Graph Methods for Generating Test Cases with Universal and Existential Constraints

Sylvain Hallé, Edmond La Chance, Sébastien Gaboury

Graph Methods for Generating Test Cases with Universal and Existential Constraints

Sylvain Hallé[1], Edmond La Chance[1], and Sébastien Gaboury[1]

Laboratoire d'informatique formelle
Département d'informatique et de mathématique
Université du Québec à Chicoutimi, Canada

Abstract. We introduce a generalization of the t-way test case generation problem, where parameter t is replaced by a set Φ of Boolean conditions on attribute values. We then present two reductions of this problem to graphs; first, to graph colouring, where we link the the minimal number of tests to the chromatic number of some graph; second, to hypergraph vertex covering. This latter formalization allows us to handle problems with constraints of two kinds: those that must be true for every generated test case, and those that must be true for at least one test case. Experimental results show that the proposed solution produces test suites of slightly smaller sizes than a range of existing tools, while being more general: to the best of our knowledge, our work is the first to allow existential constraints over test cases.

1 Introduction

In recent years, combinatorial testing has gathered increasing interest as a testing technique that can exercise interactions between parameters in an efficient way. Various testing tools, such as PICT [9], have been developed to produce a series of test cases that cover all interactions of t parameters through a test suite containing as few tests as possible. Various approaches attempt to expand the problem by providing *constraints* on what values each parameter can take; for example, one may require that when $a = 0$, then $b \neq 0$, thereby limiting the number of valid test cases that are available. A variety of tools support constraints of such kind, which must be fulfilled by *every* test case returned by the tool.

However, the problem of generating test cases where a set of conditions has to be fulfilled *at least once* by some test has seldom been studied. In this paper, we introduce the concept of Φ-way covering, a generalization of classical t-way test case generation where each Boolean condition in an arbitrary set must be validated by at least one test case. Section 2 formally defines Φ-way covering, and shows a number of testing scenarios where existential (rather than universal) constraints are required, and which cannot be handled by current combinatorial test generators.

In Section 3, we show how Φ-way test case generation can be reduced to finding a colouring of some graph, and link the minimal number of tests to its chromatic number. However, this reduction is shown to work only when the

resulting graph satisfies a property called *maximal satisfiability*. Therefore, in Section 4, we construct a second reduction, this time to the problem of hypergraph vertex covering, which drops the maximal satisfiability requirement and works for arbitrary conditions. Moreover, this second construction also allows us to handle universal constraints that are supported by existing combinatorial testing tools and algorithms.

We have implemented a test-generation tool that uses our graph-based approaches to produce a set of test cases for parameters with arbitrary domains. Experimental results, presented in Section 5, show that a straightforward application of existing graph heuristics, without any further optimization, already produces test suites whose size is comparable to the output of existing tools, and in particular shares the same asymptotic complexity, both in terms of the number of attributes and the number of values. Moreover, this is obtained while our proposed solutions solve a more general problem than existing tools, as they allow both universal and (most importantly) *existential* conditions on test cases.

2 Φ-way Covering

In this section, we introduce the problem of Φ-way test covering and describe a number of testing scenarios where such form of test covering naturally arises.

2.1 Formalization

Let $D_0, D_1, \ldots D_{n-1}$ be domains (sets of possible values) for n different parameters $p_0, \ldots, p_{n-1}$. Let $\Phi = \{\varphi_0, \ldots, \varphi_{m-1}\}$ be a set of m Boolean formulæ whose ground terms are of the form $p_i = d$, for p_i one of the parameters and $d \in D_i$. A Φ-way covering is a set $\Sigma \in 2^{D_0 \times \cdots \times D_{n-1}}$, such that for every $\varphi_i \in \Phi$, there exists an assignment of values $\sigma \in \Sigma$ that makes φ_i evaluate to true (which we shall note $\sigma \models \varphi_i$).

One can see how this problem is a generalization of classical t-way test case generation. Given parameters $p_0, \ldots, p_{n-1}$ with domains $D_0, D_1, \ldots D_{n-1}$, we can construct Φ as the smallest set such that for every set of t parameters $p_1, \ldots, p_t$ and values $d_1, \ldots, d_t$ in their respective domains, the formula $p_1 = d_1 \wedge \cdots \wedge p_t = d_t$ is in Φ. Since the resulting Φ-way covering ensures that every formula in Φ is true, it follows that every combination of values for t parameters is present in the set. As an example, Table 1 gives the set of constraints representing the 2-way covering of the set of parameters a, b and c, each having two possible values.

However, we can show that Φ-way covering is a strict generalization of t-way test case generation, as finding *one* solution to the problem is in the same complexity class as finding the *best* solution for a t-way test case generation problem [11].

