ELSEVIER

Available online at www.sciencedirect.com

SciVerse ScienceDirect

NasBASE (2015) 000–000

NasBASE

Many-Objective Software Engineering using Preference-based Evolutionary Algorithms: A Case Study in Software Refactoring

Mohamed W. Mkaouer[a], Marouane Kessentini[a], Slim Bechikh[a], Ali Ouni[b]

[a]*{firstname@umich.edu}, University of Michigan, USA*
[b]*ouniali@iro.umontreal.ca, University of Montreal, Canada*

Abstract. Recently, researchers have proposed several solution approaches to tackle many-objective optimization problems (e.g., objective reduction, new preference ordering relations, decomposition, etc.). However, these techniques are not yet widely explored in SBSE. In this paper, we discuss the different challenges and potential benefits of many-objective optimization techniques in software engineering. In addition, we propose the use of preference-based evolutionary many-objective optimization technique (P-EMO) for automating code refactoring to fix design defects. Very promising results are presented on three large open source systems. More than 85% of suggested refactorings are valid and fixes around 82% of code-smells.

Keywords: refactoring, software metrics, many-objective.

1. Introduction

Search-based software engineering (SBSE) mostly studies the application of metaheuristic optimization techniques to software engineering problems [9]. Once a software engineering task is framed as a search problem, by defining it in terms of solution representation, objective function, and solution change operators, there are a multitude of search algorithms that can be applied to solve that problem. Search-based techniques are widely applied to solve software engineering problems such as in testing, modularization, refactoring, planning, etc.

According to recent SBSE surveys [9], most of existing work treats software engineering (SE) problems from a single-objective point of view, where the main goal is to maximize or minimize one objective (e.g., correctness, quality, etc.). However, most SE problems are naturally complex in which many conflicting objectives need to be optimized such as model transformation, design quality, testing, etc. The number of objectives to consider for SE problems is, in general, high (more than three objectives). For example, evaluating the quality of a design after applying refactorings can be performed using a set of quality metrics where each metric can be consider as an objective. In this situation, the use of traditional multi-objective techniques (e.g., NSGA-II [2]), widely used in SBSE, is not sufficient as the context has a high number of objectives.

Recently, researchers have proposed several solution approaches to tackle many-objective optimization problems [1], [4], [5], [6], [8] (e.g., objective reduction, new preference ordering relations, decomposition, etc.). However, these techniques are not explored yet in SBSE [9]. In this paper, we discuss the different challenges and potential benefits of many-objective optimization techniques in software engineering. In addition, we propose the use of preference-based evolutionary many-objective optimization techniques (P-EMO) [5] for the specific software engineering problem of code refactoring to fix design defects [10]. In fact, P-EMO allows the incorporation of decision maker (DM), i.e., designer,

preferences in multi-objective optimization techniques by restricting the Pareto front to a region of interest (ROI) easing the decision making task. We report experiments on the use of P-EMO on the code refactoring problem where very promising results are obtained on three large open source systems [11], [12], [13]: More than 85% of suggested refactorings are valid and around 82% of code-smells in these systems are fixed. The best set of refactoring solutions also satisfies the preferences of developers expressed in terms of conflicting quality metrics (seven conflicting objectives to satisfy). Based on our empirical evaluation, we found also that P-EMO outperforms NSGA-II over 30 runs.

The remainder of this paper is structured as follows. The next section gives an overview of the open problems in software engineering that can be addressed by many-objective optimization techniques. Section 3 describes our adaptation of P-EMO to automate code refactoring and the results obtained from our experiment are discussed in Section 4. Section 5 concludes and gives future directions.

2. Many-Objective Software Engineering: Challenges and Benefits

2.1. Many-Objective Optimization

Recently, many-objective optimization has attracted much attention in evolutionary multi-objective optimization (EMO) which is one of the most active research areas in evolutionary computation [1]. By definition, a many-objective problem is multi-objective, but with a high number of objectives M, namely $M>3$. Analytically, it could be stated as follows [2]:

$$\begin{cases} Min\ f(x) = [f_1(x), f_2(x), \ldots, f_M(x)]^T \\ g_j(x) \geq 0 \qquad j = 1, \ldots, P; \\ h_k(x) = 0 \qquad k = 1, \ldots, Q; \\ x_i^L \leq x_i \leq x_i^U \qquad i = 1, \ldots, n \end{cases} \tag{1}$$

where M is the number of objective functions and is *strictly greater* than 3, P is the number of inequality constraints, Q is the number of equality constraints, x_i^L and x_i^U correspond to the lower and upper bounds of the decision variable x_i (i.e., i^{th} component of x). A solution x satisfying the (P+Q) constraints is said to be feasible and the set of all feasible solutions defines the feasible search space denoted by Ω.

In this formulation, we consider a minimization MOP since maximization can be easily transformed to minimization based on the duality principle by negating each objective function. Over the past two decades, several Multi-Objective Evolutionary Algorithms (MOEAs) have been proposed with the hope to work with any number of objectives M. Unfortunately, it has been demonstrated that most MOEAs are ineffective in handling many-objective problems. For example, NSGA-II [3], which is one of the most used MOEAs, compares solutions based on their non-domination ranks. Solutions with higher ranks are emphasized in order to converge to the Pareto front. When M>3, there is a high probability that almost all population individuals become non-dominated with each other, resulting in them all being lumped together in a single rank. Thus, NSGA-II is not able to maintain selection pressure in high-dimensional objective spaces.

