an association rule mining algorithm from data mining, to extract the frequent itemsets of responsibilities from $D_p$. In data mining, a frequent itemset contains the elements, which have a high probability to appear together in records. The generated $L_p$ from the Apriori algorithm is composed of all frequent itemsets of $D_p$, and it can be used to highlight general trends in $D_p$. The details of the Apriori algorithm can be found in [21]. In our approach, when Step 6.2 is completed, the DK about which responsibilities have a high probability to be assigned to the same class in PG individuals can emerge and be extracted.

3) Generate rule set $R_p$

When Step 6.2 is completed, the DK about which responsibilities have a high probability to be assigned together emerges from PG individuals. However, before exploiting the DK to IG individuals, the proposed approach should further know which responsibilities make other responsibilities be assigned to the same class in those frequent itemsets in $L_p$. For instance, if $\{R_2, R_3\}$ is a frequent itemset acquired in Step 6.2, we need to know whether $R_2$ leads to the assignment of $R_3$ to the same class with $R_2$ (i.e., $R_2 \Rightarrow R_3$), or vice versa (i.e., $R_3 \Rightarrow R_2$). Each situation above (e.g., $R_2 \Rightarrow R_3$) is called a rule, and a rule set $R_p$ is generated in Step 6.3. Each rule in $R_p$ represents a new form of knowledge about which responsibilities make other responsibilities be assigned to the same class in PG individuals, and consequently the generated rule set $R_p$ in Step 6.3 contains the DK about how to optimize responsibilities assignment for CRA design problems.

Algorithm 1. The algorithm for generating rule set $R_p$

<table>
<thead>
<tr>
<th>Input $L_p$: frequent itemsets</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output $R_p$: the generated rule set from $L_p$</td>
</tr>
<tr>
<td>Algorithm:</td>
</tr>
<tr>
<td>01 for a frequent itemset $f_i \in L_p$</td>
</tr>
<tr>
<td>02 $s \leftarrow {s</td>
</tr>
</tbody>
</table>
To generate the rule set $R_p$ from $L_p$ in Step 6.3, an algorithm is developed as shown in Algorithm 1. For each frequent itemset $fi$ in $L_p$, the algorithm first generates all of its non-empty subsets $S$ in Line 2. For a specific frequent itemset $fi$, $S$ contains $2^{n-1}$ non-empty subsets, in which $n$ is equal to the number of responsibilities that $fi$ contains. From Line 3 to Line 7, the algorithm constructs a specific rule with each non-empty subset $s$ of $fi$. During the construction procedure, it first generates a rule that has the form “$s$=(fi-s)”, in which $fi-s$ denotes the set that contains the elements in $fi$ but not in $s$. In our approach, generated rules represent DK, which can be applied by IG individuals to improve their design quality. However, not all the generated rules in Line 4 deserve to be applied by IG individuals. We use cosine measure, which is widely used in data mining [22] to measure the correlation between the non-empty subset $s$ of $fi$ and its corresponding set of $fi$-s in Line 5. The closer cosine$(s, fi)$ is to 1 (contrast to 0), the more records containing $s$ also contain $fi-s$, and vice versa. In our approach, a rule $r$ is omitted if $\text{cosine}(s, fi)$ is less than or equal to 0.65. The calculation of cosine measure is shown in Formula (2), in which $P(X)$ represents the probability that a record of the constructed dataset $D_p$ contains all responsibilities in set $X$. In addition, we use confidence measure, which is also widely used in data mining [23] to measure the priority of an association rule in Line 6. The calculation of confidence measure is shown in Formula (3). A higher confidence value of a rule means that the rule has a higher probability to be applied by IG individuals in Step 6.4. It is worth noting that some generated rules in the rule set $R_p$ are redundant. Suppose that there are two rules: $r_1$ and $r_2$ in $R_p$, if the set of responsibilities in the left part of $r_1$ is a subset of the set of responsibilities in the left part of $r_2$, and both the cosine and confidence measures of $r_1$ are greater than or equal to that of $r_2$, then $r_2$ is a redundant rule. The algorithm removes the redundant rules from Line 8 to 11.

$$\text{cosine}(s, fi-s) = \frac{P(su((fi-s))}{\sqrt{P(s)P((fi-s))}}$$ (2)

$$\text{confidence}(s \Rightarrow (fi-s)) = P((fi-s)|s)$$ (3)