Theorem 1 *Finding a solution to the Φ-way Covering problem is NP-complete.*

Proof. Given a set Σ, verifying that each $\varphi_i \in \Phi$ is satisfied at least once amounts to evaluating it with every $\sigma \in \Sigma$, which is done in polynomial time; hence the

$$\{a = 0 \wedge b = 0, a = 0 \wedge b = 1, a = 1 \wedge b = 0, a = 1 \wedge b = 1,$$
$$a = 0 \wedge c = 0, a = 0 \wedge c = 1, a = 1 \wedge c = 0, a = 1 \wedge c = 1,$$
$$b = 0 \wedge c = 0, b = 0 \wedge c = 1, b = 1 \wedge c = 0, b = 1 \wedge c = 1\}$$

Table 1: The set of Boolean constraints representing the 2-way coverage of three 2-valued parameters named a, b, c.

problem is in NP. Solving Φ-way covering when Φ consists of a single Boolean expression φ is nothing but solving the satisfiability problem (SAT) for φ; hence the problem is NP-hard.

2.2 A Case for Φ-way Covering

While t-way test case generation has proved useful in many scenarios, it was shown how in some situations, constraints must be added to the original t-way requirement to correctly handle the problem at hand. Current tools and algorithms have focused up to now on *universal* constraints, which must hold true on every test case of the test suite. For example, in some situations, a combination of values for two parameters might be mutually exclusive: one can imagine a function which can send its output either to stdout or to a file; it does not make sense, in such a case, to set the `output` parameter to stdout, and to have a non-empty value for parameter `filename`. This constraint must apply to every test case produced by an algorithm.

However, the question of *existential* constraints, as is the case in the Φ-way covering presented above, has been studied much less often. Yet, in the following, we show three test case scenarios where such constraints are required —that is, the same functionality cannot be obtained, or expressed differently, using only universal constraints.

Example 1: Completing an existing test suite As a first example, suppose we have an existing test suite, which does not cover all t-combinations of values. One would like to extend that test suite, keeping existing test cases but adding the minimum number of new tests so that 100% coverage of all t-combinations is achieved.[1] Imagine for example parameters a, b and c, each with domain $\{0, 1\}$, and a test suite T composed of only two tests: $\{(a = 0, b = 0, c = 0), (a = 0, b = 0, c = 1)\}$.

To find an extension to that test suite, it suffices to first generate the set Φ of conditions corresponding to t-way coverage as described above (for example, for $t = 2$, we obtain the set in Table 1). We then create one existential condition for each test in T; in the present case, this would lead to the addition of conditions

[1] Note how this is different from finding the minimum number of tests from scratch, as the existing test cases may not be part of an optimal solution, yet must be kept.

$a = 0 \wedge b = 0 \wedge c = 0$ and $a = 0 \wedge b = 0 \wedge c = 1$ to Φ. We then solve the Φ-way covering problem. By construction, any minimal solution for this problem is the smallest way to obtain t-way covering while preserving the existing tests. With a graph theory approach, completing a test suite is faster than producing one from scratch because the generated graph is smaller.

Example 2: Equivalence classes In some cases, the original requirement of t-way coverage might prove needlessly strict. Φ-way coverage allows one to relax these conditions, while still looking for an optimal test suite. Consider for example parameters a, b and c with domains $\{0, \ldots, 9\}$, $\{0, 1\}$ and $\{0, 1\}$, respectively. Suppose that some values of a are equivalent in its relationship with b; for example, all values of $a < 5$ behave in the same way when $b = 0$. In other words, when $b = 0$ and c has some arbitrary values, all values of $a < 5$ exercise the same functionality, and do not need to be tested separately.

It is not possible to express such a fact using standard t-way test covering algorithms, which will produce an overly strong (and large) test suite, as they will try to cover all combinations of values for a, b, and c. Moreover, it is not possible either to recover from this issue by writing universal constraints, unless one forces the selection of one value for a when $b = 0$, which in turn might prevent the resulting test suite from being optimal. Finally, one can imagine more complex dependencies on parameter values where imposing fixed values to parameters in an optimal way amounts to nothing but hard-coding the solution by hand inside the constraints.

On the other hand, this situation is nicely handled in Φ-way covering. When generating the set Φ of constraints for t-way covering, it suffices to replace all conditions $a = x \wedge b = 0 \wedge c = x'$ (where $x_i < 5$ and some fixed value x') by the *single* condition $a < 5 \wedge b = 0 \wedge c = x'$, and then to solve the resulting Φ-way problem.

The reader shall remark that the same thing can be achieved by replacing the values 0 to 4 in the domain of a by a symbolic value meaning "$a < 5$", and to solve the corresponding t-way problem. Indeed, all combinations of values of parameters a and c are still considered distinct; these combinations will be missed if values 0 to 4 are amalgamated into a single symbol.

