2014 IEEE International Conference on Software Testing, Verification, and Validation

Multi-Objective Test Suite Optimization for Incremental Product Family Testing

Hauke Baller*, Sascha Lity*, Malte Lochau† and Ina Schaefer‡
*TU Braunschweig, Institute for Programming and Reactive Systems, Braunschweig, Germany
Email: {baller, lity}@ips.cs.tu-bs.de
†TU Darmstadt, Real Time Systems Lab, Darmstadt, Germany
Email: malte.lochau@es.tu-darmstadt.de
‡TU Braunschweig, Institute of Software Engineering and Automotive Informatics, Braunschweig, Germany
Email: i.schaefer@tu-bs.de

***Abstract*—The design of an adequate test suite is usually guided by identifying test requirements which should be satisfied by the selected set of test cases. To reduce testing costs, test suite minimization heuristics aim at eliminating redundancy from existing test suites. However, recent test suite minimization approaches lack (1) to handle test suites commonly derived for families of similar software variants under test, and (2) to incorporate fine-grained information concerning cost/profit goals for test case selection. In this paper, we propose a formal framework to optimize test suites designed for sets of software variants under test w.r.t. multiple conflicting cost/profit objectives. The problem representation is independent of the concrete testing methodology. We apply integer linear programming (ILP) to approximate optimal solutions. We further develop an efficient incremental heuristic for deriving a sequence of representative software variants to be tested for approaching optimal profits under reduced costs. We evaluated the algorithm by comparing its outcome to the optimal solution.**

***Keywords*—Testing Strategies, Test Coverage of Specifications, Constrained Optimization, Linear Programming**

I. Introduction

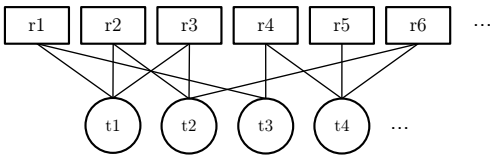
Software testing constitutes a viable verification methodology for reliable, yet practicable quality assurance of software systems. Testing allows for a flexible tradeoff between efficiency, e.g., in terms of testing cost reductions, and sufficiency, e.g., in terms of coverage measures. Thus, a tester is faced with the problem of selecting a finite set of *representative* test cases from a usually infinite input domain into a test suite to be applied to the software system under test. The design of an adequate test suite is often guided by specifying *test requirements* to be satisfied by the selected set of test cases. Test suite minimization heuristics aim at eliminating redundancy from existing test suites [1].

A sample instance of the classical test suite minimization problem is illustrated in Fig. 1(a). A given test suite, i.e., a set of test cases t_1, t_2, t_3, t_4 is to be reduced w.r.t. a finite set of requirements $r_1, r_2, \ldots, r_6$ such that each requirement is related to at least one test case via a satisfaction relation (denoted as solid lines). The subset t_1, t_4 constitutes the unique minimal test suite for this problem. This optimization problem is known as the minimum set cover problem which is NP-complete [1]. Various, mainly greedy-based heuristics have been proposed to approximate an optimal solution [2], [3].

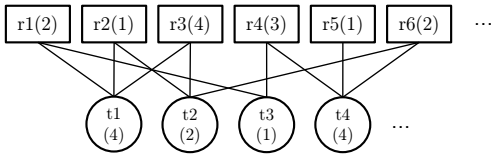
To allow for a more fine-grained tradeoff between profits and costs of test case selections, the problem setting is to be enriched to incorporate further information about (1) the individual costs, i.e., testing efforts caused by the selection of particular test cases and (2) the estimated profits gained for satisfying a certain requirement [4], [5], [6]. The goal of designing an adequate test suite may be generalized to (1) minimizing the aggregated testing costs and/or (2) maximizing the overall profit. The resulting optimization problem in Fig. 1(b) imposes a weighted set cover problem. Here, test cases and requirements are equipped with additional *weights* (in parenthesis) denoting cost and profit values, respectively. The prior minimal test suite t_1, t_4 now has overall costs of 8, whereas for the test suite t_2, t_3, t_4 we have 7 constituting the cheapest minimal test suite. The problem may be further enriched by *constraints*, e.g., a predefined profit goal to be at least met and/or a cost budget not to be exceeded leading to a *partial* set cover problem. For instance, by reducing the requirement profit goal from 13 to 11, requirement r_1 might be neglected making test case t_3 redundant.

Nowadays software systems often comprise *families of system variants* rather than singleton implementations developed for a particular customer's needs. For instance, the software product line paradigm propagates the adoption of mass customization principles also to software engineering [7]. A product line implementation is built on top of a common core application which is enriched by configuration parameters to control the (de-)activation of variable parts. As a result, the overall set of test requirements and test cases obtained for a *product line under test* is partitioned into variant-specific test artifact subsets [8]. Due to commonality among the variants, those artifact subsets are, in general, non-disjoint. As illustrated in Fig. 1(c), the product variants p_1, p_2, p_3 introduce an additional dimension to the test suite minimization problem. Each variant refers to a subset of the global test case set being valid for that product variant, again, denoted by solid lines. As a result, each variant is (indirectly) associated with a subset of test requirements from the global set of requirements. The extended problem statement then consists of selecting an optimized set of test cases covering all product family members [9], [10], [11] and/or an appropriate product variant subset for applying a product family test suite [12], [13], [14], [15], [16], [17]. For instance, selecting p_1, p_3 as product subset covers all test requirements defined for the product

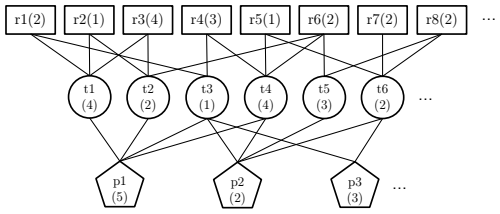
978-0-7695-5185-2/14 $31.00 © 2014 IEEE
DOI 10.1109/ICST.2014.43



(a) Test Suite Minimization Problem



(b) Multi-Objective Test Suite Optimization Problem



(c) Product Family Test Suite Selection Problem

Fig. 1. The Test Suite Minimization Problem and its Extensions

family. This additional dimension of product variants might also be equipped with profit/cost values for prioritizing product selections [18] thus resulting in a generalized and even more complex constrained multi-objective optimization problem.

To tackle this problem, we propose a formal framework for cost/profit-sensitive test suite optimization that takes product families into account. We first extend the underlying minimum set cover problem [1] to a weighted partial set cover problem [19] with additional constraints for product family coverage. Hence, in contrast to [20], we treat not only test cases, but also test requirements, products and their interrelations, as well as the respective weights as first class entities. The resulting constrained multi-objective optimization problem is reformulated as an integer linear program (ILP) for computing optimal solutions. To cope with the computational complexity of those problems, we further present an efficient heuristic by means of an incremental algorithm for approaching optimal cost/profit tradeoffs. As a side effect, the algorithm implicitly produces an optimized testing order on the set of product variants. Our heuristic is extensively evaluated concerning efficiency/accuracy tradeoffs w.r.t. optimal solutions.

This paper is organized as follows. In Sect. II, we review the foundations of the original test suite minimization problem and define extensions as described above. A corresponding ILP representation for obtaining optimal solutions is described in Sect. III. An efficient heuristic for incrementally approximating those optimal solutions is presented in Sect. IV and evaluated in Sect. V w.r.t. efficiency/optimality tradeoffs. We discuss related work in Sect. VI and conclude in Sect. VII.

II. Foundations and Extensions of the Test Suite Reduction Problem

We introduce the fundamental notions of requirement-based design and reduction of test suites for software products

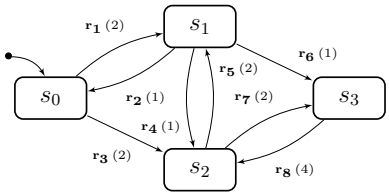


Fig. 2. Sample State Machine Test Model

under test. We then extend this setting to a constrained multi-objective optimization problem to design cost/profit-optimal test suites for families of similar product variants under test.

