Visualization of Combinatorial Models and Test Plans

Rachel Tzoref-Brill
School of Computer Science,
Tel Aviv University
and IBM Research, Israel

Paul Wojciak
IBM Systems, USA

Shahar Maoz
School of Computer Science,
Tel Aviv University, Israel

ABSTRACT

Combinatorial test design (CTD) is an effective and widely used test design technique. CTD provides automatic test plan generation, but it requires a manual definition of the test space in the form of a combinatorial model. One challenge for successful application of CTD in practice relates to this manual model definition and maintenance process. Another challenge relates to the comprehension and use of the test plan generated by CTD for prioritization purposes.

In this work we introduce the use of visualizations as a means to address these challenges. We apply three different forms of visualization, matrices, graphs, and treemaps, to visualize the relationships between the different elements of the model, and to visualize the strength of each test in the test plan and the relationships between the different tests in terms of combinatorial coverage. We evaluate our visualizations via a user survey with 19 CTD practitioners, as well as via two industrial projects in which our visualization was used and allowed test designers to get vital insight into their models and into the coverage provided through CTD generated test plans.

CCS Concepts

•Software and its engineering → Software testing and debugging; •Human-centered computing → Visualization;

Keywords

Combinatorial Testing, Software Visualization

1. INTRODUCTION

Combinatorial Test Design (CTD), a.k.a. combinatorial testing, see, e.g., [2, 4, 6, 7, 12, 24, 25], is a widely used test design methodology and is supported by many academic and commercial tools, e.g., [3,5,10,15,18]. Close to 40 CTD tools are listed in [17]. CTD is based on a *combinatorial model*, consisting of a set of parameters, their respective values, and constraints on the value combinations. A valid test in the test space is defined to be an assignment of one value to each parameter that satisfies the constraints. A CTD algorithm automatically constructs a *test plan*, consisting of a (small) subset of the set of valid tests that covers all valid value combinations of every t parameters, where t is usually a user input. The effectiveness of CTD is based on empirical data that shows that in most cases, the occurrence of a bug depends on the interaction between a small number of features of the system under test [6, 13, 21].

Successful application of CTD in practice is, however, challenging. One challenge relates to the manual process of defining combinatorial models and maintaining them, i.e., as the system under test evolves, since it requires both domain knowledge of the system under test and testing expertise. Another challenge relates to the comprehension and use of the test plan. Although a test plan is in many cases automatically generated from the model, understanding its tests, the relations between them, and their relation to the model in terms of its t-way coverage requirements, e.g., in order to identify the more important tests and prioritize them or in order to gain higher confidence in using them, is difficult.

In this work we introduce the use of visualizations as a means to address these challenges. Specifically, we apply three different forms of visualization – matrices, graphs, and treemaps – to visualize the relationships between the different elements of the model, such as parameters, constraints, and parameter combinations, and to visualize the strength of each test in the test plan in terms of t-way coverage, both globally and with respect to every other test in the test plan.

Our work follows the classic overview first, zoom and filter, details-on-demand paradigm [20] in a number of ways. The parameter matrix and the treemap allow the engineer to drill-down from the parameter combinations level to the value combinations level. The graphs allow the engineer to focus on a selected node and explore its relations with the other nodes in the graph. The treemap for the test plan enables filtering by parameters, to allow the engineer to focus on parameters of higher importance. These features of interactive exploration are an important aspect of our work.

We have implemented all the proposed visualizations as part of IBM Functional Coverage Unified Solution (IBM FOCUS), an industry-strength CTD tool [9, 18], developed by IBM Research. All screenshots in the paper are taken from IBM FOCUS. As color plays an important role in our visualizations, we directly refer to colors in some of the views and related screenshots.

We further report on an evaluation consisting of two parts:

Table 1: Example online shopping model

Parameter	Values
ItemStatus (IS)	InStock, OutOfStock, NoSuchProduct
ExportControl (EC)	Yes, No
ShippingDestination (SD)	Domestic, Foreign
PricingScheme (PS)	Scheme1, Scheme2, Scheme3
DeliveryTimeframe (DT)	Immediate, OneWeek, OneMonth
OrderShipping (OS)	Ground, Sea, Air

Constraints
DT = Immediate → OS = Air
OS = Sea → DT = OneMonth
DT = OneMonth → OS ≠ Air
SD = Foreign → PS ≠ Scheme3
SD = Domestic → OS ≠ Sea

a user survey with 19 CTD practitioners, and two real-world industrial projects in which our visualizations were used. The evaluation shows that the visualizations were well-received by IBM FOCUS users who participated in the survey, and allowed the test designers in the studied projects to get vital insight into their models and into the coverage provided through the CTD generated test plans. Thus the visualizations helped them better tune, trust, and ultimately improve their test design and execution approach.

The next section presents a running example of a combinatorial model and two test plans that we will use throughout the paper. Section 3 presents the visualizations. Section 4 presents the data structures and algorithms we use as well as an overview of our implementation. Section 5 presents the evaluation. Section 6 discusses related work and Section 7 concludes.

2. RUNNING EXAMPLE

We use a running example shown in Table 1, both to demonstrate the basic concepts of CTD, and to demonstrate our various views in Section 3. The table depicts the parameters, values, and constraints of a combinatorial model for an online shopping system.

The constraints define which value combinations are invalid in the test space. For example, the first constraint in the online shopping model defines the following invalid combinations: (DT = Immediate, OS = Ground) and (DT = Immediate, OS = Sea). A valid test is an assignment of one value to each parameter in the model that does not violate any of the constraints. For example, (IS = OutOfStock, EC = Yes, SD = Domestic, PS = Scheme1, DT = OneWeek, OS = Air) is a valid test.

All tests produced by a CTD algorithm must be valid tests. To generate a test plan from a combinatorial model, the CTD algorithm automatically constructs a subset of the test space so that it covers all valid value combinations of every t parameters, where t is usually a user input. For example, the test plan in Table 2 containing 4 valid tests is a 1-way test plan, since every single value of the model appears in at least one test of the test plan. However it is not a 2-way test plan, since it does not cover all valid pairs of values from the model. For example, the valid pair (DT = OneWeek, OS = Ground) does not appear in any of the 4 tests. In contrast, the test plan in Table 3 containing 12 valid tests is a 2-way (a.k.a. pairwise) test plan, since every valid pair of values of the test space appears in at least one of its tests.