Example 3: MC/DC testing The modified condition/decision coverage [6] (MC/DC) is a code coverage criterion for test cases which, among other things, requires that each condition in a decision takes on every possible outcome, and that each condition in a decision is shown to independently affect the outcome of the decision.

The question of performing both t-way testing and MC/DC testing in a single test suite has only been recently studied [13]. Current approaches have measured the amount of MC/DC coverage provided by a t-way test suite, without providing any specific adaptation of existing algorithms for MC/DC coverage (and vice versa). Therefore, any good MC/DC coverage obtained by a t-way test suite is, for the moment, unintentional.

However, this can be handled with Φ-way covering. One first generates the set Φ of conditions required for t-way coverage, as was described earlier. To obtain MC/DC coverage, one simply adds to Φ the conditions corresponding to MC/DC. By construction, any solution to the resulting problem will make sure that each of the t-way conditions is met by at least one test case, and that each of the MC/DC conditions will also be met by at least one test case. Note that this may result in more test cases than for any problem taken separately; however, any minimal solution of the combined problem is guaranteed to be optimal.

3 Reduction to Graph Colouring

Assuming a few restrictions on the set of conditions Φ, we shall now show how the problem of generating Φ-way test cases can be reduced to a well-known graph problem.

3.1 Construction

Given a set of Boolean formulas Φ, the conjunctive closure of Φ is the smallest set $\Phi' \supseteq \Phi$ such that if $\varphi, \varphi' \in \Phi'$, then $\varphi \wedge \varphi' \in \Phi'$.[2] A set Φ is said to be *maximally satisfiable* if every formula in its conjunctive closure is satisfiable.

We shall now devise a construction that reduces the problem of Φ-way covering to the problem of graph colouring. Let $G = \langle V, E \rangle$ be a graph such that $V = \Phi$, and E is such that there is an edge between two vertices φ, φ' if and only if $\varphi \wedge \varphi'$ is unsatisfiable. Let C be a set of vertex "colours" and $\kappa : V \to C$ a function assigning a colour to every vertex of the graph. For $k = |C|$, the function κ is called a *k-colouring* if it is surjective and moreover, any two adjacent vertices are not assigned the same colour. We will define $V_c \subseteq V$ as the set of vertices that are assigned colour c by κ (and by extension, the set of all Boolean expressions assigned to these vertices). Figure 1 shows an example of such a colouring, for the set of constraints given in Table 1.

Theorem 2 *Let Φ be a set of formulæ, G be the graph constructed from Φ as described above and κ be a k-colouring of G. If V_c is maximally satisfiable for every $c \in C$, then there exists a Φ-way covering Σ such that $|\Sigma| = k$.*

Proof. Define Φ_c as the conjunction of all expressions in V_c. One can construct a test case from V_c by choosing any satisfying assignment of variables of Φ_c. Such an assignment exists, since V_c is maximally satisfiable. Let Σ be the set of all test cases constructed in such a way, for every colour $c \in C$. Since every vertex is given a colour, taken together, the test cases in Σ satisfy every condition of Φ at least once, and hence constitute a Φ-way covering.

[2] Although Φ' is potentially infinite, it can be restricted to a finite subset by avoiding constructing expressions of the form $\varphi \wedge \varphi$, which are equivalent to φ.

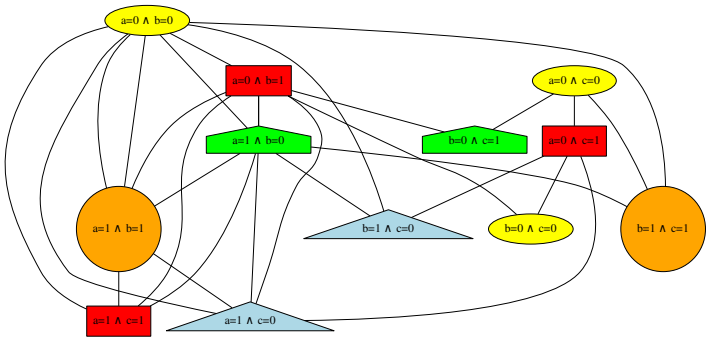


Fig. 1: The graph constructed from the constraints given in Table 1, and a possible 5-colouring.

This result shows that a *minimal* Φ-way coverage can be computed by converting the problem into the classical graph colouring problem, provided that the resulting colouring produces maximally satisfiable sets of vertices. This is not always true in the general case; Figure 2 shows a graph whose colouring is not maximally satisfiable. One can see that for every pair of vertices, it is possible to find values for a and b that satisfy both conditions; this is why no vertex is connected to any other. Hence it is possible to assign the same colour to every vertex; however, one can see that the conjunction of the condition of all three nodes is a contradiction —in other words, it is not possible to form a test case out of it.

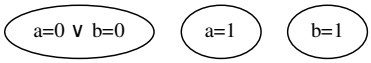


Fig. 2: A graph whose colouring is not maximally satisfiable.