A. Multi-Objective Test Suite Design for Single Products

The design of a *test suite* for a software product under test requires to select from the (potentially infinite) set $T = \{t_1, t_2, \ldots, t_n, \ldots\}$ of all valid test cases a finite subset $TS \subseteq T$. The *adequacy* of a selected test suite *TS* is usually measured by means of a finite set $R = \{r_1, r_2, \ldots, r_m\}$ of product-specific *test requirements*. Each test requirement $r_i \in R$ is satisfiable by a subset $T_i \subseteq T$. A test suite $TS \subseteq T$ satisfies a test requirement $r_i \in R$ if at least one test case $t \in T_i$ is contained in *TS*. A test suite is *complete* if it satisfies all test requirements.

Definition 1 (Complete Test Suite): Let $TS \subseteq T$ be a *test suite*. *TS* is *complete* for a finite set R of test requirements iff

$$\forall r_i \in R : TS \cap T_i \neq \emptyset.$$

The induced satisfaction relation between test cases and test requirements is, in general, an $n : m$ correspondence with $n \geq 0$ and $m \geq 0$. However, as a convention we assume that

1) for each $t \in T$ there exists at least one test requirement $r_i \in R$ with $t \in T_i$, i.e., we restrict the set T to only contain *relevant* test cases and
2) for each $r_i \in R$ it holds that $T_i \neq \emptyset$, i.e., every test requirement is satisfiable by at least one test case.

As a consequence, at least one complete test suite always exists for any given set R of test requirements. The corresponding optimization problem of finding a *minimal* complete test suite coincides with the minimum set cover problem [1].

The problem definition introduced so far is independent of the actual testing methodology and the corresponding test artifact representation. For the following illustrative examples, we assume a model-based testing approach [21]. A *test model* given as finite state machine serves as a behavioral specification of the system under test. Each valid test case refers to a path of the state-transition graph and the set of test requirements usually corresponds to certain syntactical model elements selected by structural *coverage criteria*.

Example 1: Consider the sample state machine test model depicted in Fig. 2. We apply *all-transitions* as coverage criterion and use the resulting set $R = \{r_1, r_2, \ldots, r_8\}$ of requirements as transition labels for convenience. The set $T = \{(r_1), (r_1 r_2), (r_1 r_2 r_3), \ldots\}$ of test cases contains all transition sequences corresponding to valid paths in the test model. A complete test suite is, e.g., given as $TS = \{(r_1 r_2), (r_1 r_4), (r_3 r_7 r_8), (r_3 r_5 r_6)\}$. The test suite $TS' = \{(r_1 r_2 r_3 r_5 r_4 r_7 r_8 r_5 r_6)\}$ constitutes a minimal test suite.

Test suite minimization aims at reducing testing efforts by minimizing the number of test cases to be executed on an implementation under consideration. In practice, not only the number of test cases, but also the *costs* for executing each test case affects the overall testing efforts for a selected test suite [4]. For instance, although the test suite TS' in Ex. 1 is complete, its test case requires multiple traversals within the test model thus potentially causing higher testing costs than *TS*. Those testing costs introduce test case specific *weights*, i.e., natural numbers assigned to test cases via a weight function

$$w_T : T \to \mathbb{N}.$$

The test suite minimization problem is, therefore, generalized to find a complete test suite *TS* with minimum overall costs

$$c_{TS} = \sum_{t \in TS} w_T(t).$$

The original test suite minimization problem constitutes a special case with $w_T(t) = 1$ for each $t \in T$. A corresponding decision problem consists of deciding whether a complete test suite *TS* exists with $c_{TS} \leq k_c$ for a given *cost budget* $k_c \in \mathbb{N}$.

Example 2: We extend the setting of Ex. 1 with test case costs, where we assume the path length of each test case to denote its cost value. The complete test suite TS'' = $\{(r_1 r_2), (r_3 r_5 r_4), (r_1 r_6 r_8 r_7)\}$ with overall costs of 9 now constitutes besides TS' from Ex. 1 another minimal solution, whereas TS'' does not require multiple traversals within the test model. There exists no solution for a cost budget $k_c < 9$.

The completeness notion for a test suite may be enhanced in a similar way to only demanding a *sufficient* satisfaction of the test requirements. For this, a *profit* value is assigned to each test requirement denoting the importance of satisfying a particular requirement, e.g., depending on its safety level etc. Those profits are, again, specified by a weight function

$$w_R : R \to \mathbb{N}$$

assigning to each requirement its profit value. The corresponding optimization goal is to find a (minimal) test suite *TS* that is adequate to meet an overall *profit goal* $k_p \in \mathbb{N}$ with $k_p \leq \sum_{r \in R} w_R(r)$. For this, we require

$$p_{TS} = \sum_{\substack{r_i \in R \\ T_i \cap TS \neq \emptyset}} w_R(r_i) \geq k_p,$$

i.e., the sum of profits of all test requirements satisfied by *TS* meets k_p. Hence, satisfying a profit goal k_p does not necessarily require to cover every test requirement, but rather a subset with a sufficient sum of profit values. This problem is known as (weighted) partial set covering problem [22]. Alternatively, this problem can be equivalently represented as a combination of a pricing and penalty value [19].

Example 3: In Fig. 2, the values given in parentheses denote requirement profits, e.g., $w_R(r_1) = 2$. Assuming a profit goal of $k_p = 10$, the aforementioned complete test suites TS, TS' and TS'' all satisfy k_p. In addition, also partially covering test suites such as $TS''' = \{(r_1 r_4 r_5 r_2), (r_3 r_7)\}$ with $p_{TS} = 10$ suffice to satisfy profit goal k_p.

Combining test case costs and test requirement profits, the resulting decision problem consists of deciding whether a test suite *TS* exists that (1) meets an overall profit goal k_p and (2) does not exceed a cost budget k_c. We call a test suite *TS* meeting those optimization goals for a given set of weight functions an *adequate test suite*.

Definition 2 (Adequate Test Suite): Let $TS \subseteq T$ be a test suite. *TS* is *adequate* for a finite set R of test requirements, a cost bound k_c and a profit goal k_p iff

$$c_{TS} \leq k_c \text{ and } p_{TS} \geq k_p.$$

The bounds k_c and k_p serve as constraints when selecting an adequate set of test cases. If no bounds are given for costs and/or profits, those values constitute optimization objectives for the test suite design. Correspondingly, the optimization problem of finding a test suite *TS* with (1) minimum costs c_{TS} and (2) maximum profit p_{TS}, in general, constitutes a *multi-objective optimization problem*.

Example 4: Combining the profit goal $k_p = 10$ and cost budget $k_c = 10$, TS''' from Ex. 3 constitutes an adequate test suite. By omitting the cost budget, the overall costs are to be minimized w.r.t. the profit goal k_p. The resulting optimization problem is solved, e.g., by the test suite $TS'''' = \{(r_3 r_7 r_8), (r_1)\}$ with $c_{TS} = 4$ and $p_{TS} = 10 = k_p$. By omitting the profit goal, the overall profit is optimized w.r.t. the given cost budget k_c. Test suite TS''' from Ex. 2 gains maximum profit by satisfying each test requirement with overall costs of $c_{TS} = 9$. By removing both cost budget k_c and profit goal k_p, test suite TS''' also constitutes one possible solution for the resulting multi-objective optimization problem.

