Interaction-Based Test-Suite Minimization

Dale Blue
IBM Systems & Technology Group
2455 South Road
Poughkeepsie, NY 12601, USA
dblue@us.ibm.com

Itai Segall
IBM, Haifa Research Lab
Haifa University Campus
Haifa, 31905, Israel
itais@il.ibm.com

Rachel Tzoref-Brill
IBM, Haifa Research Lab
Haifa University Campus
Haifa, 31905, Israel
rachelt@il.ibm.com

Aviad Zlotnick
IBM, Haifa Research Lab
Haifa University Campus
Haifa, 31905, Israel
aviad@il.ibm.com

***Abstract*—Combinatorial Test Design (CTD) is an effective test planning technique that reveals faults resulting from feature interactions in a system. The standard application of CTD requires manual modeling of the test space, including a precise definition of restrictions between the test space parameters, and produces a test suite that corresponds to new test cases to be implemented from scratch.**

In this work, we propose to use Interaction-based Test-Suite Minimization (ITSM) as a complementary approach to standard CTD. ITSM reduces a given test suite without impacting its coverage of feature interactions. ITSM requires much less modeling effort, and does not require a definition of restrictions. It is appealing where there has been a significant investment in an existing test suite, where creating new tests is expensive, and where restrictions are very complex. We discuss the tradeoffs between standard CTD and ITSM, and suggest an efficient algorithm for solving the latter. We also discuss the challenges and additional requirements that arise when applying ITSM to real-life test suites. We introduce solutions to these challenges and demonstrate them through two real-life case studies.

I. Introduction

As software systems become increasingly complex, verifying their correctness is more challenging. For example, in highly configurable systems, which gain more and more attention in recent years, the many coexisting optional features might unintentionally interact with each other in faulty ways. While verification approaches such as formal verification and model based testing might require extremely expensive resources due to their sensitivity to the size and complexity of the software, functional testing is prone to omissions, as it always involves a selection of what to test from a possibly enormous test space. Therefore, careful consideration of what to include in the testing is required. The process of test planning refers to the definition and selection of tests out of the space of potential tests, with the goals of eliminating redundancy and reducing the risk of bugs escaping to the field as much as possible.

Combinatorial Test Design (CTD), also known as combinatorial testing, is an effective test planning technique, in which the test space is modeled by a set of parameters, their respective values, and restrictions on the value combinations. The test space represented by this model is any assignment of one value to each parameter, that does not violate the restrictions. A subset of the space is then automatically constructed so that it covers all valid value combinations (a.k.a interactions) of every t parameters, where t is usually a user input. In other words, for every set of t parameters, any combination of t values to them will appear at least once in the test plan (unless there is no valid test that contains it, according to the restrictions). In general, one can require different levels of interaction for different subsets of parameters. The most common application of CTD is known as pairwise testing, in which the interaction of every pair of parameters must be covered. Each test in the result of CTD is an assignment of values to all the parameters, and represents a high level test, or a test scenario, that needs to be translated to a concrete executable test.

The reasoning behind CTD is the observation that in most cases the appearance of a bug depends on the combination of a small number of parameter values of the system under test. Experiments show that a test set that covers all possible pairs of parameter values can typically detect 50% to 75% of the bugs in a program [3], [10]. Other experimental work has shown that typically 100% of bugs can be revealed by covering the interaction of between 4 and 6 parameters [6].

As indicated by various studies and reports [1], [11], [4], [2], CTD is very effective for a variety of system types and testing domains, and is considered best practice when the tested functionality depends on multiple factors such as inputs, configuration elements and data items. However, we observe two main requirements of CTD that limit its application in practice. The first is the requirement to precisely define the restrictions between the different parameters. If the restrictions are not accurately defined or not defined at all, there is a very high probability that CTD will generate tests that violate the relationships between the parameters, and therefore do not translate to concrete tests that can be implemented and executed. While the requirement to define restrictions is not a limitation for simple cases like configuration spaces in which the restrictions are obvious, when the relationships between the parameters are complex, e.g., as in the healthcare insurance domain, manually capturing the restrictions is a labor-intensive task that might be infeasible in practice. The second requirement has to do with the fact that the test suite constructed by CTD corresponds to new test cases that need to be implemented. Since the test space in real-life cases is usually enormous, the probability that CTD will produce a test that corresponds to an existing one is close to zero. When

The research leading to these results has received funding from the European Community Seventh Framework Programme [FP7/2007-2013] under grant agreement 257574 (FITTEST).

the cost of implementing a test is high, e.g. when large and specific amounts of data are required for each test, constructing a complete new test suite might be infeasible in practice.

In this work, we propose to use Interaction-based Test-Suite Minimization (ITSM) as a complementary approach to CTD, for cases where standard CTD may be best practice but cannot be applied due to the requirements described above. Rather than constructing a new test suite that provides full interaction coverage, ITSM reduces an existing test suite, while preserving its interaction coverage. Similarly to CTD, ITSM requires defining the parameters of the test space and their values, but it does not require defining restrictions between the values. It is then given a test suite, where each test is in the form of an assignment of values to the parameters, and selects a subset of the test suite that preserves its t-wise value combinations. Clearly, like other test minimization techniques, ITSM is applicable only when there is an existing test suite that is on the one hand extensive and representative enough so that omissions are not a concern, and on the other hand is too large to run to completion and may contain redundant test cases.

Since ITSM is a test suite minimization technique rather than a test design technique, it is not guaranteed to provide full interaction coverage of the test space – any interactions that are missed by the original test suite will clearly be omitted also by the subset selected by ITSM. However, ITSM can be applied without the two potentially problematic requirements of CTD – it does not require defining restrictions, nor implementing new tests. Its result contains only existing tests, and these are presumed to be valid, i.e., they do not violate the (unspecified) test space restrictions. Another point to consider is that while ITSM does not require defining restrictions, it does require translating test cases to combinations of parameter values. When the original test cases are represented in a structured or semi-structured format, the values can be extracted automatically. Otherwise, the translation might be a labor-intensive manual effort.

