This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TEVC.2017.2778560, IEEE Transactions on Evolutionary Computation

Integrating Weight Assignment Strategies with NSGA-II for Supporting User Preference Multi-Objective Optimization

Shuai Wang, Shaukat Ali, Tao Yue, *Member, IEEE*, Marius Liaaen

***Abstract*—Driven by the needs of several industrial projects on the applications of multi-objective search algorithms, we observed that user preferences must be properly incorporated into optimization objectives. However, existing algorithms usually treat all the objectives with equal priorities and do not provide a mechanism to reflect user preferences. To address this, we propose an extension—User-Preference Multi-Objective Optimization Algorithm (UPMOA), to the most commonly applied, non-dominated sorting genetic algorithm II (NSGA-II) by introducing a user preference indicator $\wp$, based on existing weight assignment strategies (e.g., Uniformly Distributed Weights (UDW)). We empirically evaluated UPMOA using four industrial problems from three diverse domains (i.e., Communication, Maritime and Subsea Oil&Gas). We also performed a sensitivity analysis for UPMOA with 625 algorithm parameter settings. To further assess the performance and scalability, 103500 artificial problems were created and evaluated representing 207 sets of user preferences. Results show that the UDW strategy with UPMOA achieves the best performance and UPMOA significantly outperformed other three multi-objective search algorithms, and has the ability to solve problems with a wide range of complexity. We also observed that different parameter settings led to the varied performance of UPMOA, thus suggesting that configuring proper parameters is highly problem-specific.**

***Index Terms*— User Preference, Multi-objective Optimization, Weight Assignment Strategies, Search Algorithms**

I. Introduction

Over the last two decades, Search-Based Software Engineering (SBSE) has been widely used as an efficient means for solving a wide range of software engineering problems such as in software testing [1][2][3]. The fundamental principle of SBSE is to encode software engineering problems into optimization problems and solve them with search techniques (e.g., Genetic Algorithms) in a cost-effective manner [4][5][6]. According to a well-known SBSE repository hosted by the CREST center [45], in total 1657 published works tackled a variety of challenges faced by different domains (e.g., communication [7]).

Due to the increased complexity of modern software, Multi-objective Optimization Problems (MOPs) become even more critical since many software engineering problems are multi-objective [2][5][7]. Researchers have proposed to apply various multi-objective search algorithms [8][20][21] (e.g., non-dominated sorting genetic algorithm II (NSGA-II)) to address different MOPs such as requirements assignment [48] test case selection [5] and test case generation [6]. However, existing works usually treat all the objectives with equivalent priorities and do not take user preferences into account. Based on the experience of our industrial collaborations with several companies (e.g., Cisco Systems, Norway [12]), we learned that it is critical to incorporate user preferences when dealing with MOPs in practice. For instance, when working on the test execution system optimization for automated testing of a product line of Videoconferencing Systems (VCSs) namely *Saturn* at Cisco, we observed that priorities of optimization objectives should be distinguished based on domain knowledge of test engineers rather than treating all the objectives equally. For example, when eliminating redundant test cases for testing a new VCS (coined as the *test suite minimization problem*) [22], the fault detection capability of test cases should have higher priority than the number of test cases that are reduced since it does not make much sense in practice for test engineers to put focus on the number of redundant test cases to be eliminated without considering the fault detection capability.

In the past, we studied the existing multi-objective search algorithms (e.g., NSGA-II) [8][11] and found that most of them treat objectives equally, which requires the selection from users based on their preferences. When applying these algorithms in practice (with user preferences), at least three challenges are posed: 1) Optimal solutions that satisfy user preferences may not be in non-dominated solutions and thus the solutions may be suboptimal; 2) Users have to manually choose the "best" ones based on their preferences, which may take a lot of effort when facing hundreds of non-dominated solutions. This challenge is particularly raised by our

This work was supported by the Research Council of Norway (RCN) funded Certus SFI. Shuai Wang is also supported by RFF Hovedstaden funded MBE-CR project. Tao Yue and Shaukat Ali are also supported by RCN funded Zen-Configurator project, the EU Horizon 2020 project funded U-Test, RFF Hovedstaden funded MBE-CR project and RCN funded MBT4CPS project.

S. Wang, Simula Research Laboratory, Martin Linges vei 25, 1364 Fornebu, Norway (shuai@simula.no). Corresponding author.

S. Ali, Simula Research Laboratory, Martin Linges vei 25, 1364 Fornebu, Norway (shaukat@simula.no).

T. Yue, Simula Research Laboratory, department of Informatics, University of Oslo, Martin Linges vei 25, 1364 Fornebu, Norway (tao@simula.no).

M. Liaaen, Cisco Systems, Philip Pedersens vei 1, 1366 Lysaker, Norway (marliaae@cisco.com).

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TEVC.2017.2778560, IEEE Transactions on Evolutionary Computation

industrial collaborators (e.g., test engineers at Cisco) since they usually prefer an approach that can directly produce the solutions that satisfy all the required preferences instead of taking extra manual effort as follow-up; 3) Weight-based search algorithms (e.g., Random-Weighted Genetic Algorithm (RWGA)) can easily specify user preferences by assigning weights to each objective, but it is challenging to determine a good weight assignment strategy [8]. Moreover, weight-based search algorithms solve multi-objective optimization problems by converting them to single-objective optimization problems, which may lose a set of non-dominated optimal solutions [8].

Furthermore, several existing works [36][37][38] focus on integrating weights generated by a specific weight distribution function into existing multi-objective search algorithms (e.g., NSGA-II) for incorporating user preferences. However, user preferences are usually provided in a qualitative way in practice (e.g., fault detection capability has the highest priority) and it is quite difficult to determine beforehand which set of quantitative weights is the most appropriate one. Thus, it requires thorough evaluation of various weight sets that satisfy user preferences, which is not addressed in the literature [37-44]. In addition, there is less evidence to show that these approaches work well for solving software engineering problems with user preferences. Particularly, very few works have focused on assessing the scalability of the approaches when solving problems of varying complexity.

To address the above-mentioned challenges, this paper proposes an improved User-Preference Multi-Objective Optimization Algorithm (UPMOA) by extending the most commonly used multi-objective search algorithm NSGA-II, which has also shown promising results in the literature [5][29]. The main idea lies in defining a user-preference indicator (called $\mathcal{P}$) based on existing weight assignment strategies (e.g., Fix Weights assignment strategy), which is used to assess the quality of solutions by considering user preferences. We also propose a mechanism to integrate $\mathcal{P}$ into NSGA-II such that UPMOA can reflect user preferences when guiding search towards finding optimal solutions. To evaluate UPMOA, we employ four industrial software engineering problems with different sets of user preferences from three diverse industrial domains (i.e., *Communication*, *Maritime* and *Subsea Oil&Gas*) [22][48][49][50]. To further assess the performance and scalability of UPMOA, in total 103500 artificial problems were created and UPMOA was evaluated with 207 different sets of user preferences.

More specifically, the empirical evaluation was conducted from three aspects. First, we generated in total 625 parameter settings derived from four algorithm parameters (i.e., crossover rate, mutation rate, population size and maximum number of fitness evaluations) and determined the best weight assignment strategy for $\mathcal{P}$ by comparing three existing weight assignment strategies with respect to each parameter setting, i.e., Fixed Weights (*FW*), Randomly-Assigned Weights (*RAW*) and Uniformly Distributed Weights (*UDW*). Results show that *UDW* can help UPMOA to achieve the best performance for all the 625 parameter settings. Second, we compared UPMOA with three representative multi-objective search algorithms in terms of each parameter setting [5][22][26], i.e., NSGA-II, RWGA and UDW-(1+1) Evolutionary Algorithm (EA). Results show that UPMOA significantly outperformed the others for all the parameter settings and has the capability of solving problems with a wide range of complexity. Third, we conducted a sensitivity analysis for UPMOA using the 625 parameter settings. Results show that different parameter settings indeed led to large variance of the performance of UPMOA and setting proper parameter values is highly problem-specific. We also observed that the default settings suggested by the literature [52] were not able to achieve the best performance of UPMOA for all the four industrial problems, which suggests the need for parameter tuning in practice.

This paper is an extension of a conference paper [47] and the significant differences with the conference version include: 1) The evaluation has been further extended to generalize the results by applying UPMOA to two additional real world case studies from the *Maritime* and *Subsea Oil&Gas* domains; 2) The performance and scalability of UPMOA was further evaluated with additional 39500 (in total 103500) artificial problems inspired from the two additional industrial problems with 79 sets of user preferences; 3) UPMOA was extensively evaluated with in total 625 parameter settings; 4) A sensitivity analysis was conducted using the 625 parameter settings; and 5) The background and case studies have been improved with illustrations, e.g., examples showing how the three weight assignment strategies work (Section II), examples of non-optimal solutions (Section II) and examples of artificial problems (Section IV). The related work has been updated to include more comparisons with the literature.

The rest of the paper is organized as follows: Section II describes the relevant background and Section III details UPMOA. Our four industrial case studies and 103500 artificial problems are described in Section IV. Section V presents the empirical evaluation and Section VI provides the related work. The paper is concluded in Section VII.

II. Background

In this section, we first introduce four industrial problems that motivated this work (Section A) followed by presenting the three weight assignment strategies (Section B).

A. Industrial Problems

This work is mainly driven by the needs of several projects from diverse industrial domains in a research-based innovation center named Certus Software Verification and Validation Centre[1]. Three different industrial domains are involved, which are 1) *Communication*: focusing on optimizing the test execution system for automated testing of the *Saturn* product line of Cisco's VCSs [12][27]; 2) *Maritime*: aims to develop a new test case execution system for testing robustness of real-time embedded systems deployed in diverse maritime applications [49]; and 3) *Subsea Oil&Gas*: focuses on allocating a large number of requirements to various stakeholders for inspection [48].

[1] http://certus-sfi.no

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TEVC.2017.2778560, IEEE Transactions on Evolutionary Computation

Communication Domain. *Saturn* has a set of VCSs (e.g., C20 and C90) and testing *Saturn* has a test case repository that includes more than 2000 test cases. Notice that not all test cases are relevant for a given VCS and new test cases are constantly developed for *Saturn* every day. When a new VCS needs to be tested, we learn that two industrial problems are required to be addressed, i.e., 1) ***Cost-effective Test Suite Minimization***: eliminating redundant test cases without significantly reducing the effectiveness (e.g., feature pairwise coverage) at the same time minimizing the cost (execution time); 2) ***Cost-effective Test Case Prioritization***: prioritizing test cases into an optimal order within a limited test resource budget with the aim to maximizing the effectiveness (e.g., fault detection capability) and minimizing the cost (e.g., execution time of test cases)[4][7][22]. However, based on the expertise of VCS testing, we discovered that the test engineers have preferences for the objectives depending on the test requirements. For instance, fault detection capability is more important than the total execution time taken by the test cases since the ultimate goal is to detect faults rather than reducing execution time of test cases. Notice that user preferences for both industrial problems are detailed in Section IV.

Maritime Domain. Our industrial partner plays a key role as one of the leading maritime service companies in Norway [49]. The team we are collaborating with focuses on robustness testing of embedded systems and a large number of test cases have been designed and implemented by test engineers. During one particular test cycle, we learn that it is practically infeasible to execute all the test cases when there is a limited time budget (e.g., 10 hours in our case) and thus the industrial problem to be addressed in this context can be coined as ***Cost-effective Test Case Selection***, i.e., select a maximum number of test cases within a given time budget for execution while maximizing pre-defined effectiveness objectives [49]. As we investigated the problem in a deeper way, we learnt that the process of selecting test cases should be able to incorporate user preferences of the optimization objectives according to the domain knowledge, e.g., achieving high fault detection capability should have higher priority than the number of test cases to be executed (see Section IV for more details for user preferences). From the perspective of test engineers, test case selection without considering user preferences may result in solutions that are suboptimal. For instance, using NSGA-II to address this problem, we obtained two non-dominated solutions: *A* and *B*. *A* holds the fault detection capability value as 0.8 (see Section IV.C for more details to calculate fault detection capability) and the number of selected test cases as 158 while *B* has the values of 0.3 and 37 for these two objectives. In practice, *A* is better than *B* as *A* can manage to detect more faults than *B* though *A* requires executing more test cases. Thus, when taking user preferences into account, *B* becomes not optimal.