In general, more than one solution for a multi-objective optimization problem potentially exists, each constituting a different tradeoff between the multiple objectives. The selection of a final solution, therefore, requires a further prioritization among the objectives, e.g., by means of additional prioritization factors α_c and α_p (cf. Sect. III). The adequate test suite design introduced so far may be further enriched by supporting multiple, independent weight functions for both test cases and test requirements. However, we assume those additional weights to be integrated into compound weight functions w_T and w_R, e.g., by means of a suitable prioritization among the different values. In addition, optimization problems are usually formulated as minimization problems. Therefore, profit maximization can be reformulated correspondingly to also impose a minimization problem.

Until now, adequate test suite design is based on the assumption that every test case is applied to a singleton product under test p and every test requirement is associated with some entity of p. We now enhance this setting to *sets* of similar products under test organized in a so-called *product family*.

B. Test Suite Design for Product Families

Recent approaches for a structured development of *families* of similar (software) products such as Software Product Line (SPL) Engineering propose techniques for efficiently engineering variant-rich software systems [7]. Those approaches aim at explicit specifications of commonality and variability among the product family members at any level of abstraction. Based on a common core platform, SPL engineering propagates a preplanned, extensive reuse of shared development artifacts

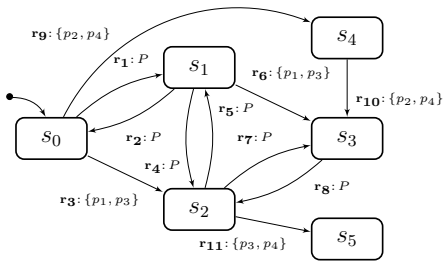


Fig. 3. Sample Annotated State Machine Test Model

among different *product variants* throughout all development phases. This *reuse* principle also applies to test artifacts [8].

We extend our formal test suite optimization introduced so far, by considering sets $P = \{p_1, p_2, \ldots, p_l\}$ of products under test. Each product $p_j \in P$ may be tested in isolation by designing and applying product-specific adequate test suites TS^j as described before. However, as all product implementation variants of P are derived from a common platform, they also share a common domain of test cases T and test requirements R [8], [10]. Hence, for each product variant $p_j \in P$, we have

1) a subset $T^j \subseteq T$ of test cases that are valid for p_j and
2) a subset $R^j \subseteq R$ of test requirements relevant for p_j.

For the satisfaction relation between the sets T and R, we make the same assumptions as for the single product case. We assume further meaningful properties to hold for T and R in case of product family testing, namely

1) for each $r_i \in R^j$ there exists some $T^j \cap T_i \neq \varnothing$ *(satisfiability of product test requirements)*,
2) if $t \in T^j \cap T_i$ for some $r_i \in R^j$ then t satisfies r_i when executed on p_j *(consistency of product test requirement satisfaction)*, and
3) for each $t \in T$ there exists some $p_j \in P$ with $t \in T^j$ and the same for $r \in R$, accordingly *(minimality of product family test case/requirement set)*.

In case $T^u \cap T^v \neq \varnothing$ holds for product variants p_u and p_v, they share common test cases and, correspondingly, for test requirements if $R^u \cap R^v \neq \varnothing$ holds. However, from $r_i \in R^u \cap R^v$ and $t \in T_i \cap T^u$ it does not necessarily follow that $t \in T_i \cap T^v$ holds, i.e., different test cases are potentially required for test requirements shared among different product variants.

Example 5: The state machine depicted in Fig. 2 now serves as test model for a product p_1 of a product family $P = \{p_1, p_2, p_3, p_4\}$. The product family test model in Fig. 3 integrates every test model variant into one generic state machine. Each transition is parametrized with the subset of product variants whose product-specific test model contains this transition. For instance, r_1 is part of every test model, whereas r_9 is only contained in the test model variants of p_2 and p_4. The set $R = \{r_1, r_2, \ldots, r_{11}\}$ now comprises further requirements to be satisfied for products p_2, p_3 and p_4 and the set T is extended to the set of all valid product family test cases satisfying the consistency properties outlined above.

Applying a complete test suite $TS \subseteq T$ selected for a set R of test requirements as described before to a set P of products requires to select for each test case $t \in TS$ some product $p_j \in P$ such that $t \in T^j$. Hence, the test cases selected into test suite *TS* implicitly determine the (sub-)set of product variants from P required to satisfy all test requirements in R. To avoid the test suite selection to determine the subset of products under test, we improve the notion of a complete test suite as follows.

TABLE I. SAMPLE TEST SUITE FOR PRODUCT FAMILY P

#	Path	Products	Cost
t_1	$r_1 r_2$	P	2
t_2	$r_1 r_4$	P	2
t_3	$r_3 r_7 r_8$	p_1, p_3	3
t_4	$r_3 r_5 r_6$	p_1, p_3	3
t_5	$r_1 r_2 r_3 r_5 r_4 r_7 r_8 r_5 r_6$	p_1, p_3	10
t_6	$r_3 r_5 r_4$	p_1, p_3	3
t_7	$r_1 r_6 r_8 r_7$	p_1, p_3	4
t_8	$r_1 r_4 r_5 r_2$	P	4
t_9	$r_3 r_7$	p_1, p_3	2
t_{10}	r_1	P	1
t_{11}	$r_1 r_4 r_7$	P	3
t_{12}	$r_9 r_{10} r_8$	p_2, p_4	3
t_{13}	$r_3 r_{11}$	p_3	2
t_{14}	$r_1 r_4 r_{11}$	p_4	3

Definition 3 (Complete Product Family Test Suite): Let *TS* be a test suite. *TS* is complete for a finite set R of test requirements and a product family P iff

$$\forall p_j \in P : \forall r_i \in R^j : \left(TS \cap \left(T_i \cap T^j\right)\right) \neq \varnothing.$$

The corresponding optimization problem consists of finding a *minimal complete product family test suite*.

Again, *costs* $w_T(t)$ of test cases $t \in T$ and *profit* values $w_R(r)$ may be introduced. Due to the multiple occurrences of test requirements in different product configurations, the overall profit p_{TS} of a product family test suite *TS* may be defined in different ways. Here, we require for a profit value of requirement r_i to count for a test suite *TS* that r_i is satisfied for *all* products $p_j \in P$ with $r_i \in R^j$, i.e.,

$$p_{TS} = \sum_{\substack{r_i \in R \\ \forall p_j \in P, r_i \in R^j : (T^j \cap T_i) \cap TS \neq \varnothing}} w_R(r_i).$$

Example 6: In Table I a sample subset of T for the sample product family in Fig. 3 is given containing 14 test cases satisfying all 11 requirements on P. Every test case is represented by its path, i.e., the sequence of requirements, the set of products it is reusable for, and its cost value. Some test cases are applicable for all product variants, e.g., test case t_8, whereas others are only valid for product subsets, e.g., t_3, or even singleton products, e.g., t_{13}.

As the adequacy of a test suite depends on the satisfaction of product-specific test requirements, the individual products might be also equipped with weights using a function

$$w_P : P \to \mathbb{N}.$$

The weight $w_P(p)$ of a product $p \in P$ represents *costs* and/or *profits* arising when applying test cases to product p during product family testing. For instance, the profit for testing product p_j is gained by a product family test suite *TS* if

$$\forall r_i \in R^j : (TS \cap (T^j \cap T_i)) \neq \varnothing,$$

i.e., all product-specific test requirements relevant for p_j are satisfied by *TS*. Summarizing, we enhance the notion of adequate test suites to families of products under test.

To distinguish between test requirement profits and product profits, we use the indexes *rp* and *pp*, respectively.