In Step 6.3, the correlation relationships between responsibilities in frequent itemsets $L_p$ are transformed into the causal relationships of responsibilities in the rule set $R_p$. It shows that for the CRA problem, the DK about which responsibilities have a high probability to be assigned to the same class in PG individuals is transformed through this step into a new form of knowledge about which responsibilities make other responsibilities be assigned to the same class in PG individuals. When Step 6.3 is completed, the DK for CRA design problems is automatically extracted with the proposed algorithm, and individuals in IG can apply the DK from the rule set $R_p$ in Step 6.4.

4) Apply design knowledge adaptively for IG

In Step 6.4, a rule with a higher value of confidence measure will have a higher learning rate, which means this rule has a higher probability to be applied by individuals in IG. For each IG individual, it is limited to apply learningNums rules at most in the current generation, and the repeatLearning parameter in the proposed approach decides whether an IG individual can apply the same rule repeatedly in the current generation. In Step 6.4, each individual in IG tries to apply rules in the generated rule set $R_p$, and adaptively changes the IG individual itself with these rules. However, each rule has its precondition, which should be satisfied before the rule is applied to individuals in IG. We define the precondition of a rule according to the left part of the rule. For example, in the rule “$\{R_1, R_2\} = R'_{cr}$”, the left part of the rule is $\{R_1, R_2\}$, and the precondition of the rule is that $R_1$ and $R_2$ should belong to the same class in an individual. If $R_1$ and $R_2$ are assigned to different classes in an individual in IG, this rule cannot be applied to this individual. In Step 6.4, individuals in IG apply the extracted DK from individuals in PG to make adaptive changes in order to adapt against CRA design problems.

With the learning operator in LGA, our proposed approach for CRA can extract implicit DK from individuals which have better design quality in the population, based on the DK emergence mechanism. In addition, our approach allows individuals to adaptively change themselves to adapt against design problems with the extracted DK.

IV. EVALUATION PLAN

This section describes the experiment design that evaluates the learning-based approach for CRA, and is composed of the research questions (Section IV.A), the selected problem instances (Section IV.B), the rival approaches to the proposed approach (Section IV.C), the configuration of the parameters (Section IV.D), and the metrics used to evaluate the proposed approach (Section IV.E).

A. Research Questions

We used the following research questions (RQs) to evaluate the proposed approach.

RQ1: Do the solutions generated by the learning based approach have better structural measurements compared with the solutions generated by the rival approaches?

Rationale: Using structural measures is a common way to evaluate the quality of object-oriented design. The most widely used structural measures are based on coupling, cohesion, and complexity metrics [24], which mainly measure the modifiability of design [25]. To answer RQ1, we plan to adopt the widely used structural measures to compare the design quality of generated solutions between the proposed approach and the rival approaches.

RQ2: Are the solutions generated by the learning based approach closer to the “gold standard” of a given design problem compared with the solutions generated by the rival approaches?
Rationale: We need to further investigate and report the similarity degree of the generated solutions by different approaches to the “gold standard” of a given design problem on the basis of RQ1. Existing work has noticed that search-based approaches will produce complex designs that have a high conceptual distance from the designs produced by designers [26][27][28], which means that automated optimization is highly disruptive [28]. Since developers tend to avoid such disruption in practice [26][27][28], the similarity degree between generated design and the “gold standard” acts as an essential criterion to evaluate automated software design methods [27][28]. The used “gold standard” in this evaluation is also used in existing work for the CRA problem [3][6].

B. Problem Instances

Three problem instances (cases), a cinema booking system (CBS) [29], a graduate development program (GDP) [30], and a select cruises system (SC) [31], are used for the evaluation. We chose these three systems because (1) they were designed independently from our research, which mitigates the bias in evaluation; (2) the “gold standard” design of these three cases is available in [32]; and (3) existing work on CRA also used these cases to evaluate their proposed approaches [4][5][6][33].

The three problem instances have distinct sizes. The number of classes in the “gold standard” design for CBS and GDP is only single-digit, while the number of classes in the “gold standard” design for SC is double-digit. However, the number of classes is an indeterminable variable in automated approaches for the CRA problem, and the number of possible design solutions for even small problem instances is beyond human comprehension [5]. For the smallest problem instance (i.e., CBS) in our evaluation, the cardinality of its resulting design solution search space exceeds $2.77248 \times 10^{19}$ [5], which means the smallest size problem instance in this study is sufficient to evaluate the effectiveness of different approaches.

