A New Approach to Feature-based Test Suite Reduction in Software Product Line Testing

Arnaud Gotlieb[1], Mats Carlsson[2], Dusica Marijan[1] and Alexandre Pétillon[1]
[1]*Certus Software Validation and Verification Centre, Simula Research Laboratory, Lysaker, Norway*
[2]*Swedish Institute of Computer Science, Uppsala, Sweden*

Keywords: Test Suite Reduction, Software Product Line Testing.

Abstract: In many cases, Software Product Line Testing (SPLT) targets only the selection of test cases which cover product features or feature interactions. However, higher testing efficiency can be achieved through the selection of test cases with improved fault-revealing capabilities. By associating each test case a priority-value representing (or aggregating) distinct criteria, such as importance (in terms of fault discovered in previous test campaigns), duration or cost, it becomes possible to select a feature-covering test suite with improved capabilities. A crucial objective in SPLT then becomes to identify a test suite that optimizes reaching a specific goal (lower test duration or cost), while preserving full feature coverage. In this paper, we revisit this problem with a new approach based on constraint optimization with a special constraint called GLOBAL_CARDINALITY and a sophisticated search heuristic. This constraint enforces the coverage of all features through the computation of max flows in a network flow representing the coverage relation. The computed max flows represent possible solutions which are further processed in order to determine the solution that optimizes the given objective function, e.g., the lowest test execution costs. Our approach was implemented in a tool called Flower/C and experimentally evaluated on both randomly generated instances and industrial case instances. Comparing Flower/C with MINTS (Minimizer for Test Suites), the State-Of-the-Art tool based on an integer linear formulation for performing similar test suite optimization, we show that our approach either outperforms MINTS or has comparable performance on random instances. On industrial instances, we compared three distinct models of Flower/C (using distinct global constraints) and the one mixing distinct constraints showed excellent performances with high reduction rates. These results opens door to an industrial adoption of the proposed technology.

1 INTRODUCTION

1.1 Context

Testing a software product line entails at least the selection of a test suite which covers all the features of the product-line. Indeed, even if it may not guarantee that each product would behave correctly, ensuring that each feature is tested at least once is a minimum requirement of Software Product Line Testing (SPLT) (Martin Fagereng Johansen and Fleurey, 2011; Henard et al., 2013). However, among the various test suites which cover all the features, some have higher fault-revealing capabilities than other, some have reduced overall execution time or energy consumption properties (Li et al., 2014). Dealing with different criteria when selecting a feature-covering test suite is thus important. Yet, at the same time, the budget allocated to the testing phase is usually limited and reducing the number of test cases while maintaining the quality of the process is challenging. For example, selecting a feature-covering test suite which minimizes its total execution time is desirable for testing some product lines which are developed in continuous delivery mode (Stolberg, 2009). Similarly, if the execution of each test case is associated a cost (because the execution requires access to cloud resources under some service level agreement), then there is a challenge in selecting a subset of test cases which can minimize this cost. Of course, ideally one would like to deal with all the criteria (feature coverage, execution time, energy consumption, ...) at the same time in a multi-criteria optimization process (Wang et al., 2015). Unfortunately, this approach cannot offer strong guarantee on the coverage of features or reachability of a global minimum, which is often not acceptable for validation engineers. Thus, there is

Gotlieb, A., Carlsson, M., Marijan, D. and Pétillon, A.
A New Approach to Feature-based Test Suite Reduction in Software Product Line Testing.
DOI: 10.5220/0005983400480058
In *Proceedings of the 11th International Joint Conference on Software Technologies (ICSOFT 2016) - Volume 1: ICSOFT-EA*, pages 48-58
ISBN: 978-989-758-194-6

room for approaches which offer guarantees in terms of feature coverage and optimize individually some criteria such as test execution time or energy consumption.

1.2 Existing Results

Test suite reduction has received considerable attention in the last two decades. Briefly, we can distinguish greedy techniques (Rothermel et al., 2002; Tallam and Gupta, 2005; Jeffrey and Gupta, 2005), search-based testing techniques (Ferrer et al., 2015; Wang et al., 2015), and exact approaches (Hsu and Orso, 2009; Chen et al., 2008; Campos et al., 2012; Li et al., 2014; Gotlieb and Marijan, 2014). Test suite reduction should not be confounded with test selection and generation for software product lines which has also received considerable attention these last years (Henard et al., 2013).

Greedy techniques for test suite reduction are usually based on variations of the Chvatall algorithm which selects first a test case covering the most features and iterates until all features are covered. In the 90's, (Harrold et al., 1993) proposed a technique which approximates the computation of minimum-cardinality hitting sets. This work was further refined with different variable orderings (Offutt et al., 1995; Agrawal, 1999). More recently, (Tallam and Gupta, 2005) introduced the delayed-greedy technique, which exploits implications among test cases and features or requirements to further refine the reduced test suite. The technique starts by removing test cases covering the requirements already covered by other test cases. Then, it removes test requirements which are not in the minimized requirements set, and finally it determines a minimized test suite from the remaining test cases by using a greedy approach. Jeffrey and Gupta extended this approach by retaining test cases which improve a fault-detection capability of the test suite (Jeffrey and Gupta, 2005). The technique uses additional coverage information of test cases to selectively keep additional test cases in the reduced suites that are redundant with respect to the testing criteria used for the suite. Comparing to (Harrold et al., 1993), the approach produces bigger solutions, but with higher fault detection effectiveness.