The difficulty faced when solving a many-objective problem can be summarized as follows. Firstly, most solutions become of equivalent quality to each other according to the Pareto dominance relation which deteriorates dramatically the search process' ability to converge towards the Pareto front and the MOEA behaviour becomes very similar to random search. Secondly, a search method requires a very high number of solutions (some thousands or even more) to cover the Pareto front when the number of objectives increases. For instance, it has been shown that in order to find a good approximation of the Pareto front for problems involving 4, 5 and 7 objective functions, the number of required non-dominated solutions is about 62 500, 1 953 125 and 1 708 984 375 respectively [4]; which makes the decision making task very difficult. Thirdly, the objective space dimensionality increases significantly which makes promising search directions very hard to find. Finally, the Pareto front visualization becomes more complicated for the DM, thereby complicating the interpretation of the MOEA's results. Recently, researchers have proposed several solution approaches to tackle many-objective optimization problems which are described as follows:

- ***Objective reduction*:** This technique consists of finding the minimal subset of objective functions that are conflicting with each other. The main idea is to study the different conflicts between the objectives. The objective reduction approach attempts to eliminate objectives that are not essential to describe the Pareto optimal front [1]. If the number of essential objectives is found to be two or three, the problem could be solved by existing MOEAs. It implies that objective reduction could make an otherwise unsolvable many-objective problem solvable. Even when the essential objectives are four or more, the reduced representation of the problem will have favorable impact on the search efficiency, computational cost and decision making.

- ***Incorporating DM's preferences*:** When the number of objective functions increases, the Pareto optimal approximation is composed of a huge number of non-dominated solutions (some hundreds and even some thousands). Consequently, the selection of the final alternative would be very difficult for a human DM. In reality, the DM is not interested in the entire Pareto front, but rather just in the portion of the front that best matches his/her preferences, called ROI. Recently, several preference-based approaches have been proposed in the specialized literature [5], [6]. The main idea is to exploit DM's preferences in order to differentiate between Pareto equivalent solutions so that we can direct the search towards the ROI on problems involving more than 3 objectives. P-EMO algorithms have demonstrated several promising results that are: (1) improving the algorithm's ability to tackle many-objective problems, (2) providing the DM with a good approximation of the ROI and thereby easing the decision making task, and (3) saving the computational effort required to find the remainder of the Pareto front in which the DM is not interested.

- ***New preference ordering relations*:** Since the Pareto dominance's ability to differentiate between solutions decreases as the number of objectives increases, researchers have proposed several alternative relations. These relations try to circumvent the failure of the Pareto dominance by using additional information such as the ranks of the particular solution regarding the different objectives and the related population [7].

- ***Decomposition*:** This technique consists of decomposing the problem into several sub-problems and then solving these sub-problems simultaneously by exploiting the parallel search ability of evolutionary algorithms. The most reputable decomposition-based MOEA is MOEA/D [8] where the basic idea is to assign each population individual the task of optimizing a particular aggregation of the objectives (weighted aggregation or Chebyshev norm) based on a particular weight vector.

We investigate, in this paper, the applicability of these recent approaches in tackling SE problems such as the many-objective refactoring problem. As depicted in Figure 1, large scale software includes a large number of components (e.g., classes, methods, etc.) with high degree of complexity and interactions among them. In the next section, we identify the potential use of many objective optimization techniques in SBSE.

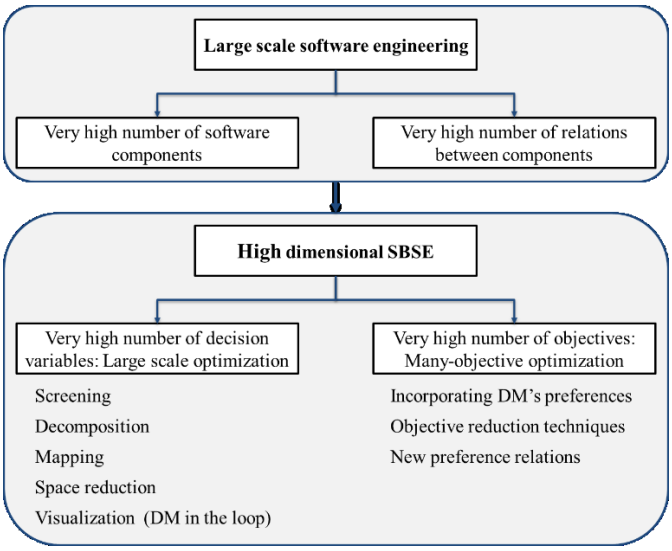


Fig.1. Handling High dimensionality in SBSE.

2.2. Benefits, Challenges and Open Problems in Many-Objective Software Engineering

According to a recent survey by Harman et al. [9], most existing search-based software engineering (SBSE) work treats SE issues as mono-objective. However, SE problems are typically multi-objective by nature. Consequently, research has been shifted towards multi-objective approaches in handling SE issues. As noted by Deb (2012) during his keynote speech in SSBSE'12 [14], EMO methods are actually ready to be applied to SE problems. One of the major areas that Deb noted is the incorporation of DM preferences in multi-objective SBSE. Consequently, it is very interesting for the SBSE community to apply P-EMO algorithms, such as r-NSGA-II and PBEA [5], [15], to SE problems ranging from requirement engineering to software testing and maintenance with an attempt to provide the DM with a ROI that corresponds to the set of non-dominated solutions that best match the DM's preferences. These preferences can be expressed in different ways.

