The Optimal Refactoring Selection Problem –
A Multi-Objective Evolutionary Approach

Camelia Chisăliţă-Creţu

(1) Faculty of Mathematics and Computer Science,
Babeş-Bolyai University of Cluj-Napoca
1, Mihail Kogălniceanu Street, RO-400084 Romania
E-mail: cretu@cs.ubbcluj.ro

Abstract

Refactoring is a commonly accepted technique to improve the structure of object-oriented software. The Optimal Refactoring Selection Problem (ORSP) is the general identification problem of the optimal refactorings that may be applied to software entities, such that the internal structure is kept or improved in order to meet the requested demands. ORSP is an example of a Feature Transformation Subset Selection (FTSS) search problem in the Search-Based Software Engineering (SBSE) field. The paper states a special case of ORSP, the Multi-Objective Single Refactoring Selection Problem (MOSgRSP), that treats the refactoring cost and refactoring impact as conflicting selection criteria. MOSgRSP is based on the general form of the Single Refactoring Selection Problem (SgRSP). A weighted objective genetic algorithm is proposed. Different weight-based experiments on a didactic case study are presented and compared.

Keywords: Refactoring, Object-oriented programming, Search-based engineering, Multi-objective optimization

Introduction

Software systems continually change as they evolve to reflect new requirements, but their internal structure tends to decay. Refactoring is a commonly accepted technique to improve the structure of object oriented software (Fowler, 1999). ORSP is the identification problem of the optimal refactorings that may be applied on software entities, such that several objectives are kept or improved.

ORSP is an example of a Feature Transformation Subset Selection (FTSS) search problem in SBSE field. The paper presents the formal definition of the MOSgRSP which is based on SgRSP and performs a proposed weighted objective genetic algorithm on an experimental didactic case study. Obtained results for our case study are presented and compared.

The rest of the paper is organized as follows: Section 2 shortly reminds the general form of the Multi-Objective Optimization Problem (MOOP), while Section 3 presents the formal definition of the investigated MOSgRSP. Local Area Network (LAN) Simulation source code was used in order to validate our approach, being refered by Section 4. Section 5 unfolds the discussed evolutionary approach with several details related to the genetic operators of the proposed genetic algorithm. Different weight-based experiments on the LAN Simulation case study are presented and compared in Section 6. The paper ends with conclusions and future work.

MOOP Model

MOOP is defined in (Zitzler et al., 2001) as the problem of finding a decision vector

\[ x = (x_1, \ldots, x_n) \]

which optimizes a vector of \( M \) objective functions \( f_i(x) \) where
1 ≤ i ≤ M , that are subject to inequality constraints \( g_j(x) \geq 0 , \) \( 1 \leq j \leq J \) and equality constraints \( h_k(x) = 0 , \) \( 1 \leq k \leq K \). A MOOP may be defined as:

\[
[1] \quad \text{maximize} \{ F(x) \} = \text{maximize} \{ f_1(x), ..., f_M(x) \},
\]

with \( g_j(x) \geq 0 , 1 \leq j \leq J \) and \( h_k(x) = 0 , 1 \leq k \leq K \) where \( x \) is vector of decision variables and \( f_i(x) \) is the \( i \)-th objective function; and \( g(x) \) and \( h(x) \) are constraint vectors.

There are several ways to deal with a multi-objective optimization problem. In this paper the weighted sum method (Kim and deWeck, 2005) is used.

Let us consider the objective functions \( f_1, f_2, ..., f_M \). This method takes each objective function and multiplies it by a fraction of one, the "weighting coefficient" which is represented by \( w_i, 1 \leq i \leq M \). The modified functions are then added together to obtain a single fitness function, which can easily be solved using any method which can be applied for single objective optimization.

