Random-Weighted Search-Based Multi-objective Optimization Revisited

Shuai Wang[1, 2], Shaukat Ali[1], and Arnaud Gotlieb[1]

[1]Certus Software V&V Center, Simula Research Laboratory, Norway
[2]Department of Informatics, University of Oslo, Norway
{shuai,shaukat,arnaud}@simula.no

Abstract. Weight-based multi-objective optimization requires assigning appropriate weights using a weight strategy to each of the objectives such that an overall optimal solution can be obtained with a search algorithm. Choosing weights using an appropriate weight strategy has a huge impact on the obtained solutions and thus warrants the need to seek the best weight strategy. In this paper, we propose a new weight strategy called Uniformly Distributed Weights (*UDW*), which generates weights from uniform distribution, while satisfying a set of user-defined constraints among various cost and effectiveness measures. We compare *UDW* with two commonly used weight strategies, i.e., Fixed Weights (*FW*) and Randomly-Assigned Weights (*RAW*), based on five cost/effectiveness measures for an industrial problem of test minimization defined in the context of Video Conferencing System Product Line developed by Cisco Systems. We empirically evaluate the performance of *UDW*, *FW*, and *RAW* in conjunction with four search algorithms ((1+1) Evolutionary Algorithm (EA), Genetic Algorithm, Alternating Variable Method, and Random Search) using the industrial case study and 500 artificial problems of varying complexity. Results show that *UDW* along with (1+1) EA achieves the best performance among the other combinations of weight strategies and algorithms.

Keywords: Uniformly distributed weights, multi-objective optimization, search algorithms.

1 Introduction

A weight-based multi-objective optimization problem requires finding an optimal set of weights for all the objectives, while satisfying the required constraints (e.g., the priority of various objectives defined by users) on weights capturing the complex tradeoff relationships among the objectives with the aim of finding an overall optimal solution for the problem. Such weight-based multi-objective problems have been solved efficiently together with search algorithms (e.g., Genetic Algorithms) in the literature [1-6]. As compared with other types of commonly used techniques (i.e., Pareto-based techniques) [1, 5], weight-based techniques offer the following advantages: 1) These techniques balance all the objectives to find an optimal solution rather than obtaining a set of non-dominated solutions thus eliminating the effort for users to

C. Le Goues and S. Yoo (Eds.): SSBSE 2014, LNCS 8636, pp. 199–214, 2014.

select among the obtained solutions; 2) These techniques are straightforward to implement with computational efficiency; and 3) When objectives have different priorities, weight-based techniques can tackle such situation easily by assigning customized weights to each particular objective.

In the literature [1], the following two weight assignment strategies are commonly used: 1) Assigning fixed weights (equal or unequal) to each objective, termed as *Fixed Weight* (*FW*) strategy in this paper; 2) Assigning random weights to each objective based on a set of pre-defined constraints, coined as *Randomly-Assigned Weights* (*RAW*) strategy. Even though these two strategies have shown promising results [1-4, 6], they still suffer from some limitations. With *FW*, it is rare that all objectives have equal weights and determining appropriate weights usually depends on domain-expertise. A potential solution is to ask users to specify a set of constraints among weights rather than giving exact values. With *RAW*, we observe that it does not guarantee equally-distributed uniformity of selection of weight, i.e., all the potential weights do not have the same probability to be selected. Since one set of fixed weights can determine a specific search direction, there may be no equivalent probability to choose various search directions to find an optimal solution [1].

To reduce the randomness in weight selection using *RAW*, we propose a new weight assignment strategy called *Uniformly Distributed Weights* (*UDW*), which generates weights from a uniform distribution, while meeting a set of user-defined constraints. The strategy gives an equal importance to the generation of weight for each criterion, while preserving the relative importance of criterion. Said otherwise, the goal is to keep the advantages of uniform random generation, while having priorities defined among the various criteria. We evaluate the proposed weight strategy *UDW* as compared with the *FW* and *RAW* using an industrial problem of test minimization for Video Conferencing Systems (VCSs) product line from Cisco Systems. Being specific, this test minimization problem is a multi-objective optimization problem having five distinct objectives: Test Minimization Percentage (*TMP*), Feature Pairwise Coverage (*FPC*), Fault Detection Capability (*FDC*), Average Execution Frequency (*AEF*), and Overall Execution time (*OET*). A fitness function defined on all these five objectives is used in conjunction with the following search algorithms: Genetic Algorithm (GA), (1+1) Evolutionary Algorithm (EA), and Alternating Variable Method (AVM) to compare the three distinct weight assignment strategies. Random Search (RS) is used as the comparison base line. Moreover, inspired by the industrial problem, we created 500 artificial problems of varying complexity to evaluate the three weight assignment strategies in conjunction with all the four algorithms.

The obtained results show that: 1) With *FW*, *RAW* and *UDW*, (1+1) EA significantly outperformed the other search algorithms; 2) With (1+1) EA, *UDW* significantly performed better than *FW* and *RAW*; 3) The performance of (1+1) EA and GA with *UDW* was significantly improved with the increasing complexity of problems.

The rest of the paper is organized as follows: Section 2 provides a relevant background on *FW* and *RAW*. Section 3 presents *UDW* strategy followed by the description of our industrial and artificial case studies (Section 4). Section 5 presents the empirical evaluation with an overall discussion and Section 6 addresses threats to validity. Related work is discussed in Section 7 and Section 8 concludes the paper.