To the best of our knowledge, there exists only a single work in the SBSE community which discusses the challenges of incorporating preferences in many-objective SBSE [16]. However, this paper does not really discuss the problematic of integrating user preferences in multi-objective SBSE, but rather the problematic of the *many-objective* resolution of SE problems. In addition, the case study presented in [16] is related to a proof of concept related to software product lines using PBEA. We recently proposed in [18] the use of P-EMO for model transformation as a proof of concept using only three objectives.

Most existing approaches to address software engineering issues (i.e., model transformation, design quality, testing, evolution, and comprehension, etc.) are based on a mono-objective view, i.e., to mainly satisfy (e.g., maximize or minimize) one performance metric (e.g., correctness, quality, meta-model coverage, etc.) or an aggregation of up-to three objectives. We believe that many-objective optimization algorithms, such as P-EMO, are very suitable for most SE problems. In fact, most of these problems, such as modelling or quality or testing, are difficult to fully-automate due to the need for interaction with the user especially when the number of objectives to satisfy becomes high. Due to the lack of space, we focus in this paper on three main important SE problems.

***Software refactoring*:** code refactoring consists of improving the design quality of systems by detecting and fixing *code-smells* using refactoring operations (such as move method, extract class, etc.) [17]. Unlike software bugs, there is no general consensus on how to decide if a particular design violates a quality heuristic. There is a difference between detecting symptoms and asserting that the detected situation is an actual bad-smell. Code-smells [17] are generally described using natural language and their detection relies on the interpretation of the developers. Indeed, different experts can have divergent opinions when identifying symptoms for the same bad-smell type. Overall, evaluating the quality of a design is subjective. Thus, incorporating DM (designer/expert) preferences can address different quality improvement objectives during the detection process. These objectives can be formulated in terms of quality metrics which means that the number of objectives can be high. Many designers can specify different reference/ideal points depending on their preferences. Another issue in model refactoring is that detecting dozens of bad-smell occurrences in a system is not always helpful, except if the list of defects is sorted by priority. In addition to the presence of false positives that may create a rejection reaction from development teams, the process of using the detected lists, understanding the defect candidates, selecting the true positives, and correcting them, is long, expensive, and not always profitable. However, the list of defects can be reduced based on the developers' preferences. For instance, they can focus only on some specific bad-smells. Some other issues are related to the fixing step. In fact, developers have many preferences to satisfy in addition to improving the quality. Reducing the effort required to improve the design quality, preserving the semantic coherence when improving the quality, and minimizing the number of suggested refactorings, can all be additional DM preferences, especially considering that many refactoring alternatives are sometimes equivalent from a quality point of view. In some cases, correcting some anomalies corresponds to re-implementing many parts of the system or even the entire system. In such situations, we need to find a compromise between improving code quality and reducing the adaptability effort and this depends on the developers' preferences. When considering these refactoring objectives, the designers can run into difficulties to define the relative importance of each one (quality metrics, effort, semantic, etc.); however, it might be easier for them to identify some reference points (e.g., 90% quality, 60% effort and 100% semantic). These reference points are one of the inputs of a P-EMO algorithm. In this paper, we are considering quality metrics as objectives to satisfy when applying refactoring.

Software evolution: For understanding the evolution of a source code (different versions), dedicated mono-objective change detection approaches have been proposed for models [19]. The majority of existing approaches [19], [20], [21] are successful in detecting atomic changes. However, composite changes, such as refactorings, are difficult to detect due to eventually hidden changes in intermediate model versions that may be no longer available. Thus, a huge number of equivalent refactoring solutions can be found to describe some changes between different model versions. The developers can formulate some preferences to select the best solutions from these equivalent ones. These preferences could be minimizing the number of refactoring operations (describing the changes), maximizing the number of complex refactorings, and reducing the number of atomic changes. These objectives can be different from one developer to another.

Software testing: In software testing, one of the important problems is the generation of test cases (models) from the meta-model description [22]. The main criterion, used by existing work, to evaluate test cases, is the coverage of meta-model elements. However, this objective is insufficient since there are some other important objectives such as the number of generated test cases, the number of detected mutants, and the number of covered changes (in case of meta-model evolution or regression testing). Furthermore, the developers can specify some other preferences like testing only some meta-model elements. It is also difficult to specify the relative importance of each objective since coverage, number of test cases, and number of covered mutants, all are very important objectives. Instead, the developer can specify only some reference points that can be used to find the best trade-off between all these objectives.

Software migration: the goal of existing software migration approaches is to provide rules generating target code/models, from source language, without errors [23]. However, other important objectives are how to minimize the complexity of transformation rules (e.g., the number of rules, number of matchings in the same rule) while maximizing the quality of target models and rules correctness. The transformation mechanism can be considered as a many-objective problem where the goal is to find the best rules maximizing target-model's quality and rules-correctness, and minimizing rules-complexity. The correctness can be evaluated regarding the number of satisfied constraints (specified in the metamodel level). Then, the quality of the proposed solution (rules) can be evaluated by calculating the number of rules and matchings in each rule, and assessing the quality of generated target models using a set of quality metrics.

3. Case Study: Many-Objective Software Refactoring

In this section, we first introduce definitions of important concepts related to software refactoring. Then, we emphasize the adaptation of P-EMO to the refactoring problem.

3.1. Overview

Refactoring is defined as the process of improving a code after it has been written by changing its internal structure without changing the external behavior [17]. The idea is to reorganize variables, classes and methods mainly to facilitate future adaptations and extensions. This reorganization is used to improve different aspects of software-quality such as maintainability, extensibility, reusability, etc. Some modern integrated development environments (IDEs), such as Eclipse, NetBeans, and Refactoring Browser [19], provide semi-automatic support for applying the most commonly used refactorings, e.g., move method, rename class, etc. However, automatically suggesting/deciding where and which refactorings to apply is still a real challenge in SE.