One shortcoming of greedy algorithms is that they only approximate true global optima without providing any guarantee of test suite reduction. Search-based testing techniques have been used for test suite reduction through the exploitation of meta-heuristics. (Wang et al., 2013) explores classical evolutionary techniques such as hill-climbing, simulated annealing, or weight-based genetic algorithms for (multi-objective) test suite reduction. By comparing 10 distinct algorithms for different criteria in (Wang et al., 2015), it is observed that random-weighted multi-objective optimization is the most efficient approach. However, by assigning weights at random, this approach is unfortunately not able to place priority over the various objectives. Other algorithms based on meta-heuristics are examined in (Ferrer et al., 2015).

All these techniques can scale up to problems having a large number of test cases and features but they cannot explore the overall search space and thus they cannot guarantee global optimality.
On the contrary, exact approaches, which are based either on boolean satisfiability or Integer Linear Programming (ILP) can reach true global minima. The best-known approach for exact test suite minimization is implemented in MINTS (Hsu and Orso, 2009). It extends a technique originally proposed in (Black et al., 2004) for bi-criteria test suite minimization. MINTS can be interfaced with either MiniSAT+ (Boolean satisfiability) or CPLEX (ILP). It has been used to perform test suite reduction for various criteria including energy consumption on mobile devices (Li et al., 2014). Similar exact techniques have also been designed to handle fault localization (Campos et al., 2012). Generally speaking, the theoretical limitation of exact approaches is the possible early combinatorial explosion to determine the global optimum, which exposes these techniques to serious limitations even for small problems. In the context of feature covering for software product lines, an approach based on SAT solving has been proposed in (Uzuncaova et al., 2010). In this approach, test suite reduction is encoded as a Boolean formula that is evaluated by a SAT solver. An hybrid method based on ILP and search, called DILP, is proposed in (Chen et al., 2008) where a lower bound for the minimum is computed and a search for finding a smaller test suite close to this bound is performed. Recently, another ILP-based approach is proposed in (Hao et al., 2012) to set up upper limits on the loss of fault-detection capability in the test suite. In (Mouthuy et al., 2007), Monthuy *et al.* proposed a constraint called SC for the set covering problem. They created a propagator for SC by using a lower bound based on an ILP relaxation. Finally, (Gotlieb and Marijan, 2014) introduced an approach for test suite reduction based on the computation of maximum flows in a network flow. This theoretical study was further refined in (Gotlieb et al., 2016) where a comparison of different constraint models was given, but there was no multi-objectives test suite optimization.

In this paper, we propose a new approach of

feature-based test suite reduction in software product line testing. Starting from an existing test suite covering a set of features of a software product line, our approach selects a subset of test cases which still covers all the features, but also minimizes one additional criteria which is given under the form of sum of priorities over test cases. This is an exact approach based on the usage of a special tool from Constraint Programming, called the *global_cardinality* constraint (Régin, 1996). This constraint enforces the link between test cases and features while constraining the cardinality of the subset of features each test case has to cover. By combining this tool with a sophisticated search heuristics, our approach creates a constraint optimization model which is able to compete with the best known approach for test suite reduction, namely MINTS/CPLEX (Hsu and Orso, 2009).

Associating a number to each test case is convenient to establish priorities when selecting test cases. Indeed, such a priority-value can represent or aggregate distinct notions such as execution time, code coverage, energy-consumption (Li et al., 2014), fault-detection capabilities (Campos et al., 2012) and so on. Using these priorities, feature-based test suite reduction reduces to the problem of selecting a subset of test cases such that all the feature are covered and the sum of test case priorities is minimized. Feature-based test suite reduction generalizes the classical test suite reduction problem which consists in finding a subset of minimal cardinality, covering all the features. Indeed, feature-based test suite reduction where each test case has exactly the same priority yields to the de-facto size-minimisation of the test suite. However, solving feature-based test suite reduction is hard as it requires in the worst case to examine a search tree composed of all the possible subsets of test cases. Typically, for a test suite composed of N, there are 2^N such subsets and N typically ranges from a few teens to thousands, which makes exhaustive search intractable.

1.3 Contributions of the Paper

The work presented in this paper is built on top of previously-reached research results. In 2014, we proposed a mono-criteria constraint optimization model for test suite reduction based on the search of max-flows in a network flow representing the problem (Gotlieb and Marijan, 2014). Unlike the method based on search-based test suite minimization presented in (Wang et al., 2013), our approach is exact which means that it offers the guarantee to reach a global minimum.

This paper introduces a new constraint optimization model based on the usage of the GLOBAL_CARDINALITYconstraint for performing multi-criteria test suite optimization in the context of SPLT. This model also features a dedicated search heuristic which permits us to find optimal test suite in a very efficient way. According to our knowledge, this is the first time a multi-criteria test suite minimization approach based on advanced constraint programming techniques is proposed in the context of SPLT. This constraint optimization model has been put at work to select test suites on both randomly-generated instances of the problem and also real cases.

1.4 Plan of the Paper

Next section introduces the necessary background material to understand the rest of the paper. Section 3 presents our approach to the feature-based test suite reduction problem. Section 4 details our implementation and experimental resuls, while section 5 discusses of the related works. Finally, section 6 concludes the paper.

2 BACKGROUND

This section introduces the problem of feature-based test suite reduction and briefly reviews the notion of global constraints. It also presents the global constraint called *global_cardinality*.