Table 2: A 1-way test plan for the online shopping model, consisting of 4 tests

1	(IS = OutOfStock, EC = Yes, SD = Domestic, PS = Scheme1, DT = OneWeek, OS = Air)
2	(IS = NoSuchProduct, EC = No, SD = Foreign, PS = Scheme2, DT = OneMonth, OS = Ground)
3	(IS = InStock, EC = Yes, SD = Domestic, PS = Scheme3, DT = Immediate, OS = Air)
4	(IS = OutOfStock, EC = Yes, SD = Foreign, PS = Scheme1, DT = OneMonth, OS = Sea)

Table 3: A pairwise test plan for the online shopping model, consisting of 12 tests

1	(IS = OutOfStock, EC = No, SD = Foreign, PS = Scheme1, DT = Immediate, OS = Air)
2	(IS = InStock, EC = No, SD = Domestic, PS = Scheme3, DT = OneWeek, OS = Ground)
3	(IS = NoSuchProduct, EC = Yes, SD = Foreign, PS = Scheme2, DT = OneMonth, OS = Ground)
4	(IS = InStock, EC = Yes, SD = Domestic, PS = Scheme2, DT = Immediate, OS = Air)
5	(IS = InStock, EC = Yes, SD = Foreign, PS = Scheme1, DT = OneMonth, OS = Sea)
6	(IS = NoSuchProduct, EC = Yes, SD = Domestic, PS = Scheme3, DT = Immediate, OS = Air)
7	(IS = OutOfStock, EC = Yes, SD = Foreign, PS = Scheme2, DT = OneWeek, OS = Ground)
8	(IS = NoSuchProduct, EC = No, SD = Domestic, PS = Scheme1, DT = OneWeek, OS = Ground)
9	(IS = OutOfStock, EC = No, SD = Foreign, PS = Scheme2, DT = OneMonth, OS = Sea)
10	(IS = NoSuchProduct, EC = No, SD = Foreign, PS = Scheme1, DT = OneMonth, OS = Sea)
11	(IS = OutOfStock, EC = Yes, SD = Domestic, PS = Scheme3, DT = OneMonth, OS = Ground)
12	(IS = InStock, EC = Yes, SD = Domestic, PS = Scheme3, DT = OneWeek, OS = Air)

3. VISUALIZATIONS

We provide an overview of the new visualization views that we defined and support in IBM FOCUS. For each we provide motivation for the view, what information it displays and how it is presented, using the running example discussed in Section 2.

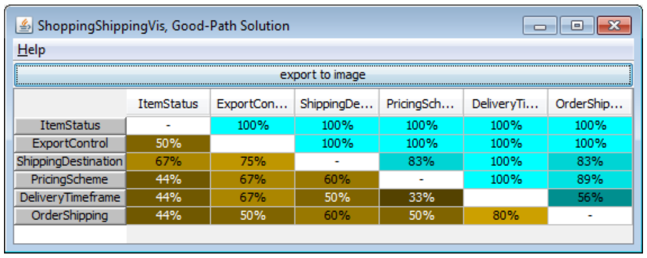
The examples and descriptions in this section are for demonstration purposes only. In Section 5 we will describe how these views were used in real-world projects involving real-world use cases and data.

We divide the different views according to their visualization form. We support three such forms: matrices, graphs, and treemaps [19]. We describe the views per visualization form, from the simplest form to the most complex one.

3.1 Matrix-Based Visualizations

Matrices are one of the most basic and popular forms for the presentation of data. In the context of CTD, matrices are a natural way to visualize the relationship between pairs of parameters in a model and in its derived test plan. The matrix helps explore the parameter pairs space as a means to review and confirm understanding of which pairs are constrained more than others in the test space and which pairs are covered more than others in the test plan.

Figure 1 illustrates the visualization of the online shopping model from Table 1, together with the 1-way test plan from Table 2. The upper half of the matrix represents the validity status of the parameter pairs. Each cell reflects the percentage of valid value pairs out of all value pairs of the corresponding parameter pair. Darker cells draw attention

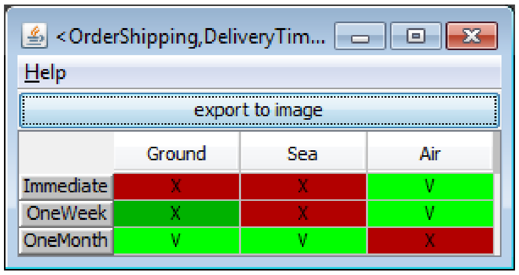

ShoppingShippingVis, Good-Path Solution

Help

export to image

	ItemStatus	ExportCon...	ShippingDe...	PricingSch...	DeliveryTi...	OrderShip...
ItemStatus	-	100%	100%	100%	100%	100%
ExportControl	50%		100%	100%	100%	100%
ShippingDestination	67%	75%	-	83%	100%	83%
PricingScheme	44%	67%	60%	-	100%	89%
DeliveryTimeframe	44%	67%	50%	33%		56%
OrderShipping	44%	50%	60%	50%	80%	-

Figure 1: Matrix visualization for online shopping model and its 1-way test plan. Upper half visualizes percentage of valid value pairs out of all pairs for each parameter pair. Lower half visualizes percentage of covered pairs out of all valid pairs for each parameter pair. See Section 3.1.


<OrderShipping,DeliveryTim...

Help

export to image

	Ground	Sea	Air
Immediate	X	X	V
OneWeek	X	X	V
OneMonth	V	V	X

Figure 2: Drill-down to the values level for the parameter pair `OS` and `DT`. Valid pairs are marked green, invalid ones are red. Covered pairs are marked V, uncovered ones with an X. Green X cells are darker green than green V cells.

to pairs that are more constrained than other pairs, and may call for an additional review by the practitioner. Similarly, cells that are expected to be constrained but are marked with light colors can also draw attention from the practitioner. For example, the darkest cell in the upper half of the matrix in Figure 1 is for the parameter pair (`DT`,`OS`), indicating the multiple constraints on their relationship, resulting in only 56% of the value pairs being valid.

The lower half of the matrix represents the coverage status of the parameter pairs with respect to the test plan. Each cell reflects the percentage of covered value pairs out of all valid value pairs of the corresponding parameter pair. Darker cells draw attention to pairs that are less covered than other pairs. For example, the darkest cell in the lower half of the matrix in Figure 1 is for the parameter pair (`DT`, `PS`), indicating that only 3 of 9 valid value pairs are covered by this 1-way test plan.

Of course, in a pairwise or high interaction level test plan we expect all cells to have 100% value coverage. We note that IBM FOCUS can also analyze the coverage of existing, manually-designed test plans, where each test case was mapped to a combination of values from the model. In this case we expect different levels of coverage for different parameter pair cells.

Double-clicking on a parameter pair cell will drill down to the value pairs level, where one can view their validity and coverage status, as depicted in Figure 2. For example, the value pair (`DT` = `OneWeek`, `OS` = `Ground`) is the only valid pair (marked with green) that is uncovered by the test plan (marked with an X).

We added support in IBM FOCUS also for creating a matrix visualization for the model only, without a corresponding test plan. In this case, only the validity status will be visualized, hence the lower half of the matrix will be empty. In Section 5 we refer to this case as a separate view than the one that incorporates both validity and coverage visualization.

3.2 Graph-Based Visualizations

We use graphs to visualize elements and their relationships via three different views.