In order to identify which parts of the source code need to be refactored, most existing work (see [10] for example) relies on the notion of design defects or bad smells. In this paper, we do not focus on the first step related to the detection of refactoring opportunities. We consider that different design defects are already detected, and need to be corrected. Typically, design defects, also called anomalies [17], design flaws, bad smells, or anti-patterns, refer to design situations that adversely affect the development of software. When applying refactoring to fix design defects, software metrics can be used as an indication of the quality of the new design. For instance, high cohesion and low coupling between most of the classes in the systems give an indication of the good quality of a system. In addition, most design defects can be detected using quality metrics (symptoms). In [25], the authors used quality metrics as fitness functions to evaluate suggested refactoring. However, the number of objectives was limited to one (aggregation of metrics in one objective [25]) or two (considering only two quality metrics [26]).

In this paper, we are adapting a many-objective optimization technique by the mean of preference-based evolutionary many-objective algorithm [5] described in Section 2.1. In the next section, we describe the adaption of P-EMO for software refactoring.

3.2. Problem Formulation

In the following, we propose our formulation for the code refactoring problem using P-EMO. We give first the solution representation, then the objective function descriptions and change operators.

A solution S is a sequence of refactorings represented in a vector where each refactoring is a dimension in this vector. For each of these refactorings we specify pre- and post-conditions that are already studied in [17] to ensure the feasibility of applying them. In addition, for each refactoring, a set of controlling parameters, e.g., actors and roles (name of source and target classes, etc.) are randomly picked from the program to be refactored. In the majority of existing works, the fitness function evaluates a generated solution by verifying its ability to improve an aggregation of metrics (as one objective) or two objectives (only two metrics). In our case, we are considering a set of n metrics to improve where each metric is considered as an objective. Thus, the objective functions are the following:

P-EMO pseudocode for the refactoring problem

```
01. Input
02.   M : the number of objective functions
03.   n : the number of decision variables (refactoring sequence length)
04.   N : the population size
05.   SYS : the system to evaluate
06.   R : the user's reference point
07. Output
08.   Pt : the set of preferred non-dominated (non-r-dominated) solutions
09. Begin
10.   t ← 0 ;
11.   Pt ← random_initialization (n, N, M);
12.   Pt ← evaluate (Pt, SYS);
13.   While (NOT termination-criterion) do
14.     Qt ← genetic_operations (Pt) ;
15.     Qt ← evaluate (Qt, SYS);
16.     Ut ← Union (Pt, Qt);
17.     Ut ← Non-r-dominated_sorting (Ut, M, R);
18.     Ut ← Crowding_distance_assignment (Ut, M);
19.     Pt ← environmental_selection (Ut-1, N);
20.     t ← t+1;
21.   End While
22. End
```

Fig.2. Pseudocode of the P-EMO algorithm adaptation to the refactoring problem.

Quality: $f_i(S) = \sum_{j=1}^{m} M_i(e_j)$, where m is the number of code elements in the system (e.g., classes) and M_i is the i^{th} quality metric. These fitness functions are to be maximized or minimized. Quality assessment has been widely investigated in software engineering, multiple metric-based models have been introduced as mean of quality measurement. One of most widely used metrics suite in evaluating the Object-Oriented (OO) design has been defined by Chidamber and Kemerer [24]. It is generally used to analyze the internal structural properties of the system in order to mainly gather insights about the software quality. More specifically, CK metrics suite has not only been utilized in quality evaluation, but also in design flaws detection. Bansyia and Davis have introduced QMOOD [30] to measure quality based on various OO metrics. These metrics' high values can be considered good indicators for healthy design and can be calculated early during the development process to help maintain a good system standing and reduce its complexity. Martin [31] defined the instability indicator based on the ratio of afferent and efferent coupling between classes. In this context we consider a set of the metrics used in the above mentioned literature in assessing the objectives fitness functions. Table 1 summarizes the used metrics, their description and their associated objective.

Figure 2 illustrates the pseudocode of the adaptation of the P-EMO algorithm to the many-objective refactoring problem. The algorithm begins by randomly generating the population P_0 of N refactoring sequences. Once these sequences are generated, each refactoring sequence is executed on SYS, the system to be evaluated, and then evaluated based on the obtained metric values. After that, the offspring population Q_t is created by applying crossover and mutation on P_t. Each offspring is then evaluated. Next, both populations are merged to form U_t. Afterwards, we apply non-r-dominated sorting and crowding assignment to U_t. We now increment the generation index ($t \leftarrow t+1$) and we update P_t for the next generation by means of environmental selection on U_{t-1}. This preference-based evolutionary process is repeated until the termination criterion is satisfied. To better explore the search space, crossover and mutation operators are defined. For crossover, we use a single, random, cut-point crossover. It starts by selecting and splitting at random two parent solutions. Then crossover creates two child solutions by putting, for the first child, the first part of the first parent with the second part of the second parent, and, for the second child, the first part of the second parent with the second part of the first parent. This operator must respect the length limits by eliminating as necessary randomly some refactoring operations. The mutation operator picks at random one or more operations from its associated sequence and replaces them by other ones from the initial list of possible refactorings.

Table 1. Metrics used in assessing Objectives.