CTD and ITSM can also be combined to produce an optimized test plan while still reusing existing tests. In this approach, CTD is first applied to produce a complementary test suite that covers the t-wise interactions that are not covered by the existing test suite [5]. The two test suites are then combined, and ITSM is applied to reduce it. This approach is suitable when restrictions can be defined, implementing new tests is costly but possible, and there has been a significant investment in an existing test suite, and thus reusing it is desirable. This paper, however, focuses on the scenario where only ITSM is applied.

We present two real-life case studies that demonstrate the scenario where standard CTD was inapplicable and ITSM was successfully used instead. In the first case study, from the healthcare insurance domain, the restrictions were too complex to specify, and implementation of new tests was highly costly. However, the existence of numerous insurance claims from field usage enabled the usage of ITSM. In the second case study we look at extending a legacy computer terminal interface via web services. The test space in use was represented explicitly test by test, and translating it to restrictions between the test space parameters was too costly. Instead, the original representation was used as input to ITSM. For both case studies, we describe the challenges and requirements we encountered while applying ITSM, such as challenges related to translating existing tests to combinations of parameter values, and non-standard coverage requirements, and introduce our solutions for them.

In addition, we discuss different approaches for solving the ITSM problem. That is, given a test suite in the form of parameter value combinations and interaction coverage requirements, how to select a small subset of the test suite that preserves the same interaction coverage as the entire suite. Though it is possible to reduce an ITSM problem to a CTD problem and apply existing CTD algorithms to solve it, translating a test suite to a CTD model is very costly. Instead, we propose an efficient algorithm for solving ITSM directly.

A well known concern of test suite minimization techniques is that of loss of fault detection. The effectiveness of minimization techniques is measured not only by their reduction power, but also by their ability to maintain the same level of fault detection as the original suite. Since studying the fault detection capabilities of ITSM is left for future work, this is a threat to validity that needs to be considered. Studies that investigate the loss of fault detection concern have conflicting findings. While some studies [13], [12], [7] showed no significant decrease in fault detection effectiveness after test suite minimization, in other empirical studies [8], fault detection capabilities of test suites were severely compromised. The reasoning behind our minimization approach is that most software faults are caused by an interaction between a small number of parameters, as demonstrated in several studies [3], [6], [10]. Therefore, the fault detection effectiveness of the minimized test suite is expected to be similar to that of the original one. However, actual empirical data must be obtained to support these expectations.

The rest of the paper is organized as follows. In Section II we introduce ITSM via a simplified example. Section III presents our algorithm for solving ITSM. In section IV we present in detail two case studies for ITSM, as well as additional results from applying ITSM to various systems. Section V discusses related work. Finally, Section VI draws our conclusions and future research directions.

II. Interaction-Based Test-Suite Minimization

In this section, we introduce the main idea of interaction-based test-suite minimization via a simplified example. Consider testing the claims system of a healthcare insurance provider. Each filed claim consists of information about the patient and the treatment received. Each of these further expands to more detailed information. For example, the patient's information may consist of their age, history (e.g. number of previous insurance claims, and their general health status), type of insurance etc. Similarly, the treatment information may consist of the exact type of treatment given, the geographic

location in which it was given, the type of clinic, the involved personnel, etc.

Table I depicts a highly simplified model for such a system. Table II further lists a pairwise test plan for this model (i.e., CTD with $t = 2$). This test plan consists of 15 tests, and achieves full pairwise coverage of the model. Note that since no restrictions have been defined for the model, all combinations in the model are considered valid.

TABLE I
EXAMPLE HEALTHCARE MODEL

Parameter	Values
Gender	Female, Male
Age	Child, Adult, Elderly
Past claims	None, One, Many
Health	Healthy, Some non-chronic history, Chronic disease
Caregiver	Doctor, Paramedic, Nurse
Treatment	Surgery, Vaccination, Emergency
Location	Home town, Home state, Out-of-state
Clinic	Hospital, Private clinic, Home

In order to implement the test plan suggested in Table II, one needs for each of the rows to either find or generate patients, caregivers, clinics, etc., that exactly match the characteristics defined by the tests. For example, in order to file the claim dictated by the first row, one needs to have in the system a female child with many past claims who is healthy. A claim should then be filed for her name, for getting a surgery by a doctor in an out-of-state private clinic. Such a doctor and a clinic need to also exist in the system. While this may seem possible for this small, almost toy-sized, model – for real-life cases the chances of finding patients, doctors, etc. which exactly match the required characteristics are slim-to-none. Alternatively, one may consider to generate this data in the system specifically for testing purposes. However, the effort of setting up the exact required data is also high and often impractical.

Moreover, the model as it is now contains no restrictions. Therefore some tests suggested by the CTD algorithm are not even valid ones. For example, in the last test in the table, a claim is filed for a nurse performing surgery, a case which should clearly be excluded. For systems with complex business logic, correctly identifying and defining the restrictions is a daunting task which is often infeasible, or at least not cost-worthy.

We therefore propose to use interaction-based test-suite minimization. The idea behind this approach is, given a set of existing tests, to select a subset that maintains the same coverage of value combinations as the existing set (up to a certain user-supplied interaction level, t). In the above healthcare example, one could collect all claims filed in a certain period of time (say, a few months), and select a subset of them to maintain as a test suite. If a long enough period of time is taken, then most interactions of interest would, with high probability, be covered by the existing set, thus will also be covered by the minimized one.

Now consider Table III. It illustrates a possible set of existing tests (picked randomly from the model, for illustration). For our example, let's assume that this table represents the set of claims filed in the last month in the system. Clearly, all these claims are valid ones (i.e. ones that are implementable and executable. After all, these claims were indeed filed in the system recently). Moreover, all patients participating in these tests actually do exist, and similarly, so do the caregivers, clinics, etc. Therefore, it is easy to re-execute each of these tests. On the other hand, this test plan might be too large, and needs to be diluted.