3.2.1 Parameter graphs

Parameter graphs visualize the relationships between parameters, specifically to indicate the amount of constraints they share. The motivation is the same as for the matrix upper half – to assist the practitioner in reviewing and debugging the CTD model during its development.

We construct the parameters graph as follows. Nodes represent parameters, edges represent constraints, and two parameters are connected if there is a constraint in which they both appear. The more constraints are in common between the two parameters, the thicker the edge between them. Hovering over an edge displays all its constraints in a pop-up message.

Figure 3 depicts the parameter graph for our online shopping example. The thickest edge in the graph indicates that the parameters `DT` and `OS` share the largest amount of constraints. Clicking on a certain node highlights all nodes it shares constraints with and the respective edges. When a node is selected, the user can view the graph in radial tree layout, as shown in Figure 3, to visually emphasize the relations between the selected node and the other nodes.

3.2.2 Constraint graphs

Constraint graphs are the dual of parameter graphs, where each node represents a constraint, and edges between two nodes represent common parameters between the two respective constraints. Constraint graphs can be used to locate complex constraints that may need to be simplified and similar constraints that may be merged. As opposed to parameter graphs, constraints graphs are hyper-edged graphs, i.e., there may be multiple edges between two nodes, one for each common parameter. The constraints graph for our online shopping example is depicted in Figure 3.

3.2.3 Test-plan graphs

The motivation for visualizing combinatorial test plan via graphs is to provide an understanding of how much powerful each test is with respect to the other tests in the test plan and help prioritize the test plan. This is achieved by visualizing (1) the amount of unique value tuples that each test case covers, and (2) how many of its value tuples are common with other tests (and with which ones).

We construct the test plan graph as follows. Nodes represent tests, numbered according to their numbering in IBM FOCUS, and the more unique value combinations the corresponding test covers, the larger the node. Edges represent value combinations. Two tests are connected if there are value combinations which they both cover. The more combinations are in common between the two tests, the thicker the edge between them.

It is important to note that the value combinations are determined according to the interaction coverage requirements

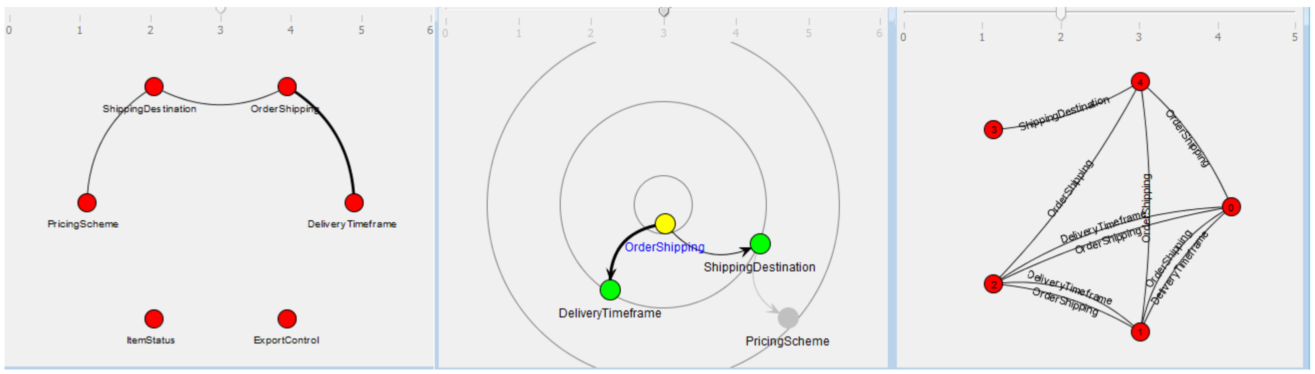


Figure 3: **On the left, the parameter graph for our example. In the middle, the same graph where the OS parameter is selected by the engineer, in radial tree layout, highlighting the parameters it shares constraints with and the respective edges. On the right, the constraint graph for our example: nodes represent constraints, numbered according to their order in IBM FOCUS; edges represent parameters and are labelled with the parameter names. See Sections 3.2.1 and 3.2.2.**

that were used to generate the test plan. For example, when one requests 3-way coverage, the edges represent triplets of values that the two related nodes are both covering.

When hovering over a node, its parameter values are displayed in a pop-up message. Similarly, when hovering over an edge, all the value combinations it represents are displayed in a pop-up message.

Figure 4 depicts the test plan graph for the pairwise test plan of our online shopping example, which was presented in Table 3. According to the nodes size, it is easy to see that test 1 contains the most unique value pairs. When clicking it, the tests it shares value pairs with are highlighted together with their respective edges. One can see that test 1 shares value pairs with 7 out of 11 other tests, though only a few value pairs in each case. The next most unique test is test 5, and it shares value pairs with 8 out of 11 other tests.

3.3 Treemap-Based Visualizations

Treemap [19] is a visualization method for displaying hierarchical data by using nested rectangles of different sizes and colors. Each of the dimensions of size and color represents a different aspect of the visualized data. We use treemaps for two types of visualization.

3.3.1 Test plan treemap

The motivation for visualizing test plans using treemaps is similar to that explained in Section 3.2.3 for graph visualization, i.e., understanding the respective strength of each test case and prioritizing the test plan. Similarly to test-plan graphs, we use the value combination coverage of each test as the main measure to be visualized. However, there are differences in the nature of the displayed information.

Each square in the treemap represents a test of the test plan, numbered according to its numbering in IBM FOCUS. The size of the square reflects the percentage of value tuples it uniquely covers in the test plan, while its shade reflects the same information only with respect to a user-selected subset of parameters. This allows for prioritizing tests according to specific parameters of high importance. The tests are always visualized ordered according to their size.

We present two ways to calculate the unique coverage of each test: incremental coverage and non-incremental coverage. A test incrementally covers a value tuple if this tuple is

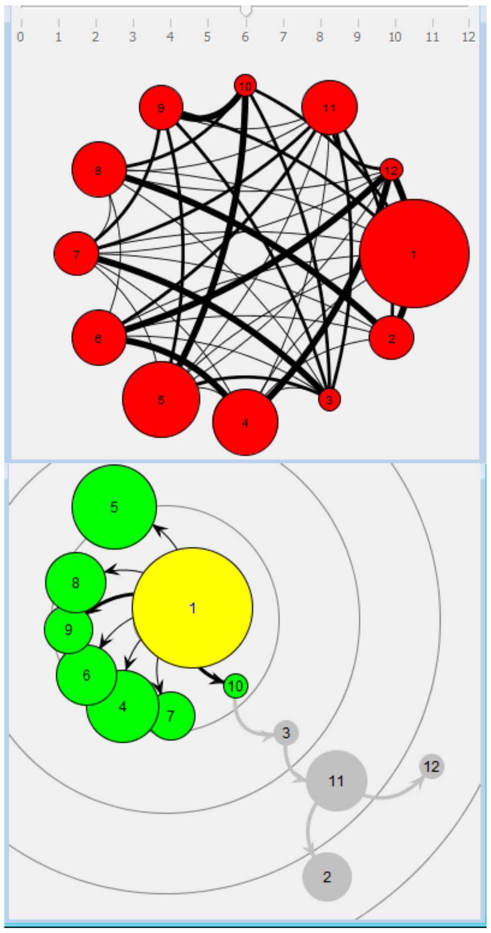


Figure 4: **At the top, the test plan graph for the online shopping example and its derived pairwise test plan containing 12 tests. At the bottom, the same graph after the largest node representing test 1 is clicked, in radial tree layout, highlighting the nodes it shares value pairs with and the respective edges. See Section 3.2.3.**

not covered by any preceding test (according to the order in which they are produced by IBM FOCUS when creating the test plan). A test non-incrementally covers a value tuple if this tuple is not covered by any other test in the test plan.