Mathematically, the new mapping may be written as:

\[
[2] \quad F(x) = \sum_{i=1}^{M} w_i \cdot f_i(x), \quad 0 \leq w_i \leq 1, \quad \sum_{i=1}^{M} w_i = 1.
\]

MOSgRSP Definition

The complete definition for the SgRSP is presented in (ChisăliŃă-CreŃu and Vescan, 2009). In order to understand the problem a brief summary is given here. Let \( SE = \{ e_1, ..., e_m \} \) be a set of software entities, e.g., a class, an attribute from a class, a method from a class, a formal parameter from a method or a local variable declared in the implementation of a method. The weight associated with each software entity \( w_i, 1 \leq i \leq m \) is kept by the set \( Weight = \{ w_1, ..., w_m \} \), where \( w_i \in [0, 1] \) and \( \sum_{i=1}^{m} w_i = 1 \). A set of possible relevant chosen refactorings (Fowler, 1999) that may be applied to different types of software entities of \( SE \) is gathered up through \( SR = \{ r_1, ..., r_t \} \). There are various dependencies between such transformations when they are applied to the same software entity, a mapping emphasizing them being defined by:

\( rd: SR \times SE \rightarrow \{ \text{Before, After, AlwaysBefore, AlwaysAfter, Never, Whenever} \} \).

The effort involved by each transformation is converted to cost, described by the function \( rc: SR \times SE \rightarrow N \). Changes made to each software entity \( e_i, i=1,m \) by applying the refactoring \( r_l, 1 \leq l \leq t \) are stated and a mapping is defined: \( effect: SR \times SE \rightarrow Z \). The overall effect of applying a refactoring \( r_l, 1 \leq l \leq t \) to each software entity \( e_i, i=1,m \) is defined by the mapping \( res: SR \rightarrow Z \).

The MOSgRSP is the identification problem of a refactoring that may be applied to a software entity such that the proposed objectives are kept or improved. This is to find a refactoring
for each software entity $e_i \in SE, i=1,m$ such that the cost objective is minimized and the overall effect objective is maximized.

Multi-objective optimization often means compromising conflicting goals. For our MOSgRSP formulation there are two objectives taken into consideration in order to maximize refactorings effect upon software entities. The first objective function minimizes the total cost. In order to have a maximized objective, it was subtracted from $MAX$, the biggest possible total cost, as:

$$\text{maximize} \left\{ f_1(r) \right\} = \text{maximize} \left\{ MAX - \sum_{i=1}^{m} \sum_{i=1}^{n} \mathbf{c}(r_i, e_i) \right\}; r = (r_1, ..., r_m)$$

The second objective function maximizes the total effect of applying refactorings upon software entities, considering the weight of the software entities in the overall system, like:

$$\text{maximize} \left\{ f_2(r) \right\} = \text{maximize} \left\{ \sum_{i=1}^{m} \sum_{i=1}^{n} \mathbf{w}(r_i, e_i) \right\}; r = (r_1, ..., r_m)$$

The goal is to identify those solutions that compromise the refactorings costs and the overall impact on transformed entities. The final fitness function for MOSgRSP is defined by aggregating the two objectives and may be written as:

$$F(r) = \alpha \cdot f_1(r) + (1-\alpha) \cdot f_2(r), \quad 0 \leq \alpha \leq 1.$$ 

**Case Study: LAN Simulation**

The algorithm proposed was applied to a simplified version of the LAN Simulation source code that was presented in (Demeyer et al, 2002). Figure 1 shows the class diagram of the studied source code. It contains 5 classes with 5 attributes and 13 methods, constructors included.

![Class Diagram for LAN Simulation](image)

The current version of the source code lacks of hiding information for attributes since they are directly accessed by clients. The abstraction level and clarity may be increased by creating a new superclass for PrintServer and FileServer classes, and populate it by moving up methods in the class hierarchy.

Thus, for the studied problem the software entity set is defined as: $SE = \{c_1, ..., c_5, a_1, ..., a_5, m_1, ..., m_5\}$. The chosen refactorings that may be applied are: renameMethod, extractSuperClass, pullUpMethod, moveMethod, encapsulateField, addParameter, denoted by the set $SR = \{r_1, ..., r_6\}$.
in the following. The dependency relationship between refactorings is defined in what follows:

\[(r_1, r_2) = B, (r_1, r_3) = AA, (r_2, r_1) = B, (r_2, r_1) = A, (r_1, r_2) = AB, (r_1, r_2) = A, (r_2, r_3) = N, (r_2, r_3) = N, (r_3, r_2) = N, (r_3, r_2) = N, (r_3, r_3) = N, (r_3, r_3) = N\].

The values of the final effect were computed for each refactoring, but using the weight for each existing and possible affected software entity, as it was defined in Section 3. Therefore, the values of the \(r_{es}\) function for each refactoring are: 0.4, 0.49, 0.63, 0.56, 0.8, 0.2.