Subsea Oil&Gas Domain. The application domain is about designing and developing subsea production systems (SPSs) that control and monitor physical processes such as oil and gas production platforms [48]. At the early phase of developing such a SPS, stakeholders are required to review, and inspect a large number of requirements (e.g., contracts, regulations) and thus the problem to be addressed in this context is ***Cost-effective Requirements Allocation***, i.e., allocating requirements to stakeholders in order to maximize stakeholders' familiarities to assigned requirements while balancing the overall workload of each stakeholder. Through further investigation, we observed that the optimization objectives hold different priorities that should be taken into account during the process of allocating requirements to stakeholders. For instance, in certain contexts, the familiarities and overall workload are considered more important than the number of requirements assigned to the stakeholder.

Based on the above discussions, there is a strong need in practice to incorporate user preferences into the search process. These observations inspired and encouraged us to extend the most widely used multi-objective search algorithm NSGA-II to meet user preferences when guiding search towards finding optimal solutions.

B. Weight Assignment Strategies

Fixed Weights (FW) Assignment Strategy. *FW* assigns pre-defined quantitative weights (between 0 to 1) to each objective (e.g., 0.3 to the objective of fault detection capability (*FDC*)) based on the domain knowledge of user preferences from experts (e.g., test engineers at Cisco) [8]. The assigned weights to each objective will not be changed during the entire search process. However, based on our experience, domain experts (e.g., test engineers) usually specify user preferences in a qualitative way (e.g., *FDC* is more important than overall test case execution time (*OET*)) rather than in a quantitative manner (e.g., *FDC* has 0.2 higher priority than *OET*).

Randomly-Assigned Weights (RAW) Assignment Strategy. *RAW* randomly generates normalized weights (between 0 and 1) for each optimization objective, which satisfy required constraints based on user preferences. For example, we suppose a constraint $w_2 > w_1$ representing one objective (e.g., *FDC*) is more important than another objective (e.g., *OET*). *RAW* first generates a value between 0 and 1 for w_2 followed by generating a second value for w_1 between 0 and w_2 for satisfying the constraint. During the search process, *RAW* assigns normalized weights (meeting the constraints) to each objective and dynamically changes the weights at each generation until the best solution is obtained or the termination criterion is met. *RAW* helps to explore the search space with diverse search directions (one set of weights determines a specific search direction) with the aim to reduce the chances of losing optimal solutions [8]. However, *RAW* cannot ensure that all the potential weights can have equivalent possibilities to be generated and thus each search direction cannot be explored in a uniform way during the search process [26].

Uniformly Distributed Weights (UDW) Assignment Strategy. To address the challenge of *RAW*, our previous work [26] proposed *UDW* with the aims to: 1) generate normalized weights for objectives while satisfying the required constraints (Similarly as *RAW*); and 2) guarantee the uniformity for the selection of weights so that each search direction can be explored with equal possibilities. The core of *UDW* is to form each individual weight domain (for each objective between 0 and 1) as Cartesian product for the input domain followed by pre-computing subdomains of the input domain [26]. More specifically, as shown in Fig. 1, by employing an arbitrary

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TEVC.2017.2778560, IEEE Transactions on Evolutionary Computation

division parameter k, *UDW* first initializes a set of weight tuples WK as empty (*Step 1*) and divides the input domain W of weights into k^m equivalent subdomains (*Step 2*), where m is the number of weight variables to be assigned to objectives (i.e., the number of optimization objectives). *UDW* further employs a systematic refutation to eliminate subdomains, which do not satisfy the inputted constraint C (*Step 3*). Afterwards, *UDW* selects one subdomain from the remaining ones randomly and generates a uniform tuple of weights that complies with the required constraint set C. This process will be repeated until the number of generated tuple sets reach to K (*Step 4*). Last, the generated weight tuples WK will be returned, which will be used as weights assigned to optimization objectives (*Step 5*).

***UDW*: Uniformly-Distributed Weight Strategy**
Input: $W = \{w_1, w_2, \dots, w_m\}$, $C = \{c_1, c_2, \dots, c_n\}$, division parameter k (Integer) and length of the expected weight tuples K (Integer)
Output: $W_1, W_2, \dots, W_K$ for Objectives O or $\emptyset$
Step 1: $WK := \theta$
Step 2: $(WD_1, \dots, WD_{k^m}) := Divide(\{w_1, w_2, \dots, w_m\}, k)$;
Step 3: forall $WD_i \in (WD_1, \dots, WD_{k^m})$ **do**
if WD_i is fully unsatisfiable w.r.t. C **then** remove WD_i from $(WD_1, \dots, WD_{k^m})$;
Step 4: Suppose $\{WD'_1, WD'_2, \dots, WD'_p\}$ is the remaining list of domains;
if $p \geq 1$ **then**
while $K > 0$ **do**
choose WD'_j uniformly and randomly from $\{WD'_1, WD'_2, \dots, WD'_p\}$;
choose W uniformly and randomly from WD'_j;
if W satisfies C **then** add W to WK; $K := K - 1$;
Step 5: return WK for Objectives O.

Fig. 1. Pseudocode for *UDW*

For instance, suppose we have two objectives (i.e., *OET* and *FDC*) and one constraint on their weights (ranging from 0 to 1), which is $w_2 > w_1$ (representing *FDC* has a higher priority than *OET*). Given the division parameter is k=2 (which can be any integer number theoretically), the expected length of weight tuples K=100 (i.e., in total producing 100 sets of weights for *OET* and *FDC*), *UDW* first divides the entire input domain into four equivalent subdomains (i.e., $2^2 = 4$), i.e., subdomain 1 ($w_1 \in [0,0.5)$ and $w_2 \in [0,0.5)$), subdomain 2 ($w_1 \in [0,0.5)$ and $w_2 \in [0.5,1]$), subdomain 3 ($w_1 \in [0.5,1]$ and $w_2 \in [0,0.5)$) and subdomain 4 ($w_1 \in [0.5,1]$ and $w_2 \in [0.5,1]$). Second, the subdomains that violate the constraint (i.e., the subdomain 3) will be eliminated. Afterward, *UDW* randomly selects a subdomain from the subdomains 1, 2, 4 and generates a uniform tuple of weights for the two objectives that satisfy the defined constraint. The process of selecting subdomains and generating uniform tuples of weights is repeated until there are 100 tuples of weights produced by *UDW*. Note that as compared with *RAW*, the main improvement for *UDW* is to ensure that generated weights are uniformly distributed and thus each search direction can be explored with the same probability.

III. The UPMOA Algorithm

The core idea for UPMOA lies in defining a user-preference indicator named $\mathcal{P}$ used to assess the quality of a specific solution s based on user preferences for various objectives. First, we classify all the objectives O into two categories, i.e., effectiveness objectives $o\_effect = \{o\_effect_1, o\_effect_2 \dots o\_effect_{n_{effect}}\}$ that need to be maximized (e.g., fault detection capability) and cost objectives $o\_cost = \{o\_cost_1, o\_cost_2 \dots o\_cost_{n_{cost}}\}$ that require to be minimized (e.g., execution time). The function of $\mathcal{P}$ is defined as below where s refers to an individual solution. A higher value of $\mathcal{P}(s)$ indicates a better solution s.

$$\mathcal{P}(s) = \sum_{i}^{n_{effect}} w_i * Norm(o\_effect_i(s)) + \sum_{j}^{n_{cost}} w_j * (1 - Norm(o\_cost_i(s)))$$

Where $w = \{w_1, w_2, w_3 \dots w_n\}$ is the set of weights assigned to each objective. Notice that w should satisfy all the user preferences for the objectives. For instance, if o_1 (e.g., fault detection capability) has higher priority than o_2 (e.g., overall execution time) based on the domain expertise, w_1 should be greater than w_2, i.e., $w_1 > w_2$. All the weights should of course satisfy the constraint: $\sum_{i=1}^{n} w_i = 1$. In addition, to be in the same magnitude, the values of the cost/effectiveness objectives should be normalized using the normalization function: $Norm(x) = \frac{x}{x+1}$ [19]. We choose this normalization function because the upper limits of the values are unknown for some of the objectives beforehand [19]. For instance, the overall execution time for test cases can range from several minutes to several hours and thus it is practically impossible to determine the upper threshold of the overall execution time. Notice that our approach does not prevent using other normalization functions if applicable. Once the user-preference indicator $\mathcal{P}$ is defined, the pseudo code for UPMOA is shown as Fig. 2.

Algorithm UPMOA
1. Create a random population pop with the size of N.
2. Set generation number t as 1 and population $P_t = pop$;
3. Set offspring set $Q_t = \phi$;
4. Select individuals from P_t using the selection operator, create offspring by applying crossover and mutation operator and add the offspring into Q_t;
5. Combine population P_t and offspring Q_t as R_t;
6. Sort R_t into several non-dominated fronts $\mathcal{F} = \{\mathcal{F}_1, \mathcal{F}_2, \dots, \mathcal{F}_n\}$ using the ranking algorithm from NSGA-II;
7. $P_{t+1} = \phi$ and i=1
8. Until $|P_{t+1}| + |\mathcal{F}_i| \leq N$
$P_{t+1} = P_{t+1} \cup \mathcal{F}_i$; $i = i + 1$;
9. Generate $w = \{w_1, w_2, w_3 \dots w_n\}$ satisfying all the user preferences for the objectives O;
10. For each individual s in $\mathcal{F}_i$, assign a $\mathcal{P}(s)$ value based on the defined user-preference indicator;
11. Sort $\mathcal{F}_i$ based on $\mathcal{P}(s)$ in an ascending order;
12. Select individuals from $\mathcal{F}_i$ according to the order until $|P_{t+1}| = N$;
13. $t = t + 1$;
14. Repeat from step 3 until the termination condition is satisfied.

Fig. 2. Pseudocode for UPMOA

Similarly to the original NSGA-II, UPMOA starts with creating a random population pop for initial population P_1 that includes N individual solutions (*Steps 1* and *2*). For each generation t, an offspring set Q_t will be first created by applying selection, crossover and mutation operators to the individuals of parent population P_t (*Step 4*). Afterwards, best individual solutions will be chosen from the combinations of

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TEVC.2017.2778560, IEEE Transactions on Evolutionary Computation

P_t and Q_t and filled into the next population P_{t+1} (*Steps 5-12*). More specifically, a set called R_t will first be created by adding up P_t and Q_t (*Step 5*), followed by sorting R_t into a set of non-dominated fronts $\mathcal{F} = \{\mathcal{F}_1, \mathcal{F}_2, \dots, \mathcal{F}_n\}$, using the ranking algorithm from NSGA-II [20] (*Step 6*). *Step 8* aims at filling individual solutions from the ordered fronts $\mathcal{F}$ into P_{t+1} until there is a front $\mathcal{F}_i$ that its individual solutions cannot be fully added into P_{t+1} when the size of P_{t+1} becomes to *N*. As for *Step 9* and *10,* a set of weights w will be generated based on a weight assignment strategy that satisfies the user preferences for the objectives *O*. Using the generated weights; a value of $\wp$ will be calculated and assigned to each individual solution *s* belonging to $\mathcal{F}_i$. Notice that at this stage, we are not aware which weight assignment strategy (Section II.A) is the best for the user-preference indicator $\wp$ and thus all the three weight assignment strategies will be evaluated (Section V) to determine the best strategy for $\wp$. For the *FW* strategy, the same weight set w provided by the experts will be applied for all the generations during the search. As for the *RAW* and *UDW* strategies (Section II.A), for each generation, a new set of w will be generated that meets the user preferences for the objectives *O*. In *Step 11*, the individual solutions in $\mathcal{F}_i$ will be sorted in an ascending order (*Step 11*) and the best individual solutions will be selected and filled into P_{t+1} according to the order until the size of P_{t+1} meets *N* (*Step 12*). The algorithm will keep running until the termination condition is satisfied, e.g., reaching the maximum number of generations (*Step 14*).