Figure 5 depicts the test plan treemap for the same pairwise test plan of our online shopping example (as in Section 3.2.3). When using incremental coverage, the tests coverage more or less follows the order in which they were generated by the CTD tool, due to the greedy nature of the CTD generation algorithm. When using non-incremental coverage, we get that test 1 and 5 have the most unique coverage of value pairs, in accordance with the test plan graph visualization (note the size of nodes 1 and 5 in Figure 4).

3.3.2 *Model and test plan treemap*

This view visualizes similar information to the one visualized by the matrix described in Section 3.1, i.e., amount of validity and amount of coverage of value combinations. However, as opposed to matrices, which are two-dimensional, the treemap is not limited to pairs only. Each square represents a parameter combination (tuple). The size of the square reflects its coverage percentage, while its shade reflects its validity percentage. For example, if the test plan was generated using 3-way requirements, then the squares in the treemap represent parameter triplets.

Figure 6 depicts the model and test plan treemap for the online shopping example and its derived pairwise test plan (which appears in Table 3). All squares are of equal size, indicating that all parameter pairs are equally covered by the test plan. The darker squares indicate lower percentage of valid pairs for the respective parameter pairs. As seen before, (`DT`, `OS`) is the darkest square due to its lowest percentage of valid value pairs.

As in the matrix view, by clicking a square, the treemap view allows the user to drill down to the value tuples level.

4. COMPUTATION AND IMPLEMENTATION

In this section we provide details on the algorithmic aspects of the data queries required for the visualizations as well as on the implementation of the visualization views.

4.1 Data Query Computation

We describe the data queries computation for each of the visualization views. We implemented these computations as part of IBM FOCUS.

Matrix data query computation

The matrix visualization requires two types of information for each pair of parameters: (1) the percentage of valid value pairs out of the total number of value pairs, and (2) the percentage of value pairs covered by the test plan out of all valid value pairs. In the matrix view we display both the actual percentage (in the cell itself) and the numerator and denominator values (in a tooltip when hovering over the cell). The percentage is also used to determine the color and shade of the cell.

Algorithm 1 presents the pseudo-code for computing the above information. It receives as input a combinatorial model $S = (P, V, C)$, where P is the set of parameters, V is the set of value sets (one value set per parameter), and C is the set of constraints, a Binary Decision Diagram (BDD) [1] $Valid_S$ representing the set of all valid tests in the model (will be explained in the following), and optionally also a test plan T, which is a set of value combinations, where

```
input  : A combinatorial model S=(P,V,C)
         The BDD $Valid_S$ of all valid tests of $S$
         Optional: a test plan $T$
output : An array total of the number total value pairs per
         parameter pair
         An array valid of the number of valid value
         pairs per parameter pair
         Optional: an array covered of the number of
         covered value pairs per parameter pair
1  for $p_i, p_j \in P$ do
2      if $i < j$ then
3          $total(i,j) \leftarrow |V(i)| \times |V(j)|$
4          $validPairs \leftarrow \texttt{project}(Valid_S, (i,j))$
5          $valid(i,j) \leftarrow |coveredPairs|$
6          if $|T| > 0$ then
7              $covered(i,j) \leftarrow 0$
8              for $(v_1, v_2) \in validPairs$ do
9                  $covered(i,j) \leftarrow$
                   $covered(i,j) + \texttt{isCovered}(T, (v_1, v2))$
10             end
11         end
12     end
13 end
```

Algorithm 1: Query for matrix visualization. See Section 4.1.

each value combination $t \in T$ assigns a single value to each parameter in P. The algorithm iterates over all parameter pairs in the model, and for each pair computes the total number of value pairs, the valid number of value pairs, and the number of value pairs covered by the test plan (if given). The total number of value pairs can be easily computed by multiplying the number of values of the first parameter with that of the second parameter (see line 3). However, computing the number of valid pairs is much more challenging. Note that we cannot simply evaluate the constraints in C against a given value pair, as there may be insufficient information to conclude whether or not they are satisfiable. Instead, to precisely and efficiently compute the number of valid value pairs for a given parameter pair, we use a symbolic representation based on Binary Decision Diagrams (BDDs) [1], a compact data structure for representing and manipulating Boolean functions. Specifically, we follow Segall et al. [18] and represent the set of all valid tests in the combinatorial test space using a single BDD, which is given to our algorithm as input ($Valid_S$). Note that by nature of construction, $Valid_S$ considers both the constraints explicitly specified by the user who defined the model, and those implicitly derived from combinations of explicit constraints. To compute the number of valid value pairs, we project the $Valid_S$ BDD on the parameter pair by existentially quantifying all other parameters from the BDD (line 4). Finally, we compute the coverage of the parameter pair in the input test plan T by iterating over all valid value pairs, and for each valid pair, checking whether it appears in at least one test in T (line 9).

The time complexity of Algorithm 1 is proportional to the number of pairs in P which is $\binom{|P|}{2}$, multiplied by the complexity of the most expensive operation inside the loop which is the projection operation, by itself proportional to the size of $Valid_S$.

The values level drill-down uses the validity and coverage data computed for the parameter level matrix to determine