2.1 Feature-based Test Suite Reduction

Feature-based Test Suite Reduction (FTSR) aims to select a subset of test cases out of a test suite which minimizes the sum of its priorities, while retaining its coverage of product features. Formally, a FTSR problem is defined by an initial test suite $T = \{t_1, \ldots, t_m\}$, each test case being associated a priority-value $p(t_i)$, a set of n product features $F = \{f_1, \ldots, f_n\}$ and a function $cov : F \rightarrow 2^T$ mapping each feature to the subset of test cases which cover it. Each feature is covered by at least one test case, i.e., $\forall i \in \{1, \ldots, n\}, cov(f_i) \neq \emptyset$. An example with 5 test cases and 5 features is given in Table 1, where the value given in the table denotes the priority of the test case. Given T, F, p the priorities, and cov, a FTSR problem aims at finding a subset of test cases such that every feature is covered at least once, and the sum of priorities of the test cases is minimized. For the sake of simplicity, we consider minimization of the priorities, but maximization can be considered instead without changing the difficulty of the problem.

Table 1: A FTSR Example.

	f_1	f_2	f_3	f_4	f_5
t_a	2	2	-	-	-
t_b	1	-	1	-	-
t_c	-	3	3	-	3
t_d	-	-	-	2	2
t_e	-	-	-	1	-

Definition 1 (Feature-based Test Suite Reduction (FTSR))**.** *A* FTSR *instance is a quadruple* (T,F,p,cov) *where* T *is a set of* m *test cases* $\{t_1,\ldots,t_m\}$ *along with their priorities* $p(t_i)$*,* F *is a set of* n *product features* $\{f_1,\ldots,f_n\}$*,* $cov : F \rightarrow 2^T$ *is a coverage relation capturing which test cases cover each feature. An optimal solution to FTSR is a subset* $T' \subseteq T$ *such that for each* $f_i \in F$*, there exists* $t_j \in T'$ *such that* $(f_i,t_j) \in cov$ *and* $\sum_{t_j \in T'} p(t_j)$ *is minimized.*

A labelled bipartite graph can be used to encode any FTSR problem, with edges denoting the relation *cov* and labels denoting p, the priorities over the test cases, as shown in Fig.1. Note that the priorities are associated to the test cases and not to the features. In fact, in feature-based test suite reduction, all the features must be covered at least once, so that it is pointless to define priorities over features. As an extension, it is possible to consider for each test case distinct priorities for covering the features but this complexifies the problem without bringing much benefice as it is too complex for validation engineers to manage complex priority sets. Note also that the optimal solution shown in Fig.1 is not unique. For example, $\{t_a,t_b,t_d\}$ covers all the features and has also $TotalPriorities = 5$. When all the priorities are the same, then the FTSR problem reduces to the problem of finding a subset of minimal size.

2.2 Global Constraints

Constraint Programming is a declarative language where instructions are replaced by constraints over variables which take their values in a variation domain (Rossi et al., 2006). In this context, any constraint enforces a symbolic relation among a subset of variables, which are known only by their type or their domain. Formally speaking, a *domain variable* V is a logical variable with an associated domain $D(V) \subset \mathbb{Z}$ which encodes all possible labels for that variable. In the rest of the paper, upper-case letter or capitalized word will denote domain variables, while lower-case letter or word will denote constant values. For example, the domain variable *COLOR* takes its values in the domain $\{black,blank,blue,red,yellow\}$ which is encoded as 1..5, and the constraint $primary\_color(COLOR)$ enforces *COLOR* to be $blue,red,yellow$, but not *black* or *blank*. We use to say that $3,4,5$ satisfies *primary_color* and that $1,2$ are *inconsistent* with respect to the constraint. In the rest of the paper, we will use $a..b$ to denote $\{a,a+1,\ldots,b-1,b\}$ and $\{a,b\}$ to denote the enumerate set composed only of a and b. Note that in our context, we consider only finite domains, i.e., domains containing a finite number of possible distinct labels.

A constraint program is composed of both regular instructions and constraints over domain variables. Interestingly, constraints come with filtering algorithms which can eliminate some inconsistent values. For example, the constraint $same\_color(COLOR1,COLOR2)$ enforces $COLOR1$ and $COLOR2$ to take the same color. Let suppose that the variable $COLOR1$ has domain $\{blue,red,yellow\}$ and $COLOR2$ has domain $\{blank,blue,red\}$, then the consraint prunes both domains to $\{blue,red\}$ because all other label are inconsistent with the constraint *same_color*. Among the possible type of constraints, we have *simple constraints*, which include domain, arithmetical and logical operators, and *global constraints* (Régin, 2011). The constraints *primary_color* and *same_color* are simple constraints as they can be encoded with a domain and an equality operators. A *global constraint* is a relation which holds over a non-fixed number of variables and implements a dedicated filtering algorithm. A typical example of global constraint is $\text{NVALUE}(N,(V_1,\ldots,V_m))$, introduced in (Pachet and Roy, 1999), where $N,V_1,\ldots,V_m$ are domain variables and the constraint enforces the number of distinct values in $V_1,\ldots,V_m$ to be equal to N. This constraint is useful in several application areas to solve tasks assignment and time-tabling problems. For example, suppose that N is a domain variable with domain 1..2 and $FLAG_1,FLAG_2,FLAG_3$ are three domain variables with domains $FLAG_1 \in \{blue,blank\}$, $FLAG_2 \in \{yellow,blank,black\}$ and $FLAG_3 \in \{red\}$, then the constraint $\text{NVALUE}(N,(FLAG_1,FLAG_2,FLAG_3)$ can significantly reduce the domains of its variables. In fact, the value 1 is inconsistent with the constraint and can thus be filtered out of the domain of N, as there is no intersection between the domains of $FLAG_1$ and $FLAG_3$. It means that, if there is a solution of the constraint, it should at least contain two distinct values, constraining N to be equal to 2. In addition, the domains of $FLAG_1$ and $FLAG_2$ have only a single value in their intersection (*blank*), meaning that they can only take this value and all the other val-