The main difference between UPMOA and NSGA-II is that the crowd distance indicator of NSGA-II is replaced with our defined user-preference indicator $\wp$ (*Steps 9-12*), i.e., individual solutions are added into the next population based on $\wp$ rather than the crowd distance indicator. This can be justified based on the fact that our aim is to guide the search and output optimal solutions considering various user preferences rather than maximizing the diversity of outputted solutions designed in the original NSGA-II. Notice that, besides NSGA-II, $\wp$ can also be combined with other multi-objective search algorithms.

IV. Case Studies and Artificial Problems

In this section, we detail the four industrial problems from the three diverse domains i.e., communication, maritime and subsea oil&gas as shown in TABLE I (shown in Appendix A). To further assess the scalability of UPMOA with varying complexity, we design a large number of artificial problems inspired by each industrial problem (TABLE I).

A. Test Suite Minimization Problem

To address this problem, we defined four cost/effectiveness objectives [4][22] (TABLE I). Furthermore, we observe that test engineers have several priorities for the defined cost/effectiveness objectives, i.e., *FDC* and *FPC* should have higher priority than *TMP* and *OET* since the aim is to minimize the redundant test cases without significantly reducing *FDC* and *FPC* (TABLE I).

Industrial Case Study. To evaluate UPMOA, we chose four VCSs from *Saturn*: C20, C40, C60 and C90 (TABLE I). Notice that each feature can be tested by at least one test case and each test case can be used to test at least one feature. Each test case tc_i has a success rate for execution ($SucR_{tc_i}$) for calculating *FDC*, an average execution time (AET_{tc_i}) for measuring *OET*. In general, for *Saturn*, each feature is associated to 5-10 test cases and each test case tc_i is associated with 1-5 features with $SucR_{tc_i}$ ranging from 50% to 95%, and AET_{tc_i}ranging from 2 to 60 minutes [22].

Artificial Problems. We simulate 64 different sets of user preferences based on the four objectives (TABLE I), i.e., C(4,1)*A(1,1)=4 when considering the user preference for only one objective, C(4,2)*A(2,2)=12 when two objectives have user preferences, C(4,3)*A(3,3)=24 and C(4,4)*A(4,4)=24 for three and four objectives, respectively. For each set of user preferences, we defined 500 artificial problems. We first created a feature repository including 1200 features and a test case repository of 60,000 test cases. For each test case tc_i, three key attributes are assigned (inspired by our industrial problems but with expansion for generality), i.e., $SucR_{tc_i}$ ranges from 0% to 100%, for calculating *FDC* and AET_{tc_i} ranges from 1 minutes to 100 minutes for calculating *OET*. We created artificial problems with the increasing number of features and test cases, i.e., we used a range of 10 to 1000 with an increment of 10 for the number of features and each feature can be associated with test cases ranging from 5 to 45 with an increment of 10. So that 100*5 = 500 artificial problems were obtained for each set of user preferences in this way and in total 64*500=32000 artificial problems were obtained for the 64 sets of user preferences (TABLE I). For instance, the *500th artificial problem* is to minimize 45000 test cases for testing 1000 features with the user preference of that *TMP* has the highest priority.

B. Test Case Prioritization Problem

To address this problem, we defined four cost-effectiveness measures (TABLE I). Notice that test resources in our case refer to correct software versions deployed on hardware since test cases aims at testing different software versions. It may take a different amount of time to allocate particular test resources for a specific test case. More details related with the objective functions can be consulted in [7]. Moreover, we observe that *FDC* has the highest priority since the main aim is to detect faults as early as possible. *FPC* is more important than *PE* and *OEC* since another goal for testing VCS is to cover as many feature pairs as possible (TABLE I).

Industrial Case Study. To evaluate UPMOA, we chose a real testing cycle as shown in TABLE I. Each test case tc_i has a success rate for execution ($SucR_{tc_i}$) ranging from 50% to 100%, an average execution time AET_{tc_i} ranging from 2 to 60 minutes, and time for allocating relevant test resources (TTR_{tc_i}) ranging from 1 to 30 minutes.

Artificial Problems. We simulate another 64 different sets of user preferences for the four objectives related with test case prioritization problem (TABLE I). For each set of user preferences, we created 500 artificial problems. We first created a feature repository including 600 features, a test case repository of 3000 test cases and a test resource repository of 1000 available test resources. More specifically, one feature can be tested by 1-5 test cases whereas one test case can be

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TEVC.2017.2778560, IEEE Transactions on Evolutionary Computation

used for testing 1-5 features. Executing one test case requires allocating 1-5 test resources, while one test resource can be used for executing 1-5 test cases. Therefore, we created artificial problems with the increasing number of features and test resources, i.e., we used a range of 10 to 500 with an increment of 10 for feature number and the given available test resources can be from 50 to 500 with an increment of 50. So that 50*10 = 500 artificial problems were obtained and in total 32000 artificial problems were obtained for the 64 sets of user preferences. For instance, the 3000th *artificial problem* focuses on prioritizing 2500 test cases for testing 500 features within 500 available test resources. The user preference is that *FPC* is more important than *PE* while *PE* has a higher priority than *FDC* and *OEC*.

C. Test Case Selection Problem

Notice that one of the key differences between this problem and test case prioritization problem (from communication domain) is that different budget is taken into account, i.e., time budget is important in this case while test resource budget (i.e., hardware) is critical in communication domain. To tackle this challenge, we defined four cost-effectiveness measures together with our partner as shown in TABLE I. More details of these objective functions can be consulted in [49]. We also extracted key user preferences for these four objectives (TABLE I), i.e., *MPO* has highest priority since the primary goal for the problem is to select test cases that can detect faults. Moreover, *MPR* and *MC* should have higher priority than *TT* since time is usually the last factor to be considered as compared with covering important test requirements and causing high impact of systems (TABLE I).

Industrial Case Study. To evaluate UPMOA, we employed a real world case study (TABLE I) and each test case have four key attributes associated with average execution time AET, priority pr, probability po and consequence c.

Artificial Problems. Based on the four defined objectives, in total 64 different sets of user preferences were simulated and 500 artificial problems were created for each set of user preferences. We created a test case repository that includes 5500 test cases and each test case has four attributes, i.e., AET ranging from 1 minute to 50 minute, pr, po and c (ranging from 0 to 1). The 500 artificial problems were created with an increasing number of test cases from 100 to 5090 with an increment of 10 and for each set of test cases is associated with a time budget that ranges from 200 minutes to 20000 minutes. Thus, in total 64*500=32 000 artificial problems were created for the 64 user preference sets. For instance, the *1000th artificial problem* is to select a subset of test cases from the 5090 ones with a 20000 minute time budget and the user preference is that MPR has the highest priority.

D. Requirement Allocation Problem

To deal with this problem, we defined three cost-effectiveness measures as shown in TABLE I. More details of these objective functions can be consulted in [48]. We also observed a set of user preferences, i.e., *FAM* should have highest priority since the main goal to solve the problem is to let each stakeholder review the requirements that he/she is more familiar. Second, *OWL* should have higher priority than *ASSGIN* since the workload balance between stakeholders is more important than the overall number of requirements assigned to each stakeholder.

Industrial Case Study. To evaluate UPMOA, a real-world case study was employed (TABLE I). Notice that each requirement has three key attributes, named complexity (CM) ranging from 0 to 9, dependency index (DP) ranging from 0 to $n_R - 1$ (n_R refers to the number of total requirements) and importance (IM) ranging from 0 to 9. Each stakeholder has one attribute called the value of familiarity for reviewing a specific requirement i (FM_{ji}) that ranges from 0 to 9.

Artificial Problems. We simulated 15 sets of user preferences based on the three defined objectives (i.e., C(3,1)*A(1,1)+C(3,2)*A(2,2)+C(3,3)*A(3,3)=15) and for each set of user preferences, we created 500 artificial problems. To be specific, we created a requirement repository that includes 2000 requirements and a stakeholder repository that includes 200 stakeholders. Each requirement has three attributes inspired from the real-world case study, i.e., CM ranging from 0 to 9, DP from 0 to $n_R - 1$ (n_R is the number of total requirements) and IM from 0 to 9 and each stakeholder holds an attribute to indicate the familiarity for a given requirement ranging from 0 to 9. We increased the number of requirements to be allocated for reviewing from 300 to 1800 with an increment of 30 and the number of stakeholders who are responsible for reviewing from 60 to 150 with an increment of 10. Thus, 50*10=500 artificial problems were obtained for each set of user preferences and in total 15*500=7500 artificial problems were created. For instance, the 2000th *artificial problem* aims at allocating 1800 requirements to 150 stakeholders for reviewing and the user reference is that *Assign* is more important than *FAM* while *FAM* has a higher priority than *OWL*.

V. Empirical Evaluation

In this section, we first present the experiment design (Section A) followed by experiment settings and evaluation metrics (Section B). A description of applied statistical tests is provided in Section C and Section D presents the results. Last, we provide the overall discussions and threats to validity in Section E and Section F, respectively.

A. Experiment Design

Through the experiment, four research questions are addressed:

RQ1: Which weight assignment strategy (i.e., *FW*, *RAW* and *UDW*) can assist UPMOA in achieving the best performance with varying algorithm parameter settings?

RQ2: How does UPMOA compare with other multi-objective search algorithms (i.e., NSGA-II, RWGA and UDW-(1+1) EA) with varying algorithm parameter settings?

RQ3: How do different parameter settings affect the performance of UMPOA (sensitivity analysis)?

RQ4: How do UPMOA scale with the increasing complexity of problems?

TABLE II (shown in Appendix B) shows an overview for the experiment design including four main tasks (T_1 – T_4). Since we are not aware which weight assignment strategy can help UPMOA to achieve the best performance, T_1 is first performed to compare the three existing weight assignment strategies in conjunction with UPMOA with varying algorithm

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TEVC.2017.2778560, IEEE Transactions on Evolutionary Computation

parameter settings (RQ1). As for the algorithm parameter settings, we investigated four algorithm parameters (TABLE II) based on the existing literature [52]. Notice that it is practically infeasible to investigate all possible values for each parameter and thus we chose the five values for each parameter as evaluated in the existing literature [52], e.g., we chose the crossover rate from {0, 0.2, 0.5, 0.8, 1}. Therefore, in total 625 combinations of algorithm parameters were investigated in our experiment. Notice that other values for the parameters can be chosen as well and the key aim is to assess how the algorithms perform with different parameter settings.