C. Rival Approaches

SOGA is a single objective genetic algorithm proposed for the CRA problem. The major difference between SOGA and the proposed LGA approach is that the latter introduces the extra learning mechanism in the approach. By comparing the results between SOGA and LGA, we can evaluate whether automated approaches for the CRA problem can benefit from exploiting implicit design knowledge.

SLINK, CLINK, and UPGMA are three hierarchical clustering approaches used for the CRA problem [3]. Their difference is in the calculation of clusters’ distance. The details of calculating the distance between two clusters with these three approaches can be found in [3].

MMAS utilizes an ant colony optimization (ACO) algorithm for the CRA problem, which is a graph-based search approach presented in [6].

Random Search (RS) works by iteratively moving to better positions in the search space, which are sampled from neighbors surrounding the current position. Existing work suggested a search algorithm should always be compared with a very minimum random search in order to check that the success is not due to the search problem being easy [34].

In the abovementioned approaches, the proposed LGA approach, SOGA, MMAS and RS are randomized approaches, while SLINK, CLINK, and UPGMA are deterministic approaches. Randomized approaches employ a degree of randomness as part of their logic, and different output results will be given with the same input data in different runs. By contrast, deterministic approaches will acquire the same output results with the same input data in different runs. For randomized approaches, we run each evaluation 30 times for each problem instance according to the guidelines in [35], and the output results of deterministic approaches are treated as the average value of that approach with a standard deviation of 0, because these approaches only ran once.

D. Evaluation Setup

We use the same parameter settings for SOGA from [2]. Since the major difference between SOGA and the proposed LGA approach is that the latter introduces an extra learning mechanism in the approach, the basic parameters of the LGA, which are independent of the learning mechanism, are same as the parameter settings for SOGA. Range of the remaining parameters with regard to the learning mechanism is tested for the proposed LGA (The bold value is the default value of that parameter for the LGA), which is shown in Table 1. To ensure the fair comparisons between approaches, the total fitness evaluations are the same for randomized approaches.

<table>
<thead>
<tr>
<th>Approach</th>
<th>Parameters</th>
</tr>
</thead>
</table>
| LGA SOGA | Population Size: 320  
Crossover Operation: Single Point Crossover  
Crossover Rate: 0.7  
Mutation Operation: Swap Mutation  
Mutation Rate: 1 / length of chromosome  
Selection Operation: Binary Tournament  
firstPercent: 0.1, 0.3 (for LGA only)  
lastPercent: 0.1, 0.3 (for LGA only)  
learningNums: 5, 20, 50 (for LGA only)  
learningSteps: 1, 10 (for LGA only)  
repeatLearning: true, false (for LGA only) |
| CLINK SLINK UPGMA | Cluster Number: From 1 to the number of responsibilities |
| MMAS | Number of Ants: 320  
Ant Selection for Updating Pheromone: random selection  
between best of iteration and best so far  
Initial pheromone limits: 0.1  
Pheromone evaporation rate: 0.02  
3 runs, 32000 fitness evaluations per run |
| RS | Fitness Evaluations: 32000 fitness evaluations per run, 30 runs |

E. Evaluation Metrics

For RQ1, we adopted TCC [36] and LCOM5 [37] to evaluate the cohesion measurement of a generated solution, and adopted CBO [38] and CSC [3] to evaluate the coupling and complexity measurement respectively. Similar to the existing work on CRA [3][6], we use F-Score for RQ2 to evaluate the similarity degree of the design solutions generated by automated approaches to the “gold standard”. We adopted F-Score computation from [3] to calculate the
In Fig. 4(a), we compare the fitness with different values of the firstPercent parameter when using different learningSteps. Higher fitness means generated solutions have a better design quality. We use Wilcoxon–Mann–Whitney test to determine whether two independent samples are selected from populations having the same distribution. When we set learningSteps equal to 1 (i.e., the learning operator in Fig. 1(b) is executed in each generation), it has significant differences on the fitness (i.e., p-value < 0.05 at a 5% significance level in the evaluation) between the two samples which have different values of firstPercent. We can observe that the smaller value for firstPercent tends to improve the design quality, but limits the diversity of generated solutions because of the small variances between generated solutions. However, the result is completely different when learningSteps is set to 10 (i.e., the learning operator in Fig. 1(b) is only executed in each 10th generation). The p-value in this situation reveals that there is no difference on the fitness between different firstPercent values. We conjecture that a bigger learningSteps reduces the total execution times of the learning operator, which may impair the influence of the parameters with regard to the learning process.