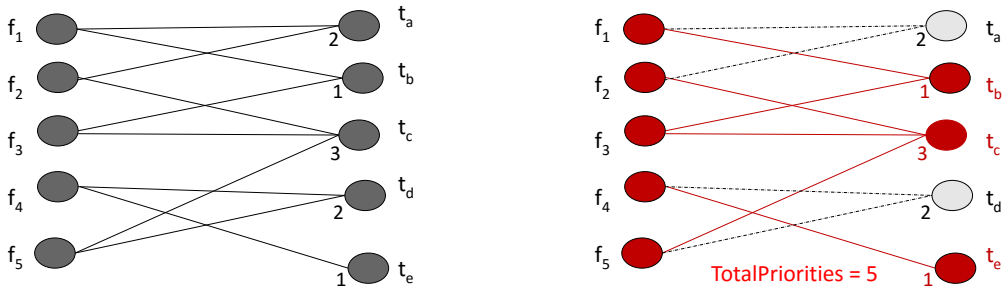


Figure 1: FTSR as a bipartite graph (a), with an optimal sol. (b).

ues are inconsistent. So, in conclusion, the constraint NVALUE$(N,(FLAG_1,FLAG_2,FLAG_3)$ is solved and $N=2$, $FLAG_1=FLAG_2=blank$ and $FLAG_3=red$. Of course, this is a favourable case and other instances may lead to only prune some of the inconsistent assignments without being able to solve the constraint. In this case, a search procedure must be launched in order to eventually find a solution. This search procedure selects an unassigned variable and will try to assign it a value from its current domain. The process is repeated until all the unassigned variables become instantiated or a contradiction is detected. In the latter case, the process backtracks and makes another value choice. This process is parametrized by a search heuristic which selects the variable and the value to be assigned first.

In our framework, we will use a powerful global constraint, which can be seen as an extension of NVALUE: the GLOBAL_CARDINALITYconstraint or *GCC* for short (Régin, 1996). The GLOBAL_CARDINALITY(T,d,C) constraint, where $T=(T_1,\ldots,T_n)$ is a vector of domain variables, $d=(d_1,\ldots,d_m)$ is a vector of distinct integers, and $C=(C_1,\ldots,C_m)$ is a vector of domain variables. GLOBAL_CARDINALITY(T,d,C) holds if and only if for each $i\in 1..m$ the number of occurrences of d_i in T is C_i. The C_i variables are called the *occurrence variables* of the constraint. The filtering algorithm associated to GLOBAL_CARDINALITY is based on the computation of max-flows in a network flow. Interestingly, this algorithm has a cubic time-complexity (Régin, 1996) which means that exploiting GLOBAL_CARDINALITY for filtering inconsistent values can be realized in polynomial time.

3 FTSR THROUGH GLOBAL CONSTRAINTS

In this section, we show how a constraint optimization model based on *GCC* can actually encode the solutions of a Feature-based Test Suite Reduction problem. This encoding is explained in Sec.3.1, while Sec.3.3 introduces a dedicated search heuristics to deal with priority-based test case selection.

3.1 A Constraint Optimization Model For FTSR

The FTSR problem can be encoded with the following scheme: each feature can be associated with a domain variable F having the following finite domain: $\{T_1,...,T_n\}$, where each T_i corresponds to a number associated to a test case which covers F. So, for example, the problem reported in Tab.1 can be encoded as follows: $F_1\in\{1,2\},F_2\in\{1,3\},F_3\in\{2,3\},F_4\in\{4,5\},F_5\in\{4\}$ where T_a is associated to 1, T_b is associated to 2, and so on.

Given a FTSR problem (F,T,p,cov), a constraint optimization model can be encoded as shown in Fig.2, where the domain variables C_i denote the number of times test case i is selected to cover any feature in $F_1,\ldots,F_n$, let B_i be Boolean variables denotating the selection of test case i. In this model, $(F_1,\ldots,F_n),(C_1,\ldots,C_m)$ are decision variables, only known through their domain. The boolean variables $B_1,\ldots,B_m$ are local variables introduced to establish the link with the priorities. First, the goal of this model is to minimize *TotalPriorities*, the sum of the priorities of selected test cases. For computing *TotalPriorities*, we have used the global constraint SCALAR_PRODUCT$((B_1,\ldots,B_m),(p_1,\ldots,p_m),$ *TotalPriorities*$)$ which enforces the relation *TotalPriorities* $=\sum_{1\leq i\leq m}B_i*p_i$. Actually, the non-zeroed B_i correspond to the selected test cases. Second, the constraint GLOBAL_CARDINALITY allows us to constrain the variables C_i which are

Minimize $TotalPriorities$ s.t.

$\text{GLOBAL\_CARDINALITY}((F_1,\ldots,F_n),(1,\ldots,m),(C_1,\ldots,C_m)),$
for $i = 1$ to m do $B_i = (C_i > 0)$,
$\text{SCALAR\_PRODUCT}((B_1,\ldots,B_m),(p_1,\ldots,p_m),TotalPriorities).$

Figure 2: A constraint optimization model for solving FTSR.

associated to number of times test case are selected. The model can be solved by searching the space composed of the possible choices for $(F_1,\ldots,F_n),(C_1,\ldots,C_m)$. Interestingly, it allows us to branch either on the choice of features or on the choice of test cases. Solving this model allows us to find an optimal solution to FTSR, as proved by the following sketch of proof.
($\Rightarrow$) An optimal solution of FTSR corresponds to an assignment of $(F_1,\ldots,F_n)$ with test cases which minimizes the sum of priorities. Let us call $\{t_p,\ldots,t_q\}$ this solution and *minimum* this sum. This is also an optimal solution of our model. In fact, the variables $\{C_p,\ldots,C_q\}$ are strictly positive because their associated test case is selected in the solution through GLOBAL_CARDINALITY, which means that only the corresponding $\{B_p,\ldots,B_q\}$ are equal to 1 and thus SCALAR_PRODUCT$((B_1,\ldots,B_m),(p_1,\ldots,p_m),$ *TotalPriorities*) is equal to *minimum*.
($\Leftarrow$) An optimal solution of our constraint optimization model is also an optimal solution of FTSR. In the model, *TotalPriorities* is assigned to the sum of priorities of selected test cases, which is exactly the definition of FTSR.