Is has been shown [3], [10] that most software defects are caused by an interaction of a small number of parameters. Therefore, a reasonable criterion for choosing a subset of the tests to maintain is that the same set of interactions up to a small given size are covered. For example, one may require that all pairs of values that appear in the large set also appear in the selected one. As mentioned above, since the existing tests represent the set of claims filed in the system during a relatively long period, it is reasonable to believe that most interesting interactions of parameters are covered by this set of tests. Therefore the sacrifice in coverage resulting from applying ITSM, as opposed to "standard" CTD, is small. Moreover, some of the interactions that do not appear in the existing tests are such that are in fact invalid. Our approach thus removes the need of explicitly specifying the invalid combinations (i.e., restrictions).

Getting back to our example, the set of tests in Table III consists of 50 tests. These tests cover 222 out of the 231 pairs of values in the model. Some of these pairs are uncovered since they are in fact invalid (e.g., a surgery performed at home, or by a nurse), while others are due to the fact that no claim including them has been filed during the collection period (e.g., an adult having surgery). The test plan in table IV gives the result of running the ITSM algorithm on this input, with pairwise coverage requirement. The table consists of only 22 tests, all chosen from the given 50, which achieves the same pairwise coverage as the original 50.

III. IMPLEMENTATION

We now discuss the implementation of ITSM. We refer to the value combinations that are to be covered as *coverage targets*. Coverage targets may be given in different forms, such as a Cartesian product (e.g., "every combination of size t", just as in standard CTD), or as explicit sets of value combinations to be covered. We say that a test t covers a coverage requirement c if the values specified by c are used in t.

Given a suite of tests $T = t_i, i = 1 \dots n$, a set of coverage targets $C = c_j, j = 1 \dots m$, and a mapping $M : T \rightarrow 2^C$ that specifies the coverage targets that are covered by each test in T, the objective of an interaction-based test-suite minimization algorithm is to find $S = s_i, i = 1 \dots k$, a subset of T that covers all the targets that T covers – $S \subseteq T s.t. \cup_{i=1 \dots n} M(t_i) = \cup_{i=1 \dots k} M(s_i)$.

TABLE II
PAIRWISE TEST PLAN FOR THE EXAMPLE HEALTHCARE MODEL

Gender	Age	PastClaims	Health	CareGiver	Treatment	Location	Clinic
Female	Child	Many	Healthy	Doctor	Surgery	OutOfState	PrivateClinic
Male	Adult	None	SomeNonChronicHistory	Doctor	Vaccination	HomeTown	Home
Male	Child	One	ChronicDisease	Nurse	Emergency	HomeState	Hospital
Female	Elderly	None	ChronicDisease	Paramedic	Surgery	OutOfState	Home
Female	Elderly	Many	Healthy	Paramedic	Emergency	HomeTown	Hospital
Male	Elderly	One	SomeNonChronicHistory	Doctor	Emergency	OutOfState	PrivateClinic
Female	Elderly	None	Healthy	Nurse	Vaccination	OutOfState	Hospital
Male	Adult	One	Healthy	Nurse	Surgery	HomeState	Home
Male	Elderly	Many	SomeNonChronicHistory	Paramedic	Vaccination	HomeState	PrivateClinic
Female	Adult	None	SomeNonChronicHistory	Nurse	Emergency	OutOfState	PrivateClinic
Female	Adult	None	SomeNonChronicHistory	Doctor	Surgery	HomeState	Hospital
Male	Child	None	SomeNonChronicHistory	Paramedic	Emergency	HomeTown	Home
Female	Adult	One	ChronicDisease	Paramedic	Vaccination	HomeTown	PrivateClinic
Male	Child	One	ChronicDisease	Doctor	Vaccination	OutOfState	Home
Male	Adult	Many	ChronicDisease	Nurse	Surgery	HomeTown	Home

A naïve approach to implementing test suite minimization would be by reduction to CTD, where the set of tests that the CTD algorithm may choose from is limited to the input tests. Exact implementation details depend on the CTD algorithm used, but common to all is the need to represent the set of possible tests (or the complementary set, of excluded tests). Unfortunately, compact representations that are very efficient for CTD turn out to be inefficient for test suite minimization. For example, [9] presents a BDD-based algorithm for CTD. In order to use it for ITSM, one needs to capture the set of existing tests as a BDD, e.g., by disjuncting the BDDs representing the individual tests. BDDs are an effective data structure for representing *structured* sets, i.e. ones for which the characteristic formula is relatively simple. Since existing tests can very rarely be characterized by a simple formula, this representation becomes very inefficient for large sets of existing tests. Alternatively, one could try to capture the complementary set, of excluded test, using standard restriction notations. This is typically also an infeasible approach.

To mitigate these problems, we describe a fast algorithm that uses a low overhead data structure.

First, consider the following simple and greedy algorithm:

- For $i = 1 \ldots n$ if t_i covers a target that is not yet covered by S then add t_i to S.

It is easy to see that at the end of the loop, S covers all the targets that T covers. However, it is also easy to see that S is not always the best solution. For example, if the test suite has two tests, the first of which covering one target and the second covering the same target and another one, this algorithm will select both tests, whereas the second test suffices.

This algorithm visits each test in T at most once. Its time complexity is $O(|T| \cdot |C|)$.

The following algorithm is less greedy, produces better results, but works harder:

- While S covers less than T, add to S the test that covers the most targets that are covered by T but not yet covered by S.

This algorithm also computes a correct result, that is, in the end S covers the same targets as T does. Its result is not optimal, but it works well for the example above.

This algorithm visits $n - i$ tests in its i-th iteration, hence its time complexity is $O(|T| \cdot |C| \cdot |S|)$. In the worst case, $|S| = |T|$, but in practice $|S|$ is frequently orders of magnitude less than $|T|$.

We next explore three ways to improve this algorithm. Two reduce the constant factor in the O( ) complexity expression of this algorithm, and one results in a smaller output test suite.

- Avoiding unnecessary calculations
- Test prioritization
- Counting uncovered targets

1) Avoiding unnecessary calculations: We introduce an improvement that significantly reduces the number of times that uncovered targets are counted.