2 Background

Fixed Weights (*FW*) assigns fixed normalized weights (between 0 and 1) to each objective [1]. These weights can be obtained from domain knowledge of experts. For instance, our case requires five weights (w_1, w_2, w_3, w_4, w_5) corresponding to each objective (*TMF*, *FPC*, *FDC*, *OET*, *AEF*).

Randomly-Assigned Weights (*RAW*), inspired by Random-Weighted Genetic Algorithm (RWGA) mainly used for weight-based optimization [1], generates a set of distributed normalized weights for each objective while still satisfying the user defined constraints. For instance, we have a constraint $w_2 > w_1$ and using *RAW*, a random distributed value is first generated for w_2 from 0 to 1 and then another distributed value is generated for w_1 from 0 to w_2. At each generation when running search algorithms, each objective is assigned to a distributed normalized weight as above mentioned and meets all the defined constraints, i.e., the weights for each objective are dynamically changed during each generation until the best solution is found or the termination criteria for the algorithms is met. Using *RAW*, multiple search directions can be stipulated by assigning dynamic weights during each generation [1]. However, *RAW* cannot guarantee that each point within the range for weights has the same probability to be selected thus each search directions cannot be reached uniformly.

3 Uniformly Distributed Weights

Inspired by our previous work in [7], the Uniformly Distributed Weights (*UDW*) uniformly selects weights at random when these weights are subject to a set of defined constraints. Notice that solving this problem efficiently is challenging as sampling uniformly tuples of values from an unknown domain is not trivial. More specifically, the tuples of weights generated by *UDW* must satisfy: 1) all the user-defined constraints and 2) an equi-probability selection that guarantee that each search direction has equivalent probability to be reached[1]. Formally speaking, consider a multi-objective optimization problem P involving m optimization objectives $O = \{o_1, o_2, \dots, o_m\}$, each objective has a specific weight, corresponding to the set $W = \{w_1, w_2, \dots, w_m\}$. Relations among these objectives are captured by a set of n arithmetic constraints $C = \{c_1, c_2, \dots, c_n\}$ over the variables in *W*. Notice that such constraints can be pre-defined by the users based on their particular domain expertise.

The core idea of *UDW* lies in the pre-computations of subdomains of the input domain (formed by the Cartesian product of each individual weight domain) and consideration of subdomains out of which uniform random sampling is trivial. More precisely, using an arbitrary division parameter *k*, *UDW* first divides the input domain into k^m equivalent subdomains, where *m* is the number of weight variables. Second, a systematic refutation is used to eliminate subdomains that do not satisfy the constraints, and finally, uniform random generation of tuples of weights is realized by selecting a remaining subdomain and a tuple at random. Notice that the selected subdomain and tuple may still not satisfy the constraints, and thus rejection may still happen. But, the systematic refutation process would have eliminated most parts of the input domain that do not contain any solution (fully conflicts with the constraints).

The *UDW* algorithm (shown as below) takes as inputs, the set of weights *W*, the constraint set *C*, the division parameter *k*, and the length of the expected sequence of weight tuples (*K*), i.e., the number of generations during the search[1]. The output of *UDW* is a sequence of *K* weight tuples for *W,* called *WK,* such that the sequence is uniformly distributed over the solution set of *C*. Notice that *UDW* returns fail ($WK = \emptyset$) when none of the weight tuples satisfy the constraint set *C*, which shows none of the solutions will be found for the optimization problem based on the defined constraints. The Algorithm works as follows: a weight tuples sequence (*WK*) is first initialized as empty (*Step 1*) and then the input space is divided into k^m subdomains, i.e., $\{WD_1, \dots, WD_{k^m}\}$ (*Step 2*). The refutation process eliminates a number of subdomains with respect to the constraint set *C* (*Step 3*). The remaining subdomains (i.e., $\{WD'_1, WD'_2, \dots, WD'_p\}$) are sampled uniformly and from them, uniform tuples of weights are generated at random until *K* sets are generated (*Step 4*). Notice that at this step, only the generated tuples that satisfy the constraint set *C* are kept. The generated set of tuples *K* is returned and used to assign weights to each objective, in order to guide the search in the search algorithms (*Step 5*). Notice that despite some similarities with the algorithm presented in [7, 8], our *UDW* algorithm is original in terms of uniformly generating random weights for multi-objective test suite optimization.

Algorithm *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.

---

4 Case Studies

Industrial Case Study. Our industrial partner is a product line of Video Conferencing Systems (VCSs) called *Saturn* developed by Cisco Norway [8]. *Saturn* has several products, e.g., C20 (low end product) and C90 (high end product). Test suite minimization for testing a product is essential since it is practically impossible to execute all the test cases developed for the whole product line within the allocated budget [9, 10]. However, such minimization may have descendent impact on the effectiveness of testing (e.g., fault detection capability) when reducing the number of test cases. Thus, this minimization problem can be formulated as a multi-objective optimization

[1] Each generation requires assigning a new set of weights to each objective during the search.

problem that is well solved by various search algorithms [1, 5], i.e., we aim at reducing the cost of testing while preserving the effectiveness. We chose this problem to evaluate our proposed weight assignment strategy and such problem can be represented formally as: search for a solution s_k (a subset of the test cases) from the solution space S (all combinations of the test cases for testing a given product) to achieve the following objectives (i.e., maximum effectiveness and minimum cost): $\forall s_l \in S \cap s_l \neq s_k$:

$$Effectiveness(s_k) \geq Effectiveness(s_l) \text{ and } Cost(s_k) \leq Cost(s_l)$$