Note that even if the model given in Fig.2 is generic and solves the FTSR problem, it includes a search within a search space of exponential size $O(D^n)$ where D denotes the size of the greatest domain of the features and n is the number of test cases. This does not come as a surprise as the feature covering problem is a variant of the set covering problem which is NP-hard (Hsu and Orso, 2009).

3.2 Search Heuristics

Search heuristics consist of both a variable selection strategy and a value assignment strategy, which both relate to the finite domain variables used in the constraint optimization model. Regarding variable-selection, a first idea is to use the *first-fail principle* in the model of Fig.2, which selects first a variable representing a feature that is covered by the least number of test cases. As all the features have to be covered, it means that those test cases are most likely to be selected. However, this strategy ignores the selection of the test case having the greatest priority or the test cases covering the most features, which would be very interesting for our FTSR problem. Regarding value-selection, it is also possible to define a special heuristic for our problem.

3.3 A FTSR-dedicated Heuristic

Unlike static variable selection heuristics used in greedy algorithms such as for example, the selection of variables based on the number of features they cover, our strategy is more dynamic and the ordering is revised at each step of the selection process. It selects first the variable O_i associated to the test case with the greatest priority. Then, among the remaining test cases which cover features not yet covered, it selects the variable O_j with the greatest priority and iterates until all the features are covered. In case of choice which may not lead to a global minimum, then the process backtracks and permits us to select a distinct test case, not necessarily associated with the greatest priority. Regarding value-selection heuristics, it is possible to combine it with the variable-selection heuristics so that, each time a choice is made, it selects first the test cases which cover the most features. These ingredients have made the FTSR-dedicated heuristic a very powerfull method for solving the FTSR problem as shown in our experimental results.

4 IMPLEMENTATION AND RESULTS

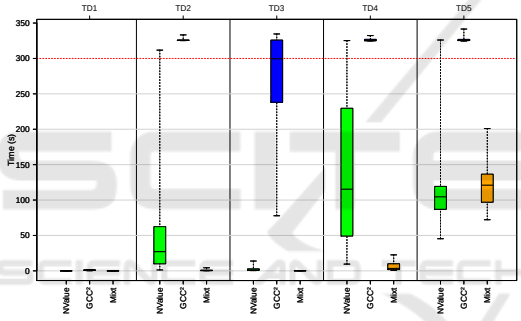
We implemented the constraint optimization model and search heuristic described above in a tool called Flower/C. The tool is implemented in SICStus Prolog and utilizes the clpfd library of SICStus which is a constraint solver for finite domain variables. It reads a file which contains the data about test cases, covered features, priorities, execution time, etc. and processes these data by constructing a dedicated constraint optimization model. Solving the model requires to implement the search heuristics and tuning the input format for a better preprocessing. These steps are encoded in SICStus Prolog and a runtime is embedded into a

tool with a GUI, in order to ease the future industrial adoption of the tool.

We performed experiments on both random and industrial instances of FTSR. For random problems, we created a generator of FTSR instances, which takes several parameters as inputs such as the number of features, the number of test cases along with their associated priorities, and the density of the relation *cov* which is expressed a number *d* representing the maximum arity of any links in *cov*. For industrial instances, we took a feature model designed to represent video-conferencing systems and a test suite available for testing the products of this software product line. The generator draws a number *a* at random between 1 and *d* and creates *a* edges in the bipartite graph which represents *cov*.

4.1 Results and Analysis

4.1.1 Comparison of CP Models

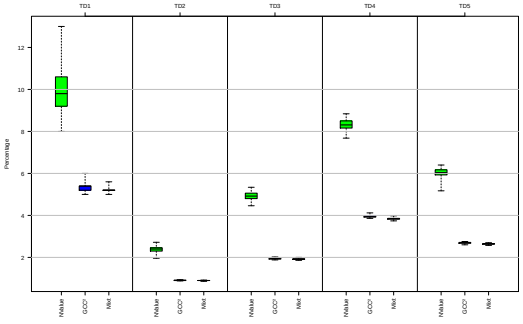


	TD1	TD2	TD3	TD4	TD5
Requirements	20	90	60	60	30
Test cases	70	100	100	200	500
Density	8	20	20	20	8

Figure 3: Comparison of FLOWER/C with other CP models (to=300s).

Fig. 3 contains a comparison of the CPU time used to solve instances of FTSR for three distinct CP models used in our implementation FLOWER/C. The goal here is to observe the time taken to find an actual global optimum of the constraint optimization models. The data used for this experiment have been randomly generated and we show the results on 20 random samples. A time-out of 300 seconds was set up in order to keep reasonable time for the results analysis. The model based on two *GCC* global constraints (called GCC^2) exhibits time-out for all the data sets but TD1. On the contrary, the results obtained for the model with NVALUE shows better performences for this model as it achieves good results in all the cases. Even better, the mixt model which combines NVALUE and *GCC* guarantees optimal results in all the five case studies. Comparing the CPU time taken by the three models is obviously interesting but it may hide differences in terms of reduction rates obtained in a given amount of time. This is the objective of the following experiment.