Note that the number of uncovered targets that a test can contribute in one iteration is never higher than the number it could contribute in a previous iteration. Hence, if the current iteration has already found a test that contributes $maxSoFar$ new targets, then the $O(|C|)$ process of counting uncovered targets for any test that could contribute less than $maxSoFar$ in previous iterations can be skipped. This is illustrated in Algorithm 1, where $test_i.prevCount$ is the latest computed contribution for $test_i$. Note that this value is not necessarily computed in every iteration.

```
1 if test_i.prevCount > maxSoFar then
2 |   Compute count, the number of uncovered
  |   targets that test_i covers
3 |   test_i.prevCount ← count
4 |   if count > maxSoFar then
5 |   |   best ← test_i
6 |   |   maxSoFar ← count
7 |   end
8 end
```

Algorithm 1: Skipping unnecessary counts

TABLE III
TEST PLAN FOR THE EXAMPLE HEALTHCARE MODEL, CONSISTING OF 50 EXISTING TESTS

Gender	Age	PastClaims	Health	CareGiver	Treatment	Location	Clinic
Female	Child	None	ChronicDisease	Paramedic	Emergency	HomeTown	Hospital
Female	Elderly	One	Healthy	Nurse	Vaccination	OutOfState	PrivateClinic
Male	Child	Many	Healthy	Paramedic	Emergency	OutOfState	Hospital
Female	Adult	Many	ChronicDisease	Paramedic	Emergency	OutOfState	Home
Male	Child	None	SomeNonChronicHistory	Nurse	Emergency	HomeState	Hospital
Male	Adult	Many	SomeNonChronicHistory	Paramedic	Emergency	OutOfState	Home
Male	Elderly	One	ChronicDisease	Paramedic	Emergency	HomeState	Home
Female	Child	None	Healthy	Nurse	Vaccination	HomeTown	Home
Male	Adult	One	Healthy	Nurse	Vaccination	HomeTown	PrivateClinic
Female	Adult	Many	Healthy	Nurse	Emergency	OutOfState	Home
Male	Elderly	None	ChronicDisease	Doctor	Surgery	OutOfState	Hospital
Male	Elderly	One	SomeNonChronicHistory	Doctor	Surgery	OutOfState	PrivateClinic
Male	Adult	Many	SomeNonChronicHistory	Doctor	Emergency	HomeState	Home
Male	Elderly	Many	Healthy	Nurse	Emergency	HomeState	Hospital
Female	Elderly	Many	ChronicDisease	Doctor	Surgery	OutOfState	PrivateClinic
Male	Child	Many	ChronicDisease	Nurse	Emergency	OutOfState	Home
Female	Adult	One	ChronicDisease	Nurse	Vaccination	HomeTown	Hospital
Male	Elderly	Many	ChronicDisease	Nurse	Vaccination	OutOfState	Hospital
Male	Adult	None	Healthy	Nurse	Emergency	HomeTown	PrivateClinic
Male	Elderly	Many	ChronicDisease	Doctor	Surgery	OutOfState	Hospital
Male	Child	Many	ChronicDisease	Doctor	Surgery	OutOfState	PrivateClinic
Male	Elderly	None	Healthy	Paramedic	Emergency	OutOfState	Home
Male	Elderly	One	SomeNonChronicHistory	Nurse	Emergency	HomeTown	PrivateClinic
Female	Elderly	Many	SomeNonChronicHistory	Nurse	Emergency	OutOfState	Hospital
Male	Elderly	One	ChronicDisease	Paramedic	Emergency	HomeTown	PrivateClinic
Female	Elderly	Many	ChronicDisease	Paramedic	Emergency	OutOfState	Home
Male	Adult	Many	ChronicDisease	Paramedic	Emergency	HomeState	Home
Female	Elderly	None	ChronicDisease	Nurse	Vaccination	OutOfState	Home
Male	Child	None	SomeNonChronicHistory	Paramedic	Emergency	HomeState	Home
Male	Elderly	None	SomeNonChronicHistory	Doctor	Emergency	OutOfState	Home
Female	Elderly	Many	ChronicDisease	Paramedic	Emergency	OutOfState	Home
Male	Adult	Many	Healthy	Paramedic	Emergency	OutOfState	Hospital
Female	Child	Many	Healthy	Doctor	Surgery	OutOfState	PrivateClinic
Female	Adult	None	ChronicDisease	Paramedic	Emergency	HomeTown	Home
Female	Child	Many	ChronicDisease	Nurse	Vaccination	HomeState	Home
Female	Elderly	Many	ChronicDisease	Nurse	Emergency	OutOfState	PrivateClinic
Male	Adult	One	SomeNonChronicHistory	Paramedic	Emergency	OutOfState	Hospital
Female	Elderly	None	ChronicDisease	Doctor	Emergency	HomeState	PrivateClinic
Male	Child	None	Healthy	Nurse	Vaccination	OutOfState	Home
Female	Child	None	SomeNonChronicHistory	Paramedic	Emergency	HomeTown	PrivateClinic
Male	Elderly	Many	SomeNonChronicHistory	Doctor	Emergency	HomeTown	Hospital
Male	Elderly	One	Healthy	Paramedic	Emergency	OutOfState	Home
Male	Child	One	ChronicDisease	Paramedic	Emergency	HomeTown	PrivateClinic
Male	Adult	One	ChronicDisease	Nurse	Vaccination	HomeTown	Home
Male	Elderly	One	Healthy	Paramedic	Emergency	OutOfState	Home
Male	Elderly	Many	ChronicDisease	Doctor	Emergency	OutOfState	Hospital
Male	Elderly	None	SomeNonChronicHistory	Nurse	Emergency	HomeState	Hospital
Male	Elderly	Many	Healthy	Paramedic	Emergency	HomeTown	PrivateClinic
Male	Elderly	One	Healthy	Paramedic	Emergency	OutOfState	Home
Male	Child	Many	ChronicDisease	Nurse	Emergency	HomeState	Home

Table V shows that this improvement provides a significant speedup. We show results for three data sets. The second column (Tests) shows the number of initial tests per data set. The third column (Targets) shows the number of targets that are to be covered. The next two columns (CountsWithout-Skipping and TimeWithoutSkipping) show the total number of times that a test's possible contribution to coverage is computed and the total execution time (in seconds) without the above improvement, respectively. The last two columns show the same where the improvement is used. For large data sets the improved algorithm is about 32 times faster than the straightforward solution.