Moreover, three effectiveness measures were previously defined in [6]: *TMP* was used to measure the amount of reduction for the number of test cases as compared with the original test suite; *FPC* and *FDC* were defined to calculate the feature pairwise coverage (each feature represents a testing functionality for VCS testing) and fault detection capability achieved by a potential solution. More detailed definitions and mathematical formulas for these effectiveness measures can be consulted in [6]. Through more investigation with Cisco, an additional effectiveness measure called Average Execution Frequency (*AEF*) was defined to count the average execution frequency for a solution thereby measuring its priority and a cost measure called Overall Execution Time (*OET*) was defined to measure the execution time cost for the potential solution obtained by search algorithms. Moreover, a fitness function based on the cost/effectiveness measures was defined to guide the search. This fitness function converted multi-objective minimization problem into single objective problem based on the weight-based theory [1], which is shown as follows:

$$Fit\_F = 1 - w_1 * nor\ (TMP) - w_2 * nor\ (FPC) - w_3 * nor\ (FDC) - w_4 * \big(1 - nor(OET)\big) - w_5 * nor\ (AEF)$$

Notice $nor\ (x)$ is a normalization function, which is computed as follows: $x/(x+1)$. A lower value of fitness function *Fit_F* (we called such value as Overall Fitness Value (*OFV*)) represents a better solution. Moreover, w_1, w_2, w_3, w_4 and w_5 are a set of weights assigned to *TMP*, *FPC*, *FDC*, *OET* and *AEF* respectively and are required to satisfying a basic constraint $\sum_{i=1}^{5} w_i = 1$ in addition to other constraints capturing tradeoff relationships among the objectives.

We chose four products C20, C40, C60 and C90 from *Saturn* that includes 169 features and each product can be represented by a subset of these features. Each feature can be tested by at least one test case. More specifically, C20, C40, C60 and C90 includes 17, 25, 32 and 43 features respectively and 138, 167, 192 and 230 test cases relevant for testing these products, respectively. 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* ($OET = \sum_i AET_{tc_i}$) and an execution frequency (EF_{tc_i}) for obtaining *AEF* ($AEF = \frac{\sum_i EF_{tc_i}}{n_s}$, n_s is the number of test cases included in a specific solution). In summary, for *Saturn*, each feature is associated with 5-10 test cases and each test case is associated with 1-5 features with $SucR_{tc_i}$ ranging from 50% to 95%, AET_{tc_i} ranging from 2 minutes to 60 minutes and EF_{tc_i} ranging from 1 time to 50 times per week.

Artificial Problems. We further defined 500 artificial problems to evaluate the performance of *FW*, *RAW* and *UDW*. Notice that the artificial problems are inspired by our industrial case study but with expansion for generality. To achieve this, we first created a feature model containing 1200 features and a test case repository of 60,000 test cases. For each test case tc_i, three key attributes are assigned randomly, i.e., $SucR_{tc_i}$ ranging from 0% to 100%, AET_{tc_i} ranges from 1 minutes to 100 minutes and EF_{tc_i} ranges from 1 time to 100 times per week. Moreover, we created artificial problems with the increasing number of features and associated test cases for each feature, i.e., we used a range of 10 to 1000 with an increment of 10 for features number and each feature can be associated with test cases ranging from 5 to 45 with an increment of 10. Thus, 100*5 = 500 artificial problems were obtained in this way. Notice that such artificial problems are also designed based on the domain expertise of VCS testing and thorough discussions with test engineers at Cisco.

5 Empirical Evaluation

5.1 Experiment Design

Recall that our main goal is to minimize the test suite for testing a product with high effectiveness (*TMP*, *FPC*, *FDC* and *AEF*) while meeting the time budget (*OET*).

To generate suitable weights, it is of paramount importance to provide a set of pre-defined constraints in a given context. In our industrial case study, through the domain analysis and several thorough discussion with test engineers at Cisco, we observed that: 1) *OET* has the highest priority among all the objectives; 2) *FPC* and *FDC* are more important than *TMP* and *AEF*; and 3) *AEF* has the lowest priority among all the objectives. So we came up with five key independent constraints for the test minimization problem, i.e., $w_4 > w_2$, $w_4 > w_3$, $w_2 > w_1$, $w_3 > w_1$ and $w_1 > w_5$. Based on the defined constraints, specific weights for each objective can be generated using different weight strategies for the industrial case study and artificial problems.

Using the experiments, we want to address the following research questions:

RQ1: With each weight strategy (i.e., *UDW*, *FW* and *RAW*), which search algorithm achieves the best performance for each objective and *OFV*?

RQ2: With the best search algorithm, which weight strategy can achieve the best performance for each objective and *OFV*?

RQ3: How does the increment of the number of features and associated test cases influence the performance of the search algorithms along with each weight strategy?

Being specific, our experiment first compares each pair of the four algorithms with each weight strategy to determine which algorithm can achieve the best performance (RQ1). Afterwards, we choose the best algorithm and compare its performance with each weight strategy to evaluate whether *UDW* can outperform the other two weight strategies (RQ2), which also shows which combination of search algorithms and weight strategies can achieve the best performance. Notice that for industrial case study, we evaluate the values for each objective (i.e., *TMP*, *FPC*, *FDC*, *OET* and

AEF) and as for artificial problems, we only evaluate the values of *OFV* to assess the performance and scalability for the selected algorithms with different weight strategies. To address RQ3, Mean Fitness Value for each problem ($MFV_{i,j} = \frac{\sum_{r=1}^{nr} OFV_r}{nr}$) is defined to measure the mean overall fitness value for a certain number of runs *nr* (in our case, *nr* = 100), where i is the feature number and j is the test case number ($10 \leq i \leq 1000$ with an increment of 10 and $5 \leq j \leq 45$ with an increment of 10). OFV_r is the obtained fitness value after the r_{th} run.