T_2 is further performed to compare performance of UPMOA with three representative multi-objective search algorithms from the literature with respect to the 625 algorithm parameter settings (RQ2, TABLE II). The three selected algorithms are: **a) NSGA-II** since UPMOA is designed based on NSGA-II and the existing works (e.g., [5][29]) show that NSGA-II performs well for solving MOPs such as test selection [5]; **b) RWGA** since our previous work [4][22] shows that RWGA can achieve promising results for solving MOPs when user preferences are not taken into account, e.g., test suite minimization problem [22]; and **3) UDW-(1+1) EA** since several existing works show that (1+1) EA usually performs the best [28][48], such as for requirements assignment [48]. In addition, combining *UDW* and (1+1) EA (i.e., UDW-(1+1) EA) performs well for solving multi-objective optimization problems [26]. To guide the search, a fitness function is defined for the problems *TM*, *TP*, *TS* and *RA* as shown in TABLE II (*Problem* column), which computes a value ranging from 0 to 1 and a lower value shows a better solution. $Norm\ (x) = \frac{x}{x+1}$ is a normalization function for making all the values in the same magnitude [19].

To address RQ3, T_3 was conducted to perform a sensitivity analysis for UPMOA with respect to the 625 parameter settings. Notice that we only analyze and report the results of UPMOA with the best weight assignment strategy (from RQ1), which will be integrated into the industrial practice for solving MOPs. Furthermore, to tackle RQ4, we chose the best algorithm parameter setting from RQ3 and further assessed the performance and scalability of UPMOA using the large number of artificial problems (Section IV).

Moreover, to generate suitable weights, we defined key independent constraints based on the user preferences as shown in TABLE III. For the *FW* strategy, we provided a set of pilot weights for each objective based on the user preferences together with our industrial partners (TABLE III). Notice that since it is practically very challenging to provide quantitative values for these weights, we are not aware if the weights provided for the *FW* strategy are the most suitable ones for each objective. As for artificial problems, 207 sets of constraints are defined based on the 207 different sets of user preferences for *TM*, *TP*, *TS* and *RA*, respectively (Section IV). For instance, when only the objectives *TMP* and *FPC* are considered for the test suite minimization problem and *FPC* is more important than *TMP*, the constraints will be defined as $w_2 > w_1$, $w_1 > w_3$ and $w_1 > w_4$. In terms of the *FW* strategy, since it is practically impossible to ask test engineers or stakeholders to provide quantitative weight values for all the simulated sets of user preferences, we decide to choose the first tuple of weights generated by the *UDW* strategy as the weights applied by *FW* in the experiment for all the 103500 artificial problems.

TABLE III An Overview of the User Preferences to the Four Problems*

Weight Assignment	Problem	Constraints (User Preferences)
FW	*TM*	$w_1 = 0.2, w_2 = 0.3, w_3 = 0.3, w_4 = 0.2$
	TP	$w_1 = 0.15, w_2 = 0.3, w_3 = 0.4, w_4 = 0.15$
	TS	$w_1 = 0.1, w_2 = 0.25, w_3 = 0.4, w_4 = 0.25$
	RA	$w_1 = 0.2, w_2 = 0.3, w_3 = 0.5$
RAW UDW	*TM*	$w_2 > w_1, w_3 > w_1, w_2 > w_4, w_3 > w_4$
	TP	$w_3 > w_2, w_2 > w_1, w_2 > w_4$
	TS	$w_3 > w_2, w_3 > w_4, w_2 > w_1, w_4 > w_1$
	RA	$w_2 > w_3 > w_1$

* w_1, w_2, w_3 and w_4 are the weights assigned to 1) *TMP*, *FPC*, *FDC* and *OET* for *TM*; 2) *PE*, *FPC*, *FDC* and *OEC* for *TP*; and 3) *TD*, *MPR*, *MPO* and *MC* for *TS*. As for *RA*, w_1, w_2 and w_3 are the weights assigned to *ASSIGN*, *FAM* and *OWL*.

Notice that for UPMOA, w_1, w_2, w_3 and w_4 are used to calculate values of the user-preference indicator $\wp$. For NSGA-II, the generated weights will be not applied since all the objectives are treated with equivalent priority. For RWGA and UDW-(1+1) EA, the generated weights are used to convert multi-objective optimization problem to a single-objective optimization problem.

B. Experiment Settings and Evaluation Metrics

We employ jMetal [24] to implement UPMOA since jMetal integrates most of the existing search algorithms into a Java framework and has been widely applied for addressing various MOPs [22][30]. Each algorithm is run 100 times to account for random variation inherited in search algorithms [13]. Notice that all the experiments were run on the Abel cluster at the University of Oslo[2].

To address the research questions, we chose the following evaluation metrics to assess the quality of the solutions: 1) For each defined objective of one specific case study, the best solution with the highest value of the objective function is selected for comparison. For instance, 100 non-dominated optimal solutions are found by NSGA-II and we will choose solutions with the best objective function value for each objective; 2) To compare the overall performance, followed by the guidelines in [39][41][46], Hypervolume (*HV*) is employed as a metric to assess the quality of the solutions obtained by search algorithms from the perspectives of convergence and diversity, which represents the volume of the objective space that is covered by the produced solutions P (i.e., Pareto front). *HV* can be calculated using $HV = volume\ (\bigcup_{i=1}^{P} v_i)$ [44][46]. For each solution $i \in P$, v_i refers to the diagonal corners of the hypercube between the solution i and a reference point that is a vector of worst objective function values, e.g., (1,1,1,1) in our case for test suite minimization problem. A higher value of *HV* demonstrates a better performance of the solutions.

To answer RQ1 and RQ2, we compare and analyze results using the two defined metrics. Note that *HV* for each problem (TABLE I) refers to HV_TM, HV_TP, HV_TS and HV_RA, respectively. To address RQ3 and RQ4 (for the artificial problems), we only evaluate results with *HV*. To address RQ4, *Mean HV Value* (MHV) is defined for each artificial problem after a certain number of runs *nr* (in our case, *nr* = 100).

[2] http://www.uio.no/english/services/it/research/hpc/abel/.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TEVC.2017.2778560, IEEE Transactions on Evolutionary Computation

More specifically, we defined ten dependent variables to evaluate the performance and scalability of UPMOA as shown in TABLE IV. For *TM*, three variables were defined, where i refers to the feature number, j refers to the test case number and u refers to the number of user preference sets ($10 \leq i \leq 1000$ with an increment of 10, $4 \leq j \leq 45$ with an increment of 10 and $1 \leq u \leq 64$ with an increment of 1). $HV\_TM_{i,j,u,r}$ is the hypervolume value (HV) after the r^{th} runs with i features, j test cases and u^{th} user preference set. Notice that $MHVF\_TM$ and $MHVTC\_TM$ are defined to measure the mean HV value in a given number of features or test cases and RQ4 can be addressed using statistical analysis. For *TP*, another three variables were also defined, where i refers to the feature number, j refers to the number of test resources and u refers to the number of user preference sets ($10 \leq i \leq 500$ with an increment of 10 and $50 \leq j \leq 500$ with an increment of 50). $HV\_TP_{i,j,u,r}$ is the values of HV after the r^{th} runs with i features, j test resources and u^{th} user preference set. Similarly, $MHVF\_TP$ and $MHVTR\_TP$ are defined to measure the mean HV value within a number of features or test resources.

TABLE IV Variables and Formulas for Each Problem

Problem	Variable	Formula
Test Suite Minimization (*TM*)	*MHV* for *TM* (*MHV_TM*)	$MHV\_TM_{i,j} = \frac{\sum_{r=1}^{nr}\sum_{u=1}^{64} HV\_TM_{i,j,u,r}}{64 * nr}$
	MHV_TM for Feature (*MHVF_TM*)	$MHVF\_TM_i = \frac{\sum_{j=5,j=j+10}^{45} MHV\_TM_{i,j}}{5}$
	MHV_TM for Test Cases (*MHVTC_TM*)	$MHVTC\_TM_j = \frac{\sum_{i=10,i=i+10}^{1000} MHV\_TM_{i,j}}{100}$
Test Case Prioritization (*TP*)	*MHV* for *TP* (*MHV_TP*)	$MHV\_TP_{i,j} = \frac{\sum_{r=1}^{nr}\sum_{u=1}^{64} HV\_TP_{i,j,u,r}}{64 * nr}$
	MHV_TP for Feature (*MHVF_TP*)	$MHVF\_TP_i = \frac{\sum_{j=50,j=j+50}^{500} MHV\_TP_{i,j}}{10}$
	MHV_TP for Test Resource (*MHVTR_TP*)	$MHVTR\_TP_j = \frac{\sum_{i=10,i=i+10}^{500} MHV\_TP_{i,j}}{50}$
Test Case Selection (TS)	*MHV* for *TS* (*MHV_TS*)	$MHV\_TS_i = \frac{\sum_{r=1}^{nr}\sum_{u=1}^{64} HV\_TS_{i,u,r}}{64 * nr}$
Requirement Allocation (RA)	*MHV* for *RA* (*MHV_RA*)	$MHV\_RA_{i,j} = \frac{\sum_{r=1}^{nr}\sum_{u=1}^{15} HV\_RA_{i,j,u,r}}{15 * nr}$
	MHV_RA for Requirement (*MHVR_RA*)	$MHVR\_RA_i = \frac{\sum_{j=60,j=j+10}^{150} MHV\_RA_{i,j}}{10}$
	MHV_RA for Test Stakeholder (*MHVS_RA*)	$MHVS\_RA_j = \frac{\sum_{i=300,i=i+30}^{1800} MHV\_RA_{i,j}}{50}$

As for *TS* (TABLE IV), one variable was defined, i.e., MHV_TS to measure the average HV value with a given number of test cases, where i refers to the number of test cases, u is the number of user preference sets ($100 \leq i \leq 5090$ with an increment of 10). $HV\_TS_{i,u,r}$ refers to the HV value after the r^{th} runs with i test cases and u^{th} user preference set. In terms of *RA*, we also defined three variables to measure the mean values of HV within a given number of requirements ($MHVR\_RA$) or stakeholders ($MHVS\_RA$). To be specific, i refers to the number of requirements, j refers to the number of stakeholders and u refers to the number of user preference sets ($300 \leq i \leq 1800$ with an increment of 30, $60 \leq j \leq 150$ with an increment of 10 and $1 \leq u \leq 15$ with an increment of 1). $HV\_RA_{i,j,u,r}$ refers to the HV value after the r^{th} runs with i requirements, j stakeholders and u^{th} user preference set.

C. Statistical Tests

Based on the guidelines in [13], we apply the Vargha and Delaney statistics and Mann-Whitney U test to evaluate the results. The Vargha and Delaney statistics calculates $\hat{A}_{12}$, which is defined as a non-parametric effect size measure. In our case, $\hat{A}_{12}$ evaluates the probability of yielding higher values for each objective and HV for two algorithms A and B. Each pair of algorithms is further compared using the Mann-Whitney U test (p-value) to determine the significance of the results with the significance level being 0.05, i.e., there is a significant difference if p-value is less than 0.05. Based on the above description, we define that algorithm A has better performance than algorithm B, if the $\hat{A}_{12}$ value is greater than 0.5 and such better performance is significant if p-value is less than 0.05. To address RQ4, the Spearman's rank correlation coefficient (ρ) is used to measure the correlations of the MHV of the algorithms and the increasing complexity of problems [16]. The value ranges from -1 to 1, i.e., a value of ρ that is greater than 0 indicates a positive correlation and vice versa. A ρ value close to 0 shows that there is no correlation between two sets of data. Moreover, we also report significance of correlation using $Prob>|\rho|$; a value lower than 0.05 means that the correlation is statistically significant.

D. Experiment Results

This section presents the experiment results for each research question.