Fig. 4 Comparison of different firstPercent and lastPercent parameters

Compared with the firstPercent parameter, the lastPercent has less influence on the fitness of generated solutions, which is shown in Fig. 4(b). No matter what value of the learningSteps parameter is used, there is no significant difference between the samples with different lastPercent. However, when the learning operator is executed in each generation, it should be noted that bigger lastPercent limits the diversity of generated solutions because of the small variances between generated solutions. We can conclude that if more individuals adaptively change themselves, the less diversity of the solutions will be generated by the LGA.

To investigate the collaborative effect of firstPercent and lastPercent, we compare the fitness with different combos of the two parameters. As shown in Fig. 4(c), nonparametric Kruskal-Wallis test is used to compare the different groups, and pairwise test is used to compare each two groups with Bonferroni adjustments [40]. When the learning operator is executed in each generation, the p-value indicates that there is significant difference between these groups. It further confirms that the firstPercent and lastPercent parameters are related to the effectiveness of the LGA. However, since a
bigger learningSteps reduces the execution times of the learning operator, it makes the influence of these two parameters weaken as mentioned before.

Another important parameter for LGA is learningNums, which is designed to limit the maximum number of rules that an IG individual can be applied in a generation. The comparison of the fitness when using different learningNums is shown in Fig. 5: three different values are used for the learningNums parameter, which are 5, 20, and 50. The resulting p-value of the Kruskal-Wallis test shows the learningNums parameter has the important effect on the performance of the LGA. Pairwise test is further used to compare each two groups with Bonferroni adjustments. From the result of Fig. 5, there is no significant difference on the fitness when the learningNums value is 20 and 50. This result reveals that learning 20 rules in a generation for an IG individual is enough, and there is no need to pay more cost for learning more rules to improve the fitness. However, if the maximum number of rules that an IG individual can be applied in a generation is limited to only 5 rules, it impairs the influence of the learning process in the LGA.

Fig. 5 Comparison of different learningNums values

learningSteps and repeatLearning are another two key parameters which influence the effectiveness of the learning process of the LGA. learningSteps decides how often the learning process occurs, and repeatLearning decides whether an IG individual can apply a rule many times in a generation. We investigated the collaborative effect of these parameters to the fitness of generated solutions, and the result is shown in Fig. 6.

On one hand, when the value of repeatLearning is false, the p-values acquired from the Kruskal-Wallis test indicate that there is no significant difference on the fitness of generated solutions regardless of the different values of learningSteps and learningNums. One explanation is that learning a rule many times for an IG individual in a generation may improve the flexibility of the learning process, which makes generated solutions be diverse with more different fitness. On the other hand, the effect of the repeatLearning parameter on the fitness is more complicated when the value of repeatLearning is true. When learningSteps is small (e.g., 1), there is no evidence to show that using different learningNums values make the fitness of generated solutions be different. In contrast, if the learning operator is executed less frequently (i.e., the learning operator is only executed in certain generations), the fitness of the groups with different learningNums are different. Hence the repeatLearning parameter is one of the keys to the fitness of generated solutions when using a big value of learningSteps.

Fig. 6 Comparison of different learningSteps, learningNums, and repeatLearning parameters