Moreover, Mean Fitness Value for Feature ($MFV\_F_i = \frac{\sum_{j=5}^{45} MFV_{i,j}}{5}$) and Mean Fitness Value for Test Case ($MFV\_TC_j = \frac{\sum_{i=10}^{1000} MFV_{i,j}}{100}$) are further defined to measure the mean fitness function in a given number of features or associated test cases. In this way, a specific *MFV_F* and *MFV_TC* can be calculated for each number of features and test cases and RQ3 can be addressed using statistical analysis.

In addition, for all the search algorithms, the maximum number of evaluation for the fitness function is set as 5000 and we collected the optimal solution after the 5000_{th} fitness function evaluation. For GA and (1+1) EA, the mutation of a variable is done with the standard probability $1/n$, where n is the number of variables. We used a standard one-point crossover with a rate of 0.75 for GA and set the size of population as 100. RS was used as the comparison baseline to assess the difficulty of the problems [11]. According to the guidelines in [11, 13], each algorithm is run for 100 times to account for random variation inherited in search algorithms.

5.2 Statistical Tests

To analyze the obtained result, the Vargha and Delaney statistics and Mann-Whitney U test are used based on the guidelines for reporting statistical tests for randomized algorithms [11]. The Vargha and Delaney statistics is used to calculate $\hat{A}_{12}$, which is a non-parametric effect size measure. In our context, $\hat{A}_{12}$ is used to compare the probability of yielding higher values for each objective function and overall fitness value (*OFV*) for two algorithms *A* and *B* with different weight strategies. If $\hat{A}_{12}$ is 0.5, the two algorithms have equal performance. If $\hat{A}_{12}$ is greater than 0.5, *A* has higher chances to obtain better solutions than *B*. The Mann-Whitney U test is used to calculate *p*-value for deciding whether there is a significant difference between *A* and *B*. We choose the significance level of 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 RQ3, we choose the Spearman's rank correlation coefficient (ρ) to measure the relations between the *MFV_F* and *MFV_TC* obtained by the algorithms with different number of features and test cases [12]. More specially, there is a positive correlation if ρ is greater than 0 and a negative correlation when ρ is less than 0. A ρ close to 0 shows that there is no correlation between the 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.

5.3 Results and Analysis

Results and Analysis for Industrial Case Study. Notice that due to limited space, all detailed results can be consulted in the technical report [18] (i.e., Table 4 and Table 5). We only present the key findings for each research question (RQ1-RQ2).

Results and Analysis for RQ1. Based on the obtained results (Table 4 in [18]), we concluded that all the three search algorithms (i.e., AVM, GA and (1+1) EA) significantly outperformed RS for each objective and *OFV* with each weight strategy (i.e., *FW*, *RAW* and *UDW*). Moreover, (1+1) EA achieved significantly better performance than GA and AVM for each objective and *OFV* (RQ1). In addition, AVM had significantly worse performance than GA for each objective and *OFV*. In summary, for each objective and *OFV* for the four *Saturn* products, the performance of the four algorithms can be sorted as (1+1) EA, GA, AVM and RS, from better to worst.

Results and Analysis for RQ2. Based on the results of RQ1, we chose the best algorithm (1+1) EA and compared its performance in conjunction with *FW*, *RAW* and *UDW* for each objective and *OFV* in the four *Saturn* products (Table 5 in [18]). According to the results, we concluded that (1+1) EA along with *RAW* and *UDW* significantly outperformed (1+1) EA with *FW* for each objective and *OFV*. Moreover, the performance of (1+1) EA with *UDW* was also significantly better than (1+1) EA with *RAW* for *TMP*, *FPC*, *FDC*, *OET*, *AEF* and *OFV*. Meanwhile, we report the average time used by each algorithm per run, i.e., 2.17 seconds for AVM, 1.78 seconds for GA, 1.55 seconds for (1+1) EA and 1.20 seconds for RS, which shows running search algorithms require similar effort (i.e., time) as compared with RS. In summary, we concluded (1+1) EA along with *UDW* achieved the best performance.

Results and Analysis for Artificial Problems. Recall that the evaluation for artificial problems is based on the overall fitness value (*OFV*) for each of the 500 problems. To answer RQ3, we calculated the spearman's rank correlation using mean fitness values *MFV_F* and *MFV_TC* (Section 5.1) for each algorithm with different weight strategies. The detailed results and analysis are discussed in detail as below.

Results and Analysis for RQ1. Table 1 summarizes the results for comparing the selected search algorithms with *FW*, *RAW* and *UDW* for the 500 artificial problems. Two numbers are shown in each cell of the table split by a slash. The first number in the column *A>B* shows the number of problems out of 500 for which an algorithm *A* has better performance than *B* ($\hat{A}_{12} > 0.5$), *A<B* means vice versa ($\hat{A}_{12} < 0.5$), and *A=B* means the number of problems for which *A* has equivalent performance as *B* ($\hat{A}_{12} = 0.5$). The second number after *"/"* in the column *A>B* means the number of problems out of 500 for which *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 is no significant difference in performance between *A* and *B* ($p \geq 0.05$) .We concluded the results as below.

AVM vs. GA: AVM outperformed GA for on average 40.8% problems (i.e., (167+231+214)/3/500*100% = 40.8%), but for 34.94% problems, AVM performed significantly better than GA. AVM performed worse than GA for on average 59.2%

problems and in 52.4% problems, AVM was significantly worse than GA. There were no significant differences between AVM and GA for 12.66% problems.