However, we can show that a colouring is always maximally satisfiable if we impose restrictions on Φ.

Theorem 3 *Let Φ be a set of Boolean formulæ $\{\varphi_1, \ldots, \varphi_n\}$, where each φ_i is a conjunction of atomic propositions of the form $p_j = d$. If $\varphi_i \wedge \varphi_j$ is satisfiable for any pair of formulæ $\varphi_i, \varphi_j \in \Phi$, then Φ is maximally satisfiable.*

Proof. Suppose the contrary. Then there exists a set $S \subseteq 2^{[1,n]}$ of indices such that $\bigwedge_{i \in S} \varphi_i$ is unsatisfiable. Since every φ_i is a conjunction of parameter-value

equalities, this entails that there exist two assertions $p_j = d$ and $p_j = d'$ such that $d \neq d'$. Let $k, k' \in [1, n]$ such that the first assertion occurs in φ_k and the second occurs in $\varphi_{k'}$. Then $\varphi_k \wedge \varphi_{k'}$ is unsatisfiable, which contradicts the hypothesis.

3.2 Complexity Results

This result has a couple of important consequences. First, we can show that any classical t-way covering problem generates a graph that satisfies the hypotheses of Theorem 3.

Corollary 1 *Let $G = \langle V, E \rangle$ be a graph generated from a t-way problem instance and let $\kappa : V \to C$ be a colouring of G as defined above. For any colour $c \in C$, V_c is maximally satisfiable.*

Proof. Since every vertex in V_c is a conjunction of expressions of the form $p_j = d$, the result follows from Theorem 3.

Hence finding a set of k test cases, if it exists, can be solved by converting the problem to k-colouring. Second, and perhaps most importantly, the previous construction provides us with a means of calculating the *exact* lower bound to the number of tests required to achieve t-way coverage. This bound is nothing but the *chromatic number* of the graph. Moreover, it is possible to compute this optimal size without even generating the solution, by constructing the *chromatic polynomial* of the graph; the chromatic number (and hence the smallest number of tests cases required) is the smallest integer that is not a root of this polynomial.

The current best known algorithm can find a k-colouring in time $O(2^n \cdot n)$, where n represents the number of constraints. Given the chromatic number can be known directly through the method described above, this gives us the same complexity for generating a minimal set of test cases. In the absence of the known value, one can fall back on simple dichotomic search, as the desired value k lies between 1 and $C(t, n) \times |D|^t$, the total number of combinations for values of t parameters.

3.3 Heuristics for Graph Colouring

One main advantage of reducing the test case generation problem to an existing mathematical problem is that one can then tap directly into a number of optimizations specific to that problem, rather than trying to solve it from scratch. This is precisely the case for graph colouring, which has been the subject of extensive study for decades. In particular, a number of methods for computing an approximate solution, called *heuristics*, can be used to produce a test suite whose size can be bounded in terms of the size of the optimal solution.

The most well known heuristic for graph colouring is DSATUR from Brélaz [2]. This heuristic is partially built on the observation that more often than not, vertices with a bigger degree generally end up with a colour of their own instead of being assigned an existing colour. The algorithm starts out by selecting the

biggest degree vertex, colouring it, and then it colours the neighborhood of that vertex using the DSAT (Degree Saturation) measure. When the DSAT of two vertices is the same, the degree is used for tie-breaker. The DSAT of all vertices is updated at every colouring of a vertex. The DSAT of a vertex is an integer representing how many adjacent colours are in the neighborhood.

The DSATUR heuristic is unfortunately not great for combinatorial testing because in the generated graph files, every graph node has the same degree. Therefore, the heuristic cannot really work like it is supposed to.

The depth-first search (DFS) greedy heuristic is perhaps the most simple heuristic of all. The graph is simply explored in a typical depth-first-search fashion and the colours are assigned turn by turn. The quality of the solution depends exclusively on the sequence of the vertices. In fact, the DFS algorithm can be modified into an exact backtracking algorithm. For instance, if the backtracking algorithm is called with the task of finding a 13-colour permutation and the current colour count is 14, it does not make any sense to keep colouring the graph. The algorithm needs to backtrack and change it's permutation until it can colour the whole graph with the proper number of colours. This backtracking algorithm is almost impossible to use in practice for combinatorial testing because the graphs are too big.

Since trying every path leads to the exact solution, sampling the paths by using randomness is also a good way to go and leads to better solutions than greedy algorithms. The random DFS heuristic works in this way. For each vertex exploration, the neighborhood will be explored in a random way a number of times. When the k random explorations fail, the algorithm falls back to regular neighborhood exploration, until it can return or go to the next vertex. This is done because of course, true random exploration could slow the algorithm to a stall.

Since adding randomness to the DFS exploration scheme turns out to be very cheap and not increasing the time complexity, this heuristic is usually ran hundreds of times and the best result is used.