2) Test prioritization: Changing the priority of selecting tests results in a smaller output test suite.

So far, we only considered the number of uncovered targets in preferring one test over another. Typically, this results in the algorithm ending with many iterations selecting tests that contribute only one target. We experimented with several weighting schemes that give a higher weight to targets that appear less in the input, and preferring higher weight tests to lower ones.

TABLE IV
ITSM RESULT FOR THE EXAMPLE HEALTHCARE MODEL, CONSISTING OF 22 OUT OF THE 50 EXISTING TESTS

Gender	Age	PastClaims	Health	CareGiver	Treatment	Location	Clinic
Female	Child	Many	Healthy	Doctor	Surgery	OutOfState	PrivateClinic
Male	Child	None	SomeNonChronicHistory	Paramedic	Emergency	HomeState	Home
Female	Adult	One	ChronicDisease	Nurse	Vaccination	HomeTown	Hospital
Male	Elderly	None	ChronicDisease	Doctor	Surgery	OutOfState	Hospital
Male	Elderly	One	Healthy	Paramedic	Emergency	OutOfState	Home
Female	Child	None	SomeNonChronicHistory	Paramedic	Emergency	HomeTown	PrivateClinic
Male	Elderly	Many	Healthy	Nurse	Emergency	HomeState	Hospital
Male	Adult	Many	SomeNonChronicHistory	Doctor	Emergency	HomeState	Home
Male	Adult	None	Healthy	Nurse	Emergency	HomeTown	PrivateClinic
Female	Child	Many	ChronicDisease	Nurse	Vaccination	HomeState	Home
Female	Elderly	Many	ChronicDisease	Nurse	Emergency	OutOfState	PrivateClinic
Male	Elderly	One	SomeNonChronicHistory	Doctor	Surgery	OutOfState	PrivateClinic
Female	Adult	Many	ChronicDisease	Paramedic	Emergency	OutOfState	Home
Male	Elderly	Many	SomeNonChronicHistory	Doctor	Emergency	HomeTown	Hospital
Female	Child	None	Healthy	Nurse	Vaccination	HomeTown	Home
Female	Elderly	None	ChronicDisease	Doctor	Emergency	HomeState	PrivateClinic
Male	Elderly	One	ChronicDisease	Paramedic	Emergency	HomeState	Home
Male	Child	None	SomeNonChronicHistory	Nurse	Emergency	HomeState	Hospital
Male	Adult	One	SomeNonChronicHistory	Paramedic	Emergency	OutOfState	Hospital
Female	Elderly	One	Healthy	Nurse	Vaccination	OutOfState	PrivateClinic
Male	Child	One	ChronicDisease	Paramedic	Emergency	HomeTown	PrivateClinic
Male	Elderly	Many	ChronicDisease	Nurse	Vaccination	OutOfState	Hospital

TABLE V
EFFECT OF THE COUNT SKIPPING IMPROVEMENT

DataSet	Tests	Targets	CountsWithoutSkipping	TimeWithoutSkipping	CountsWithSkipping	TimeWithSkipping
1	539	9,904	126,374	0.516	5,321	0.141
2	280,884	4,378	27,195,508	96.502	1,661,447	3.016
3	4,008,037	2,392	1,250,607,544	618.873	18,455,267	19.331

The intuition behind this approach is that when such weights are used, the first iterations select tests that cover many hard to find targets, and the last iterations easily find many easy to find targets.

Indeed, using weights may reduce the size of the selected suite by up to 15%.

3) Counting uncovered targets: Finally, ITSM can be speeded up by performing several bit operations at a time.

We maintain a mapping from each test to the targets that it covers. Covered targets are represented by bits in integer variables, which we call *elements*, and counting the number of set bits in a bitmap is done using a lookup table for an element at a time. We get a significant speedup by using 16 bit elements - a `short int` in C/C++, or a `char` in Java. Such bitmaps are associated with each input test, representing the targets that each test covers, and are also used for intermediate results, representing the targets that still have to be covered.

Algorithm 2 shows counting the number of uncovered targets that $test_i$ contributes. $test_i.covered_j$ is the j-th element in the bitmap that describes the targets covered by $test_i$, $bitCount[]$ is a lookup table that is initialized to the count of set bits in every possible element, and $count$ is the number of uncovered targets that $test_i$ can contribute.

This speed up can be combined with the priority criterion above by using $nElementsInBitmap$ $bitCount[]$ tables, i.e., using $bitCount_j[new]$ instead of $bitCount[new]$ in line 4 of Algorithm 2.

```
1 count ← 0
2 for j = 0 to nElementsInBitmap do
3 |   new ← test_i.covered_j & uncovered_j
4 |   delta ← bitCount[new]
5 |   count ← count + delta
6 end
```

Algorithm 2: Counting uncovered targets

In Section IV, and in Table VI in particular, we present results that compare variants of the proposed algorithm, with and without the improvements suggested above.

IV. EVALUATION

Interaction-based test-suite minimization has been applied in several real-life cases within and outside of IBM. While clearly we cannot share the exact details of these applications, we do present two cases in some detail here as case studies. We also give some results from several other applications.

A. Case Study – Healthcare

The first case study is in fact the real-life version of the example in Section II. The system under test was a claims processing system of several large healthcare providers. The model for testing this system consists of 59 parameters, some with only a few values, and some with hundreds. For this case, both requirements presented in Section I are problematic. First,

the business logic in the system is very complex, therefore it is practically impossible to correctly capture all the restrictions between values in the model. Moreover, even if one could specify these restrictions, generation of a test that corresponds to an arbitrary combination of values to the parameters (even if valid) requires generating all the corresponding data and is therefore a highly laborious task that was infeasible in this case. Due to these limitations, we opted for applying ITSM.