Metric	Description	Objective
Weighted Methods per Class (WMC)	It measures the complexity of class using the weighted sum of all class' methods. The class complexity is calculated as the sum of the McCabe's Cyclomatic Complexity (McCC) values of its methods and constructors. (McCC) is defined as $[E - N + 2]$, where E and N are the number of edges and nodes in the module's control flow graph. Low values are recommended.	Complexity
Data Access Metric (DAM)	It measures the invisibility degree of attributes in a class. It is calculated by dividing the number of private attributes by the total number attributes in a class. Ideally, all attributes should be hidden, as part of the OO. High values are recommended.	Encapsulation
Direct Class Coupling (DCC)	It counts the number of classes that a class is directly related to. This metric includes the classes directly related by attribute declaration and message passing (list of parameters) in methods. Low values are recommended.	Coupling
Tight Class Cohesion (TCC)	It counts the number of direct and indirect connections between each pair of methods that belong to the same class. The connection may be based on calls, shared attributes or classes. High values are recommended.	Cohesion[1]
Low-Level Similarity-Based Class Cohesion (LSCC)	It counts the number of direct and indirect shared attributes between each pair of methods belonging to the same class. It has been empirically shown [29] that LSCC and TCC may be conflicting thus they were both considered as separate objectives. High values are recommended.	Cohesion[2]
Measure of Functional Abstraction (MFA)	It divides the number of inherited methods by a class, by the total number of methods that belong to that class (inherited, overridden and defined). A large number of inherited methods is usually explained by either multiple inheritance or deep hierarchy that the class belongs to, in both cases, it more complex to predict its behavior. Low values are recommended.	Inheritance
Afferent Coupling (AC)	It calculates the number of entities being dependent to the measured class. These entities include classes, methods, parameter types, variable types, inheritance etc.	Stability
Efferent Coupling (EC)	Contrary to Afferent Coupling, this metric calculates the number of reachable entities by the class being measured. It includes classes, methods, parameter types, variable types, inheritance etc. The stability objective is being calculated based on these metrics: $Stability = 1 - (EC / (EC + AC))$. High values are recommended.	

In our problem formulation, all fitness functions are to be minimized. Consequently, objectives, such as Cohesion and stability, that need to be maximized, will be transformed using the duality principle. The following section will present our work evaluation in terms of research questions, settings and detailed results of our findings.

4. Experiments

In order to evaluate the feasibility and the efficiency of our approach for generating good refactoring suggestions, we conducted an experiment based on three large open source systems. We start by presenting our research questions. Then, we describe and discuss the obtained results and their threats to validity.

4.1. Research Questions

In our study, we assess the performance of our refactoring proposal by investigating whether it could generate meaningful sequences of refactorings to fix code-smells by improving as much as possible a set of seven conflicting quality metrics. Indeed, our study aims at addressing two research questions, which are defined below. We also explain how our experiments are designed to address them. The two research questions are: RQ1: To what extent can the proposed approach correct design defects? RQ2: How does the proposed P-EMO approach perform compared to a classical multi-objective algorithm (NSGA-II) and a mono-objective approach (aggregating all metrics in only one objective)?

To answer RQ1, we validate the proposed refactoring operations to fix design defects. To this end, we calculate the Defect Correction Ratio (DCR). It corresponds to the number of defects that are corrected after applying the suggested refactoring solution.

$$DCR = \frac{\#\,\text{corrected defects}}{\#\,\text{defects before applying refactorings}} \in [0,1]$$

To answer RQ2, we assessed the performance of the multi-objective algorithm NSGA-II compared to our P-EMO algorithm, and a basic evolutionary algorithm (EA) where one fitness function is used (an average of the seven metrics).

4.2. Setting

To conduct our experiments, we used three well-known open-source java projects: Xerces-J [11], JFreeChart [12] and AntApache [13]. Table 2 provides some descriptive statistics about these three programs. The design defects (Blob, Spaghetti Code and Functional Decomposition) on these systems are detected using our previous work based on genetic programming and are validated manually by a set of experts [27]. We also used these detection rules in order to detect design defects after applying the best set solutions proposed by P-EMO, NSGA-II and genetic algorithm (EA).

Table 2. Program statistics.

Systems	Release	Classes number	Defects number	KLOC
Xerces-J	v2.7.0	991	66	240
JFreeChart	v1.0.9	521	57	170
AntApache	v1.8.2	1191	82	255

In our experiments, we used and compared our P-EMO algorithm with NSGA-II and EA. For each algorithm, to generate an initial population, we start by defining the maximum solution length (maximum number of refactorings per solution). The vector length is proportional to the number of refactorings that are considered and the size of the program to be refactored. A higher number of operations in a solution does not necessarily mean that the results will be better. Ideally, a small number of operations should be sufficient to provide a good trade-off between the fitness functions. This parameter can be specified by the user or derived randomly from the sizes of the program and the used refactoring list. For all algorithms, we set the maximum vector length to 500 refactorings, the population size to 50 individuals (refactoring solutions) and the termination criterion to 100 generations.

We calculated an average *DCR* for all these comparisons over 30 runs. In fact, since the used algorithms are stochastic meta-heuristics, they produce different results every run when applied to the same problem instance. To this end, the use of rigorous statistical tests is then essential to provide support for the conclusions derived by analyzing such data. Thus, we used the p-values of the Wilcoxon rank sum test [28] when comparing between different algorithms over 30 runs. We used a different reference point at each run to verify the stability of our P-EMO approach.