RQ1 (Assessing the three weight assignment strategies). The results showed that for each industrial problem (*TM*, *TP*, *TS* and *RA*, Section *IV*), *RAW* and *UDW* significantly outperforms *FW* for each objective and HV (HV_TM, HV_TP, HV_TS and HV_RA) in terms of each parameter setting (in total 625 settings). Notice that we do not provide detailed results in the paper due to the limited space since all the values for $\hat{A}_{12}$ are less than 0.5 and the p-values are less than 0.0001. This indicates that asking experts to provide quantitative weight values based on their preferences may not be an accurate representation of user-preferences since it may be difficult to provide quantitative values (e.g., for *TM*, *FDC* with 0.3 has higher priority than *TMP* with 0.2) even though experts can specify them in a qualitative way (e.g., *FDC* is more important then *TMP*). Moreover, for all the 625 parameter settings, we find that the performance of *UDW* is significantly better than *RAW* for all the objectives and HV. This shows that the generated weights with uniformly normal distribution (Section II.A) can significantly improve the performance of the search algorithms.

Running Time. TABLE V reports the average running time for each algorithm with respect to each industrial problem. For each industrial problem, the average running time is calculated by taking all the 625 parameter settings into account. From TABLE V, we can observe that there is no practical difference in terms of running time for applying the three weight assignment strategies (RQ1). By applying the Whitney U Test

TABLE V Average Running Time of Search Algorithms

RQ	Algorithm	Average Running Time (seconds)				Whitney U-Test
		TM	*TP*	*TS*	*RA*	
1	UPMOA (*FW*)	1.47	2.07	3.01	5.45	No significant differences
	UPMOA (*RAW*)	1.58	2.19	2.96	5.86	
	UPMOA (*UDW*)	1.50	2.10	2.87	5.77	
2	UPMOA (*UDW*)	1.50	2.10	2.87	5.77	
	NSGA-II	1.57	2.30	3.13	5.69	
	RWGA	1.59	2.05	2.79	5.60	
	UDW-(1+1)EA	1.46	1.99	2.80	5.63	

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TEVC.2017.2778560, IEEE Transactions on Evolutionary Computation

for the average running time, we observed that all the *p*-values are greater than 0.05. We therefore can conclude that applying *RAW* and *UDW* do not significantly increase the algorithm running time as compared with *FW*. Thus, we can answer RQ1 as: *UDW* can help UPMOA to achieve the best performance for solving the four multi-objective minimization problems.

RQ2 (Comparing UPMOA with the state-of-the-art). The results for comparing UPMOA (with *UDW*) with NSGA-II, RWGA and UDW-(1+1) EA showed that UPMOA achieved significantly better performance than the selected three search algorithms for each objective and *HV* (HV_TM, HV_TP, HV_TS and HV_RA) for all the industrial problems in terms of each parameter setting since all the values for $\hat{A}_{12}$ are less than 0.5 and the *p*-values are less than 0.0001. Moreover, we report the average time taken by each algorithm as shown in TABLE V and no significant differences were observed among them based on results of the Whitney U Test (RQ2). We can conclude that applying UPMOA does not significantly increase the running time. Therefore, we answer RQ2 as: UPMOA can significantly improve the performance of multi-objective search algorithms in terms of solving the four industrial optimization problem with user preferences.

RQ3 (Sensitivity Analysis of UPMOA). Based on the results of RQ1, we chose UPMOA with *UDW* for the sensitivity analysis. Recall that we aim at assessing the performance of UPMOA (with *UDW*) with in total 625 parameter settings (TABLE II), e.g., crossover rate is 0.8, population size is 100, mutation rate is 0.2 and maximum number of fitness evaluations is 5000.

TABLE VI summarizes the results by reporting the parameter settings that lead the best performance of UPMOA (*Best Setting* column) and the parameter setting that results in the worst UPMOA performance (*Worst Setting* column) in terms of each industrial problem (measured by HV_TM, HV_TP, HV_TS and HV_RA, respectively). We also conducted the Whitney U Test to determine the significance between the best and worse parameter settings and the results showed that for each industrial problem, the best parameter setting significantly outperformed the worse parameter setting in terms of the performance of UPMOA.

TABLE VI Results for Sensitivity Analysis *

Problem	Best Setting				Worse Setting				Whitney U-Test
	C	Mu	P	Max	C	Mu	P	Max	
TM	0.8	0.5	200	25000	0	0	5	5000	Significant Differences
TP	0.8	0.8	200	25000	0	0	5	5000	
TS	1	0.9	200	25000	0	0	5	5000	
RA	0.5	0.2	200	25000	0	0	5	5000	

*C: crossover rate; Mu: mutation rate; P: Population size; Max: maximum number of fitness evaluations

We can also observe that the parameter setting that led to the best performance of UPMOA is highly sensitive to the problem to be solved (problem-specific). We also find that the population size and maximum number of fitness evaluation of the best parameter settings are always 200 and 25000 for the four problems, which indicates that a higher number of population size and fitness evaluations may help to improve the performance of search algorithms (e.g., UPMOA).

Moreover, as stated in [13][52], default parameter settings usually lead to good performance of search algorithms. To test this, for each problem, we employed the Vargha and Delaney statistics and Mann-Whitney U test for comparing the best (and worst) parameter setting with the default setting suggested by jMetal [24] (i.e., crossover rate =0.9, mutation rate =1/*n* where *n* is the number of variables, population size is 100 and the maximum number of fitness evaluations is 25000). The results showed that for all the problems, the best parameter settings always managed to produce significantly better performance of UPMOA than the default setting. In addition, UPMOA with the default setting was able to significantly outperform UPMOA with the worst parameter setting. Such interesting observation suggests that a parameter tuning (sensitivity analysis) might be necessary in practice to determine the best parameter setting for UPMOA.

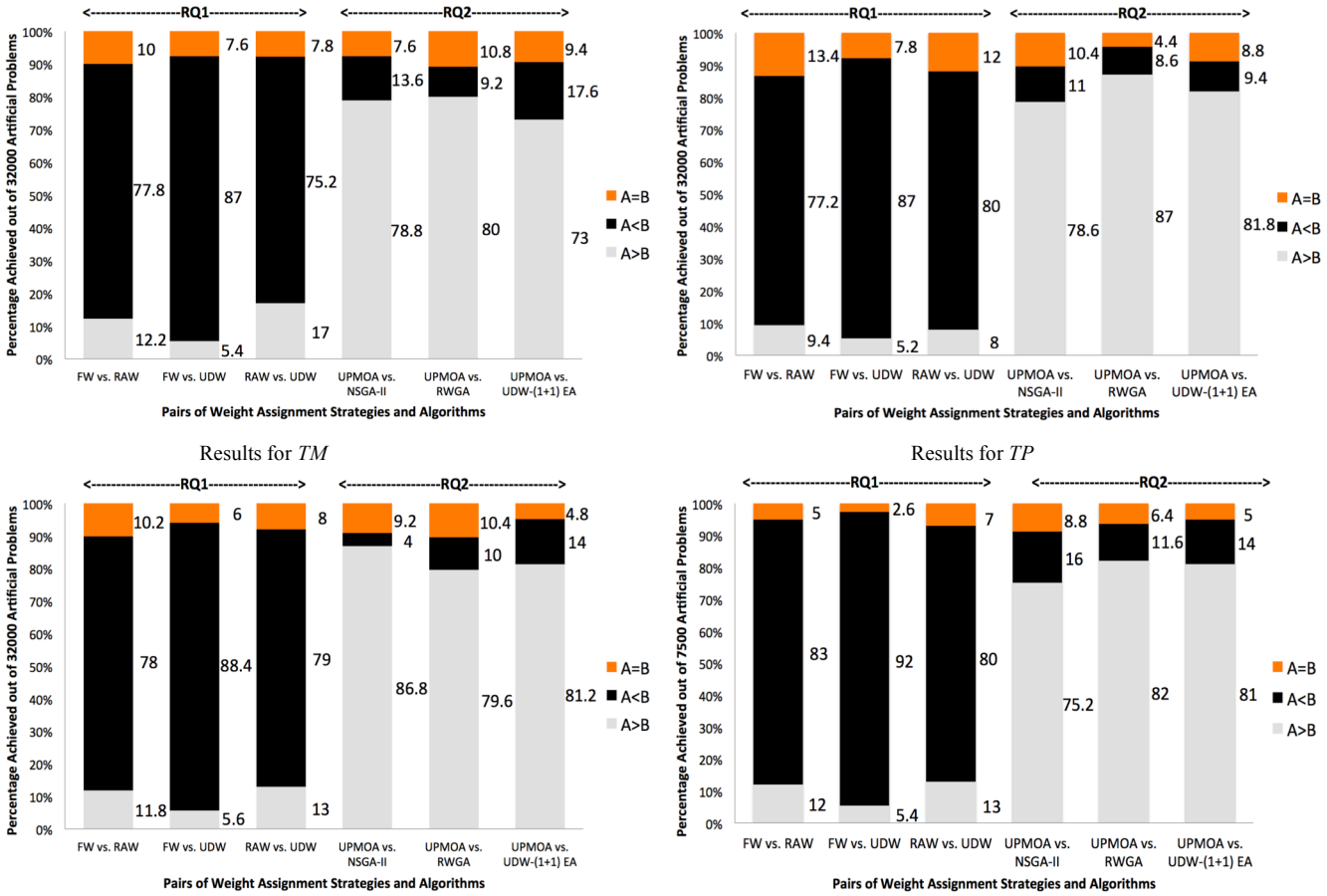
Thus, we can answer RQ3 as: setting parameters for UPMOA is highly problem-specific, which can cause very large variance of the UPMOA performance.

RQ4 (Assessing the performance and scalability of UPMOA with artificial problems). The evaluation for the artificial problems is based on HV_TM for *TM*, HV_TP for *TP*, HV_TS for *TS* and HV_RA for *RA* (Section B) and a higher value shows a better performance of an algorithm. Recall that for RQ4, we only chose the best parameter settings for *TM*, *TP*, *TS* and *RA* based on the results of RQ3 (Section A). Fig. 3 illustrates the results for the artificial problems for *TM*, *TP*, *TS* and *RA*. *A*>*B* shows the percentage of problems for which an algorithm *A* has significantly better performance than *B* ($\hat{A}_{12} < 0.5 \,\&\&\, p < 0.05$), *A*<*B* means vice versa ($\hat{A}_{12} > 0.5 \,\&\&\, p < 0.05$) and *A*=*B* means the number of problems for which there are no significant differences in performance between *A* and *B* ($p \geq 0.05$).

From Fig. 3, we can first observe that *UDW* performs significantly better than *FW* and *RAW* for on average 81.1% problems for *TM*, 83.5% for *TP*, 83.7% for *TS* and 86% for *RA*. The results indicate that even with the varying complexity of problems with 207 sets of user preferences, *UDW* along with UPMOA still performs the best as compared with *FW* and *RAW*. Moreover, we observe that UPMOA always achieves the best performance in a significant manner for most of the problems, i.e., on average 77.3% problems for *TM*, 82.5% for *TP*, 82.5% for *TS*, 79.4% for *RA*. Thus, we can conclude that UPMOA has a good capability for solving different optimization problems with different sets of user-preferences with increasing complexity.