Since many of the parameters in this case study have a very large number of values which cannot be abstracted in a reasonable way, the standard practice of specifying coverage requirements as t-way coverage of all parameters is infeasible, since it will require too many tests in order to be fully satisfied. Therefore, the following two types of coverage requirements were used: a) explicit values and combinations of interest which were defined manually; b) requirements such as "cover the pairs of values for parameters A and B that correspond to 90% of the claims". A preprocessing step translated requirements of the latter type into concrete requirements. The reasoning behind the latter type is that for huge amounts of data, such as the ones dealt with in this case study, it is often very hard to manually consider all combinations of values for parameters of interest. On the other hand, typically a small number of such combinations cover the most common cases, so by requiring to cover a certain top percent of the claims, one can with small effort make sure that the most frequently used combinations are covered, with a relatively small number of tests.

Table VI summarizes the reductions achieved for seven different healthcare providers. In all of them, claims were collected over a certain period of time, and the ITSM technology was used in order to select a subset of them that fully covers all the interactions that are defined as required and are covered by the complete sets. The table shows the number of selected results for three variants of the greedy algorithm: a) the most naïve greedy algorithm (referred to as "Simple Greedy" in the table), b) the less greedy algorithm, without improvements (referred to as "Uniform"), and c) the improved algorithm, including test prioritization ("Prioritized"). The magnitude of reduction in all cases was from hundreds of thousands, and even millions of claims (four million for the largest case) to around eight-hundred with the most naïve algorithm and around three-hundred for the prioritized one. Computation time for the prioritized algorithm is about 0.3 seconds per 100,000 input tests on a standard PC.

TABLE VI
ITSM RESULTS FOR SEVEN HEALTHCARE PROVIDERS

Total # Claims	# Claims After ITSM		
	Simple Greedy	Uniform	Prioritized
280,885	709	311	270
374,102	830	352	304
438,560	736	342	296
572,519	814	364	313
1,250,549	820	343	297
3,984,877	945	329	289
4,008,088	857	311	273

Note that some preprocessing had to be performed in order to translate the claims into parameters and values. One example of a type of preprocessing operations is that of abstraction, such as merging several different concrete values into the more abstract one, e.g. replacing the patient's concrete age with child/adult/elderly. Another example is table-based replacements, e.g., looking up a caregiver in a table and replacing her code with the relevant parameter values.

As mentioned above, the input to ITSM is a set of claims collected over a relatively long period of time. The reasoning behind using ITSM for this case is that over such a period, most relevant values and interactions of values are used at least once, and will therefore be covered by the selected set. In other words, in this case the potential coverage omissions, as opposed to using CTD (which would have guaranteed full coverage), are not a big concern, especially given the significantly higher cost that CTD would have demanded, which was unacceptable.

B. Case Study – Interface Panels

In the second case study a legacy computer terminal interface was extended via web services to work in a modern web browser environment. Because implementation was done in a fashion that made use of the existing legacy GUI panel definitions, the original requirement for testing was to verify every existing legacy panel. With about 19,500 panels, this testing requirement was unrealistic, especially since verification required visual inspection of the resulting web page.

The most unique aspect of this example was that the 19,500 panels represented 100% of the test combinations. Thus we had an existing test suite that was guaranteed to exercise all necessary combinations. Because of this, the need to fully enumerate all restrictions, as required in traditional CTD, was not necessary. We relied on the existing test suite to naturally reflect the restrictions. Having this set of all actually used test combinations also removed any incentive to write additional tests. In light of these two factors, the test challenge became one of test suite minimization. One more consideration made the example a good candidate for ITSM. The panels were well suited for automation to be represented as tuples of parameter-values.

The model that was built consisted of 86 parameters, with a Cartesian product test space size of 4.72E21 combinations. Amazingly, traditional pairwise analysis (with no restrictions) reduced this to only 13 tests due to the fact that most parameters were Boolean, having only 2 values.

Similar to the first example, existing tests were available as several subsets. Each subset corresponded to the panels from a different product that made use of the terminal interface functionality. Table VII demonstrates the reduction achieved for each of the sets. In this example, the analysis was also taken one step further by performing an ITSM reduction on the test suite combining all panels. The result of that analysis appears as the last entry in the table. The table also shows the percentage of all pairwise values that appear in the test suite for each product. As expected, we see that a higher percentage

of coverage is achieved by the combined test suite. In addition, an analysis of the tests selected in the combined case shows that panels from many different products were chosen.

TABLE VII
ITSM RESULTS FOR THE INTERFACE PANELS CASE STUDY

Product ID	# Panels	% Covered	# Selected
1	343	58.1	25
2	77	30.4	3
3	1132	61.1	27
4	48	50.2	10
5	22	30.4	3
6	204	44.8	14
7	4866	60.4	30
8	1065	58.8	32
9	39	32.5	6
10	532	44.4	12
11	213	36.1	13
12	3428	89.7	87
13	43	39.6	10
14	86	35.2	7
15	1132	43.4	16
16	2280	42.4	15
17	1177	41.2	19
18	90	34.5	7
19	2759	39.6	12
ALL	19536	91.2	92

As a side effect of the combined analysis we see a clue about the number of restrictions that are associated with this data. From the high coverage percentage of 91.2% we can assume there are relatively few restrictions. Many restrictions would normally reduce the number of valid combinations as compared to the Cartesian product size to a percentage lower than the 90% plus that we observe here.

While pairwise coverage could have been accomplished with only 13 tests, we had no idea if those 13 violated the unknown restrictions. The higher number of 92 tests selected using ITSM were known not to violate restrictions. This gives us an idea of the tradeoff in effort associated with the technique. With the analytical pairwise optimization offered by traditional CTD missing, a higher number of natural test scenarios were required to achieve the same pairwise coverage. In the end, while more tests had to be run, ITSM still represented less effort than would have been required to discover all of the restrictions associated with the model and to code 13 new tests from scratch.

C. Additional Results

We also applied ITSM to several other projects from financial and health-insurance domains. The charts in Figure 1 summarize the results. Each chart depicts the original number of tests vs. the number of tests selected by ITSM. On average, ITSM reduced over 70% of the tests. The level of interaction coverage obtained by the test plans is also plotted. As expected, the coverage is identical before and after the reduction.