Definition 4 (Adequate Product Family Test Suite): Let $TS \subseteq T$ be a test suite. *TS* is *adequate* for a product set P, a finite set R of test requirements, a cost bound k_c, a test requirement profit goal k_{rp} and a product profit goal k_{pp} iff

$$c_{TS} \leq k_c, rp_{TS} \geq k_{rp} \text{ and } pp_{TS} \geq k_{pp}.$$

An alternative representation of the problem of testing a product family is to select a suitable product subset $PS \subseteq P$ under test. In the simplest case, we require for a subset *PS* that each test requirement is satisfiable by testing at least one product from the set *PS*, i.e.,

$$\forall r_i \in R : \exists p_j \in PS : r_i \in R^j.$$

Correspondingly, an appropriate subset *PS* might be selected for a given test suite *TS* such that

$$\forall t \in TS : \exists p_j \in PS : t \in T^j,$$

i.e., every test case is applicable to at least one product in the set *PS*. Again, adding profit and/or cost values further constrains the selection of suitable subsets *PS* similar to adequate test suite selections. We introduce a corresponding notion of adequate product subsets.

Definition 5 (Adequate Product Subset): Let $PS \subseteq P$. *PS* is *adequate* for a finite set R of test requirements, a cost bound k_c, a test requirement profit goal k_{rp} and a product profit goal k_{pp} iff an *adequate* test suite *TS* exists with

$$TS \subseteq \bigcup_{p_j \in PS} T^j.$$

A product subset is adequate if it suffices to apply an adequate product family test suite. The corresponding optimization problem consists of finding a *minimal* adequate product subset.

Example 7: We now want to find an adequate test suite and product subset for P in Tab. I. Product profits are simply given by the size of the test model, i.e., $w_P(p) = 2 * \#states + \#trans$. Missing requirement profits are defined as $w_R(r_9) = 2$, $w_R(r_{10}) = 2$ and $w_R(r_{11}) = 8$. We define the profit/cost bounds k_c as 45% of the maximum cost value as well as k_{rp} and k_{pp} as 75% of the maximum profit values. Therefore, the adequate test suite $TS_{P'} = \{t_7, t_8, t_{11}, t_{12}, t_{13}, t_{14}\}$ is obtainable and applicable on the adequate product subset $P' = \{p_2, p_3, p_4\}$ satisfying all test requirements except r_3.

In the following, we first present an alternative representation of the product family test suite optimization problem in order to compute an optimal solution. Then, we develop an incremental heuristic solution that computes from a set of not yet tested products the next product to be tested in order to incrementally optimize the overall cost and profit values.

III. Multi-Objective Test Suite Optimization using Linear Programming

As already mentioned in the previous section, a multi-objective test suite optimization problem constitutes a concrete instance of a (partial) weighted set cover problem. The framework of (Integer) *Linear Program* (ILP) is a viable means to formulate and solve those kinds of problems in a concise way [19]. We first describe the general principles of ILP and its application to weighted set cover problems. This way, we are able to obtain an optimal solution for designing an adequate test suite for a product family under test using an ILP solver.

A. Integer Linear Programming

A Linear Program (LP) is formulated over a finite set X of *decision variables* usually represented as a vector $X = (x_1, x_2, \ldots, x_n)^T$. Decision variables may take arbitrary continuous values, but may be restricted to (a) a given range and (b) a discrete value domain. An LP solely containing decision variables of type integer is called an *Integer Linear Program* (ILP) [23]. An LP is given as

$$\begin{aligned} \textit{min} \quad & f(X) \\ \textit{s.t.} \quad & g_i(X) \leq a_i \end{aligned}$$

consisting of an *objective function* $f(X)$ and a set of *constraints* $g_i(X) \leq a_i$ each defined by a function $g_i(X)$ and a bound a_i. The functions $f(X)$ and $g_i(X)$ are all linear functions. An ILP solution is a value assignment to all decision variables that satisfies the constraints and minimizes the values of the objective function. Variables may be further restricted to certain domains and/or ranges. A maximization problem can be converted into a minimization problem in the obvious way.

A minimum set cover problem can be formulated as an ILP as follows. Let E be a set of elements and S be a set of subsets of E. Each subset $s_i \in S$ corresponds to a decision variable x_i, that is limited to the values 0 and 1. If a variable is set to 1 the corresponding subset is selected for covering the respective subset of E. A solution for the set cover problem requires every element in E to be covered by at least one selected set from S. Thus, for each element $e_j \in E$ a constraint

$$\sum_{s_i \in S_j} x_i \geq 1$$

is to be added to the ILP, where $S_j \subseteq S$ contains the subsets s_i of S with $e_j \in s_i$. Those constraints ensure each element of E to be covered by at least one selected subset $s_i \in S_j$. To obtain a minimum set cover solution, the number of sets selected by the objective function is to be minimized, i.e.,

$$\textit{min} \sum_{s_i \in S} x_i$$

holds. This approach can be extended to weighted set cover problems by enriching the objective function with weights w_i of subsets s_i and requiring

$$\textit{min} \sum_{s_i \in S} x_i w_i$$

to hold. The objective function is minimal if the sum of the weights of the selected sets is minimal. This ILP can be further extended to solve a partial (weighted) set cover problem by adding profit values to elements of E. For those elements being covered by at least one subset, the corresponding profits are aggregated. We extend our ILP by introducing further decision variables y_j for each element $e_j \in E$ and change the corresponding constraints to

$$-y_j + \sum_{s_i \in S_j} x_i \geq 0.$$

Thus, if an element e_j is selected in a solution, at least one set s_i must be selected to satisfy the constraint. To guarantee that the covered elements satisfy the profit bound p a further constraint of the form

$$\sum_{e_j \in E} y_j p_j \geq p$$

is added with p_j denoting the profit value for element e_j. An alternative way to denote this constraint is

$$\sum_{e_j \in E} y_j p_j - \Delta_p = p \quad \text{with } \Delta_p \geq 0,$$

where the difference between the profit bound and the actual profits is now stored in the variable Δ_p. This variable may be used for further computations in both constraints and the objective function. Besides minimizing the weights of the covering set, i.e., the costs of a solution, we further want to maximize the weights of the covered elements, i.e., the profits of a solution. By Δ_p we denote the gap between the actual profits and the lower bounds imposed for the profits. A solution for an ILP with the enhanced objective function

$$min\ c + \Delta_p$$

now requires to minimize both (a) the overall costs c, i.e., the sum of weights of the covering subsets and (b) the profit gap Δ_p. Such an (I)LP with more than one objective function constitutes an *multi-objective optimization problem*. In general, we can not assume, that costs and profits are measured in the same scale. To integrate the different scale measures of cost and profit values, we introduce further normalization factors λ_c and λ_p to the objective function

$$min\ \lambda_c c + \lambda_p \Delta_p.$$

A multi-objective optimization problem has, in general, more than one optimal solution constituting different tradeoffs between contradicting objectives. These solutions are called *Pareto optimal* [24]. For the selection of a final solution, corresponding weighting factors α_c and α_p may be introduced to enforce an appropriate prioritization among cost and profit objectives. The resulting objective function is given as

$$min\ \alpha_c \lambda_c c + \alpha_p \lambda_p \Delta_p.$$

This approach is known as *weighted sum method* [24]. In the following we extend this approach for product family test suite optimization. The original set cover problem is extended to a two-layered set cover problem, where the elements of one set to be covered are, at the same time, sets for covering other elements. In particular, each test requirement must be covered by at least one test case and each test case must be applicable to at least one product. The partial weighted set cover problem is extended to arbitrary (bounded) multi-objective cost/profit optimization problems. Our approach is open for further extensions and, thus, provides a general test suite optimization framework built upon ILP representations.