4.3. Results

Table 3 summarizes our findings. To evaluate the efficiency of our approach, we compared our results to those produced by NSGA-II (based on 7 metrics/objectives) [2] and EA (an average of the 7 metrics as one objective). In [26], Harman et al. proposed a multi-objective approach that uses two quality metrics to improve (coupling between objects CBO, and tight class cohesion) after applying the refactorings sequence. This corresponds to our NSGA-II implementation (but with 7 metrics). As described in Table 2, the majority of suggested refactorings significantly improve the code quality. Over an average of 30 runs, it is clear that P-EMO performs better than NSGA-II and EA. P-EMO fixed 86%, 92% and 89% of design defects respectively on Xerces, JFreeChart and Ant-Apache. The average *DCR* scores for NSGA-II and EA are around half of the average *DCR* score for P-EMO. In fact, the average DCR score for NSGA-II and EA is around 55% on the three systems.

Table 3. Precision median values of P-EMO, NSGA-II, and EA (average of 7 metrics) over 30 independent simulation runs using 7 objective (metrics).

	P-EMO		**NSGA-II**		**EA**	
Systems	**DCR (%)**	***p-value***	**DCR (%)**	***p-value***	**DCR (%)**	***p-value***
Xerces	**86**	**< 0.01**	**57**	**< 0.01**	**49**	**< 0.01**
JFreeChart	**92**	**< 0.01**	**53**	**< 0.01**	**51**	**< 0.01**
Ant-Apache	**89**	**< 0.01**	**56**	**< 0.01**	**52**	**< 0.01**

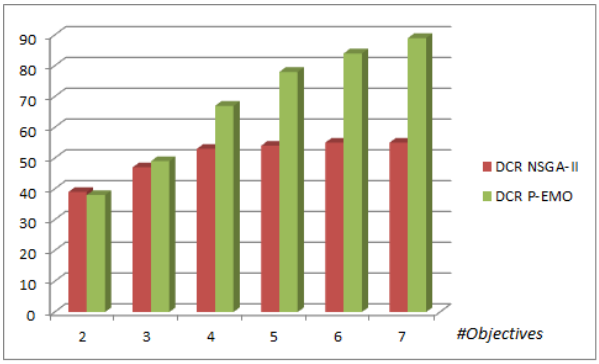
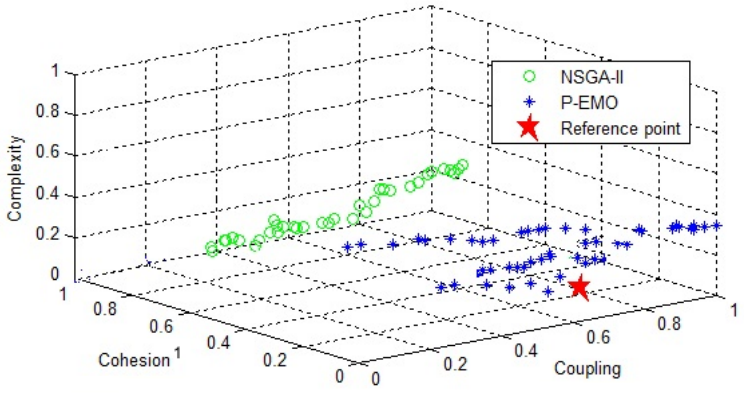


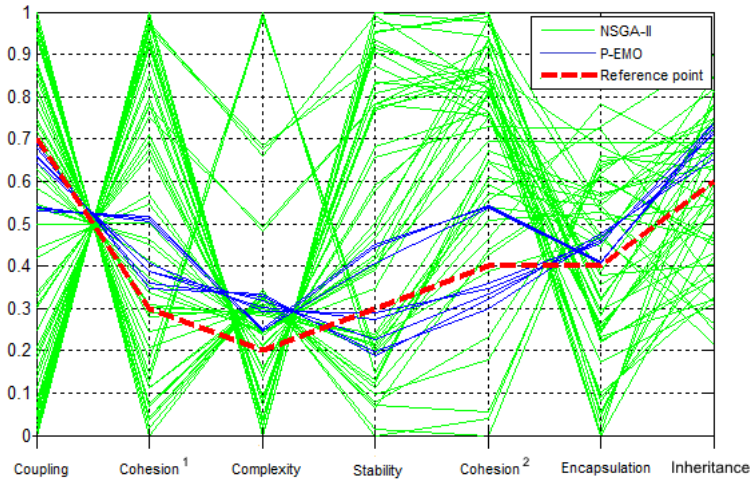
Fig.3. P-EMO vs NSGA-II with different number of objectives: DCR median values on the three systems.

We also compared between P-EMO and NSGA-II correction results based on the number of used objectives. As described in Figure 3, NSGA-II and P-EMO obtain similar results when the number of objectives is lower than 4, but P-EMO provides much better results (higher DCR scores) when the number of metrics (objectives) becomes higher than 4. For example, with more than 6 objectives, NSGA-II fixed half of the defects fixed by P-EMO. Thus, we can conclude that it is important to consider a high number of metrics to improve in order to fix a high number of defects. This is evident since design defects have different symptoms that can be characterized with at least 7 different metrics. The DCR median values are calculated over 30 runs where different reference points are used at each run to ensure the stability of our results. A reference point in the context of our experiments is a 7-dimensional vector where each dimension is an *ideal* value suggested by the user for a specific metric. In addition, at each run we select randomly metrics to consider as objectives.