4.1.2 Comparison of the Reduction Rate



	TD1	TD2	TD3	TD4	TD5
Requirements	250	500	1000	1000	1000
Test cases	500	5000	5000	5000	7000
Density	20	20	20	8	8

Figure 4: Comparison of reduction rate (in % of remain. test cases, to=30s).

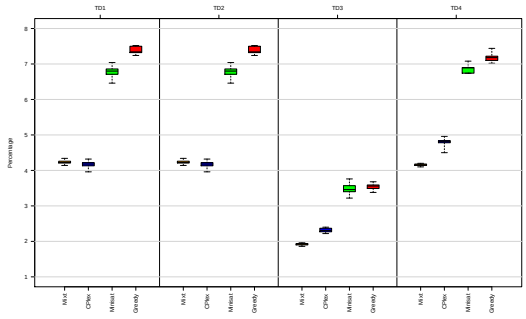
Fig. 4 shows the reduced percentage of test cases after 30 seconds of computation.

In this experiment, the constraint optimization model using NVALUE is less efficient than the two models using *GCC*. This is due to the usage of dedicated search heuristics in the latter case which allows much more efficient searches among the feasible solutions of the optimization problem. Despite the interest of comparing constraint optimization models using similar techniques based on global constraints, it is equally important to compare Constraint Programming techniques with other traditional approaches.

4.1.3 Evaluation of Flower/C against Other Approaches

In a first experiment, we compared our implementation, Flower/C, with three other approaches, namely MINTS/MiniSAT+, MINTS/CPLEX and Greedy on randomly-generated instances. MINTS is a generic tool which handles the test-suite reduction problem as an integer linear program (Hsu and Orso, 2009). For each feature to be covered, a linear inequality is generated which enforces the coverage of the requirement. The selection of test cases is ensured by the usage of auxiliary boolean variables. MINTS can be interfaced with distinct constraint solvers, including MiniSAT+ and CPLEX. Note that CPLEX is considered as the most advanced available technology to solve linear programs. We also implemented a greedy approach for solving the FTSR-problem. This ap-

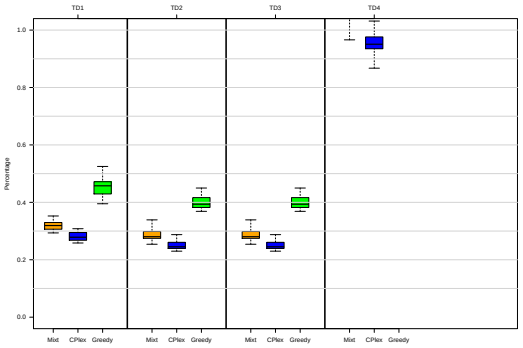
proach is based on the selection of the test cases covering the most features. All our experiments were run on a standard i7 CPU machine at 2.5GHz with 16GB RAM. Fig. 5 shows the results of experiments



	TD1	TD2	TD3	TD4
Features	1000	1000	1000	2000
Test cases	5000	5000	5000	5000
Density	7	7	20	20

Figure 5: Comparison of reduction rate of Flower/C (Mixt), MINTS/MiniSAT+, MINTS/CPLEX and Greedy on random instances with uniform priority values (to=60s).

when considering the reduction rate achieved by all the three approaches in 60*s* of CPU time. In this experiment, the same priority-values are used for all test cases. By reduction rate, we mean the ratio between the size of reduced test suite over the initial size of the test suite expressed in percentage. In this context, the smaller the better. We observe that for the four groups of random instances (ranging from 1000 to 2000 features with distinct maximal density values), Flower/C achieves equal or better results than all the three other approaches in terms of reduction rates in a limited amount of time. Regrading the two last groups (TD3 and TD4), Flower/C performs even strictly better than all the three other approaches reaching exceptional reduction rates. It is worth noticing for each group, one hundred random instances was generated which means that the results are quite stable w.r.t. random variations. It is also quite clear that CPLEX performs much better for these problems than MiniSAT+. This does not come as a big surprise by noticing that the FTSR problem has a simple formulation in terms of integer linear program which are better solved with CPLEX than MiniSAT+. In a second experience reported in Fig.6, we gave different maximum cost values to each instance and the random generator selected at random for each test case a value in between 1 and this maximum value. In this experiment, we did not get any result with MiniSAT+ because the encoding of the objective function as the sum of priority-values did not allow us to use a Boolean SAT solver. So, only the results with CPLEX, Flower/C



	TD1	TD2	TD3	TD4
Features	1000	1000	1000	2000
Test cases	5000	5000	5000	5000
Density	7	7	20	20
Cost	50	100	100	100

Figure 6: Comparison of reduction rate of Flower/C (Mixt), CPLEX and Greedy with non-uniform cost values (to=60s).

and greedy are reported. Fig.6 shows that the results are slightly in favor of MINTS/CPLEX on the three first groups of random instances, while the allocated time for the last group was insufficient to get any interesting results.

4.1.4 Evaluation on Industrial Instances

We conducted a third experience reported in Fig.7 on industrial instances of the FTSR problem. One of our industrial partners provided us with data extracted from a software product line of video-conferencing systems. This industrial case study includes a feature model to represent the product line and meta-data associated to test cases (test case execution duration, realized feature coverage, fault-detection proness, etc.). We compared three distinct constraint optimization models (all three based on the usage of the global constraints NValue and GCC). The CPU time required to solve the FTSR problem reported in Fig.7, shows that the model combining NValue and GCC is the best. Interestingly, the results also show that the reduction rate is quite high for all the five industrial instances (from 61,80% to 26,67%) which indicate that solving the FTSR problem in practice is of great importance. Finally, the last row of the table shows the number of removed features while processing the instances. A feature f_1 can automatically be removed from the constraint optimization model when the coverage of another feature f_2 necessarily entails the coverage of f_1. Here again, the results show that detecting such automatically entailed features is of paramount importance. However, it is worth noticing that we conducted our evaluation in the laboratory and further ex-