To investigate the effectiveness of the extracted rules, we show the rule evolution process for the CBS case in Fig. 7, and use responsibility Cost for the CBS case as an example. Table 2 shows the six extracted rules for responsibility Cost when the 32000 fitness evaluations are completed. Two types of rules can be found in Table 2: 1) responsibilities in a rule have an obvious structural dependency relationship (e.g., CalculateCost depends on Cost in the use cases of CBS in Rule 1). If IG individuals apply this type of rules, their design quality will be improved undoubtedly because the dependencies between corresponding responsibilities will become internal dependencies, which helps to increase the cohesion of these designs; 2) responsibilities in a rule have no structural dependency relationship. We take Rule 6 as an example for this type of rules. From the designers’ perspective, if BookingNumber and Quantity are assigned to the same class (i.e., the precondition of Rule 6), assigning Cost to that class is natural even if there is no dependency between the three responsibilities in the use cases of CBS. The reason is that the three responsibilities represent a scenario that users book the number of Quantity tickets for a film with ID BookingNumber and then calculate the Cost of that booking. If IG individuals apply this type of rules (e.g., Rule 6), it makes the generated solutions more flexible.
solutions be close to the designs produced by designers, which minimizes the structure disruption of generated solutions.

Table 2. The extracted rules for responsibility Cost

<table>
<thead>
<tr>
<th>Rule ID</th>
<th>Rules (lhs (\Rightarrow) rhs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rule 1</td>
<td>{CalculateCost} (\Rightarrow) {Cost}</td>
</tr>
<tr>
<td>Rule 2</td>
<td>{CancelBooking, BookingNumber} (\Rightarrow) {Cost}</td>
</tr>
<tr>
<td>Rule 3</td>
<td>{CancelBooking, Quantity} (\Rightarrow) {Cost}</td>
</tr>
<tr>
<td>Rule 4</td>
<td>{Seats, BookingNumber} (\Rightarrow) {Cost}</td>
</tr>
<tr>
<td>Rule 5</td>
<td>{Seats, Quantity} (\Rightarrow) {Cost}</td>
</tr>
<tr>
<td>Rule 6</td>
<td>{BookingNumber, Quantity} (\Rightarrow) {Cost}</td>
</tr>
</tbody>
</table>

B. Results of Structural Measures

Fig. 8 to Fig. 10 show the structural measurements of generated solutions in 30 runs for different approaches with the three cases. The dash line represents the corresponding value of the “gold standard” solution. The results suggest the structural measurements are significant different between approaches (\(p < 2.2e^{-16}\) for the Kruskal-Wallis test) for all cases.

If we focus on the results, we see that the CBO, TCC and LCOM5 values obtained from automated approaches are much better than those of the “gold standard” in the three cases except RS (the lower the better for the CBO, LCOM5 and CSC measures, but the higher the better for the TCC measure). However, the CSC values obtained from the “gold standard” are better than all automated approaches. A possible reason of this result is that “gold standard” tried to reduce the complexity of a system, instead of maximizing cohesion and minimizing coupling measurements. In addition, complexity mainly depends on the problem instances, and the optimization of complexity by automated approaches receives the least effects compared with coupling and cohesion when using the MQ measure in the fitness function. Furthermore, it is worth noting that deterministic approaches (i.e., SLINK, CLINK, and UPGMA) perform worse than randomized approaches except RS. Since the cardinality of design solution search space of the three cases is huge as described in Section IV.B, we conjecture that randomized approaches beat deterministic approaches because randomized approaches provide better support to explore and exploit the problem search space than that of deterministic approaches.

From the results, we can also see that the proposed LGA approach beats all rival approaches in structural measures. We can conclude that exploiting extracted DK during the optimization promotes the better exploration and exploitation of the design solution spaces for all cases.

C. Results of Similarity Degree to “gold standard”

Similar to the existing research [3][6], each randomized approach also ran 30 times independently. For each run, the similarity degrees of the generated solutions to the “gold standard” are calculated by Formula (4), and the best value of the similarity degree of generated solutions is used to represent the similarity degree result of that run. The results of the F-Score for different approaches are shown in Fig. 11.

From the results shown in Fig. 11, different approaches have significant differences on the F-Score values in all cases because of the \(p\)-values calculated from the Kruskal-Wallis test. The generated solutions of the proposed LGA approach get a high similarity degree (above 77%) to the “gold standard” solutions for all the three problem instances, and the LGA beats all rival approaches. It is worth noting that the values of the similarity degree for the CRA problem were lower than the other two problems. The results imply that LGA has the ability to generate similar solutions to “gold standard” solutions in different problem instances, and LGA has less disruptive effects for the Cra problem than rival approaches because of exploiting the implicit DK.
VI. RELATED WORK

Most (semi-)automated approaches for the CRA problem are based on SBSE techniques such as ones reported in [2][4][5][6][33]. Bowman and his colleagues used a multi-objective genetic algorithm (MOGA) for solving the CRA problem [2]. They defined the fitness functions with two objectives: coupling and cohesion, which can be optimized independently in their multi-objective approach. Optimizing coupling and cohesion independently may not work well in some situations for the CRA problem. For example, the multi-objective approach for the CRA problem prefers a candidate solution that all responsibilities are assigned to a single class because this solution has the best coupling measurement than other candidate solutions. Since both coupling and cohesion measures belong to structural measures, a single objective fitness function is appropriate.