4 Reduction to Hypergraph Vertex Cover

As we have seen in the previous section, the graph colouring approach can only be applied when the set of assertions resulting from the initial constraints is maximally satisfiable. Yet, there exist situations where this additional hypothesis does not hold. In this section, we present an alternate method that attempts to alleviate this problem.

This method is a reduction of the Φ-way covering problem to finding a vertex covering of some hypergraph. As a reminder, a *hypergraph* is a tuple $G = \langle V, E \rangle$, where V is a set of vertices and $E \subseteq 2^{2^V}$ is a set of edges. A hypergraph generalizes a classical graph by having edges that may link more than two vertices. A *vertex covering* of some hypergraph $G = \langle V, E \rangle$ is a set of vertices $V' \subseteq V$, such that for every edge $e = \{v_0, v_1, \ldots, v_k\} \in E$, we have that $e \cap V' \neq \emptyset$. Hence every hyperedge of the graph is adjacent to at least one vertex in V'.

4.1 Construction

Given parameters $p_0, \ldots, p_{n-1}$ with domains $D_0, D_1, \ldots D_{n-1}$, let Φ be a set of Boolean constraints whose ground terms are parameter-value equalities. Define $V = D_0 \times D_1 \times \cdots \times D_{n-1}$ as the set of all possible combinations of values for each parameter. The set of hyperedges E is then constructed such that $e = \{v_0, v_1, \ldots, v_k\} \in E$ if and only if there exists some $\varphi \in \Phi$, such that $v_i \in e$ if and only if $v_i \models \varphi$. In other words, each condition $\varphi \in \Phi$ is associated to exactly one hyperedge, linking all vertices giving values for parameters that make that condition evaluate to true.

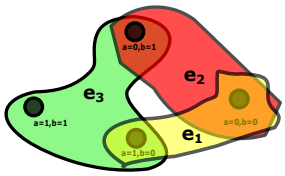


Fig. 3: The hypergraph constructed from the constraints $\Phi = \{a = 0, b = 0, a \neq 0 \vee b \neq 0\}$. Numbers are used to label segments belonging to the same hyperedge.

Figure 3 gives an example of such a construction, for the simple set of conditions $\Phi = \{a = 0, b = 0, a \neq 0 \vee b \neq 0\}$, assuming that parameters a and b have the same domain $D = \{0, 1\}$. One can see that the hyperedge labelled "1" links all vertices with values satisfying the first condition ($a = 0$); similarly, the hyperedge labelled "2" links all vertices with values satisfying the second condition ($b = 0$). Finally, the hyperedge labelled "3" links all vertices with values satisfying the last condition ($a \neq 0 \vee b \neq 0$); this last edge links three vertices.

We shall now demonstrate how one can extract a Φ-way covering out of a vertex cover.

Theorem 4 *Let $p_0, \ldots, p_{n-1}$ be parameters with domains $D_0, D_1, \ldots D_{n-1}$, and Φ be a set of Boolean constraints whose ground terms are parameter-value equalities. Let $G = \langle V, E \rangle$ be the hypergraph constructed from a set of conditions Φ as described above, and let V' be a vertex covering for G. There exists a set $\Sigma \subseteq D_0 \times D_1 \times \cdots \times D_{n-1}$ that is a Φ-way covering; moreover, this set is of size $|V'|$.*

Proof. It suffices to show that $\Sigma = V'$ is the set we are looking for. By construction, every hyperedge of G is adjacent to some $v \in V'$. By construction, this entails that for every $\varphi \in \Phi$, there exists a combination of parameter values v such that $v \models \varphi$, hence V' is a Φ-way covering.

In Figure 3, vertices forming a possible covering of size 2 have been identified in yellow. One can see how every hyperedge is indeed adjacent to some yellow

vertex. Moreover, it is easy to see that no covering of size 1 could achieve the same result. Hence, finding the the minimal Φ-way covering amounts to finding the minimal vertex covering of the hypergraph G constructed from Φ.

Universal constraints Contrary to the graph colouring reduction described in Section 3, the reduction to hypergraph vertex cover works for *arbitrary* existential conditions, thus dropping the requirement for maximal satisfiability that was necessary in the former. Moreover, universal conditions (which must be true for every test case) can also be taken into account in a very straightforward way. Given a set Φ' of such constraints, it suffices to remove from V any vertex for which one of the constraints in Φ' evaluates to false. Since each vertex completely defines the values of all parameters, such an evaluation is always possible, and the fate of every vertex can always be determined. The computation of a vertex cover can then proceed on the pruned graph.

By construction, all vertices in the graph that remains fulfill all the conditions in Φ'. In addition, the vertex cover guarantees that each hyperedge (i.e. each existential condition) is covered at least once. Hence this construction allows us to handle both arbitrary universal and arbitrary existential conditions at the same time.