AVM vs. (1+1) EA: AVM performed better than (1+1) EA for on average 22.92% problems and 17.86% out of these 22.92% problems, AVM significantly outperformed (1+1) EA. On the contrary, for on average 77.06% problems, AVM performed worse than (1+1) EA and for 72.6% problems, AVM performed significantly worse than (1+1) EA. For 9.54% problems on average, there were no significant differences between AVM and (1+1) EA.

Table 1. Results for comparing the algorithms with each weight strategy for artificial problems

Weight Strategy	Pair of Algorithms	A>B	A<B	A=B	Best Algorithm
FW	AVM vs. GA	167/132	333/297	0/71	(1+1) EA
	AVM vs. (1+1) EA	121/98	379/344	0/58	
	AVM vs. RS	377/345	123/101	0/54	
	GA vs. (1+1) EA	134/115	366/348	0/37	
	GA vs. RS	364/331	136/112	0/57	
	(1+1) EA vs. RS	436/408	64/37	0/55	
RAW	AVM vs. GA	231/199	269/238	0/63	(1+1) EA
	AVM vs. (1+1) EA	107/76	393/376	0/48	
	AVM vs. RS	312/267	188/135	0/98	
	GA vs. (1+1) EA	195/164	305/276	0/60	
	GA vs. RS	397/340	103/63	0/97	
	(1+1) EA vs. RS	449/423	51/38	0/39	
UDW	AVM vs. GA	214/193	286/251	0/56	(1+1) EA
	AVM vs. (1+1) EA	116/94	384/369	0/37	
	AVM vs. RS	336/290	164/129	0/81	
	GA vs. (1+1) EA	172/149	328/302	0/49	
	GA vs. RS	369/323	131/105	0/72	
	(1+1) EA vs. RS	427/399	73/54	0/47	

AVM vs. RS: AVM outperformed RS for 68.34% problems with the three weight strategies on average but for 60.14% problems the results were statistically significant. There were no significant differences for 15.54% problems on average.

GA vs. (1+1) EA: For 33.4% problems on average, GA was better than (1+1) EA, but for 29.2% problems, GA was significantly better than (1+1) EA. Moreover, GA was worse than (1+1) EA for 66.6% problems and for 61.74% out of these 66.6% problems, GA was significantly worse than (1+1) EA. There were no significant differences between GA and (1+1) EA for 9.74% problems.

GA vs. RS: There were on average 75.34% problems, for which GA achieved better performance than RS with the selected weight strategies and for average 66.26% problems, GA significantly outperformed RS. Moreover, for 15.06% problems, there were no significant differences between GA and RS.

(1+1) EA vs. RS: In case of (1+1) EA, it performed better than RS for 87.46% problems. Out of these 87.46% problems, on average 82% problems, (1+1) EA was significantly better than RS. For 9.4% problems, there were no significant differences between (1+1) EA and RS.

Concluding Remarks for RQ1: Based on the results, RQ1 can be answered as follows: the performance of AVM, GA and (1+1) EA are all significantly better than RS

with all the selected three weight strategies in term of finding an optimal solution for our minimization problem. Moreover, (1+1) EA outperformed GA and GA performed better than AVM together with all the weight strategies. In summary, (1+1) EA achieved the best performance for the three weight strategies in most of the problems.

Results and Analysis for RQ2. Table 2 summarizes the results for comparing *FW*, *RAW* and *UDW* with the best algorithm (1+1) EA for the 500 artificial problems. The data in the columns $A>B$, $A<B$ and $A=B$ is organized in the same way as in Table 1.

Table 2. Results for comparing the weight strategies along with (1+1) EA for artificial problems

Pair of Weight Strategies	A>B	A<B	A=B
FW vs. *RAW*	187/165	313/276	0/59
FW vs. *UDW*	119/91	381/364	0/45
RAW vs. *UDW*	197/175	303/276	0/49

FW vs. RAW: In conjunction with (1+1) EA, for 37.4% (187/500=37.4%) problems, *FW* outperformed *RAW*, but for 33% of problems, there were significant differences. On the contrary, *RAW* performed better than *FW* for 62.6% problems and for 55.2% problems, *RAW* was significantly better than *FW*. There were no significant differences between *FW* and *RAW* for 11.8% problems.

FW vs. UDW: For (1+1) EA, *FW* performed better than *UDW* for 23.8% problems, but only for 18.2% problems, *FW* significantly outperformed *UDW*. Moreover, the performance of *FW* was worse than *UDW* for 76.2% problems and for 72.8% problems, *FW* was significantly worse than *UDW*. Meanwhile, there were no significant differences between *FW* and *UDW* for 9% problems.

RAW vs. UDW: Combined with (1+1) EA, for 39.4% problems, *RAW* performed better than *UDW* and there were significant differences for 35.2% out of these 39.4% problems. *UDW* outperformed *RAW* for 60.6% problems and *UDW* significantly outperformed *RAW* for 55.2% problems. In addition, there were no significant differences between *RAW* and *UDW* for 9.8% problems.

Similarly, the average time taken by each algorithm is reported per run for the 500 artificial problems, i.e., 4.58 seconds for AVM, 3.96 seconds for GA, 3.41 seconds for (1+1) EA and 3.04 seconds for RS, which shows adapting search algorithms takes similar time as compared with RS.

Concluding Remarks for RQ3: Based on the above results, we can answer RQ3 as follows: along with the best algorithm (1+1) EA, *UDW* achieved the best performance among the three weight strategies and *RAW* outperformed *FW* significantly in most of the artificial problems, i.e., the *UDW* weight strategy in conjunction with (1+1) EA achieved the best performance in our context.