Finally, we revisit the claim from Section III, that the naïve approach to ITSM, of reduction to a CTD problem, does not scale. In Figure 2 we present computation time for solving

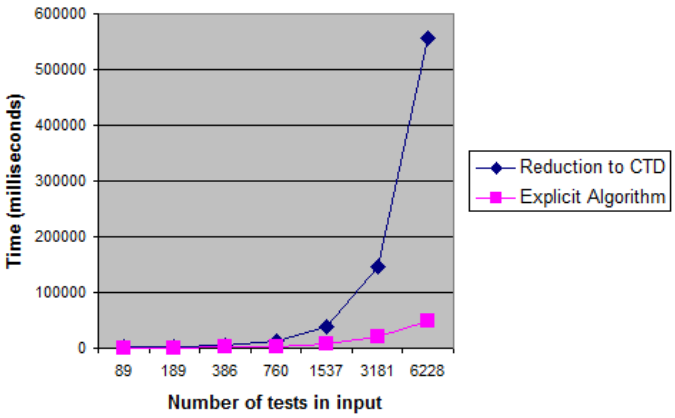

Fig. 2. Runtime of the prioritized algorithm vs reduction to standard CTD

ITSM, comparing between reduction to the BDD-based CTD algorithm presented in [9] and the algorithm presented here. It is easy to see that the BDD-based algorithm explodes at problems that are orders of magnitude smaller than those deemed problematic for the approach presented here. The inputs in this experiment are random subsets of the test suites mentioned in Section IV-A, and the coverage requirements are to cover all values, and for half of the parameters, all pairs of values. Similar behavior was observed on other input sets and other coverage requirements.

V. Related Work

Test suite minimization is a well-studied and widely used approach for increasing the efficiency of regression test suites, that tend to grow over time. This approach reduces the size of test suites by eliminating redundant test cases, according to some criteria, where the most common criterion is achieving the same code coverage as the original test suite. Problems related to test suite minimization are those of test case selection and test case prioritization. In test case selection, following a set of changes in the system under test, relevant test cases to run are identified. In test case prioritization, the test cases are ordered in a way that is intended to achieve early fault detection. A recent survey [14] studies these three problems and lists 159 related papers.

For test minimization, most techniques described in the survey use either general black-box requirements as the criterion, or structural coverage-level criteria. However, there are some other criteria described in the paper such as operational abstraction, model-based minimization and more. In our work, we choose to focus on interactions between values as the minimization criterion, motivated by the studies such as [3], [6], [10] that show that most software defects are caused by an interaction of a small number of parameters. The survey in [14] also states that only 8% of the proposed techniques were applied on industrial-scale examples. As mentioned above, in this paper we describe two industrial case studies, and survey some results from other cases, all from real-life industrial applications.