B. Product Family Test Suite Optimization using ILP

We now consider three distinguished sets of elements, i.e., requirements, test cases, and products and introduce respective decision variables.

$$\begin{aligned}
\min \quad & \alpha_{rp}\lambda_{rp}\Delta_{rp} + \alpha_c\lambda_c c_{TS} + \alpha_{pp}\lambda_{pp}\Delta_{pp} && (1)\\
\text{s.t.} \quad & \textstyle\sum_{r_i \in R} w_R(r_i)\, z_i - \Delta_{rp} = k_{rp} && (2)\\
& c_{TS} = \textstyle\sum_{t \in T} w_T(t)\, y_t \leq k_c && (3)\\
& \textstyle\sum_{p_j \in P} x_j w_P(p_j) - \Delta_{pp} = k_{pp} && (4)\\
& -y_t + \textstyle\sum_{p_j \in P: t \in T^j} x_j \geq 0 \quad \forall t \in T && (5)\\
& -z_i + \textstyle\sum_{t \in T_i \cap T^j} y_t \geq 0 \quad \forall p_j \in P: \forall r_i \in R^j && (6)\\
& -p_j + \textstyle\sum_{t \in T_i \cap T^j} y_t \geq 0 \quad \forall p_j \in P: \forall r_i \in R^j && (7)\\
& \Delta_{rp}, \Delta_{pp} \geq 0\\
& x_j, y_t, z_i \in \{0,1\} \quad \forall p_j \in P, \forall t \in T, \forall r_i \in R
\end{aligned}$$

Fig. 4. ILP for Product Family Test Suite Selection

1) Products $p_j \in P$ are represented by variables x_j indicating whether p_j is selected into a product subset,
2) Test case $t \in T$ are represented by variables y_t indicating whether t is selected into a test suite and
3) Requirements $r_i \in R$ are represented by variables z_i indicating whether r_i is satisfied by a test suite.

The corresponding ILP for computing a product family test suite is given in Fig. 4. Thus, the test case selection is intended to (1) minimize the requirement profits (Δ_{rp}) and/or staying above the bound k_{rp}, respectively, (2) minimize the overall test case costs (c_{TS}), and (3) minimize the product profits (Δ_{pp}) and/or staying above the bound k_{pp}. The objective function (Eq. 1) builds the weighted sum over these three values by applying predefined factors (α_{rp}, α_c, and α_{pp}) for objective prioritization. In addition, each value is normalized (λ_{rp}, λ_c, and λ_{pp}) for a better comparison. For the requirement profits, we minimize the difference (Δ_{rp}) between the sum

$$\sum_{r_i \in R} w_R(r_i)\, z_i$$

of requirement profits satisfied by the selected test suite and the lower bound k_{rp} for the requirement profits. This difference is calculated in Eq. (2). The product profits are handled similarly in Eq. 4. We assure that the variables storing the difference can only take positive values ($\Delta_{rp} \geq 0$ and $\Delta_{pp} \geq 0$). Eq. (3) limits the costs for the selected test cases to k_c. Summarizing, we can conclude the following.

Theorem 1: A solution of the ILP in Fig. 4 constitutes an adequate product family test suite according to Def. 4.

An adequate product family test suite must (1) satisfy each selected requirement r_i for each selected product p_j and, therefore, selecting a corresponding test case $t \in T_i \cap T^j$, (2) ensure that for each selected test case $t \in T$ at least one product $p_j \in P$ is selected with t being a valid test case for p_j, i.e., $t \in T^j$, and (3) satisfy the requirements of all selected products. Eq. (5) ensures that a corresponding product is selected for each test case. Whenever a test case is selected, at least one product must be also selected to satisfy the inequality. Note that for each test case, a dedicated inequality is added to the ILP. Analogously, Eq. (6) guarantees that for each product and each requirement, a corresponding test case is selected if the requirement is selected to be satisfied. Finally, Eq. (7) guarantees that for each requirement of a selected product a test case is selected to be satisfied. Hence, the given ILP calculates an adequate product family test suite and optimizes the test suite w.r.t. the given prioritization.

Example 8: Consider Tab. I and Ex. 7. We define α_c = 1.0, α_{rp} = 2.0, α_{pp} = 1.0 to prioritize the requirement profits when selecting an adequate test suite. We further use the same values for cost/profit bounds as in Ex. 7. Solving the resulting ILP yields an adequate test suite $TS_{P'} = \{t_3, t_4, t_8, t_{11}, t_{12}, t_{13}\}$ satisfying all test requirements except r_{11} with an objective value of 0.479 applicable to the product subset $P' = \{p_1, p_2, p_3\}$.

IV. Incremental Product Subset Selection

The ILP introduced in the previous section computes an adequate product family test suite by searching for a subset of test cases that (a) meets the given cost/profit bounds and (b) achieves an optimal tradeoff between reduced testing costs and maximal requirement/product profits. The high computational complexity of this task arises from potentially arbitrary relations between test cases, requirements, and products. In the worst case, every possible subset of test cases has to be investigated leading to the well-known combinatorial explosion problem. Recent heuristics for test suite minimization are based on incremental *greedy* selection strategies [25]. Therein, each test case is ranked w.r.t. the *number* of requirements it satisfies. This ranking imposes the ordering for iteratively selecting test cases into a reduced test suite until every requirement is eventually satisfied. Pure greedy-based approaches do not necessarily produce an optimal solution as the ranking does not take any information about correlations between test cases w.r.t. common requirements into account. Improved heuristics further consider the *cardinality* of a requirement during test case ranking. Therefore, the overall number of test cases satisfying a certain requirement influences this ranking.

We adopt the basic ideas from those heuristics and propose extensions to incrementally select an approximately minimal product subset under test until an adequate product family test suite is completely applicable. Therefore, we adopt the principles of *dynamic programming* to incrementally improve an existing product subset until finally reaching an adequate subset. In each iteration, the problem is decomposed into considering the cost/profit values of test artifacts of each remaining product. For this, our heuristic generalizes the concepts of *ranking* as well as *cardinality* to consider

1) weights of set elements, i.e., cost/profit values, and
2) multiple layers of mutually related (sub-)sets, i.e., test cases, requirements and products

in a concise and uniform way. Concerning 1) the given cost/profit bounds now serve as explicit termination conditions. For reducing the complexity of 2), we benefit from consistency properties imposed for interrelations between the different layers discussed in Sect. II.

A pseudo code representation of a product selection iteration step is depicted in Alg. 1. The algorithm has as fixed parameters the global entities of the multi-objective product family test suite optimization problem as described in the previous sections, i.e., the related sets R, T, P, profit/cost bounds k_c, k_{rp}, k_{pp} and objective prioritization factors $\alpha_{rp}, \alpha_c, \alpha_{pp}$. Each iteration receives as input the intermediate subset $PS \subseteq P$ of already selected products together with the respective set of test cases (*TS*) collected for the products in *PS*.

First, it is checked whether the intermediate profit values reached by *PS* and *TS* are already sufficient to meet the given

Algorithm 1 Incremental Heuristic

```
1: Parameters: R, T, P, k_c, k_pp, k_rp, α_c, α_rp, α_pp
2: Input: PS, TS
3: Output: p, TS_p
4: if ¬(rp_TS ≥ k_rp ∧ pp_TS ≥ k_pp) then
5:     for all p_j ∈ P \ PS do
6:         while existUncoveredReq(R^j, TS^j) do
7:             R^j' ← selectReqMinTcCardMaxProf(R^j, T^j, TS^j)
8:             R^j'' ← rankReqMinProdCard(R^j')
9:             T^j' ← selectTcMaxReqCardMaxReqProf(R^j'', T^j)
10:            T^j'' ← rankTcMaxProdCard(T^j')
11:            selectTcMinCost(T^j'', TS^j)
12:        computeProdWeights(p_j, TS^j, R̃^j)
13:    (rank_P, rank_TS, rank_R) ← rankProducts(P \ PS)
14:    p ← selectNextOptProd(rank_P, rank_TS, rank_R)
15:    if c_TS + c_TS,p > k_c then
16:        print No adequate subset found
17:    else
18:        return (p, TS_p)
19: else
20:    print Subset PS is adequate
```

profit goals k_{rp} and k_{pp} (line 4). In this case, an adequate product subset *PS* with approximately optimal cost and profit values has been found (line 20). Otherwise, the next (locally) optimal product is selected for *PS* from the set $P \setminus PS$. For estimating the next best fitting product, two steps are conducted for each product p_j (line 5-12).