Simons and his colleagues proposed an interactive search-based approach for class design [5], and they further researched on the fitness function employed in their proposed interactive approach [33]. In their approach, search is steered jointly by designer preferences and software agents, which is a semi-automated search process. Different from the work which used GA as the search algorithm to solve the CRA problem, Tawosi and his colleagues solved the CRA problem with the ACO (ant colony optimization) algorithm [6]. In their approach, Problem Domain Semantic Network (PDSN), which is extracted from requirements, is proposed to simulate the prior knowledge of designers. However, the PDSN needs to be constructed and edited manually by domain experts to ensure the accuracy of the PDSN.

Smith and Simons investigated various search algorithms and corresponding parameters of these algorithms for class design [4]. They concluded that it is difficult to improve the design quality of the generated solutions only by choosing appropriate search algorithms and corresponding parameters for design problems. They also pointed out that it is beneficial for OO design to exploit the use information between responsibilities. We propose a novel approach in this work that exploits the use information between responsibilities as DK for improving the design quality of the generated solutions according to the emergence mechanism.

Non-SBSE approaches apply other techniques to solve the CRA problem. One of these techniques is clustering, which partitions a set of unlabeled data objects into a number of groups of similar objects. Since the essence of the CRA problem is to partition a set of responsibilities into a set of classes [3], clustering techniques can be used for the CRA problem. Masoud and Jalili presented a clustering-based model for the CRA problem [3]. In contrast with the population-based SBSE approaches (e.g., GA), clustering-based approaches cannot have many candidate solutions simultaneously during the clustering process, which makes DK emergence from multi-solutions impossible, and consequently the candidate solutions have no ability to apply the implicit DK for addressing the design problems during the process. Other researches tried to extract and generate class design from natural language requirements with natural language processing (NLP) techniques [20][41][42][43]. For example, linguistics patterns are used to describe the concepts of class, attribute, and method for the CRA problem [20][41]. The definitions of these patterns have a great impact on the design quality of generated solutions; however, these definitions heavily rely on designers’ experience and preferences. SBSE approaches (e.g., the proposed LGA approach) depends less on designers for the CRA problem.

VII. CONCLUSIONS AND FUTURE WORK

The assignment of responsibilities to classes is a vital task in OOAD, and (semi-)automated approaches have been developed for the CRA problem. CRA is one of the most complex tasks in OOAD as it heavily relies on designers’ judgment and is highly dependent on implicit DK of design problems. In this work, we propose a learning based approach (LGA) for the CRA problem to extract the implicit DK from candidate solutions, and then use the extracted DK for improving the design quality of the generated solutions. From an implementation perspective, the proposed LGA approach extends the traditional GA with an extra learning operator to support the learning mechanism. The implementation of the learning mechanism in the learning operator has four main steps for addressing the CRA problem: (1) construct a dataset for extracting DK; (2) generate frequent itemsets to extract the DK about which responsibilities have a high probability to be assigned to the same class; (3) generate a rule set to extract the DK about which responsibilities make other responsibilities be assigned to the same class; and (4) candidate solutions apply the extracted DK to improve their design quality.

A comparison between the obtained results of the proposed LGA approach and the rival approaches shows that the generated solutions by LGA have better structural measurements than the solutions produced by the state-of-the-art approaches. In addition, the similarity degree measurements of generated solutions by LGA to the “gold standard” solutions are more competitive than that of the rival approaches. Since developers tend to avoid structural disruption in practice [26][27][28], the results of similarity degree confirm that the generated solutions by the proposed approach are more likely to be accepted by developers from the practical aspects. For future work, we plan to develop a dedicated tool to support our approach for a practical use and evaluation of CRA.
REFERENCES