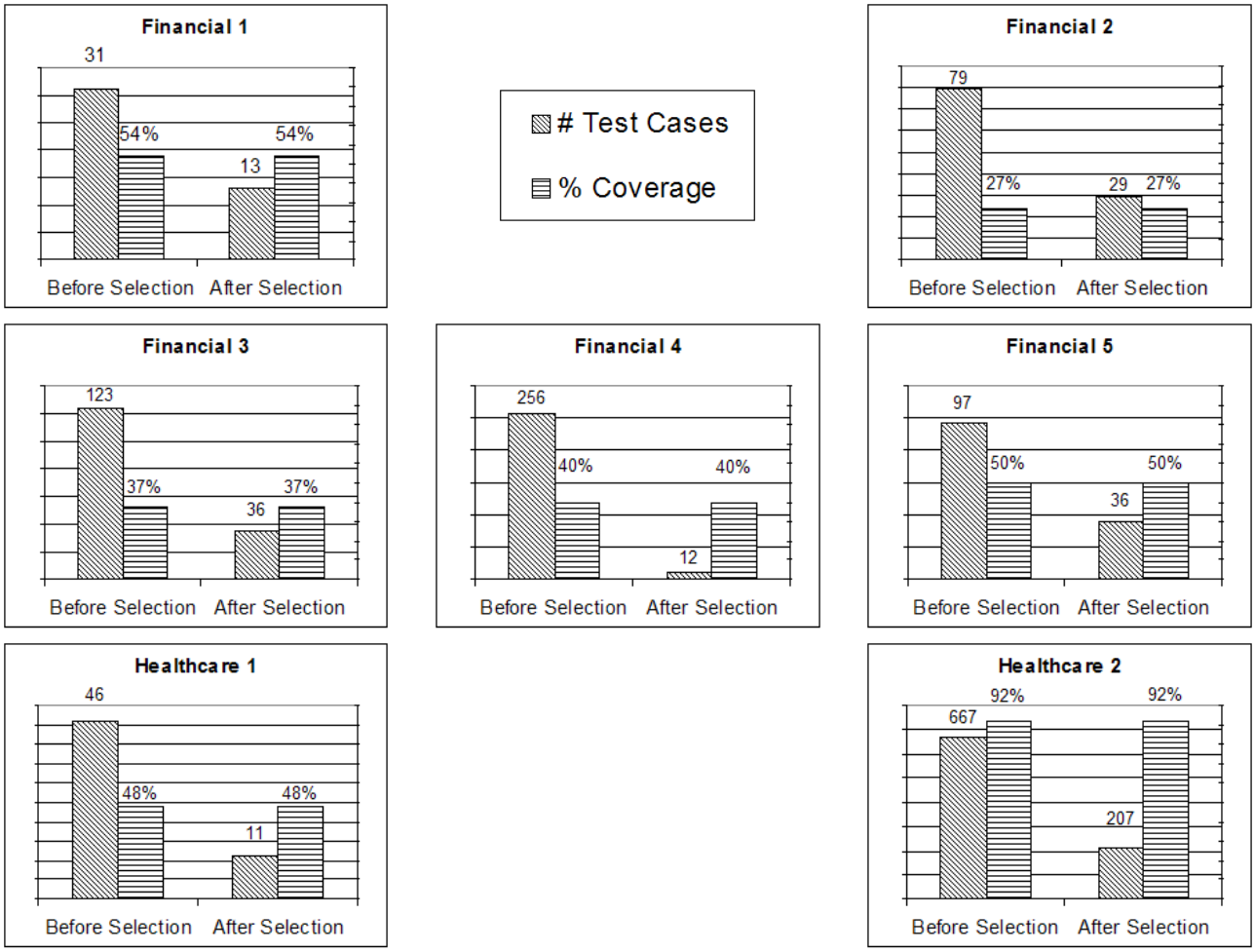

Fig. 1. Test suite minimization results for seven different real-life projects

Interaction-based test-suite minimization was introduced in [5]. The test minimization problem defined there receives an input test suite that already has 100 percent t-wise coverage. In addition, all value combinations must be considered valid by the test space, which means that all t-wise tuples between the set of parameters must appear in the input test suite. In reality, such test suites rarely exist, which significantly limits the application of this minimization approach. Indeed, no case studies were presented in [5]. We redefine the ITSM problem as a minimization of an arbitrary set of tests, while allowing any relationships between the different parameters of the test space. We thus turn ITSM into a widely applicable approach, as demonstrated by the case studies that we present.

VI. Conclusions and Future Work

In this paper we present Interaction-based Test-Suite Minimization (ITSM) – a test suite minimization approach that maintains the same coverage of value combinations as the existing set (up to a certain user-supplied interaction level, t). The approach is backed by an efficient and effective algorithm, and evaluated using two case studies and further experimental results.

In the future, we plan to further validate the approach by studying actual fault detection reports and fault escape reports, in order to substantiate the motivating claim regarding fault discovery effectiveness. It is often impractical to actually run the original suite and compare its fault detection to that of the minimized one, therefore we plan to also take into consideration escapes to the field.

Another interesting future direction is that of multiple test suites. In the second case study, presented in Section IV-B above, test suites for multiple products were considered, both separately and in conjunction. In the latter case, where all tests were considered as one test suite to be minimized, we observed that our minimization technique indeed considered tests from many different products. However, the distribution between them was not even, and some products were completely left out. In the future, we plan to extend our algorithm to better reflect the distribution of tests from multiple separate sources.

Interaction-based test-suite augmentation, or enhancement, is the process of adding tests to the existing test suite in order to achieve full interaction coverage [5]. While it "suffers" from the same requirements as CTD (need for precise restrictions, and for generating tests), it does allow users to reuse their

existing tests and reduce the number of new tests to be generated (typically at the price of a larger final test suite). We plan in the future to compare the effectiveness of the three approaches (CTD, ITSM and enhancement) in terms of test suite sizes, fault detection and costs, as well as to explore possibilities of combining these approaches.

Finally, ITSM requires that the set of existing tests be represented as tuples of values to parameters. Often, however, test suites are written in much less structured form, from relatively structured spreadsheets to free text. Automatic, or semi-automatic, tool support for translation of tests from such unstructured forms into tuples of parameter values would be significant for the applicability of the ITSM approach.

REFERENCES

[1] K. Burroughs, A. Jain, and R.L. Erickson. Improved quality of protocol testing through techniques of experimental design. In *IEEE International Conference on Record, Serving Humanity Through Communications*, volume 2, pages 745–752, 1994.

[2] M. B. Cohen, J. Snyder, and G. Rothermel. Testing across configurations: implications for combinatorial testing. *SIGSOFT Softw. Eng. Notes*, 31(6):1–9, 2006.

[3] S. R. Dalal, A. Jain, N. Karunanithi, J. M. Leaton, C. M. Lott, G. C. Patton, and B. M. Horowitz. Model-Based Testing in Practice. In *Proc. 21st International Conference on Software Engineering (ICSE'99)*, pages 285–294. ACM, 1999.

[4] M. Grindal, B. Lindström, J. Offutt, and S. F. Andler. An evaluation of combination strategies for test case selection. *Empirical Softw. Engg.*, 11(4):583–611, 2006.

[5] A. Hartman and L. Raskin. Problems and Algorithms for Covering Arrays. *Discrete Mathematics*, 284(1-3):149–156, 2004.

[6] D. R. Kuhn, D. R. Wallace, and A. M. Gallo. Software Fault Interactions and Implications for Software Testing. *IEEE Transactions on Software Engineering*, 30:418–421, 2004.

[7] G. Rothermel, M. J. Harrold, J. von Ronne, and C. Hong. Empirical studies of test-suite reduction. *Journal of Software Testing, Verification, and Reliability*, 12:219–249, 2002.

[8] G. Rothermel, M.J. Harrold, J. Ostrin, and C. Hong. An empirical study of the effects of minimization on the fault detection capabilities of test suites. In *Software Maintenance, 1998. Proceedings., International Conference on*, pages 34–43, 1998.

[9] I. Segall, R. Tzoref-Brill, and E. Farchi. Using Binary Decision Diagrams for Combinatorial Test Design. In *Proc. 20th Intl. Symp. on Software Testing and Analysis (ISSTA'11)*, pages 254–264. ACM, 2011.

[10] K.C. Tai and Y. Lie. A Test Generation Strategy for Pairwise Testing. *IEEE Transactions on Software Engineering*, 28:109–111, 2002.

[11] A. W. Williams. Determination of test configurations for pair-wise interaction coverage. In *Proceedings of the IFIP TC6/WG6.1 13th International Conference on Testing Communicating Systems: Tools and Techniques*, TestCom '00, pages 59–74, 2000.

[12] E. Wong, J. R. Horgan, A. P. Mathur, and A. Pasquini. Test set size minimization and fault detection effectiveness: A case study in a space application. In *In Proceedings of the 21st Annual International Computer Software and Applications Conference*, pages 522–528, 1997.

[13] W. E. Wong, J. R. Horgan, S. London, and A. P. Mathur. Effect of test set minimization on fault detection effectiveness. In *Proceedings of the 17th international conference on Software engineering*, ICSE '95, pages 41–50, 1995.

[14] S. Yoo and M. Harman. Regression testing minimization, selection and prioritization: a survey. *Software Testing, Verification and Reliability*, 22(2):67–120, 2012.