1) For every requirement $r_i \in R^j$ a test case is selected from T^j in a cost/profit-aware manner taking the already selected test cases in *TS* into account (line 6 – 11).
2) Product p_j is weighted w.r.t. newly selected test cases in TS^j by aggregating the test case costs, requirement profits and the product profit for p_j (line 12).

Concerning 2), profits are only taken into account for the subset $\tilde{R}^j$ of requirements being satisfied for all related products in P by the extended test suite $TS \cup TS^j$. According to the test case selection heuristic of Harrold et al. [1], the traversal order of the set of test requirements not yet satisfied by TS^j is determined by their profits *and* cardinality (line 7). Hence, those high-profit requirements are preferred in $R^{j\prime}$ for which only a few number of satisfying test cases exists. This way local optima are prevented which are usually obtained when applying a pure greedy-based approach. The product set context of test requirements is further taken into account by preferring those requirements with minimal product cardinality (line 8). In line 9, the test cases are ordered according to the requirement profits they achieve. A test case is preferred for selection into $T^{j\prime}$ if the corresponding requirement profit gained for $R^{j\prime\prime}$ is maximal w.r.t. other considered test cases. Similar to the requirement selection, test cases in $T^{j\prime}$ are further ranked by considering their product set context in order to prefer test cases being reusable for many other products. Finally, in line 11, those test cases are selected in $T^{j\prime\prime}$ having minimal costs. Summarizing, a test case is preferred for p_j if (1) it has low costs, (2) it gains high test requirement profits and (3) it is (re-)usable for high-profit products.

The aggregation of test case costs, requirement profits and product profits for each p_j is used to derive product rankings w.r.t. optimal cost and profit values (line 13). For this, the products are ordered, correspondingly. The resulting rankings deliver the optimal cost and profit values available in the current iteration. Together with the objective prioritization

factors $\alpha_{rp}, \alpha_c, \alpha_{pp}$ this ranking finally yields the next product p selected for *PS* (Line 14). For each product, the distances to the optimal values within each ranking is determined, where the objective factors further prioritize the different distances, accordingly. The product with the minimal overall distance to all optimum values is finally selected as next optimal product under test. In line 15, the algorithm checks if the cost of the existing test suite c_{TS} and the costs of the newly selected test cases $c_{TS,p}$ exceed the cost bound k_c. In this case, the heuristic fails to approximate an adequate product subset (line 16). Otherwise, the selected product p is returned as next product under test together with the product-specific test suite TS_p to be applied (line 18). By incrementally calling this procedure, an implicit ordering is obtained for applying those test suites to *PS*. Summarizing, we can conclude the following.

Theorem 2: If Algorithm 1 terminates successfully, then *PS* is an adequate product subset w.r.t. Def. 5.

Based on the bound checks performed in each iteration, an adequate (but not necessarily minimal) subset is either found that meets the cost/profit goals, or the cost budget is exceeded, i.e., the heuristic terminates without finding a potentially existing adequate subset. The frequency of missing existing solutions is further investigated in the evaluation section.

Example 9: The problem from Ex. 8 results in an adequate product subset $P'' = \{p_2, p_3, p_4\}$ with the ordering p_2, p_4, p_3 and an adequate test suite $TS_{P''} = \{t_7, t_8, t_{11}, t_{12}, t_{13}, t_{14}\}$ satisfying all test requirements except of r_3. The result differs from Ex. 8 mainly due to the profits of test requirement r_{11} and the resulting ranking of p_4.

V. Implementation and Evaluation

We implemented the incremental heuristic presented in the previous section and conducted several experiments. The input comprises a data set consisting of a set of test requirements with profits, a set of test cases with costs, a set of products with profits, corresponding mappings between those sets, cost/profit bounds and the prioritization factors for the individual weights. We use (I)LP framework CPLEX and apply the command line solver from the GNU LINEAR PROGRAMMING KIT (GLPK)[1] (version 4.52) to obtain the optimal solution for comparison with the result computed by our heuristic. In our evaluation, we focus on the following research questions.

- **RQ 1:** Do we achieve a considerable *reduction of the execution time* for our incremental heuristic compared to solving the corresponding ILP?
- **RQ 2:** Is our incremental heuristic *applicable* in the sense that it computes a sufficient approximation of the optimal solution?

To investigate these research questions, we evaluated our heuristic by means of sample generated data sets. As complexity measures, we considered (1) the size of data sets, i.e., the number of elements of the three sets, (2) the degree of set couplings, i.e., the strength of the subset mappings and the weight distributions, and (3) the variation of the prioritization factors. The different sets were generated with varying sizes of 50, 100, and 500 elements and for both mapping as well as profits and costs we applied Gaussian distributions. For the Gaussian distribution, we use a mean $\mu = |T|/5$ and standard deviation $\sigma = |T|/20$, and $|P|$ respectively. In total we obtained 51 data sets. We executed the evaluation on an INTEL CORE2 DUO E8400 (3.0 GHz) with 2.96 GB RAM running WINDOWS XP 32 bit SP3. The results of the evaluation are given in Fig. 5 and in Tab. II. Fig. 5 provides a comparison of the execution time and the gap to the estimated optimal solution for each data set and both solving strategies. The ILP solver is able to estimate a lower bound for the optimal solution being continuously improved during computation. We terminated the ILP solving (a) if the estimated gap to the optimal solution was below $0.1\,\%$, (b) search time for an integer solution exceeded 24 hours, or (c) the memory consumption exceeded $2,000$ Mb RAM. We assumed the lower bound as reference solution for comparison to both the found optimal solution and the solution of the heuristic (cf. Fig. 5). The results of the ILP solver are marked with $*$ and those of our heuristic with $+$. The gap for solutions of the ILP solver increases with growing input data and/or in case of test cases being valid for only a small number of products.

Considering RQ 1, we observed that the median execution time decreased by a factor of ~ 21.49 from ~ 15 hours for the ILP solver to about 41 minutes for the incremental approach. If not limiting the execution time of the ILP solver, the gain in efficiency further increases. Our incremental heuristic thus achieves a considerable gain in efficiency. However, due to those execution time limitations we prevent the solver from actually running into exponential complexity problems.

Concerning RQ 2 consider Fig. 5. As expected, the accuracy of the incremental heuristic decreases with growing input data. The incremental heuristic constantly shows polynomial runtime complexity w.r.t. the problem size, i.e., the number of set elements and the degree of couplings between those sets. A variation of the degree of couplings between the different sets only had a small impact on the execution time, whereas a high impact on the accuracy arose in some cases, cf. Fig. 5 and Tab. II. Summarizing, adequate results have been found in most of the cases with considerable execution time reductions compared to the drawbacks in accuracy. In singular cases, e.g., for low coupling or bad ratios between test cases and requirements and/or products, the incremental heuristic fails to find a solution although the solver succeeded. In these cases, the Pareto front is presumably rather small. Further empirical studies are to be conducted to get a precise result for the frequency of those exceptional cases. Detailed results can be found in Tab II. All results were computed for a profit bound for both requirements and products of 75% and a cost bound of 95%. Results based on excelling the time limit for the ILP solver are marked with $+$ and those excelling the memory limit with $*$. Tables II(a)-II(c) contain results for varying sizes of the data sets. Based on these results, we consider one data set (100 requirements; 100 test cases; 100 products) and independently varied the couplings between those sets, c.f. II(d), II(e). We applied both mean and standard deviation for Gaussian distribution. From these results, we selected a data set based on high accuracy and gain in execution time to vary the weights of the objective function, cf. II(f). For the data set with maximum size the ILP solver failed as it excelled the memory limit, while the incremental heuristic could determine a solution. Overall, our measurements showed the heuristic