Results and Analysis for RQ3. Table 3 provides the results for Spearman's correlation analysis (ρ) between mean fitness value for feature (*MFV_F*) with the increasing number of features and mean fitness value for test case (*MFV_TC*) with the growth of test cases for the 500 artificial problems. Recall that a lower value of *MFV_F* or *MFV_TC* represents a better performance of an algorithm with a weight strategy.

Increasing Number of Features: For *FW*, *RAW* and *UDW*, we observed that the *MFV_F* values obtained by (1+1) EA and GA decreased significantly with the growth of feature number since all the ρ values were less than 0 and the values of $Prob > |\rho|$ were all less than 0.0001 (Table 3), i.e., the performance of (1+1) EA and GA significantly improved as the increasing number of features. For AVM, the *MFV_F* values also decreased when the number of features increases but such decrease was not statistically significant, i.e., AVM also performed better as the increasing number of features but not significantly. Finally, the performance of RS was worse with the growth of the number of features (not significantly) (Table 3).

Table 3. Spearman's correlation analysis for artificial problems

Weight Strategy	Algorithms	Increasing Features		Increasing Test Cases	
		Spearman ρ	*Prob*>| ρ|	Spearman ρ	*Prob*>| ρ|
FW	AVM	-0.07	0.4513	-0.06	0.5436
	(1+1) EA	-0.68	**<0.0001**	-0.13	0.6143
	GA	-0.65	**<0.0001**	-0.12	0.5841
	RS	0.12	0.4870	0.07	0.5323
RAW	AVM	-0.14	0.3764	-0.09	0.5134
	(1+1) EA	-0.71	**<0.0001**	-0.17	0.5670
	GA	-0.64	**<0.0001**	-0.08	0.6537
	RS	0.16	0.5137	0.18	0.5758
UDW	AVM	-0.09	0.6125	-0.10	0.4582
	(1+1) EA	-0.70	**<0.0001**	-0.14	0.4319
	GA	-0.69	**<0.0001**	-0.11	0.5762
	RS	0.09	0.4626	0.15	0.4240

Increasing Number of Associated Test Cases (*Increasing Test Cases* column): The *MFV_TC* values by AVM, (1+1) EA and GA decreased but not significantly with the growth of associated test cases, i.e., the performance of AVM, (1+1) EA and GA in along with all the weight strategies improved (but not significantly) with the growth of associated test cases. On the contrary, RS performed worse (not significantly) when the number of associated test cases increased (Table 3).

Concluding Remarks: Among all the weight strategies, (1+1) EA and GA performed significantly better as the increasing number of features and the performance of AVM and RS were not significantly influenced by the number of features. Moreover, the performance of all the selected search algorithms with the weight strategies are not significantly influenced by the increasing the number of associated test cases.

5.4 Overall Discussion

First, based on the results of RQ1 and RQ2, the reason why *UDW* performs better than *FW* can be explained as follows: 1) *FW* uses fixed predefined weights, meaning that the search space, which is the Cartesian product of all weights, cannot be fully explored. Actually, the optimal solution might not be found in this restricted search space explored in a single direction; and 2) *UDW* uniformly assigns random weights generated during each generation of a search algorithm, which allows the search algorithm to explore multiple directions.

Moreover, with uniformly distributed weights (*UDW*) at each generation, a search algorithm can be guided towards an optimal solution more efficiently than randomly generated weights (*RAW*). This may be because that *RAW* does not permit us to search uniformly at each generation, as the weights cannot be selected with the same probability, due to the presence of constraints. Consider two weights w_1 and w_2 from 0 to 1; for the sake of simplicity, but without losing any generality, we suppose that $w_1, w_2 \in \{0, \frac{1}{N}, \frac{2}{N}, \dots, \frac{N}{N}\}$ and the following constraint holds: $w_1 > w_2$. Suppose that we have two distinct candidates for w_1 namely $w_1^0 = \frac{N_1}{N}, w_1^1 = \frac{N_2}{N}$, where $N_1, N_2 \in \{0,1, \dots, N\}$ and one candidate for w_2 called w_2^0 (the solution space of the constraint for w_1 and w_2 is represented as Fig. 1). Using *RAW*, if w_1^0 is selected first, the probability of selecting w_2^0 is $Prob(\{w_2^0|w_1^0\}) = \frac{1}{N} * \frac{1}{N_1}$, as the constraint $w_1 > w_2$ holds. If w_1^1 is selected, the probability of selecting w_2^0 is $Prob(\{w_2^0|w_1^1\}) = \frac{1}{N} * \frac{1}{N_2}$. As a consequence, the probability of selecting w_2^0 is not the same in both cases ($\frac{Prob(\{w_2^0|w_1^0\})}{Prob(\{w_2^0|w_1^1\})} = \frac{N_2}{N_1}$), meaning that the couples (w_1^0, w_2^0) and (w_1^1, w_2^0) do not have the same probability to be selected. Hence, there is no equi-probability when it comes to the selection of relevant search directions to find an optimal solution. Unlike *RAW*, *UDW* guarantees the equi-probability selection of w_2^0 whatever be the first weight value selected (i.e., $Prob(\{w_2^0|w_1^0\}) = Prob(\{w_2^0|w_1^1\}) = \frac{1}{\frac{N*N}{2}} = \frac{2}{N*N}$). Thus, each search direction has an equal probability to be reached and thereby, guiding search to find an optimal solution is more equally supported.