Here, the cost mapping \(rc\) is computed as the number of the needed transformations, so related entities may have different costs for the same refactoring. Each software entity has a weight within the entire system, but \[\sum_{i=1}^{23} w_i = 1\]. For the \(effect\) mapping, values were considered to be numerical data, denoting estimated impact of a refactoring applying. Due to the space limitation, intermediate data for these mappings was not included. An acceptable solution denotes lower costs and higher effects on transformed entities both objectives being satisfied.

**Proposed Approach Description**

The goal is to identify those solutions that compromise the refactoring costs and the overall impact on transformed entities. The decision vector \(r = (r_1, \ldots, r_m), r_i \in SR, 1 \leq i \leq m\) determines the refactorings that may by applied in order to transform the considered set of software entities \(SE\).

The item \(r_i\) on the \(i\)-th position of the solution vector represents the refactoring that may be applied to the \(i\)-th software entity from \(SE\), where \(e_i \in SE, 1 \leq i \leq m\). The proposed genetic algorithm that approaches an \(entity\)-based solution representation for the studied problem, is denoted by \(SgRSGAEnt\) in the following.

**Genetic Operators**

The genetic operators used are crossover and mutation. Each of them is presented below.

**Crossover Operator.** A simple one point crossover scheme is used. A crossover point is randomly chosen. All data beyond that point in either parent string is swapped between the two parents.

For example, if the two parents are: \(parent_1 = [ga \ gb \ gc \ gd \ ge \ gf]\) and \(parent_2 = [g1 \ g2 \ g3 \ g4 \ g5 \ g6]\) and the cutting point is 3, the two resulting offspring are: \(offspring_1 = [ga \ gb \ gc \ g5 \ g6]\) and \(offspring_2 = [g1 \ g2 \ g3 \ gd \ ge \ gf]\).

**Mutation Operator.** The operator used here consists of simply exchanging the value of a gene with another value from the allowed set. In other words, mutation of \(i-th\) gene consists in allocating a different refactoring to be applied to the entity \(i\). Half chromosome number genes mutation was used here.

For instance, if we have the chromosome \(parent_1 = [ga \ gb \ gc \ gd \ ge \ gf]\) and we chose to mutate fifth gene, then a possible offspring can be \(offspring_1 = [ga \ gb \ gc \ gd \ gnew \ gf]\).

**Data Normalization**
Normalization is the procedure used in order to compare data having different domain values. It is necessary to make sure that the data being compared is actually comparable. Normalization will always make data look increasingly similar. An attribute is normalized by scaling its values so they fall within a small-specified range, e.g., 0.0 to 1.0.

As we have stated above we would like to obtain a subset of refactorings to be applied to a software entity from the given set of entities, such that we obtain a minimum cost and a maximum effect. The refactoring applying cost to an entity for the LAN Simulation case study is between 0 and 100. At each step of the selection the $res$ function is considered. We must normalize the cost of applying the refactoring, i.e., $rc$ mapping, and the value of the $res$ function too. Two methods to normalize the data: decimal scaling for the $rc$ mapping and min-max normalization for the value of the $res$ function have been used here.

Obtained Results by the Evolutionary Approach

The $SgRSGAEnt$ algorithm was run 100 times and the best, worse and average fitness values were recorded. The parameters used by the evolutionary approach were: mutation probability 0.7 and crossover probability 0.7. The experiments include runs for 10, 50, 100, 200 number of generations with 20, 50, 100, 200 as number of individuals.

The following subsection shortly presents the raw results and emphasize the impact on the class diagram when the two objective have the equal weight within the fitness function, i.e., $\alpha = 0.5$. Different weight has been given to the refactoring cost and refactoring impact within other run experiments, where $\alpha = 0.3$ $\alpha = 0.7$ and their results are reminded in subsection 6.2.

Equal Weights ($\alpha = 0.5$)

The results for the run experiments that proposes equal weights, are presented by Figure 2 and Figure 3. In Figure 2 the 200 generations evolution of the fitness function (best, worse and average) for 20 chromosomes populations is presented, while the 200 chromosomes populations with 50 generations evolution is depicted by Figure 3.

![Figure 2. The Experiment with 200 generations and 20 individuals](image-url)
In the 50 generations experiments for 50 chromosomes populations the greatest value of the fitness function was $0.3455$ (38 individuals with the fitness $> 0.33$) while in the 200 evolutions experiments for 20 individuals populations the best fitness value was $0.3562$ (96 individuals having the fitness $> 0.33$), which is the best fitness value within the run experiments.