4.2 Complexity Results

We have already shown that finding a vertex covering of a given size k is NP-complete. However, finding the *minimal* value such that a covering exists is a much harder problem in the general case. Providing the latest complexity results for this problem is out of the scope of this paper, as abundant literature on the subject can easily be found. It is known, for example, that if d is the maximal cardinality of hyperedges and n is the size of the hypergraph, there exists an algorithm that finds a covering of size k in time $d^k n^{O(1)}$ [4].

Upper bounds have also been demonstrated for approximation algorithms. For example, one can construct a maximal matching by greedily adding edges and then let the vertex cover contain all endpoints of each edge in the matching. It can be shown that this algorithm produces a vertex covering at most d times larger than the optimal solution. For $d = 2$, a slightly lower factor of $2 - o(1)$ has been demonstrated [5, 7].

Hypergraph colouring is equivalent to another problem called *hitting set*. Given a set of sets of elements $E = \{e_1, \dots e_n\}$, a hitting set is a subset V that intersects with every $e_i \in E$. When E is the set of hyperedges of G, a hitting set is precisely a set of vertices covering every edge of the hypergraph.

These results can be transferred directly to Φ-way and t-way covering. Given uniform domains of size $|D|$ for each parameter, classical t-way covering for a set of n parameters generates a hypergraph with $|D|^n$ vertices whose edges are of uniform cardinality $k = |D|^{n-t}$. Applying a result from Khot and Regev [8] which assumes the so-called *unique games conjecture*, this entails that finding the minimal number of tests is hard to approximate within any constant factor better than k.

5 Experimental Results

We implemented both approaches into an open source and mixed-language combinatorial test case generator, which is publicly available.[3] First, a PHP script takes as input a file in the PICT format, giving the name of each parameter and its possible values. Since our tool also handles universal and existential constraints, we extended the PICT format to allow the expression of these conditions. They either begin by the reserved word `Once` or `Always`, and can express any Boolean combination of elementary condition on the file's declared parameters. Figure 4 shows a sample input file.

```
a : 0, 1, 2
b : 0, 1
c : 2, 3, 4, 5

Once a != b
Always !(a > b) || c == 2
Once a < b
```

Fig. 4: An extended PICT input file for our test case generator.

The script returns a file in the Graphviz format (DOT) representing either the graph for which a colouring needs to be found, or the hypergraph for which a covering must be computed. Then, a C++ implementation of both graph algorithms reads this file and applies the corresponding heuristics, and outputs the resulting colouring or vertex cover, which can then be directly reinterpreted as a set of test cases, as explained earlier.

To assess the interest of the approach, we designed a benchmark pitting our implementation against four other tools: QICT [9], AllPairs[4], Jenny[5] and TCases[6]. These where the only tools listed on the Pairwise Testing website[7] that were both free and usable from the command-line. They vary in their support of features outside standard t-way covering, as is illustrated in Figure 5. AllPairs and QICT only work for 2-way covering without constraints. Jenny supports $t > 2$, and allows the user to specify forbidden tuples, a restricted form of universal constraints, while TCases supports arbitrary universal constraints. Finally, our graph colouring approach handles existential constraints, while the hypergraph vertex cover handles both universal and existential constraints.

The task of exhaustively comparing our proposed implementation with a large sample of existing tools is clearly out of the scope of this paper, especially given that most of these tools are comparable only for the simplest case ($t = 2$ with no

[3] `https://bitbucket.org/sylvainhalle/gcases`
[4] `http://www.mcdowella.demon.co.uk/allPairs.html`
[5] `http://burtleburtle.net/bob/math/jenny.html`
[6] `https://code.google.com/p/tcases/`
[7] `www.pairwise.org/tools.asp`

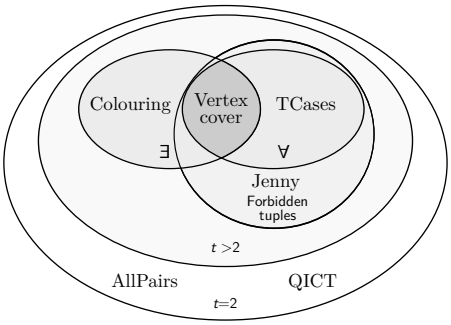
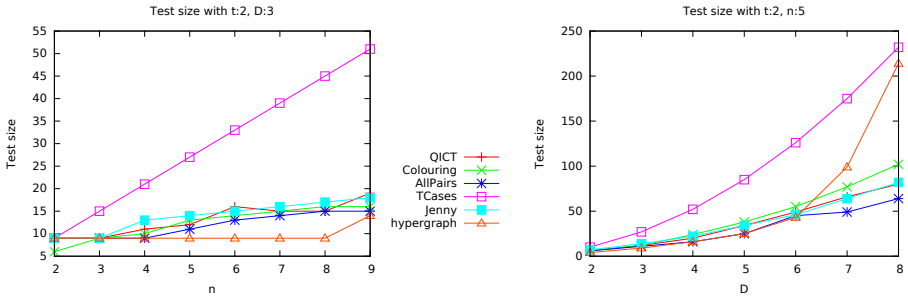


Fig. 5: Support of various features by the testing tools included in our study.

constraints). Nevertheless, we carried four sets of experiments, and measured the size of the resulting test suites for all tools that could solve the problem.