Features	59	53	50	37	37	156
Test cases	107	90	93	100	100	377
CPU Time FlowerC/Nvalue (sec)	0.00	0.10	0.01	0.01	0.01	0.03
CPU Time FlowerC/GCC² (sec)	300.00	102.00	91.80	59.16	6.09	300.00
CPU Time FlowerC/GCC_Nvalue (sec)	0.00	0.01	0.00	0.00	0.00	0.01
Reduction rate/Test cases (%)	28,97	26,67	29,03	40,00	37,00	61,80
Automatically removed features (%)	32,00	30,19	30,00	32,43	45,95	44,87

Figure 7: Evaluation of Flower/C on industrial instances.

periments are requested to understand how Flower/C could be integrated in a realistic software development chain.

5 RELATED WORKS

In this section, we analyse our approach FLOWER/C based on Constraint Programming and global constraints with other approaches for feature-based test suite reduction. Different techniques have been proposed to minimize the number of test cases in the context of feature coverage. Among these approaches, Combinatorial Interaction Testing (CIT) (Cohen et al., 1997) is the most important. As observed by Kuhn in (Kuhn et al., 2004), software defects are often due to the interactions of only a small number of parameters of features. A simple case of CIT, widely used by validation engineers, is one-way or pairwise testing. One-way testing aims at covering each feature at least once while pairwise testing aims at covering all the interactions between two features (Cohen et al., 1997). Some of the best algorithms used to generate all combinatorial interactions have been implemented in commercialized tools, such as AETG (Cohen et al., 1997), TConfig and so on. Even if these tools have demonstrated their potential for industrial adoption, they do not guarantee the reach of global minima when it comes to find the smallest subset of test cases such that all features are covered at least once. Moreover, they hardly take into account priorities and other criteria (test execution time, code coverage, etc.) when selecting test cases.

More recently, some authors have proposed to use constraint solvers to generate test cases such that all one-way or pairwise feature interactions are covered. CIT can been tuned for the coverage of feature interactions with SAT-solving as shown in (Mendonca et al., 2009). (Perrouin et al., 2012) proposes to convert variability models (used to represent all the features of a software product line) in Alloy declarative programs, so that an underlying SAT-solver can be used to generate test cases. Despite its novelty, this approach does not scale well because it is based on a generate-and-test paradigm. More precisely, it proposes a candidate test case and test wether it covers remaining uncovered features or not. Moreover, it represents the coverage relation with Boolean variables, which may lead to a combinatorial explosion in the problem representation. Unlike this approach, FLOWER/C represents the problem in a radically different way by associating a finite domain to each variable associated to a test case. This representation is efficient as it allows us to save much space. Furthermore, using global constraints, FLOWER/C can prune the search space by eliminating in advance possible choices of test cases which would lead to non-optimal feasible solutions. (Oster et al., 2010) proposes to use a greedy algorithm for solving the problem. This algorithm is very similar to the one we implemented to compare FLOWER/C with a greedy approach. Our experiments show that Constraint Programming achieves better reduction rates than greedy, as it can reach actual global minima. Another greedy algorithm coupled with clever heuristics is proposed in (Johansen et al., 2012). Although this approach allows validation engineers to deal with large industrial case studies, it is not easily comparable to FLOWER/C as it uses heuristics and does not guarantee to reach global minima.

6 CONCLUSION

In the context of software product line testing, this paper addresses the Feature-based Test suite Reduction problem which aims at minimizing a test suite where priority-values are given to the test cases, while preserving the coverage of tested features. It introduces Flower/C, a tool based on global constraints and a dedicated search heuristics to solve this problem. The tool is evaluated on both random and indus-

trial instances of the problem and the results showed that Constraint Programming with global constraints achieves good results in terms of reduction rate on both the random and industrial instances. For industrial instances, three Constraint Programming models are compared with different global constraints and we show that this is a mixture of NVALUE and GLOBAL_CARDINALITY which achieves the best result. Interestingly, these results show that Constraint Programming is competitive with other multi-objectives test suite optimization approaches.

The main persepctive of this work includes the deployment of this technique and its industrial adoption. Even if the preliminary results reported in this paper need to be further refined and extended, we believe that they are sufficiently convincing to industrialize the technology. For that purpose, its integration within an existing software development chain needs to be understood. In particular, handling meta-data about test cases such as duration, priority and code-coverage needs a proper instrumentation and the implementation or usage of specific monitiring tools to capture the required information.

REFERENCES

Agrawal, H. (1999). Efficient coverage testing using global dominator graphs. In *Workshop on Program Analysis for Software Tools and Eng. (PASTE'99)*.

Black, J., Melachrinoudis, E., and Kaeli, D. (2004). Bi-criteria models for all-uses test suite reduction. In *26th Int. Conf. on Software Eng.*, pages 106–115.

Campos, J., Riboira, A., Perez, A., and Abreu, R. (2012). Gzoltar: an eclipse plug-in for testing and debugging. In *IEEE/ACM International Conference on Automated Software Engineering, ASE'12, Essen, Germany, September 3-7, 2012*, pages 378–381.

Chen, Z., Zhang, X., and Xu, B. (2008). A degraded ILP approach for test suite reduction. In *20th Int. Conf. on Soft. Eng. and Know. Eng.*

Cohen, D. M., Dalal, S. R., Fredman, M. L., and Patton, G. C. (1997). The aetg system: An approach to testing based on combinatorial design. *IEEE Transactions on Software Engineering*, 23(7):437 – 444.