Figure 4 illustrates a comparison between NSGA-II and P-EMO for the 3-objective refactoring problem and the 7-objective one. The population size and the number of generations are set to 50 and 100 respectively for both algorithms in order to ensure fairness of comparison (5000 evaluations). We see from Figure 4 (a-b) that the P-EMO algorithm provides better refactoring sequences than NSGA-II. In fact, most P-EMO algorithm solutions Pareto dominate NSGA-II ones for the tri-objective case (cf. fig 4(a)), while matching user's preferences. For the 7-objective case, it is not easy to visualize the performance of one algorithm versus another from a dominance viewpoint. However, according to the parallel coordinate plot (cf. fig. 4(b)), NSGA-II solutions are very scattered in the objective space, which is not the case for the P-EMO algorithm which provides solutions that are very close to the user's reference point. We conclude that P-EMO outperforms NSGA-II not only in terms of matching user's preferences but also in terms of convergence. To sum up, the P-EMO algorithm is demonstrated to find better trade-off solutions for the 3- and 7-objectives cases while matching user's reference point, thereby facilitating the decision making task in selecting the best refactoring sequence.



(a)



(b)

Fig.4. P-EMO vs NSGA-II for: (a) the 3-objective case with the reference point (0.7, 0.1, 0.1) and (b) the 7-objective case with the reference point (0.7, 0.3, 0.2, 0.3, 0.4, 0.4, 0.6).

4.4. Discussion

We noticed that our technique does not have a bias towards the correction of specific design defect types. As shown in Figure 5, in all systems we had an almost equal distribution of each code-smell type (SCs, Blobs, and FDs). On some systems such as Xerces, the distribution is not as balanced. This is principally due to the number of actual defect types in the system and the choice of the reference point. Having a relatively good distribution of defect types corrected is useful for a quality engineer. Overall, all three code smell types are detected with good DCR scores in the different systems (more than 80%).

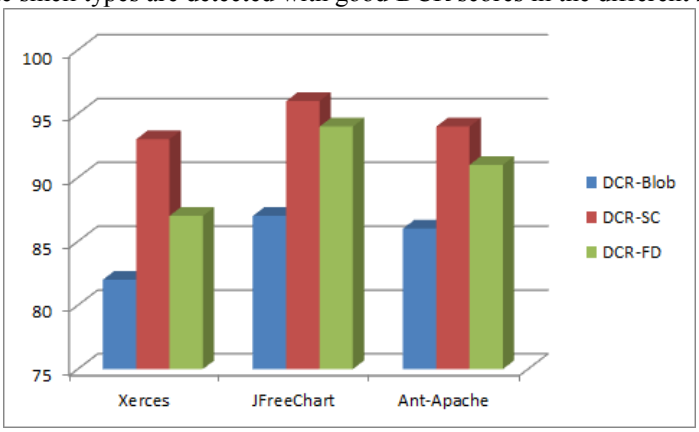


Fig.5. The impact of defect types on DCR median values of the three systems.

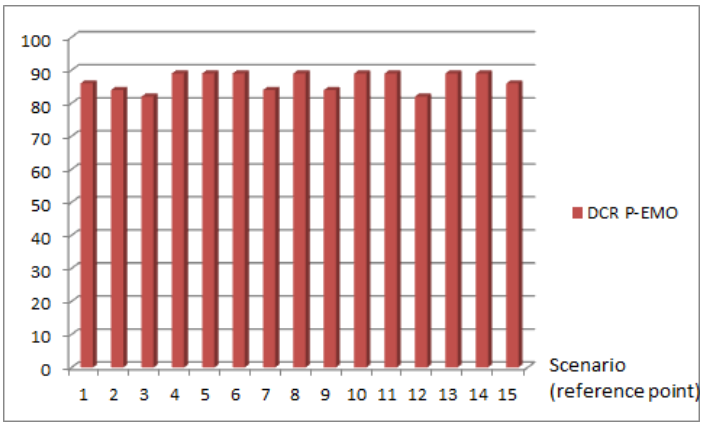


Fig.6. The impact of reference points on DCR: DCR median values on the three systems.

One of the important inputs of our P-EMO algorithm is the reference point that needs to be selected by the user. We studied the impact of different reference points selected by different users at each run using 15 scenarios. In each scenario, we asked the user to choose a reference point (7 *ideal* metrics value). Figure 6 confirms that the results are stable in the 15 scenarios where the DCR median value on the three systems is usually between 82% and 89%. Thus, we can conclude that our P-EMO technique is insensitive to a slight variation in reference points. However, the user can choose a reference point based on his goal. For example, to detect mainly blob defects, the user can specify a very low number of methods per class as main preference with acceptable values for the remaining metrics.

5. Conclusion

In this paper we introduced a new approach for code refactoring based on preference-based evolutionary multi-objective optimization (P-EMO). The experimental results indicate that P-EMO performs much better than the classical multi-objective algorithm NSGA-II and mono-objective EA. The paper provides also a set of topics for open problems in software modelling and a description of some of the benefits that may accrue through the use of many-objective optimization techniques such as P-EMO. As part of the future work, we will work on adapting P-EMO to additional SE problems and performing more comparative studies.

References

[1] Saxena, D.K., Duro, J.A., Tiwari, A., Deb, K., Zhang, Q., 2013. Objective Reduction in Many-objective Optimization: Linear and Nonlinear Algorithms. IEEE Transactions on Evolutionary Computation, 17, 1, pp. 77–99.

[2] Deb, K., 2002. Multi-Objective Optimization using Evolutionary Algorithms. John Wiley and Sons, Ltd, New York, USA.

[3] Deb, K., Pratap, A., Agarwal, S., Meyarivan, T., 2002. A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II. IEEE Transactions on Evolutionary Computation, 6, 2, pp. 182–197.

[4] López Jaimes, A., Coello Coello, C.A., Barrientos, J. E. U., 2009. Online Objective Reduction to Deal with Many-objective Problems. In: Fifth international conference on Evolutionary Multicriterion Optimization, pp. 423–437.