To further assess the scalability of UPMOA, Fig. 4 illustrates the trend of MHV_TM for *TM*, MHV_TP for *TP*, MHV_TS for *TS* and MHV_RA for *RA* (y-axis, Section B) when increasing the complexity of the problems. More specifically, the complexity of the problems is measured as: 1) for *TM*, the number of features increases from (x-axis) and number of associated test cases (five lines related with different number of associated test cases); 2) for *TP,* the number of features (x-axis) and the number of test resources (ten lines related with different number of test resources); 3) for *TS*, the number of test cases (x-axis) and 4) for *RA*, the increasing number of requirements (x-axis) and the number of stakeholders (ten lines associated with each number of stakeholder). Notice that to make the plots more visible, five points are marked for *TM*, *TP* and *RA* while 11 points are marked as circles for *TS*. Recall that a higher value of

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TEVC.2017.2778560, IEEE Transactions on Evolutionary Computation



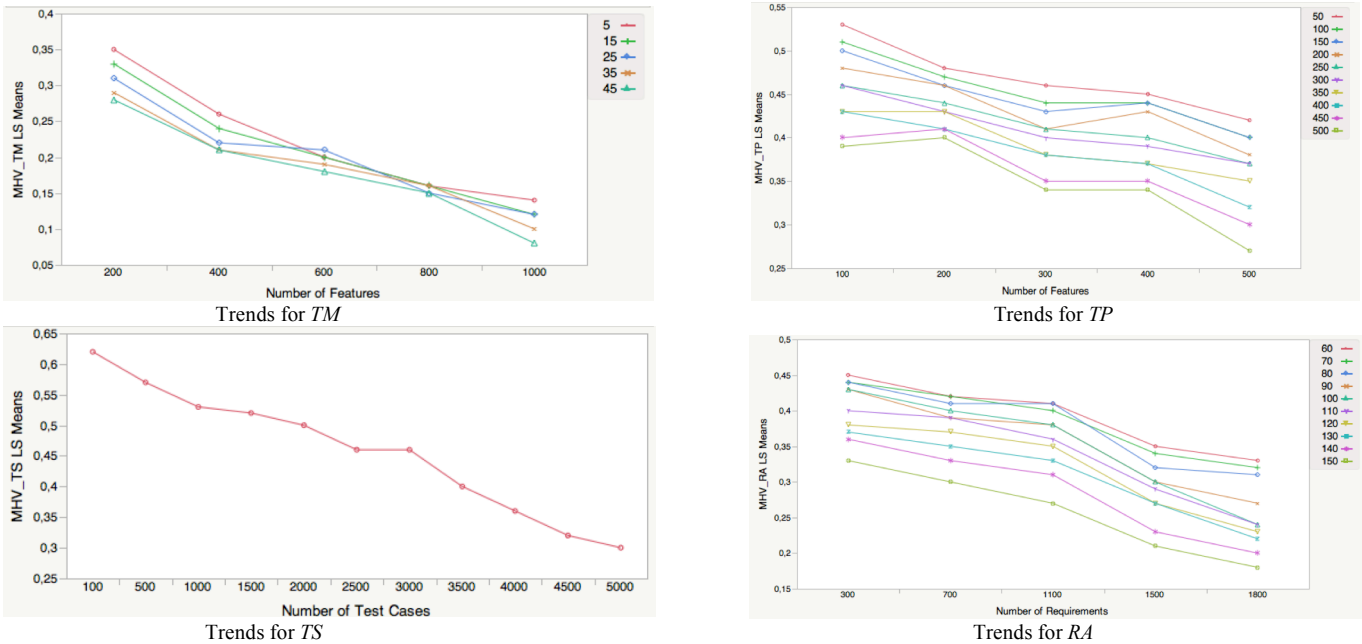
Results for *TM* Results for *TP*

Results for *TS* Results for *RA*

Fig. 3. Results of the artificial problems

MHV_TM , *MHV_TP* , *MHV_TS* and *MHV_RA* represents a better performance of a search algorithm.

From the figure, we observe that 1) for *TM*, all the lines decline from the overall view as the increasing number of features (x-axis). In addition, the lines also decline as the increasing number of associated test cases (five lines); 2) for *TP*, each line declines as the increasing number of features and the lines also decline as the increasing number of test resources (ten lines); 3) for *TS*, the line declines with the increase in the number of test cases; and 4) for *RA*: each line also declines when the number of requirements increases and the ten lines also decline with the growth number of



Trends for *TM* Trends for *TP*

Trends for *TS* Trends for *RA*

Fig. 4. Overall trends when problems becomes more complex

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TEVC.2017.2778560, IEEE Transactions on Evolutionary Computation

stakeholders. Therefore, we conclude that for all the four optimization problems, the performance of UPMOA decreases as the increasing complexity of the artificial problems.

In addition, we calculated the Spearman's rank correlation for each algorithm as shown in TABLE VII. Based on the results, we can observe that the performance of UPMOA indeed decreases when the problems become more complex, which is consistent with what we observed from the figure, since all the values for the Spearman ρ are less than 0 (i.e., negative correlation) but such a decrease is not statistically significant since all the *p*-values are greater than 0.05. We also performed the two-way Analysis of Variance (ANOVA) test [17] to assess the combined effect on UPMOA's performance in terms of 1) the number of features and the number of test cases for *TM*; 2) the number of features and the number of test resources for *TP*; and 3) the number of requirements and the number of stakeholders for *RA*. Results show that no significant interaction effects were identified since all the *p*-values are greater than 0.05 (TABLE VII). Therefore, we can conclude that UPMOA is not significantly influenced when the complexity of the problems increases and thus applying UPMOA can solve a wider range of user preference optimization problems.

TABLE VII The Results of Spearman's Ranking Correlation and ANOVA

Problem	Comparison	Spearman	ANOVA
TM	*MHVF_TM* and the number of features	-0.08	0.409
	MHVTC_TM and the number of test cases	-0.10	
TP	*MHVF_TP* and the number of features	-0.02	0.518
	MHVTR_TP and the number of test resources	-0.07	
TS	*MHV_TS* and the number of test cases	-0.11	Null
RA	*MHVR_RA* and the number of requirements	-0.12	0.310
	MHVS_RA and the number of stakeholders	-0.09	

E. Overall Discussion

Based on the experiment results, we conclude the key findings in terms for each research question as below.

For RQ1, we conclude that the *UDW* strategy can promote UPMOA to achieve the best performance for all the industrial problems with in total 625 parameter settings. The reason why *UDW* performed significantly better than *FW* and *RAW* can be explained as follows: 1) *FW* applies the fixed set of weights that determines a single direction for guiding the search towards finding optimal solutions. However, in many cases, optimal solutions might not exist in this restricted search space explored by a single direction. *UDW* assigns weight set with uniform distribution during each generation, which allows a search algorithm to explore multiple directions in search for finding optimal solutions; and 2) As mentioned in Section II.A, *RAW* can not ensure that each point of weights is selected with the same probability at each generation and thus each search direction can not be explored uniformly. Unlike *RAW*, *UDW* can guarantee the selection of weights with equal probability and thus each search direction has an equivalent chance to be reached. This also indicates that exploring each search direction uniformly is essential when guiding the search towards finding optimal solutions.

For RQ2 and RQ4, we conclude that UPMOA significantly outperforms the other three multi-objective search algorithms for all the industrial problems and 103500 artificial problems, which can be explained as follows: 1) NSGA-II treats all the objectives with equal priority and does not explicitly consider pre-defined user preferences, which may lose optimal solutions that better satisfy user preferences during the search. As for RWGA and UDW-(1+1) EA, the main workflow is to convert a multi-objective optimization problem into a single-objective optimization problem by assigning weights to each objective. Therefore, although user-preferences are reflected by using weight assignment strategies, RWGA and UDW-(1+1) EA are still more close to single-objective search algorithms, which may lose many non-dominated solutions during the search [8][11]. As compared with the afore-mentioned search algorithms, UPMOA takes the advantages of NSGA-II, RWGA and UDW-(1+1) EA by introducing a user-preference indicator $\wp$ based on the best weight assignment strategy (i.e., *UDW*) and integrating $\wp$ into NSGA-II. During the search process, UPMOA takes the user preferences as an indicator to assess the quality of produced solutions (Section III) and thus output a set of non-dominated solutions at the same time reflecting user preferences defined based on the domain knowledge. In addition, for RQ4, we conclude that UPMOA is scalable and has a good capability for solving the problems of varying complexity. This can be explained by the fact that the search algorithms can still manage to explore search space towards finding optimal solutions without significantly influenced by varying complexity of problems [1][5][8].

As for RQ3 (sensitivity analysis), we observed that different parameter settings indeed led to large variance for the performance of UPMOA and tuning parameters is highly problem-specific. Therefore, we recommend the best parameter settings (as shown in TABLE VI) for solving each industrial problem in practice, e.g., crossover rate=0.8, mutation rate=0.5, population size=200 and maximum number of fitness evaluations=25000 for solving the test suite minimization problem at Cisco (communication domain). However, if the practitioners plan to employ UPMOA for solving other multi-objective optimization problems with user preferences, we would recommend performing a sensitivity analysis (parameter tuning) for determining the parameter setting that leads to the best performance of UPMOA. Notice that parameter tuning is very computational expensive [52] and if parameter tuning is practically infeasible, we would recommend using the default parameter setting suggested by the literature [24] since our results showed that the default parameter setting significantly outperformed the worse parameter settings for all the problems (Section D).

Concluding Remarks: In summary, UPMOA with *UDW* strategy achieves the best performance in our experiments based on the four industrial problems and in total 103500 artificial problems with 207 varying sets of user preferences. Thus, as for practical implications, we would recommend the practitioners applying UPMOA (with *UDW* strategy) to deal with multi-objective software engineering problems with pre-determined user preferences.

F. Threats to Validity

A possible threat to *internal validity* is that existing works usually choose only one-default configuration setting for the parameters of the search algorithms [5][6][48]. To address this, we chose four algorithm parameters (e.g., crossover rate,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TEVC.2017.2778560, IEEE Transactions on Evolutionary Computation

mutation rate) based on the existing literature [52] and evaluated the performance of UPMOA with in total 625 parameter settings (Section A). Notice that it is also possible to assess other combinations of parameter values. The most probable *conclusion validity* threat in experiments involving randomized algorithms is due to random variations. To address it, we repeated experiments 100 times to reduce the possibility that the results were obtained by chance. We also reported the results using the Vargha and Delaney statistics (to measure the effect size) and Mann-Whitney U test (to determine the statistical significance of the results [13]). Spearman's rank correlation coefficient is used to measure the potential impact on the performance of algorithms when the problems become complex [34]. Another *conclusion validity* is that people may argue that the quality of solutions (e.g., fault detection capability) obtained by UPMOA may not be good as compared with the ones produced by real experts (e.g., test engineers). Notice that all the objectives are carefully defined together with industrial practitioners based on relevant domain knowledge and thus we assume that the solutions produced by search algorithms are comparable to the solutions produced by domain experts. We also want to emphasize that involving domain experts is very expensive in practice and it requires a huge amount of effort to conduct comparisons for solutions produced by search algorithms and experts.

For the *construct validity* threats, we looked at the validity of the comparison measures for UPMOA and the selected search algorithms. To reduce *construct validity* threats, we used the same stopping criterion for all algorithms, i.e., number of fitness evaluations. We ran each algorithm for 5000, 10000, 15000, 20000 and 25000 fitness evaluations to seek the best solution. This criterion is a comparable measure across all the algorithms since each iteration requires updating the obtained solution and comparing the computed value of fitness function. One common *external validity* threat in the software engineering experiments is about generalization of results. To address that, we evaluated the performance and scalability of UPMOA by employing four industrial problems and carefully-designed 103500 artificial problems with 207 different sets of user preferences and the results are consistent for both industrial problems and artificial problems.

VI. Related Work

Search-Based Software Engineering (SBSE) has been extensively studied and applied in software engineering disciplines from requirements to testing [2][7][9][48][53]. A comprehensive review for SBSE is available in [1], where search techniques are well described and sketched with applications to a variety of SE problems.