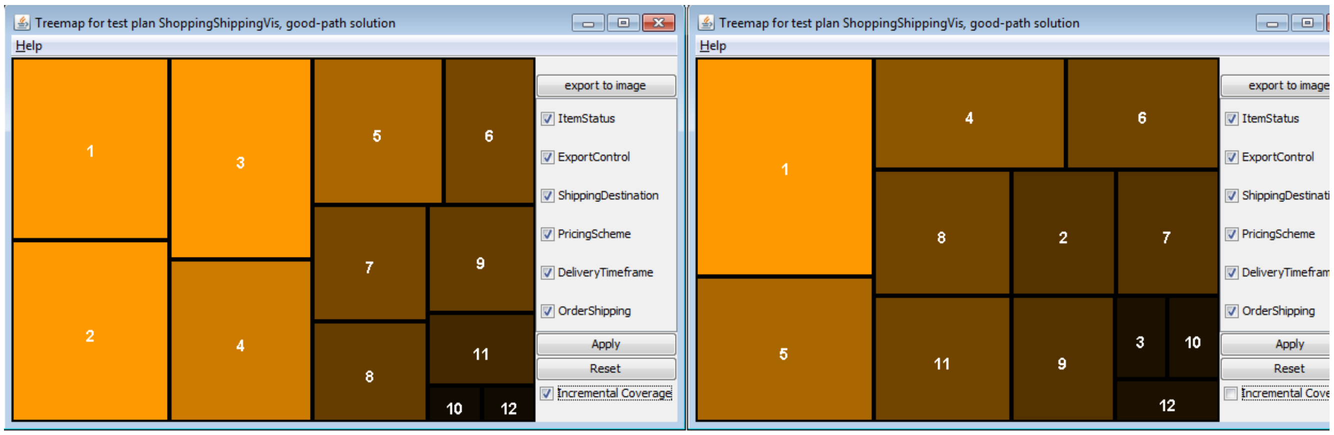


Figure 5: **On the left, the test plan treemap for the online shopping example and its derived pairwise test plan containing 12 tests, using incremental coverage. On the right, the same test plan using non-incremental coverage. See Section 3.3.1.**

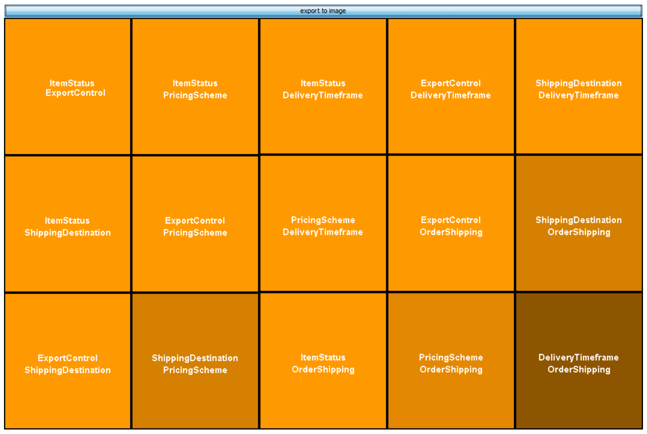


Figure 6: **The model and test plan treemap for the online shopping example and its derived pairwise test plan. The size of the squares indicates the percentage of coverage of the related parameter pairs in the test plan. They are all of equal size because the test plan achieves full pairwise coverage. The shade of the squares indicates the percentage of valid pairs for the respective parameter pairs. See Section 3.3.2.**

for specific value pairs whether or not they are valid and whether or not they are covered by the test plan.

Graph data query computation

The parameter and constraint graphs require data queries on the identity of the parameters that appear in each constraint. This information is relatively easy to extract via parsing of the constraints, and can be performed in time linear to the size of the constraints. In contrast, the test plan graph requires a more complex computation of the value tuples that are uniquely covered by each test in the test plan, and the number of value tuples shared between each two tests. Thus, for each test in the test plan we compute both the set of all value tuples it covers, and the set of value tuples it uniquely covers in the test plan. The former is used to calculate the value tuples intersection with every other test in the test plan to form the graph edges and determine their thickness. The latter is used to calculate the size of each node. Note that the value tuples are determined according to the parameter tuples that were defined in the interaction coverage requirement that were used to form the test plan. For example, in a 3-way test plan, the value tuples are all valid triplets of values. In a mixed-strength test plan, where different t level requirements are defined for different sets of parameters, different value tuples may correspond to different sizes of parameter tuples.

To compute the set of covered value tuples for each test, we iterate over the parameter tuples, and extract the corresponding value tuple from the test. To compute the set of uniquely covered value tuples, for each covered tuple we iterate over the rest of the test plan to check if there is another test that covers the same value tuple. If so, the value tuple is removed from the set of uniquely covered tuples.

The complexity of the computation of the above data queries is $O(k \times |T|^2)$, where k is the number of parameter tuples in the coverage requirements. For example, if the requirements are pairwise coverage, then $k = \binom{|P|}{2}$.

Treemap data query computation

The two treemap-based views require data queries that are similar to the queries of the other views, with slight differences. The model and test plan treemap requires validity percentage and coverage percentage for each parameter tuple, hence the calculation is similar to the symbolic calculation described in Algorithm 1. The only difference is that instead of iterating over parameter pairs, we iterate over parameter tuples based on the coverage requirements that were used when constructing the test plan. The time complexity is similar to that of the matrix computation, where rather than depending on the number of parameter pairs, it depends on the number of parameter tuples in the coverage requirements for the given test plan.

The test plan treemap requires the computation of the number of uniquely covered value tuples by each test, similarly to the computation that is performed for the test plan graph, but it also requires the same computation with respect to a selected subset of parameters. This information is computed in a similar manner, where parameter tuples that are not in the selected subset are skipped. Another difference is the option for incremental coverage, which is achieved by counting unique value tuples with respect to the previous tests only (in order of their generation by IBM FOCUS) instead of with respect to the entire test plan. The complexity is again $O(k \times |T|^2)$, as that of the computation for the test plan graph.

4.2 Implementation in IBM FOCUS

For the visualization implementation we used open source libraries which were incorporated into IBM FOCUS, as well as the Java Swing library which is an integral part of Java. Our main technological consideration was to use open source implemented purely in Java, for compatibility with the existing implementation of IBM FOCUS and in order to keep it a pure Java tool. Another consideration was to use open source libraries that provide a license that is suitable for use by a commercial tool, as required by IBM.

We implemented the matrix visualization in Java using its embedded Swing library, which supports two-dimensional tables. We implemented the graph visualization using Java Universal Network/Graph Framework (JUNG) [11]. We implemented the treemap visualization using treemaplib [23].

5. EVALUATION

We present an evaluation of our work in two different dimensions. We start with a user survey we conducted to evaluate different aspects of the effectiveness of our visualization views. We then continue with reporting on two real-world industrial projects in which our visualization was used as part of the deployment of CTD for test design.

5.1 User Survey

The research questions guiding our user survey are: **RQ1** Which of the visualization views are the most popular among the practitioners and do they consider some of them as better than others? and **RQ2** How do practitioners rate the value of the visualizations in terms of their usefulness, comprehension assistance, trust in the results, and support for communication?

5.1.1 Setup and participants

Our survey consisted of an online questionnaire that the users of IBM FOCUS were asked to fill in an IBM network. For each of the 7 views in IBM FOCUS (matrix views with and without coverage information were counted as different views), we first asked whether the participant has used this view in their work. If they answered positively, we followed with four additional questions asking them to rate their level of agreement on a Likert scale from 1 to 5 (from strongly disagree to strongly agree) with statements indicating that the view (1) was useful, (2) assisted in comprehension of the model and/or test plan, (3) increased trust in IBM FOCUS results, and (4) provided support for communication of the model and/or test plan to others. We also asked the participants to indicate whether they are "visual" by rating their level of agreement with the statement *"I am a visual person and typically like using visualizations in my work"*. We concluded the survey with two general rating questions, asking the participants to rate their level of agreement with the statements that the visualizations in IBM FOCUS made it (1) better and (2) more fun to use.