[5] Ben Said, L., Bechikh, S., Ghédira, K., 2010. The r-Dominance: A New Dominance Relation for Interactive Evolutionary Multicriteria Decision Making. IEEE Transactions on Evolutionary Computation, 14, 5, pp. 801–818.

[6] Bechikh, S., Ben Said, L., Ghédira, K., 2011. Searching for Knee Regions of the Pareto Front using Mobile Reference Points. Soft Computing – A Fusion of Foundations, Methodologies and Applications, 15, 9, pp. 1807–1823.

[7] Di Pierro, F., Khu, S-T., Savic, D.A., 2007. An Investigation on Preference Order Ranking Scheme for Multiobjective Evolutionary Optimization. IEEE Transactions on Evolutionary Computation, 11, 1, pp. 17–45.

[8] Zhang, Q., Li, H., 2007. MOEA/D: A Multiobjective Evolutionary Algorithm Based on Decomposition. IEEE Transactions on Evolutionary Computation, 11, 6, pp. 712–731.

[9] Harman, M., Mansouri, S-A., Zhang, Y., 2012. Search-based software engineering: Trends, techniques and applications. ACM Computing Surveys, 45-61.

[10] Mens, T. and Tourwé, T., 2004. A Survey of Software Refactoring. Transactions on Software Engineering, pp. 126-139.

[11] Xerces project (online): http://xerces.apache.org/

[12] JFreeChart (online): www.jfree.org/jfreechart/

[13] AntApache (online): http://ant.apache.org/

[14] Deb, K., 2012. Advances in evolutionary multi-objective optimization. In: Proceedings of Symposium on Search-Based Software Engineering, pp. 1–26.

[15] Zitzler, E., Künzli, S., 2004. Indicator-Based Selection in Multiobjective Search. In Conference on Parallel Problem Solving from Nature (PPSN VIII), Vol 3242, pp. 832– 842.

[16] Sayyad A., Menzies, T., Ammar H., 2013. On the value of user preferences in search-based software engineering: A case study in software product lines. In: international conference on software engineering.

[17] Fowler, M., Beck, K., Brant, J., Opdyke, W., Roberts, D., 1999. Refactoring – Improving the Design of Existing Code, 1st edition, Addison-Wesley.

[18] Mkaouer, M.W., Kessentini, M., Bechikh, S., Tauritz, D., 2013. Preference-based Multi-objective Software Modelling, in Proceedings of the First International Workshop on Combining Modelling and Search-Based Software Engineering.

[19] Weissgerber, P., Diehl S., 2006. Identifying Refactorings from Source-Code Changes; in Proceedings of IEEE International Conference on Automated Software Engineering, pp. 231-240.

[20] Kim, M., Notkin, D., Grossman, D., Wilson, G. Jr., 2012. Identifying and Summarizing Systematic Code Changes via Rule Inference; in Transactions on Software Engineering.

[21] Prete, K., Rachatasumrit, N., Sudan, N., Kim, M., 2010. Template-based reconstruction of complex refactorings; in: Proceedings of IEEE International Conference on Software Maintenance, pp. 1-10.

[22] Fleurey, F., Steel, J., Baudry, B., 2004. Validation in Model-Driven Engineering: Testing Model Transformations, In 15th IEEE International Symposium on Software Reliability Engineering.

[23] Czarnecki, K., Helsen, S., 2006. Feature-based survey of model transformation approaches. IBM Syst. J. (Special Issue on Model-Driven Software Development 45(3), pp. 621–645.

[24] Fenton, N., Pfleeger, S. L., 1997. Software Metrics: A Rigorous and Practical Approach, 2nd ed. London, UK: International Thomson Computer Press.

[25] O'Keeffe, M., Ó Cinnéide, M., 2008. Search-based refactoring for software maintenance, Journal of Systems and Software, 81 (4), pp. 502 – 516.

[26] Harman, M., Tratt, L., 2007. Pareto optimal search based refactoring at the design level, In Proceedings of The Genetic and Evolutionary Computation Conference, pp. 1106-1113.

[27] Kessentini, M., Kessentini, W., Sahraoui, H., Boukadoum, M., and Ouni, A., 2011. Design Defects Detection and Correction by Example. The 19th International Conference on Program Comprehension, pp 81-90.

[28] Wilcoxon, F., Katti, S. K., Roberta, A., 1973. Wilcox. Critical Values and Probability Levels for the Wilcoxon Rank Sum Test and the Wilcoxon Signed-rank Test, In Selected Tables in Mathematical Statistics, Volume I, American Mathematical Society, pp. 171-259.

[29] Ó Cinnéide, M., Tratt, L., Harman, M., Counsell, S., Moghadam, I. H., 2012. Experimental assessment of software metrics using automated refactoring. In Proceedings of the ACM-IEEE international symposium on Empirical software engineering and measurement (ESEM '12). ACM, New York, NY, USA, pp. 49-58.

[30] Bansiya, J., Davis, C. G., 2002. A Hierarchical Model for Object-Oriented Design Quality Assessment. IEEE Transactions on Software Engineering. 28, 1, pp. 4-17. DOI=10.1109/32.979986 http://dx.doi.org/10.1109/32.979986

[31] Chidamber, S. R., Kemerer, C. F., 1994. A Metrics Suite for Object Oriented Design. IEEE Transactions on Software Engineering. 20, 6, pp. 476-493.

[32] Martin, R., 1994. OO Design Quality Metrics-An Analysis of Dependencies, position paper, in Proceedings of Workshop on Pragmatic and Theoretical Directions in Object-Oriented Software Metrics.