A. Multi-objective Optimization Problems (MOPs)

There is a large body of research for applying search algorithms for addressing MOPs related with various software engineering topics [4-11, 19, 21, 24, 25, 33]. For instance, Yoo and Harman [5] used a greedy algorithm and a multi-objective search algorithm NSGA-II for test case selection for the following three measures: code coverage, fault detection history, and execution time in the context of regression testing. Walcott et al. [31] addressed a two-objective problem (i.e., code coverage and execution time) and converted it into a single-objective problem using an arithmetical combination of weights for the fitness function. However, all these existing works treated all the objectives with equivalent priorities and did not mention the user-preferences for different contexts. Based on our experience while working with industrial partners, it is practically infeasible to use an equal priority for each objective. For instance, for test case selection, fault detection capability of the selected test cases usually has the highest priority but for test case generation, coverage becomes more important than the other objectives. Thus, this work is motivated to address user-preference multi-objective optimization problems, which are different as compared with the existing works. Moreover, our previous work [26] proposed the Uniformly Distributed Weights (*UDW*) weight strategy and compared it with two existing weight assignment strategies by employing one industrial test problem. As compared with [26], this work also compared these three weight assignment strategies, but with different motivation, i.e., this work focuses on improving the current search algorithms for solving multi-objective optimization problems with user preferences. In addition, this work evaluated the performance of the proposed algorithm (i.e., UPMOA) by comparing with three other search algorithms using four industrial problems with 103500 artificial problems.

One of our published work [22] focuses on solving minimization problem based on five defined cost-effectiveness measures and empirically evaluated the performance of eight multi-objective search algorithms (including NSGA-II and RWGA). As compared with [22], the similarity of this work is that the similar test suite minimization problem is applied for the evaluation. However, there are at least three differences when comparing these two works. First, the motivation is different, i.e., this work aims at proposing an improved algorithm for solving user-preference optimization problems and the test suite minimization problem is only considered as one industrial problem for evaluation. Second, when addressing the test suite minimization problem in [22], there were no specific user preferences for the objectives and thus all of them were treated equally. In this work, four constraints were defined for user preferences (Section V.A), which were used to guide the search algorithms. Third, this work applied a multi-objective test case prioritization problem for evaluating the proposed algorithm UPMOA, which is not addressed in [22]. We also need to mention that another of our published work [7] addressed a multi-objective test case prioritization problem, which is similar as what we employed in this paper for evaluating UPMOA. But again, all the objectives in [7] were not distinguished based on the user-preferences and there were no contribution in terms of improving the current search algorithm in [7].

Furthermore, our previous work [50] studied the test case prioritization problem by combining resource-aware and multi-objective search, which uses the similar case study of prioritization presented in this work (but with different number of test cases and test resources). However, as compared with [50], there are at least two key differences, which include: 1) the focus is totally different, i.e., this work aims to propose an extended search algorithm (i.e., UPMOA) that can incorporate user preferences during the search process

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TEVC.2017.2778560, IEEE Transactions on Evolutionary Computation

and test case prioritization is only used as a case study for evaluation while [50] only applied search algorithms for solving test case prioritization problem without considering any user preferences; and 2) three more industrial case studies with a large number of artificial problems were employed for evaluating the performance and scalability of UPMOA, which was not the case for [50]. In [46], we proposed a practical guide to select quality indicators for assessing the performance of Pareto-based search algorithms for the SBSE community, which also employed case studies of test suite minimization, test case prioritization and requirement allocation. As compared with [46], the differences can be summarized as: 1) this work focuses on dealing with user preferences when guiding search towards finding optimal solutions while [46] aimed at providing a guide for choosing quality indicators without taking into account any user preference; 2) 103500 artificial problems were created with 207 sets of user preferences for evaluating UPMOA, which is not the case in [46]; and 3) one additional case study for test case selection related with maritime domain was further employed for evaluation, which was not addressed in [46].

B. Weight-based User Preferences for MOPs

There are a few existing works that focus on using weights in search as a means for incorporating user preference information [35-42]. A weighted-integration approach was proposed in [37][38] based on Hypervolume indicator (*HV*) to incorporate user preferences information with the aim to solve MOPs. The approach took the idea of extending the *HV* by integrating a weight distribution function based on the preference information. However, the running time for such approach increases exponentially when facing more than two objectives and thus it is not practically feasible to apply in practice when there is time budget for applying these approaches [35]. In addition, as compared with [37][38], our results show that there is no significant difference for the running time taken by UPMOA and the current industrial practice (i.e., random search).

In [36], an arbitrary weight distribution function was proposed to incorporate preference information directly into multi-objective search algorithms (e.g., NSGA-II). The idea is to extend the original indicator (e.g., crowd distance for NSGA-II) by multiplying a set of pre-generated weights for each objective based on the user preference information. However, in practice, it is common to acquire user preference information in a qualitative way rather than a quantitative way (e.g., *FDC* is more important than *TMP*) and thus beforehand we are not aware which set of quantitative weights is the most appropriate one. As compared with [36], the indicator p applied by UPMOA generates different sets of weights based on the user preferences at each generation and thus various sets of weights can be evaluated with the aim at obtaining the most appropriate weights to incorporate the user preference information in search to find optimal solutions.

In [51], an approach named Stepwise Adaptation of Weights (SAW) was proposed for solving constraint satisfaction problems (CSPs). The key idea of SAW is to assign weights to each sample point and dynamically adapt these weights during search to decrease the prediction error, based on a defined error function to assess how far the sample points can satisfy the defined constraints. As compared with SAW, this work poses at least three key differences: 1) weights generated by UPMOA always meet pre-defined user preferences (i.e., constraints on weights) with no need to dynamically measure and predict errors; 2) UPMOA is built on NSGA-II to particularly tackle various multi-objective optimization problems, which was not mentioned for SAW and 3) UPMOA has been empirically evaluated using four industrial problems and 103500 artificial problems, which is not the case in [51].

VII. Conclusion and Future Work

While working on several industrial problems, we found that it is important to incorporate user preferences into the search process when obtaining optimal solutions. To deal with such challenge, this paper proposes an improved user-preference multi-objective optimization algorithm (namely UPMOA) by defining and integrating a user-preference indicator (p) into the most commonly used multi-objective optimization algorithm (NSGA-II). To evaluate the performance and scalability of UPMOA, we employed four industrial problems and in total 103500 artificial problems with 207 sets of different user preferences. In addition, we performed a sensitivity analysis of UPMOA with in total 625 parameter settings. The results of evaluation demonstrate: 1) Uniformly Distributed Weights (*UDW*) strategy can assist UPMOA in achieving the best performance even with increasing complexity of problems; 2) UPMOA significantly outperforms the other multi-objective search algorithms for solving various user-preference optimization problems; and 3) different parameter setting can lead to very large variance of the performance of UPMOA and tuning parameters is highly problem-specific. In addition, results from artificial problems show that UPMOA is scalable to solve a wide range of problems of varying complexity.

As for future work, we plan to apply more problems (e.g., test case generation) with user preferences to further evaluate UPMOA. We also want to investigate the performance of integrating the user-preference indicator (p) with other search algorithms (e.g., Improved Strength Pareto Evolutionary Algorithm) as compared with UPMOA.

References

[1]. M. Harman, S.A. Mansouri and Y. Zhang, "Search Based Software Engineering: A Comprehensive Analysis and Review of Trends Techniques and Applications", Technical Report TR-09-03, 2009.

[2]. M. Harman, "Making the Case for MORTO: Multi Objective Regression Test Optimization", Proc. of the International Conference on Software Testing, Verification and Validation Workshops, pp. 111-114, 2011.

[3]. Y. Zhang, A. Finkelstein and M. Harman, "Search Based Requirements Otimization: Existing Work and Challenges", Requirements Engineering: Foundation for Software Quality, pp. 88-94, 2008.

[4]. S. Wang, S. Ali and A. Gotlieb, "Minimizing Test Suites in Software Product Lines Using Weighted-based Genetic Algorithms", Proc. of the Genetic and Evolutionary Computation Conference (GECCO), pp. 1493-1500, 2013.

[5]. S. Yoo, and M. Harman, "Pareto Efficient Multi-Objective Test Case Selection," Proc. of International Symposium on Software testing and analysis (ISSTA), pp. 140-150, 2007.

[6]. C. Henard, M. Papadakis, G. Perrouin, J. Klein, and Y.L. Traon, "Multi-objective Test Generation for Software Product Lines," Proc. of Software Product Line Conference (SPLC). pp. 62-71. 2013.

[7]. S. Wang, D. Buchmann, S, Ali, A. Gotlieb, D. Pradhan and M. Liaaen, "Multi-objective test prioritization in software product line testing: an

industrial case study", Proc. of the 18th International Software Product Line Conference, pp. 32-41, 2014.

[8]. A. Konak, D.W. Coit, and A.E. Smith, "Multi-objective Optimization using Genetic Algorithms: A Tutorial," Reliability Engineering & System Safety (91) pp. 992-1007, 2006.

[9]. G. Rothermel, R. J. Untch, C. Chu, and M. J. Harrold, " Prioritizing test cases for regression testing," IEEE TSE, 27(10): 929–948, 2001.

[10]. K. R. Walcott, M. L. Soffa, G. M. Kapfhammer, and R. S. Roos, "Time aware test suite prioritization," Proc. of the ISSTA, pages 1–11, 2006.

[11]. J. Brownlee, "Clever Algorithms: Nature-Inspired Programming Recipes," ISBN: 978-1-4467-8506-5, 2011.

[12]. Cisco Systems, "Cisco telepresence codec," Data sheet. Available from http://www.cisco.com. 2010.

[13]. A. Arcuri, and L.C. Briand, "A Practical Guide for Using Statistical Tests to Assess Randomized Algorithms in Software Engineering," Proc. of the ICSE, pp. 21-28, 2011.

[14]. S. Ali, L.C. Briand, H. Hemmati, and R. K Panesar-Walawege, "A Systematic Review of the Application and Empirical Investigation of Search-Based Test Case Generation," IEEE Transactions on Software Engineering 36 (6), pp. 742-762, 2010.

[15]. M.O. Barros and A.C. Dias-Neto, "Threats to Validity in Search-based Software Engineering Empirical Studies", Universidade Federal do Estado do Rio de Janeiro0006/2011, 2011.

[16]. D.J. Sheskin, "Handbook of Parametric and Nonparametric Statistical Procedures", 2003.

[17]. H. Klaus and K. Oscar, "Design and Analysis of Experiments", 2008.

[18]. Z. Li, M. Harman, and R.M. Hierons, "Search Algorithms for Regression Test Case Prioritization", IEEE Transactions on Software Engineering, 33(4), pp. 225–237, 2007.

[19]. D. Greer and G. Ruhe, "Software Release Planning: An Evolutionary and Iterative Approach", Information and Software Technology, 46(4), pp. 243-253, 2004.

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

[21]. E. Zitzler, M. Laumanns, and L. Thiele, "SPEA2: Improving the Strength Pareto Evolutionary Algorithm", Proc. of the EUROGEN 2001-Evolutionary Methods for Design, Optimization and Control with Applications to Industrial Problems", pp. 95-100, 2001.

[22]. S. Wang, S. Ali and A. Gotlieb, "Cost-effective test suite minimization in product lines using search techniques", Journal of Systems and Software, vol(103), pp. 370-391, 2015.

[23]. L. Zhang, S.S. Hou, C. Guo, T. Xie and H. Mei, "Time-aware test-case prioritization using integer linear programming", Proc. of the 18th ISSTA, pp. 213-224, 2009.

[24]. J.J. Durillo, A.J. Nebro, "jMetal: A Java framework for multi-objective optimization", Advances in Engineering Software, pp. 760-771. 2011.

[25]. J.H. Mcdonald, "Handbook of Biological Statistics", 2009.

[26]. S. Wang, S. Ali and A. Gotlieb, "Random-Weighted Search-Based Multi-objective Optimization Revisited", Proc. of the 6th International Symposium on Search-Based Software Engineering, pp. 199-214, 2014.