34 practitioners answered the survey, all of whom are CTD practitioners regularly involved in creating and comprehending combinatorial models for test designs, as part of their work in IBM. 19 of the 34 participants indicated that they have already used at least one of the visualization views in their work, and the results we report are based on their answers. Of these 19, 2 indicated they use IBM FOCUS for test design for more than 3 years, 8 use it for 1-3 years, and 9 for less than a year.

5.1.2 Results

Out of the 19 participants, 15 indicated they used the test plan and model matrix in their work, and 14 indicated they used the model matrix in their work. The test plan graph and the test plan treemap were used by 10 participants. The test plan and model treemap and the constraints graph were used by 7 participants. Finally, the parameter graph was used by 6 participants. These numbers indicate that the matrix-based views are the most popular ones; they were used more than twice than the non-popular ones.

To assess the perceived quality of each of the views, we report for each view its overall rating average, median, and standard deviation (SD). Note that each view was rated by a different number of participants, as detailed above.

The median rating for each of the 7 views was 4 (out of 5). The model and test plan treemap received the highest average rating of 4.21 (SD: 0.72), followed by the test plan treemap with average of 3.92 (SD: 0.91), then both model matrix and constraints graph with average of 3.79 (SD: 0.7 and 0.86, respectively). Next was the parameter graph with average of 3.75 (SD: 0.97), followed by model and test plan matrix with average of 3.72 (SD: 0.98). Closing the list is the test plan graph with average of 3.57 (SD: 1.16).

To answer [RQ1], the results show that the matrix visualizations are the most popular ones, and that the treemap visualizations and specifically the model and test plan treemap view are the ones most appreciated by the survey participants.

To assess the four quality aspects asked about in the survey, we report for each view its overall rating average, median, and standard deviation. The median rating for each of the four qualities was 4. For usefulness, the average rating was 3.85 (SD: 0.94), for comprehension assistance 3.81 (SD: 0.94), for trust in the results 3.74 (SD: 0.94), and for communicating to others 3.8 (SD: 0.89). In addition, the average rating for the statements that the visualizations made IBM FOCUS better and more fun to use were 3.95 (median: 4, SD: 0.89) and 3.8 (median: 4, SD: 0.83), respectively.

14 of the 19 participants indicated they are "visual" people, i.e., agreed or strongly agreed to the statement that they liked using visualizations in their work. When considering the answers of these 14 participants only, the overall ratings increase in all aspects, with lower standard deviation. For usefulness, the average rating is 4.07 (SD: 0.76), for comprehension assistance 4.05 (SD: 0.72), for trust in the results 3.93 (SD: 0.81), and for communicating to others 4 (SD: 0.71). The average rating for the statements that the visualizations made IBM FOCUS better and more fun to use are 4.36 (SD: 0.48) and 4.07 (SD: 0.49), respectively.

To answer [RQ2], all quality aspects were rated relatively high and with low variance. None of these four qualities was very different than the others.

5.1.3 Limitations and threats to validity

We discuss threats to the validity of the survey results, starting with internal validity. Our visualization views were introduced recently to IBM FOCUS and their use is optional. We note that out of more than 300 practitioners who are registered as users of IBM FOCUS in IBM, only 34 chose to answer our survey. From these 34, only 19 have indicated that they have already used at least one of the visualizations in their work. The results we report here are based on the answers of these 19 practitioners. This relatively small number of answers reflects the fact that the visualizations are not perceived as a critical feature of IBM FOCUS and that it may take time until more users of IBM FOCUS adopt and take advantage of them. That said, we are encouraged by comments we have received from several practitioners who answered the questionnaire, including, e.g., *"I didn't know about a couple of the later graphs until taking this survey (like the constraints one)."* and *"I really like all the new graphical features, unfortunately, I've not had the time to really understand how they all work. If I did, I think I'd use them in my work much more. Tks"*.

There are also threats to external validity. Due to the self-selection of our survey participants and the small sample, the survey results may not be representative, hence generalization is questionable.

5.2 Real-World Industrial Projects

We report on two real-world industrial projects in which our visualization was used as part of the deployment of CTD for test design. The projects are reported by the second listed author who works as a test architect in the Systems division of IBM. They were not done specifically for the purpose of evaluating the visualizations but as part of the ordinary work of this author in IBM.

5.2.1 IBM TS7700 virtual tape server (VTS)

VTS tests were designed and used for functional testing, covering functional operation variations. The emphasis was on proper completion of a single task in a given test environment. The model contained 6 parameters in total. The test environment was represented by 5 model parameters, `MachineModel` (2 values), `Number_of_Drives` (2 values), `Drive_type` (4 values), `MediaType` (3 values), and `EncryptedType` (2 values), and functional tasks were represented by another parameter `Test`, with 13 values for the different tasks. There were 41 constraints on the value combinations (for example limiting specific tasks to specific environment elements in which they can be performed), resulting in 368 legal tests. Pairwise requirements were used and IBM FOCUS created a test plan containing 39 tests.

At first, no limitation was considered with respect to affordability of total tests given test resources (people, machines, time). The 39 resulting tests were considered affordable. However, the test team was informed that an early milestone (playback) existed. This milestone entailed providing a beta version of the new function to select customers. Given that there were not enough resources to execute all the tests prior to the early milestone, the test team began using IBM FOCUS visualizations.

Before any testing was started, the test team used the visualizations to assist in making the decision on which combinations of test to prioritize. The prioritization would allow sufficient coverage prior to support of beta customers. The tests were prioritized for execution by studying the treemaps showing incremental and coverage benefits. Figure 7 depicts the incremental and non-incremental treemap for VTS pairwise test plan. Initially, the incremental treemap for the test plan was explored, pointing out how much more incremental coverage benefits the first couple tests provide. By deselecting incremental, the tests with the highest value coverage were quickly seen. These views enabled the team to think about what tests they may want to try first. From both treemaps, they got an idea that they like tests 1 and 2 for incremental benefits, but they also like tests 7, 12, and 20 for the value coverage benefits.

The test designers then used the graphs to "see" the specific relationships covered by the tests they selected to execute first (7, 12, 20, then 1 and 2). For example, Figure 8 visualizes the relationship between test 1 and the other tests, by selecting the node that represents test 1 in the graph. The graph shows how test 1 dominates most of the other tests, i.e., covers value pairs also covered by most other tests, but does not have common coverage with tests 2, 7, and 12.

The test designers then looked at the test plan model matrix view, depicted in Figure 9. This allowed them to confirm the total coverage offered by the test plan. (Incidentally, it was mentioned that it might be helpful if there were a matrix view that would show percentage coverage offered by the subset of tests they had selected for their beta support[1]). Lastly, the test designers drilled down into the cells with validity percentage less than 100% to see that the expected constraints were applied as desired.