The worst chromosome in all runs was recorded for a 200 individuals population for 50 generations evolution with a fitness value of $0.2700$ ($87$ chromosomes having the fitness $< 0.283$), while in the 20 chromosomes population with 200 generations evolution the worst individual had the fitness value $0.2772$ ($11$ individuals with the fitness value $< 0.283$ only).

The number of chromosomes with fitness value better than $0.33$ for the studied populations and generations is captured by Figure 4. It shows that smaller populations with poor diversity among chromosomes involve a harder competition within them and more, the number of eligible chromosomes increases quicker for smaller populations than for the larger ones.

**Impact on the LAN Simulation source code**

The best individual obtained by this solution representation allowed to improve the structure of the class hierarchy. Thus, for `PrintServer` and `FileServer` classes, a new base class `Server` is added. Moreover, the signature of the `print` method from the `PrintServer` class is changed in order to allow the `accept` method to be pulled up to the new base class. The `save` method signature from the `FileServer` class should be changed and then renamed too. Next, `accept` method should be pulled up to the `Server` class. But the studied best individual genes does not suggest the mentioned refactorings. The correct access to the class fields is accomplished by encapsulating them within their classes.

The current solution representation does not allow to apply more than one refactoring to each software entity, i.e., the `print` method from `PrintServer` class may be transformed by only one refactoring, e.g., `addParameter` or `renameMethod`. Figure 5 presents the LAN Simulation class diagram after the obtained solution is applied to. The refactoring cost impact on software entity have been treated with the same importance within the refactoring process ($\alpha = 0.5$).
Summary of the Obtained Results

The results of the proposed approach in Section 5 for three different values for the $\alpha$ parameter, i.e., 0.3, 0.5, 0.7, are summarized and discussed by the current section. A best chromosome list of the obtained results for all experiments is given below:

- $\alpha = 0.3$, bestFitness = 0.25272 for 100 chromosomes and 200 generations
  - bestChrom = $[1, 1, 1, 1, 4, 4, 4, 4, 2, 2, 3, 3, 3, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2]$

- $\alpha = 0.5$, bestFitness = 0.3562 for 20 chromosomes and 200 generations
  - bestChrom = $[1, 1, 1, 1, 4, 4, 4, 4, 4, 2, 2, 2, 2, 3, 2, 3, 5, 5, 2, 0, 3, 2]$

- $\alpha = 0.7$, bestFitness = 0.45757 for 50 chromosomes and 50 generations
  - bestChrom = $[1, 1, 1, 1, 4, 4, 4, 4, 4, 3, 2, 3, 2, 3, 2, 3, 5, 2, 2, 3, 2, 2]$

The data shows similar results for the structure of the best chromosome. A major difference is represented by the possible refactoring that may be applied to the `save` method from FileServer and the `accept` method from PrintServer and FileServer classes. The suggested solutions by $\alpha = 0.3$ and $\alpha = 0.5$ experiments recommend a second refactoring that may be applied to the `save` method, i.e., the `renameMethod` refactoring, while for $\alpha = 0.7$ the suggested refactoring is not appropriate, i.e., the `moveMethod` refactoring. Figure 5 reflects the changes on the class hierarchy for the $\alpha = 0.5$.

The experiments for 20 chromosomes populations have good results in each of the three runs with different values for the $\alpha$ parameter, bringing a better solution quality for the eligible individuals (see Figure 4).

This means in small populations (with few individuals) the reduced diversity among chromosomes may induce a harsher competition compared to large populations (with many chromosomes) where the diversity breeds weaker and closer individuals as fitness quality. As the run experiments revealed it, after several generations smaller populations produce better individuals (as number and quality) than larger ones, due to the poor populations diversity itself (see Figure 4).
Conclusions and Future Work

The paper defines the MOSgRSP by treating the cost constraint as an objective and combining it with the effect objective. The results of a proposed weighted objective genetic algorithm on an experimental didactic case study are presented and compared.

Current paper discusses the weighted multi-objective optimization, but the Pareto approach is a further step in current research since it proves to be more suitable when it is difficult to combine several objectives into a single aggregated fitness function. Moreover, the cost may be interpreted as a constraint, with the further consequences.

References