[27]. S. Wang, S. Ali, A. Gotlieb and M. Liaaen, "A systematic test case selection methodology for product lines: results and insights from an industrial case study", Journal of Empirical Software Engineering, 2014.

[28]. S. Ali, M. Zohaib Iqbal, A. Arcuri, and L.C. Briand, "Generating Test Data from OCL Constraints with Search Techniques", IEEE transactions on Software Engineering, 39 (10), pp. 1376-1402, 2013.

[29]. A. Ouni, M. Kessentini, H.A. Sahraoui, "Multiobjective Optimization for Software Refactoring and Evolution", Advances in Computers, 94, pp. 103-167. 2014.

[30]. J.J. Durillo, A.J. Nebro, F. Luna, and E. Alba, "Solving Three-objective Optimization Problems using a New Hybrid Cellular Genetic Algorithm", Proc. of the PPSN, pp. 661-670, 2008.

[31]. K.R. Walcott, M.L. Soffa, G.M. Kapfhammer, and R.S. Roos, "Time-Aware Test Suite Prioritization", Proc. of the International Symposium on Software Testing and Analysis (ISSTA). pp.1-12. 2006.

[32]. S. Droste, et al., "On the analysis of the (1+1) evolutionary algorithm", Theoretical Computer Science 276 (1-2). pp. 51-81. 2002.

[33]. A. Arcuri, "It really does matter how you normalize the branch distance in search-based software testing", Software Testing, Verification and Reliability 23(2). pp. 119-147. 2013.

[34]. T. Yue, L.C. Briand and Y. Labiche, "Facilitating the Transition from Use Case Models to Analysis Models: Approach and Experiments", TOSEM 22(1), 2013.

[35]. T. Friedrich, T. Kroeger and F. Neumann, "Weighted preferences in evolutionary multi-objective optimization", International Journal of Machine Learning and Cybernetics, 4 (2), pp 139-148, 2012.

[36]. T. Friedrich, T. Kroeger and F. Neumann, "Weighted preferences in evolutionary multi-objective optimization", Australasian Conference on Artificial Intelligence, pp. 291-300, 2011.

[37]. D. Brockhoff, et al., "Directed Multiobjective Optimization Based on the Weighted Hypervolume Indicator", Journal of Multi-criteria Decision Analysis, vol (20), pp. 291-317, 2013.

[38]. E. Zitzler, D. Brockhoff, and L. Thiele, "The Hypervolume Indicator Revisited: On the Design of Pareto-compliant Indicators Via Weighted Integration", International Conference on Evolutionary Multi-Criterion Optimization, pp. 862-876, 2007.

[39]. E. Zitzler and L., Thiele, "Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach", Transactions on Evolutionary Computation, vol. 3, no. 4, pp. 257–271, 1999.

[40]. A.S. Sayyad, J. Ingram, T. Menzies and H. Ammar, "Optimum feature selection in software product lines: let your model and values guide your search", Proc. of the 1st International Workshop on Combining Modelling and Search-Based Software Engineering, pp. 22-27, 2013.

[41]. J. Knowles, L. Thiele, and E. Zitzler, "A Tutorial on the Performance Assessment of Stochastic Multiobjective Optimizers", Computer Engineering and Networks Laboratory (TIK), TIK Report 214, 2006.

[42]. P. Green, T. Menzies, et al., "Understanding the Value of Software Engineering Technologies", Proc. of ASE, pp. 52-61, 2009.

[43]. D. Brockhoff, T. Friedrich, and F. Neumann, "Analyzing hypervolume indicator based algorithms", Proc. of the PPSN, 2008, pp. 651–660.

[44]. J.J. Durillo, A.J. Nebro, "jMetal: a Java Framework for Multi-Objective Optimization", Advances in Engineering Software 42, 760-771. 2011.

[45]. Y. Zhang, M. Harman, and A. Mansouri, " The SBSE Repository: A repository and analysis of authors and research articles on Search Based Software Engineering", CREST Centre, UCL.

[46]. S. Wang, S. Ali, T. Yue, Y. Li and M. Liaaen, "A Practical Guide to Select Quality Indicators for Assessing Pareto- Based Search Algorithms in Search-Based Software Engineering", Proc. of 38th International Conference on Software Engineering, pp. 631-642, 2016.

[47]. S. Wang, S. Ali, T. Yue, and M. Liaaen, "UPMOA: An improved search algorithm to support user-preference multi-objective optimization", Proc. of IEEE 26th International Symposium on Software Reliability Engineering (ISSSRE), pp. 393-404, 2015.

[48]. Y. Li, T. Yue, S. Ali, K. Nie and L. Zhang, "Zen-ReqOptimizer: A Search-based Approach for Requirements Assignment Optimization", Empirical Software Engineering Journal, 2016.

[49]. P. Dipesh, S. Wang, S. Ali, and T. Yue. Search-Based Cost-Effective Test Case Selection within Time Budget: An Empirical Study", Proc. of the Genetic and Evolutionary Computation Conference (GECCO), 2016.

[50]. S. Wang, S. Ali, T. Yue, Ø. Bakkeli, and M. Liaaen, "Enhancing test case prioritization in an industrial setting with resource awareness and multi-objective search", Proc. of 38th International Conference on Software Engineering (Companion), pp. 182-191, 2016.

[51]. J. Eggermont and J. I. Hemert, "Stepwise Adaptation of Weights for Symbolic Regression with Genetic Programming", Proc. of the Twelveth Belgium /Netherlands Conference on AI, 2000.

[52]. A. Arcuri and G. Fraser, "Parameter tuning or default values? An empirical investigation in search-based software engineering", Journal of Empirical Software Engineering, 18 (3), pp. 594-623, 2013.

[53]. P. McMinn, "Search-Based Software Testing: Past, Present and Future", Proc. of the Fourth International Conference on Software Testing, Verification and Validation Workshops, pp. 153-163, 2011.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TEVC.2017.2778560, IEEE Transactions on Evolutionary Computation

Appendix A: TABLE I An Overview of the Industrial Case Studies and Artificial Problems

Domain	Problems	Objective	Objective Formulas	Case Study	User Preferences
Communication	Test Suite Minimization (TM)	**Test Minimization Percentage (*TMP*)**: The number of test cases that can be minimized	$TMP = \left(1 - \frac{nt_{minimized}}{nt_{original}}\right) * 100\%$	1: Fours VCSs from *Saturn*: C20 with 17 features and 138 test cases; C40 with 25 features and 167 test cases; C60 with 32 features and 192 test cases; C90 with 43 features and 230 test cases. 2: 32000 artificial problems.	1: Industrial case study: *FDC* and *FPC* have higher priority than *TMP*, *OET*. 2: Artificial problem: 64 user preference sets.
		Feature Pairwise Coverage (*FPC*): How many pairs of features can be covered	$FPC = \frac{NumFP_{minimized}}{NumFP_{p_i}}$		
		Fault Detection Capability (*FDC*): How many test cases can manage to find faults	$FDC = \frac{\sum_{i=1}^{nt_{minimized}} SucR_{tc_i}}{nt_{minimized}}$		
		Overall Execution Time (*OET*): How long it takes to executed the minimized test cases	$OET = \sum_{i=1}^{nt_{minimized}} AET_{tc_i}$		
	Test Case Prioritization (TP)	**Prioritized Extent (*PE*):** The number of test cases that can be prioritized	$PE = \frac{nt_{prioritized}}{nt}$	1: A testing cycle with 257 test cases for testing 53 features (functionalities) and 59 available test resources (to setup the test environment). 2: 32000 artificial problems.	1: Industrial case study: *FDC* has the highest priority and *FPC* is more important than *PE* and *OEC*. 2: Artificial problem: 64 user preference sets.
		Feature Pairwise Coverage (*FPC*): How many pairs of features can be covered	$FPC_{s_k} = \frac{NumFP_{prioritized}}{NumFP}$		
		Fault Detection Capability (*FDC*): How many test cases can manage to find faults	$FDC = \frac{\sum_{i=1}^{nt_{prioritized}} SucR_{tc_i}}{nt_{prioritized}}$		
		Overall Execution Cost (*OEC*): How long it will take to setup the required test resources and execute the prioritized test cases	$OEC = \sum_{i=1}^{nt_{prioritized}} (AET_{tc_i} + TTR_{tc_i})$		
Maritime	Test Case Selection (TS)	**Time Difference *(TD)*:** The difference between the execution time of selected test cases and a pre-given time budget	$TD = tb - \sum_{i=1}^{nt_{selected}} AET_{tc_i}$	1: A real world case study that consists of 165 test cases with the time budget as 200 minutes. 2: 32000 artificial problems.	1: Industrial case study: *MPO* has highest priority and *MPR* and *MC* should have higher priority than *TT*. 2: Artificial problem: 64 user preference sets.
		Mean Priority (*MPR*): The overall importance of the selected test cases based on the type of requirements under test	$MPR = \frac{\sum_{i=1}^{nt_{selected}} pr_{tc_i}}{nt_{selected}}$		
		Mean Probability (*MPO*): The likelihood that a given set of selected test cases can detect a fault	$MPO = \frac{\sum_{i=1}^{nt_{selected}} po_{tc_i}}{nt_{selected}}$		
		Mean Consequence *(MC)*: The potential impact caused by failure of selected test cases	$MC = \frac{\sum_{i=1}^{nt_{selected}} c_{tc_i}}{nt_{selected}}$		
Oil&Gas	Requirement Allocation (RA)	***ASSIGN*:** What extent all the requirements can be assigned to stakeholders	$ASSIGN = \frac{\sum_{i=1}^{n_{St}} n_{AR_i}}{n_R}$	1: A real world case study including 287 requirements and 10 stakeholders. 2: 7500 artificial problems.	1: Industrial case study: *FAM* has highest priority and *OWL* should have higher priority than *ASSGIN*. 2: Artificial problem: 15 sets of user preferences.
		FAM: Overall familiarities of the stakeholders for the requirements assigned	$FAM = \frac{\sum_{i=1}^{n_{St}} \sum_{j=1}^{n_{AR_i}} \frac{FM_{ji} - FM_{min}}{(FM_{max} - FM_{min})}}{\sum_{k=1}^{n_{St}} n_{AR_k}}$		
		OWL: Overall workload differences between stakeholders	$OWL = \frac{\sum_{j=1}^{n_{St}-1} \sum_{k=j+1}^{n_{St}} abs(WL_j - WL_k)}{n_{St} * (n_{St} - 1)}$		

Appendix B: TABLE II An Overview of the Experiment Design

RQ	Task	Comparison		Problem	Algorithm Parameters
1	T_1	UPMOA	UDW	**Test Suite Minimization (TM)** with *Fit_TM= f(1-Norm(TMP),1-Norm(FPC),1-Norm(FDC),Norm(OET))* **Test Case Prioritization (TP)** with *Fit_TP= f(1-Norm(PE),1-Norm(FPC),1-Norm(FDC), Norm(OEC))* **Test Case Selection (TS)** with *Fit_TS=f(Norm(TT), 1-Norm(MPR), 1-Norm(MPO), 1-Norm(MC))* **Requirement Allocation (RA)** with *Fit_RA= f(1-Norm(ASSIGN), 1-Norm(FAM), Norm(OWL))*	**Crossover rate:** {0, 0.2, 0.5, 0.8, 1} **Population size:** {5, 10, 50, 100, 200} **Mutation rate:** {0, 0.2, 0.5, 0.8, 1} **Maximum number of fitness evaluation**: {5000, 100000, 15000, 20000, 250000}
			RAW		
			FW		
2	T_2	UPMOA, NSGA-II RWGA, UDW-(1+1) EA			
3	T_3	UPMOA with 625 parameter settings			
4	T_4	UPMOA, NSGA-II RWGA, UDW-(1+1) EA			The best algorithm parameter setting from T_3