[1] http://www.gnu.org/software/glpk/

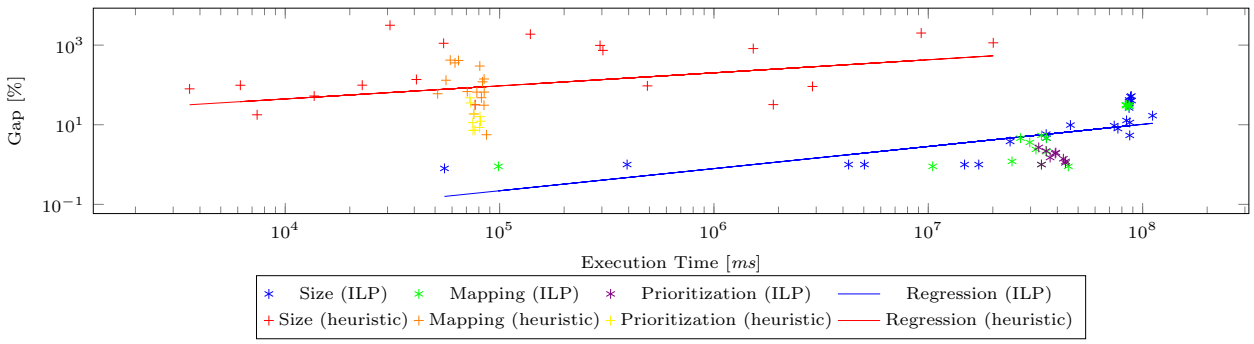


Fig. 5. Execution Time Reduction and Solution Derivation for the Heuristic Compared to the ILP Solver

TABLE II. INCREASE IN EXECUTION TIME AND DEVIATION FROM THE OPTIMAL SOLUTION FOR THE INCREMENTAL ALGORITHM COMPARED TO THE OPTIMIZATION PROBLEM FOR VARYING INPUT SIZE (A) - (C), MAPPINGS (D) & (E), AND PRIORITIZATION (F). MEASUREMENTS FOR (D) - (F) BASED ON INPUT DATA FOR 100 REQUIREMENTS, 100 TEST CASES, AND 100 PRODUCTS.

(a) 50 Test Cases ($|T|$ = 50)

$\lvert R\rvert\backslash\lvert P\rvert$	50	100	500
50	15.46 (1.78)	17.21 (1.97)	24.33 $(1.20)^*$
100	574.63 (1.17)	351.03 (∞)	18.56 No ILP
500	1932.51 $(\infty)^*$	401.43 $(\infty)^*$	4.35 No ILP$^+$

(b) 100 Test Cases ($|T|$ = 100)

$\lvert R\rvert\backslash\lvert P\rvert$	50	100	500
50	5748.33 $(1.87)^*$	123.05 (2.35)	5.97 (1.90)
100	1764.95 $(1.47)^*$	462.44 $(1.26)^*$	11.91 $(\infty)^+$
500	1179.45 $(\infty)^*$	178.00 $(1.75)^+$	2.20 $(\infty)^+$

(c) 500 Test Cases ($|T|$ = 500)

$\lvert R\rvert\backslash\lvert P\rvert$	50	100	500
50	2805.54 $(25.71)^+$	621.28 $(14.33)^+$	9.47 $(13.98)^+$
100	1576.99 $(8.98)^+$	294.05 $(7.61)^+$	4.40 $(8.13)^+$
500	286.50 $(6.04)^+$	58.19 $(6.53)^+$	∞^*

(d) Mapping (requirements/test cases)

$\mu\backslash\sigma$	$\frac{\#Sets}{40}$	$\frac{\#Sets}{20}$	$\frac{\#Sets}{10}$
$\frac{\#Sets}{10}$	351.84 $(1.26)^*$	444.85 (1.17)	636.77 (1.67)
$\frac{\#Sets}{5}$	123.71 (1.64)	462.44 $(1.26)^*$	365.77 $(1.03)^*$
$\frac{\#Sets}{2}$	1344.47 $(3.94)^+$	1348.42 $(3.48)^*$	1427.99 $(3.96)^*$

(e) Mapping (test cases/products)

$\mu\backslash\sigma$	$\frac{\#Sets}{40}$	$\frac{\#Sets}{20}$	$\frac{\#Sets}{10}$
$\frac{\#Sets}{10}$	575.93 $(\infty)^*$	693.62 $(1.57)^*$	1.76 (2.29)
$\frac{\#Sets}{5}$	327.34 $(1.41)^*$	462.44 $(1.26)^*$	313.71 $(1.64)^*$
$\frac{\#Sets}{2}$	1021.11 $(1.83)^+$	1038.76 $(1.71)^+$	1037.71 $(3.06)^*$

(f) Prioritization (α_c = 1.0)

$\alpha_{rp}\backslash\alpha_{pp}$	0.5	1.0	1.5
0.5	541.00 $(1.07)^*$	569.37 $(1.06)^*$	567.44 $(1.06)^*$
1.0	454.56 $(1.11)^*$	444.85 $(1.17)^*$	482.79 $(1.32)^*$
1.5	483.22 $(1.14)^*$	524.89 $(1.09)^*$	451.36 $(1.44)^*$

to be capable in most case to calculate adequate results with massive savings in execution time compared to ILP solving. The results of the incremental heuristic had a mean accuracy of 68.60 %. Due to the high execution time improvements we could observe a gain in efficiency by a factor of 127.48. To generalize these results, further experiments with existing product families are needed.

Threats to Validity: Our experimental results are based on generated data sets using Gaussian distributions and freely chosen bounds. Results may be precised by (a) real metrics from industry for weights and corresponding bounds and (b) generating data sets for existing product families and common coverage criteria. Until now, we do not predict a priori whether selected bounds are satisfiable. Based on further evaluation, guidelines for selecting appropriate bounds may be proposed. However, this general decision problem is independent from our approach. We only examined the effects of changing one variable for each measurement. Further evaluation should be conducted to investigate whether dependencies between variables exist and how these affect execution time and accuracy. As real world data sets usually exceed our generated data sets significantly this also potentially affects the accuracy.

The overall profit of a product family test suite may be defined in different ways. Both the incremental and the ILP based approach can be adapted to match alternative definitions. In practice, several different testing strategies, e.g., *retest strategies*, exist. To support those, further adaptions are needed, again, affecting the results. Finally, similar to [1] we assume all problem entities, i.e., test requirements, test cases and products and respective relations among those entities to be completely given a priori. An on-the-fly generation of those entities during incremental product family testing, again, requires a combination with regression testing techniques [9], [11]. In ILP weight assignment functions are limited to linear functions. Hence, adaptable weights depending on intermediate results are not supported by ILP but may be treated by the heuristic.

VI. RELATED WORK

Subset Selection for Product Family Testing: Recent approaches for subset selection are based on combinatorial testing principles. They can be categorized into approaches for the selection of a reduced set of test cases [26] and approaches for the selection of a reduced set of product variants under test, whereas no approach yet exist that combines both levels and enrich each of them with weights and respective bounds. The product selection is guided either by common coverage criteria, e.g., requirement-based coverage criteria [27] or configuration space coverage criteria [12] to determine an adequate subset, or by combinatorial combinations of configuration parameters (features). Oster et al. and Lochau et al. [28], [14] present an approach for subset selection for covering pairwise feature combinations/interactions and Perrouin et al. [13] generalize this to t-wise feature combinations. Johansen et al. [15] propose the very efficient ICPL algorithm to compute t-wise feature coverage. In [16], Johansen et al. propose to guarantee critical feature interaction coverage using weights. Further approaches enhance the notion of feature

interaction coverage by taking additional requirements and information into account [29], [17]. Besides different kinds of constraint solvers, also search-based methods have been applied to compute covering arrays enriched with feature and product weights for SPL testing [18], [30]. In contrast to our ILP-based approach, genetic algorithms are used to solve the multi-objective optimization problem where the authors focus on (pairwise) feature coverage and feature error coverage.