Ferrer, J., Kruse, P. M., Chicano, F., and Alba, E. (2015). Search based algorithms for test sequence generation in functional testing. *Information and Software Technology*, 58(0):419 – 432.

Gotlieb, A., Carlsson, M., Liaeen, M., Marijan, D., and Petillon, A. (2016). Automated regression testing using constraint programming. In *Proc. of Innovative Applications of Artificial Intelligence (IAAI'16)*, Phoenix, AZ, USA.

Gotlieb, A. and Marijan, D. (2014). Flower: Optimal test suite reduction as a network maximum flow. In *Proc. of Int. Symp. on Soft. Testing and Analysis (ISSTA'14)*, San José, CA, USA.

Hao, D., Zhang, L., Wu, X., Mei, H., and Rothermel, G. (2012). On-demand test suite reduction. In *Int. Conference on Software Engineerig*, pages 738–748.

Harrold, M. J., Gupta, R., and Soffa, M. L. (1993). A methodology for controlling the size of a test suite. *ACM TOSEM*, 2(3):270–285.

Henard, C., Papadakis, M., Perrouin, G., Klein, J., and Traon, Y. L. (2013). Multi-objective test generation for software product lines. In *17th International Software Product Line Conference, SPLC 2013, Tokyo, Japan - August 26 - 30, 2013*, pages 62–71.

Hsu, H.-Y. and Orso, A. (2009). MINTS: A general framework and tool for supporting test-suite minimization. In *31st Int. Conf. on Soft. Eng. (ICSE'09)*, pages 419–429.

Jeffrey, D. and Gupta, N. (2005). Test suite reduction with selective redundancy. In *21st Int. Conf. on Soft. Maintenance*, pages 549–558.

Johansen, M. F., Haugen, O., and Fleurey, F. (2012). An algorithm for generating t-wise covering arrays from large feature models. In *Proceedings of the 16th International Software Product Line Conference - Volume 1*, SPLC '12, pages 46–55, New York, NY, USA. ACM.

Kuhn, D. R., Wallace, D. R., and Gallo, Jr., A. M. (2004). Software fault interactions and implications for software testing. *IEEE Trans. Softw. Eng.*, 30(6):418–421.

Li, D., Jin, Y., Sahin, C., Clause, J., and Halfond, W. G. J. (2014). Integrated energy-directed test suite optimization. In *International Symposium on Software Testing and Analysis, ISSTA '14, San Jose, CA, USA - July 21 - 26, 2014*, pages 339–350.

Martin Fagereng Johansen, O. H. and Fleurey, F. (2011). Properties of realistic feature models make combinatorial testing of product lines feasible. In *Conference on Model Driven Engineering Languages and Systems (MODELS'11)*, pages 638–652.

Mendonca, M., Wkasowski, A., and Czarnecki, K. (2009). SAT-based analysis of feature models is easy. In *Proceedings of the 13th International Software Product Line Conference*, SPLC '09, pages 231–240, Pittsburgh, PA, USA. Carnegie Mellon University.

Mouthuy, S., Deville, Y., and Dooms, G. (2007). Global constraint for the set covering problem. In *Journées Francophones de Programmation par Contraintes*, pages 183–192.

Offutt, A. J., Pan, J., and Voas, J. M. (1995). Procedures for reducing the size of coverage-based test sets. In *12th Int. Conf. on Testing Computer Soft.*

Oster, S., Markert, F., and Ritter, P. (2010). Automated incremental pairwise testing of software product lines. In *Software Product Lines: Going Beyond*, volume 6287 of *Lecture Notes in Computer Science*, pages 196–210. Springer Berlin Heidelberg.

Pachet, F. and Roy, P. (1999). Automatic generation of music programs. In *Principles and Practice of Constraint Prog.*, volume 1713 of *LNCS*.

Perrouin, G., Oster, S., Sen, S., Klein, J., Baudry, B., and Traon, Y. L. (2012). Pairwise testing for software product lines: comparison of two approaches. *Software Quality Journal*, 20(3-4):605–643.

Régin, J.-C. (1996). Generalized arc consistency for global cardinality constraint. In *13th Int. Conf. on Artificial Intelligence (AAAI'96)*, pages 209–215.

Régin, J.-C. (2011). Global constraints: a survey. In *In Hybrid Optimization*, pages 63–134. Springer.

Rossi, F., Beek, P. v., and Walsh, T. (2006). *Handbook of Constraint Programming (Foundations of Artificial Intelligence)*. Elsevier Science Inc., New York, NY, USA.

Rothermel, G., Harrold, M. J., Ronne, J., and Hong, C. (2002). Empirical studies of test-suite reduction. *Soft. Testing, Verif. and Reliability*, 12:219–249.

Stolberg, S. (2009). Enabling agile testing through continuous integration. In *Agile Conference, 2009. AGILE'09.*, pages 369–374. IEEE.

Tallam, S. and Gupta, N. (2005). A concept analysis inspired greedy algorithm for test suite minimization. In *6th Workshop on Program Analysis for Software Tools and Eng. (PASTE'05)*, pages 35–42.

Uzuncaova, E., Khurshid, S., and Batory, D. (2010). Incremental test generation for software product lines. *IEEE Trans. on Soft. Eng.*, 36(3):309–322.

Wang, S., Ali, S., and Gotlieb, A. (2013). Minimizing test suites in software product lines using weight-based genetic algorithms. In *Genetic and Evolutionary Computation Conference (GECCO'13)*, Amsterdam, The Netherlands.

Wang, S., Ali, S., and Gotlieb, A. (2015). Cost-effective test suite minimization in product lines using search techniques. *Journal of Systems and Software*, 103:370–391.