5.2.2 IBM zEnterprise EC12 Enhanced Driver Maintenance (EDM)

EDM tests were of a system level. They cover interactions between different system functions. Each test represents what can be described as an end to end sequence of several steps for related and also unrelated tasks. These main steps are reflected by the parameters prefixed with `Pre`, `During`, and `Post`. In total there were 12 parameters in the model with number of values per parameter ranging from 3 to 17, and with 20 constraints, resulting in a test space containing 1.5×10^9 valid tests.

Given that many of the tasks introduced into the execution sequences are unrelated to EDM flow and that these same tasks can take hours to complete, the test designer wished to limit the appearance of some tasks in the test case matrix. Parameter value weights, as supported by IBM FOCUS, were defined in order to meet this requirement.

The coverage requirement was 3-way across three different parameters that were considered central to the execution sequence, `Pre_Condition`, `During_Workload`, and `Post_Condition`. All of the other parameters were allowed to be distributed across the resulting tests (a.k.a. to "float" in the jargon used by the test designers). This is where weights were chosen to emphasize and de-emphasize certain parameter values in the resulting tests[2]. Using the above coverage requirements and weights, the resulting test plan produced by IBM FOCUS contained 48 tests.

EDM testing had an early milestone like VTS testing. Again, this milestone was defined such that not all the tests could be attempted before that date. The test designer considered whether certain tests might be likely to provide more coverage benefits. From studying the test plan treemaps and graphs for EDM tests, it became clear that unlike VTS testing where there were coverage benefits offered by starting

[1]We have added this to IBM FOCUS list of feature requests.
[2]Assigning weights to values in a combinatorial testing model is a requirement on the distribution of values in the solution test set. This requirement reflects the importance of different values to the tester [5].

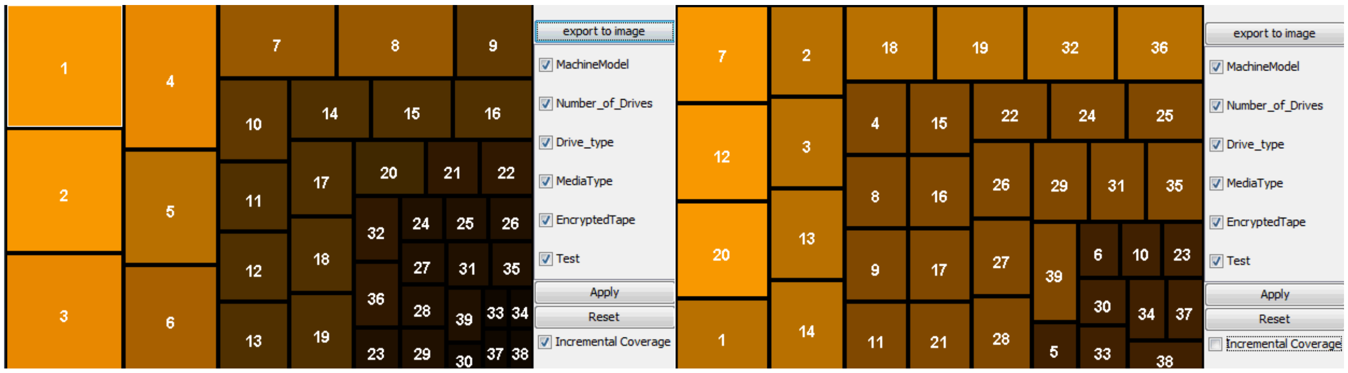


Figure 7: **On the left, the test plan treemap for VTS and its pairwise test plan containing 39 tests, using incremental coverage. On the right, the same test plan using non-incremental coverage. See Section 5.2.1.**

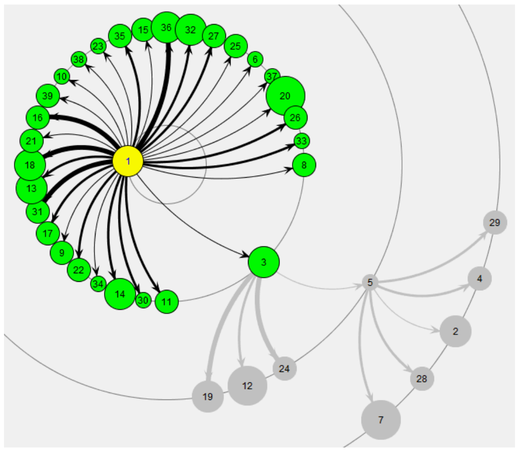
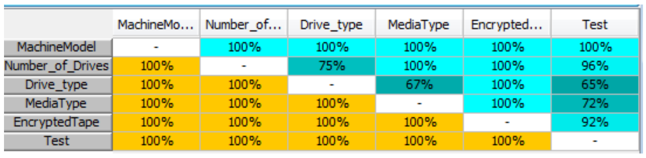


Figure 8: **The test plan graph for VTS and its pairwise test plan containing 39 tests, where the node representing test 1 is selected in order to explore its coverage relative to the other tests. See Section 5.2.1.**



	MachineMo...	Number_of...	Drive_type	MediaType	Encrypted...	Test
MachineModel	-	100%	100%	100%	100%	100%
Number_of_Drives	100%	-	75%	100%	100%	96%
Drive_type	100%	100%	-	67%	100%	65%
MediaType	100%	100%	100%	-	100%	72%
EncryptedTape	100%	100%	100%	100%	-	92%
Test	100%	100%	100%	100%	100%	-

Figure 9: **The matrix for VTS model and its derived pairwise test plan containing 39 tests. See Section 5.2.1.**

with certain tests, EDM tests did not offer such an advantage. The coverage was largely equivalent from one test to another, albeit for different parameter values. This was reflected by identical sizes and shades for the treemap, and identical node sizes for the graph. For this reason, EDM test execution leader was given the freedom to select which tests to do and when.

The matrix view was used to evaluate the impacts of the different weights, as applied to parameter values. Figure 10 top and bottom depict the matrix views for the test plan with and without the use of weights, respectively. As expected, the pairs involving the three central parameters mentioned earlier maintain 100% coverage. The intent was, as mentioned, to minimize specific parameter values' appearance in the resulting tests. By using the matrix view with the drill downs for specific cells, judgements were made by the subject matter expert test designer regarding suitability of resulting test coverage. These judgements were based on historical data relating to prior EDM test efforts and performance on customer systems. For example, previous EDM releases demonstrated that while there were many different `Pre_Level` parameter values, one particular parameter value dominated actual customer usage. Coincidentally, the dominant value was also the least costly in terms of test environment setup time. Weights were used to favor this value in the tests. By using the matrices in Figure 10, and drilling into the (`Pre_Level`, `Precondition/Post_Condition/During_Workload`) cells, the test engineer confirmed that the desired `Pre_Level` parameter value distribution was achieved. In prior EDM releases without this CTD visualization capability, the engineer resorted to doing this evaluation "by eye" over the entire test list.