Summarizing, all those approaches select a product subset based on feature parameters and their combinations, whereas our approach select an adequate product subset based on the selection of an adequate test suite and specific weights w.r.t. given product and cost bounds. In addition, most of the approaches consider the set cover problem, whereas our formal framework is extended to a partial weighted set cover problem.

Incremental Product Family Testing: Regression-based testing strategies are well-suited for incrementally testing a product family as stepping from one product variant under test to a subsequent one is similar to the testing of a subsequent product version [9]. In [11], Lochau et al. present an incremental model-based software product line testing approach, where the differences between product variants are explicitly specified by means of *regression deltas* serving as a basis to reason about test case and test result reuse. The approach is independent of the actual product subset and the ordering in which the products are tested.

VII. Conclusion

We presented a general framework for cost/profit-sensitive test suite optimization for product families under test. For approximating optimal solutions, we (1) reformulated the underlying constrained multi-objective optimization problem as ILP and (2) developed an efficient incremental heuristic for deriving adequate product subsets to be tested for approaching optimal profits under reduced costs. As future work, we plan to further enhance and evaluate our heuristic for even more accurate results compared to the optimal solution by considering real-world empirical case studies from our industrial partners and obtaining insights into suitable pricing strategies and bound selections. We plan to combine the framework with our regression-based product family testing tool [11] and to define even more fine-grained selection metrics for test cases and/or products under test. To improve our heuristic, we plan to integrate a backtracking strategy to prevent it from missing existing solutions, and to investigate alternative approaches, e.g., search-based techniques.

Acknowledgment

This work was partially supported by the DFG (German Research Foundation) under the Priority Programme SPP1593: Design For Future – Managed Software Evolution.

References

[1] M. J. Harrold, R. Gupta, and M. L. Soffa, "A Methodology for Controlling the Size of a Test Suite," *ACM Trans. Softw. Eng. Methodol.*, vol. 2, no. 3, pp. 270–285, Jul. 1993.

[2] B. Crawford, R. Soto, E. Monfroy, F. Paredes, and W. Palma, "A hybrid Ant algorithm for the set covering problem," *Physical Sciences*, vol. 6, no. 19, pp. 4667–4673, 2011.

[3] R. D. Găceanu and H. F. Pop, "An incremental approach to the set covering problem," *Studia Universitatis Babes-Bolyai Series Informatica*, vol. LVII, no. 2, pp. 61–72, 2012.

[4] A. G. Malishevsky, J. R. Ruthruff, G. Rothermel, and S. Elbaum, "Cost-cognizant test case prioritization," University of Nebraska-Lincoln, Tech. Rep., 2006.

[5] S. Xu, H. Miao, and H. Gao, "Test Suite Reduction Using Weighted Set Covering Techniques," in *SNPD*, 2012, pp. 307–312.

[6] J. P. Rout, R. Mishra, and R. Malu, "An Effective Test Suite Reduction Using Priority Cost Technique," *Computer Science & Engineering Technology*, vol. 4, no. 4, pp. 372–376, 2013.

[7] P. C. Clements and L. Northrop, *Software Product Lines: Practices and Patterns*, ser. SEI Series in Software Eng. Addison-Wesley, 2001.

[8] J. D. McGregor, "Testing a Software Product Line," Carnegie Mellon, Software Engineering Inst., Tech. Rep. CMU/SEI-2001-TR-022, 2001.

[9] E. Engström, "Exploring Regression Testing and Software Product Line Testing - Research and State of Practice," Lic Dissertation, Lund University, May 2010.

[10] H. Cichos, S. Oster, M. Lochau, and A. Schürr, "Model-based Coverage-Driven Test Suite Generation for Software Product Lines," in *MoDELS*, 2011.

[11] M. Lochau, I. Schaefer, J. Kamischke, and S. Lity, "Incremental Model-based Testing of Delta-oriented Software Product Lines," in *TAP*, 2012.

[12] M. B. Cohen, M. B. Dwyer, and J. Shi, "Coverage and Adequacy in Software Product Line Testing," in *ISSTA*. ACM, 2006, pp. 53–63.

[13] G. Perrouin, S. Sen, J. Klein, and B. Le Traon, "Automated and Scalable T-wise Test Case Generation Strategies for Software Product Lines," in *ICST*, 2010, pp. 459–468.

[14] M. Lochau, S. Oster, U. Goltz, and A. Schürr, "Model-based Pairwise Testing for Feature Interaction Coverage in Software Product Line Engineering," *Software Quality Journal*, pp. 1–38, 2011.

[15] M. F. Johansen, O. Haugen, and F. Fleurey, "An Algorithm for Generating t-wise Covering Arrays from Large Feature Models," in *SPLC*. ACM, 2012, pp. 46–55.

[16] M. F. Johansen, O. Haugen, F. Fleurey, A. Eldegard, and T. Syversen, "Generating Better Partial Covering Arrays by Modeling Weights on Sub-product Lines," in *MoDELS*. Springer, 2012, pp. 269–284.

[17] M. Kowal, S. Schulze, and I. Schaefer, "Towards Efficient SPL Testing by Variant Reduction," in *VariComp*. ACM, 2013, pp. 1–6.

[18] C. Henard, M. Papadakis, G. Perrouin, J. Klein, and Y. L. Traon, "Multi-objective Test Generation for Software Product Lines," in *SPLC*, 2013.

[19] J. Könemann, O. Parekh, and D. Segev, "A Unified Approach to Approximating Partial Covering Problems," *Algorithmica*, vol. 59, no. 4, pp. 489–509, 2011.

[20] S. Yoo and M. Harman, "Pareto Efficient Multi-objective Test Case Selection," in *ISSTA*, 2007.

[21] M. Utting and B. Legeard, *Practical Model-Based Testing: A Tools Approach*. San Francisco, CA, USA: Morgan Kaufmann, 2007.

[22] M. J. Kearns, *The Computational Complexity of Machine Learning*. MIT Press, Cambridge, Massachusetts, 1990.

[23] H. Kasana and K. Kumar, *Introductory Operations Research: Theory and Applications*. Springer, 2004.

[24] O. Grodzevich and O. Romanko, "Normalization and Other Topics in Multi-Objective Optimization," in *FMIPW*, 2006.

[25] S. Yoo and M. Harman, "Regression Testing Minimization, Selection and Prioritization: a Survey," *Software Testing, Verification and Reliability*, vol. 22, no. 2, pp. 67–120, 2012.

[26] A. Hartman, "Software and Hardware Testing Using Combinatorial Covering Suites," in *Graph Theory, Combinatorics and Algorithms*. Springer US, 2005, vol. 34, pp. 237–266.

[27] K. Scheidemann, "Verifying Families of System Configurations," Ph.D. dissertation, TU Munich, 2007.

[28] S. Oster, I. Zorcic, F. Markert, and M. Lochau, "MoSo-PoLiTe - Tool Support for Pairwise and Model-Based Software Product Line Testing," in *VaMoS*, 2011.

[29] E. N. Haslinger, R. E. Lopez-Herrejon, and A. Egyed, "Using Feature Model Knowledge to Speed Up the Generation of Covering Arrays," in *VaMoS*. ACM, 2013, pp. 16:1–16:6.

[30] F. Ensan, E. Bagheri, and D. Gašević, "Evolutionary Search-based Test Generation for Software Product Line Feature Models," in *CAiSE*, 2012.