Simplest case: $t = 2$ In the first set of experiments, we considered only pairwise testing with no constraints, and computed the impact on the size of the test suite by varying the values of $|D|$ (the size of the domain) and n (the number of parameters). The results are plotted in Figure 6b for increasing values of $|D|$, and in Figure 6a for increasing values of n. One can see that graph colouring performs comparably to QICT, AllPairs and Jenny, with TCases producing slightly larger test suites. The hypergraph vertex covering provides the best results with respect to n, but shows a steeper increase than other tools with respect to $|D|$.

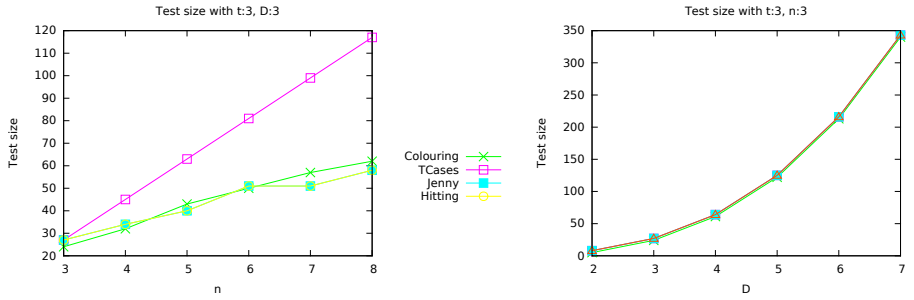


(a) $|D| = 3$, $t = 2$ (b) $n = 5$, $t = 2$

Fig. 6: Test suite size output by various tools, for a varying number of parameters when $t = 2$.

Beyond pairwise: $t > 2$ The same trend continues when the strength of the test suite is extended beyond pairwise testing. This time, AllPairs and QICT are no

longer able to solve these problems, and only Jenny, TCases and our two graph solutions remain. As Figures 7a and 7b show, our tool produces test suites that are again of size smaller than TCases', and identical to Jenny's.



(a) $|D| = 3$, $t = 3$ (b) $n = 5$, $t = 3$

Fig. 7: Test suite size output by various tools, for a varying number of parameters when $t > 2$.

Universal constraints We complexify the tests even further by introducing universal constraints. This time, only TCases and our hypergraph vertex covering can handle such a problem. We experimented both approaches with $t = 2$, and with the universal constraint $\neg(p_3 = 1 \wedge p_4 = 1)$. The results are given in Table 2. As one can see, the graph-based approach provides smaller test suites of a factor of roughly $2\times$ with respect to TCases's algorithm.

n	Hypergraph	TCases
5	60	158
6	60	203
7	118	254

Table 2: Test suite size output by various tools, for a varying number of parameters when $t = 2$.

Existential constraints In this last category, only the hypergraph approach subsists. We experimented it with $t = 2$, and with the existential constraint $\neg(p_3 = 1 \wedge p_4 = 1)$. The results are given in Table 3.

n	Hypergraph
5	65
6	66
7	110

Table 3: Test suite size output by the hypergraph approach, for a varying number of parameters when $t = 2$.

6 Related Work

As far as we could check, the problem of generating t-way test cases has never been generalized to account for a set of arbitrary Boolean conditions that must each apply in at least one test. Given some condition φ, Φ-way covering requires that there exists at least one test in T that satisfies φ, i.e. $\exists t \in T : t \models \varphi$. On the contrary, tools like TCases, PICT, Jenny and VPTag[8] to write conditions that must be true for some attribute to be allowed to take a given value, using some form of if-then construct or by specifying forbidden value combinations. A constraint ψ in these tools must be true for all test cases, i.e. $\forall t \in T : t \models \psi$.

This is in fact the exact logical dual of Φ-way covering. It follows easily that it is impossible to find φ and ψ that make both expressions equivalent for any arbitrary set T. As we have seen, the introduction of Φ-way coverage allows us to express and solve some problem instances that current approaches cannot.

The use of graph colouring for t-way test case generation was first suggested by Cheng et al. [3]. However, there are several differences between this related approach and the one suggested here. First, while the approach allows one to pick sets of parameters for which coverage is required, it is assumed that *all* combinations of their values must be present in the resulting solution. Our construction is more flexible and allows us to express arbitrary tuples of values of irregular length (e.g. $\mathrm{a} = 0 \wedge \mathrm{b} = 0$, $\mathrm{a} = 1$). Moreover, graph colouring is used by Cheng et al. only to pick a subset of parameters on which an initial t-way must then be generated by some other means; that solution is then "blown up" to include coverings for the remaining parameters; as a consequence, the minimality of the resulting covering is not ensured.