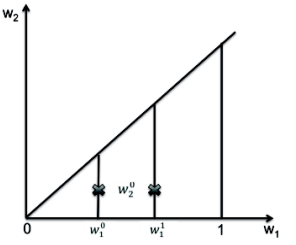

Fig. 1. An example for the probability of selecting weights

Followed by (1+1) EA, GA has significantly better performance than AVM. This can be explained from the fact that both of these algorithms are global search algorithms and thus managed to find global optimal solutions as compared to AVM, which is a local search algorithm. Notice that the performance of (1+1) EA is significantly better than GA and this may be due to the reason that (1+1) EA uses only mutation for exploring the search space as compared to GA which uses both mutation and crossover for exploration and exploitation of search space respectively requiring more generations to find a global optimal solution. By increasing the number of generations (5000 in the current experiment settings), we expect that the performance of GA can be improved, which requires further empirical evaluations.

For the experiments based on the 500 artificial problems, the results were consistent with our industrial case study (RQ1-RQ2). When we looked at the impact of varying number of features (10-1000) and associated test cases (5-45) on the performance of each algorithm along with three weight strategies (RQ3), we observed that (1+1) EA and GA performed significantly better as the increasing number of features, but for AVM and RS, the performance was not significantly influenced with the growth of features. Such interesting behavior can be explained based on the fact that (1+1) EA

and GA are global search algorithms, which can still manage to explore the search space and find a global optimal solution even with the increased complexity (more features). On the other hand, AVM's performance (local search) cannot scale with the increased complexity since AVM can be guided towards finding local solutions in the search space but the global optimal solutions might be missed. Moreover, we observed that increasing the number of test cases improved the performance of all search algorithms except RS though not significantly. This phenomenon may be due to the following two reasons: 1) Complexity of test minimization problem for a product is directly related to the number of features and thus increasing the test cases may not affect the performance of the search algorithms; 2) Increased number of test cases means that a feature can be tested with more test cases and thus increases the solution space within the entire search space, i.e., search algorithms can find solutions with better fitness since more solutions are available (though not statistically significant).

In summary, (1+1) EA along with *UDW* weight strategy achieved the best performance in our experiments and thus is suitable for test minimization problem in our industrial context. Moreover, the results suggest weights selected by domain experts might not be accurate to obtain an optimal test minimization solution in practice and thus automated weight strategies such as *UDW* are needed for an optimal solution.

6 Threats to Validity

A prominent construct validity threat is related to the measure used to compare the various algorithms and to avoid such threat we used the measured fitness value, which is comparable across all selected search algorithms. Another common construct threat to validity is the use of termination criterion for the search. In our experiments, we used number of fitness evaluations comparable across all the algorithms.

When using search algorithms, parameter settings may affect the performance of the algorithms (internal validity). In this direction, we used default parameter settings for all the algorithms and these settings have demonstrated promising results [13]. In addition, the complexity of *UDW* in the general case is exponential in the number of weight variables in the problem (i.e., $O(k^m)$ where k is an arbitrary division parameter and m is the number of weight variables). This is a limitation if one wants to consider optimization problem with more than 20 independent objectives ($m > 20$) but our experience with both academic and industrial case studies show that the number of considered objectives never goes up to seven. Consequently, this potential exponential blow-up is not considered as a threat to our approach in practice.

A common conclusion validity threat is due to random variation inherited in search algorithms and thus we repeated our experiments 100 times to reduce the probability that the results were obtained by chance. Moreover, we used appropriate statistical tests for analyzing the data, i.e., Vargha and Delaney, Mann-Whitney U test and Spearman's rank correlation coefficient based on the guideline proposed in [11].

Generalization to new case studies is required to increase the confidence on the results (external validity) and we conducted an empirical evaluation using 500 artificial problems besides an industrial case study and obtained consistent results.

7 Related Works

A comprehensive review for search-based software engineering (SBSE) is available in [5]. In particular, Harman listed a set of potential objectives used for multi-objective test optimization in regression testing [16], which have been extensively studied in the existing literature [14]. In [15], a two-objective problem (i.e., code coverage and execution time) is converted into a single-objective problem for test prioritization using an arithmetical combination of weights for the fitness function. However, there are at least two main differences with our work: 1) The *UDW* approach is not restricted to two criteria and can actually takes any number of test objectives into consideration (e.g., *TMP*, *FDC* and *AEF*) and 2) *UDW* is parameterized by a set of constraints for which it can provide a uniform sampling of weight values, which turns out to be essential when looking at the performance of various search algorithms (e.g., (1+1) EA). As compared with our previous work [6], the motivation in this paper is different. The determination of an appropriate weight assignment strategy using weight-based search algorithms turns out to be crucial to obtain an optimal solution, especially when determining the best possible weights is impossible [1].

In addition, a simple algorithm can be used alternatively to sample values uniformly at random in the presence of constraints. For example, [17] reported on such an algorithm: 1) firstly, it generates tuples of values randomly while ignoring the constraints; 2) secondly, it uses a linear constraint solver (e.g., the Simplex algorithm) to reject the generated tuples that do not satisfy the pre-defined constraints. Even if this approach is appealing by its simplicity, it does not scale up to large dimensions as shown in [7]. In fact, as soon as the constraints become complex (relational, non-linear, mixed integer-real), the number of rejected tuples grows up to a point where the number of calls to the constraint solver is intractable. Note also that using constraint propagation and refutation instead of the Simplex algorithm opens the door to the treatment of non-linear constraints (e.g., $w_1 * w_2 < 1$) but it is also incomplete to determine the exact shape of the solution space.