Recently, an optimization of the IPOG algorithm's vertical growth phase was suggested [12]. This optimization relies on a "conflict graph", whose colouring allows the algorithm to identify the next best missing tuples to add to the test suite. However, to the best of our knowledge, the present paper is the first time the smallest covering array is expressed directly in terms of the minimal colouring of some graph.

The use of hypergraphs for test case generation appears to be even less common. They have been used to reason about orthogonal arrays [10], although not directly to compute an optimal set of test cases.

[8] `http://sourceforge.net/projects/vptag/`. VPTag is limited to the case where $t = 2$, while our approach generalizes to other values of t

7 Conclusion

We have shown how Φ-way covering, a generalization of t-way test case generation, can be reduced to the classical graph colouring problem, thereby providing (in theory) a way to compute the minimal number of test cases required by any problem instance. This result applies, provided an additional hypothesis on the shape of possible solutions —a hypothesis fulfilled by any classical t-way covering problem. Moreover, in our second modelling, which relies on a translation to the hypergraph vertex covering problem, the Φ-way covering offers the possibility to express conditions in a fashion that goes outside the expressiveness of existing solutions based on dependencies, allowing both universal and existential constraints of arbitrary nature on test cases. Experimental results have shown that, compared to a set of existing test generation tools, both our approaches generate test suites of comparable size, while being strictly more general in terms of expressiveness.

These promising experimental results warrant future work on the approach. First, additional graph colouring and vertex covering heuristics could be implemented and evaluated experimentally. Second, improvements on the efficiency of the generation of the graph from a given problem instance are currently being worked on. Finally, a third graph-based reduction, merging the two approaches presented in this paper, is also planned. It is hoped that, in the longer term, the use of our graph-based algorithms shall provide a drop-in replacement for the existing methods used in current tools and systems.

References

1. 8th IEEE International Conference on Software Testing, Verification and Validation, ICST 2015, Graz, Austria, April 13-17, 2015. IEEE (2015), `http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=7102553`
2. Brélaz, D.: New methods to color the vertices of a graph. Communications of the ACM 22(4), 251–256 (1979)
3. Cheng, C.T., Dumitrescu, A., Schroeder, P.J.: Generating small combinatorial test suites to cover input-output relationships. In: QSIC. pp. 76–82. IEEE Computer Society (2003)
4. Flum, J., Grohe, M.: Parameterized Complexity Theory. Springer (2006)
5. Halperin, E.: Improved approximation algorithms for the vertex cover problem in graphs and hypergraphs. SIAM J. Comput. 31(5), 1608–1623 (2002)
6. Hayhurst, K.J., Veerhusen, D.S., Chilenski, J.J., Rierson, L.K.: A practical tutorial on modified condition/decision coverage. Tech. Rep. NASA/TM-2001-210876, NASA (2001), `http://shemesh.larc.nasa.gov/fm/papers/Hayhurst-2001-tm210876-MCDC.pdf`
7. Karakostas, G.: A better approximation ratio for the vertex cover problem. In: Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M. (eds.) ICALP. Lecture Notes in Computer Science, vol. 3580, pp. 1043–1050. Springer (2005)
8. Khot, S., Regev, O.: Vertex cover might be hard to approximate to within $2-\epsilon$. J. Comput. Syst. Sci. 74(3), 335–349 (2008)

9. McCaffrey, J.: Pairwise testing with QICT. MSDN Magazine 29(12), 28–35 (December 2009)
10. Raaphorst, S., Moura, L., Stevens, B.: A construction for strength-3 covering arrays from linear feedback shift register sequences. Des. Codes Cryptography 73(3), 949–968 (2014), http://dx.doi.org/10.1007/s10623-013-9835-2
11. Tai, K., Lei, Y.: A test generation strategy for pairwise testing. IEEE Trans. Software Eng. 28(1), 109–111 (2002), http://doi.ieeecomputersociety.org/10.1109/32.979992
12. Vilkomir, S., Anderson, D.: Improving IPOG's vertical growth based on a graph coloring scheme. In: 8th IEEE International Conference on Software Testing, Verification and Validation Workshops, ICSTW 2015, Graz, Austria, April 13-17, 2015 [1], pp. 1–8, http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=7102553
13. Vilkomir, S., Anderson, D.: Relationship between pair-wise and MC/DC testing: Initial experimental results. In: 8th IEEE International Conference on Software Testing, Verification and Validation Workshops, ICSTW 2015, Graz, Austria, April 13-17, 2015 [1], http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=7102553