Moreover, the proposed *UDW* technique is inspired from the Path-Oriented Random Testing (PRT) approach used in the context of code-based testing [7]. Even if the algorithm used to randomly generate uniformly distributed samples is similar to the one used in PRT, we see a main difference, i.e., according to our knowledge, using uniformly random distributed weights when constraints among weights are involved in search-based test minimization has never been explored before. Expressing constraints over weights is a key aspect of the proposed *UDW* technique as it releases test engineers from the tedious task of determining exact values to the weights, while preserving the benefits of *RAW*-approaches of search-based test minimization.

8 Conclusion and Future Work

In this paper, we proposed a new weight assignment strategy called Uniformly Distributed Weights (*UDW*) to generate weights by solving constraints among them with

uniform distribution for solving multi-objective optimization problems. *UDW* can guarantee the uniformity for the selection of weights at the same time meeting all the required constraints based on the domain knowledge and expertise. We compared the proposed *UDW* with two commonly-used weight strategies (i.e., Fixed Weights (*FW*) and Randomly-Assigned Weights (*RAW*)) in conjunction with the following search algorithms: (1+1) Evolutionary Algorithm (EA), Genetic Algorithm, Alternating Variable Method, and Random Search based on our industrial problem of test minimization in the context of product lines. For test minimization, a fitness function was defined based on various cost/effectiveness measures (e.g., feature pairwise coverage) identified through our industrial collaboration with Cisco Systems. We performed our empirical evaluation using a Video Conferencing Systems product line provided by Cisco Systems and 500 artificial problems of varying complexity. The results showed that (1+1) EA performs the best among all the algorithms together with *UDW* and thus we conclude that assigning weights based on uniform distribution can significantly improve the performance of (1+1) EA for multi-objective optimization, particularly for multi-objective test minimization in the context of product lines.

In the future, we plan to replicate our experiments in other industrial case studies for assessing the proposed weight strategy *UDW*. We also plan to investigate the effect of uniformly distributed weights on a diverse range of search algorithms.

References

1. Konak, A., Coit, D.W., Smith, A.E.: Multi-objective optimization using genetic algorithms: A tutorial. Reliability Engineering & System Safety (91), 992–1007 (2006)
2. Marler, R.T., Arora, J.S.: Survey of multi-objective optimization methods for engineering. Struct Multidisc Optim. 26, 369–395 (2005)
3. Jin, Y., Okabe, T., Sendhoff, B.: Adapting weighted aggregation for multiobjective evolution strategies. In: Zitzler, E., Deb, K., Thiele, L., Coello Coello, C.A., Corne, D.W. (eds.) EMO 2001. LNCS, vol. 1993, pp. 96–110. Springer, Heidelberg (2001)
4. Murata, T., Ishibuchi, H., Tanaka, H.: Multi-objective genetic algorithm and its applications to flowshop scheduling. Computer & Industrial Engineer. 30(4), 957–968 (1996)
5. Harman, M., Mansouri, S.A., Zhang, Y.: Search Based Software Engineering: A Comprehensive Analysis and Review of Trends Techniques and Applications, Technical Report TR-09-03, King College London (2009)
6. Wang, S., Ali, S., Gotlieb, A.: Minimizing Test Suites in Software Product Lines Using Weighted-based Genetic Algorithms. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO), pp. 1493–1500 (2013)
7. Gotlieb, A., Petit, M.: A uniform random test data generator for path testing. The Journal of Systems and Software 83(12), 2618–2626 (2010)
8. Cisco Systems TelePresence codec c90 (2010)
9. Wang, S., Gotlieb, A., Ali, S., Liaaen, M.: Automated Selection of Test Cases using Feature Model: An Industrial Case Study. In: Moreira, A., Schätz, B., Gray, J., Vallecillo, A., Clarke, P. (eds.) MODELS 2013. LNCS, vol. 8107, pp. 237–253. Springer, Heidelberg (2013)

10. Wang, S., Ali, S., Yue, T., Liaaen, M.: Using Feature Model to Support Model-Based Testing of Product Lines: An Industrial Case Study. In: Proceedings of International Conference of Software Quality (QSIC), pp. 75–84 (2013)
11. Arcuri, A., Briand, L.C.: A Practical Guide for Using Statistical Tests to Assess Randomized Algorithms in Software Engineering. In: Proceedings of the International Conference on Software Engineering, pp. 21–28 (2011)
12. Sheskin, D.J.: Handbook of Parametric and Nonparametric Statistical Procedures (2003)
13. Arcuri, A., Fraser, G.: On Parameter Tuning in Search Based Software Engineering. In: Cohen, M.B., Ó Cinnéide, M. (eds.) SSBSE 2011. LNCS, vol. 6956, pp. 33–47. Springer, Heidelberg (2011)
14. Yoo, S., Harman, M.: Regression testing minimization, selection and prioritization: A survey. Software Testing, Verification and Reliability 22(2), 67–120 (2012)
15. Walcott, K.R., Soffa, M.L., Kapfhammer, G.M., Roos, R.S.: Time-Aware Test Suite Prioritization. In: Proceedings of the International Symposium on Software Testing and Analysis, pp. 1–12 (2006)
16. Harman, M.: Making the Case for MORTO: Multi Objective Regression Test Optimization. In: Proceedings of the International Conference on Software Testing, pp. 111–114 (2011)
17. Smith, N.A., Tromble, R.W.: Sampling Uniformly from the Unit Simplex. Technical Report. Johns Hopkins University
18. Wang, S., Ali, S., Gotlieb, A.: Random-Weighted Search-Based Multi-Objective Test Suite Optimization Revisited. Technical Report 2013-01 (2013), https://www.simula.no/publications/TR2013-01