5.2.3 *Projects discussion*

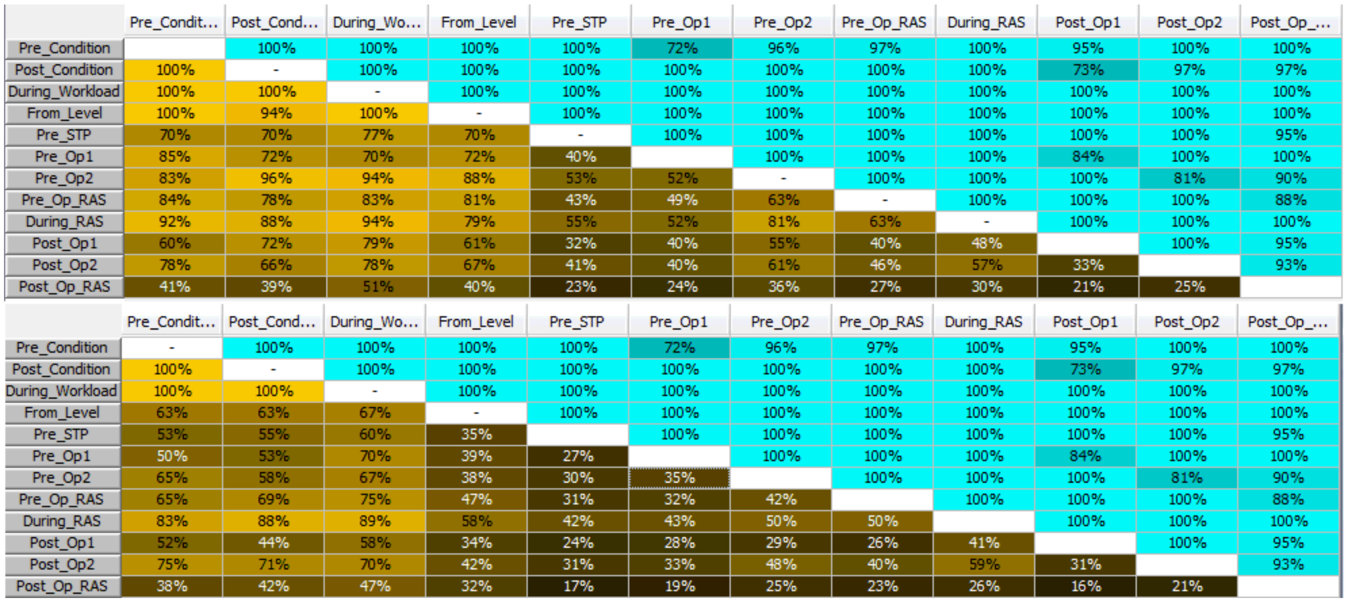
The similarities and differences between these two case studies of IBM FOCUS visualizations are interesting.

On the one hand, both test efforts were motivated by early milestone coverage objectives. The visualizations allowed the test designers to plan for test execution accordingly.

On the other hand, the two projects differed in the type of testing, functional versus system testing, and in the interaction coverage requirements. The resulting observation that the test plan treemap and graph were useful for VTS test plan and less useful for EDM test plan may be related to these differences. The reason is that while using pairwise requirements to generate the test plan induces different levels of pairwise coverage for different tests with respect to the rest of the test plan, having most of the parameters float without coverage requirements as in EDM induces similar levels of coverage for different tests.

Both test plan design efforts did benefit from the matrix view with the drill down. In VTS test plan the matrix view proved beneficial for constraint evaluation. In EDM it proved advantageous for deciding on floating parameter value distribution and for constraint evaluation.

In sum, IBM FOCUS visualizations allowed test designers to get vital insight into their models and into the coverage provided through CTD generated test plans, and thus helped



	Pre_Condit...	Post_Cond...	During_Wo...	From_Level	Pre_STP	Pre_Op1	Pre_Op2	Pre_Op_RAS	During_RAS	Post_Op1	Post_Op2	Post_Op_...
Pre_Condition		100%	100%	100%	100%	72%	96%	97%	100%	95%	100%	100%
Post_Condition	100%	-	100%	100%	100%	100%	100%	100%	100%	73%	97%	97%
During_Workload	100%	100%	-	100%	100%	100%	100%	100%	100%	100%	100%	100%
From_Level	100%	94%	100%	-	100%	100%	100%	100%	100%	100%	100%	100%
Pre_STP	70%	70%	77%	70%	-	100%	100%	100%	100%	100%	100%	95%
Pre_Op1	85%	72%	70%	72%	40%		100%	100%	100%	84%	100%	100%
Pre_Op2	83%	96%	94%	88%	53%	52%	-	100%	100%	100%	81%	90%
Pre_Op_RAS	84%	78%	83%	81%	43%	49%	63%	-	100%	100%	100%	88%
During_RAS	92%	88%	94%	79%	55%	52%	81%	63%	-	100%	100%	100%
Post_Op1	60%	72%	79%	61%	32%	40%	55%	40%	48%		100%	95%
Post_Op2	78%	66%	78%	67%	41%	40%	61%	46%	57%	33%		93%
Post_Op_RAS	41%	39%	51%	40%	23%	24%	36%	27%	30%	21%	25%	

	Pre_Condit...	Post_Cond...	During_Wo...	From_Level	Pre_STP	Pre_Op1	Pre_Op2	Pre_Op_RAS	During_RAS	Post_Op1	Post_Op2	Post_Op_...
Pre_Condition	-	100%	100%	100%	100%	72%	96%	97%	100%	95%	100%	100%
Post_Condition	100%	-	100%	100%	100%	100%	100%	100%	100%	73%	97%	97%
During_Workload	100%	100%	-	100%	100%	100%	100%	100%	100%	100%	100%	100%
From_Level	63%	63%	67%	-	100%	100%	100%	100%	100%	100%	100%	100%
Pre_STP	53%	55%	60%	35%		100%	100%	100%	100%	100%	100%	95%
Pre_Op1	50%	53%	70%	39%	27%		100%	100%	100%	84%	100%	100%
Pre_Op2	65%	58%	67%	38%	30%	35%		100%	100%	100%	81%	90%
Pre_Op_RAS	65%	69%	75%	47%	31%	32%	42%		100%	100%	100%	88%
During_RAS	83%	88%	89%	58%	42%	43%	50%	50%		100%	100%	100%
Post_Op1	52%	44%	58%	34%	24%	28%	29%	26%	41%		100%	95%
Post_Op2	75%	71%	70%	42%	31%	33%	48%	40%	59%	31%		93%
Post_Op_RAS	38%	42%	47%	32%	17%	19%	25%	23%	26%	16%	21%	

Figure 10: At the top, the matrix for EDM and its test plan derived from 3-way requirements on the three central parameters, without weights. At the bottom, the matrix for the test plan derived using the same coverage requirements with the addition of weights. Note the different shades and coverage levels for the same cells in the two matrices. See Section 5.2.2.

them better tune, trust, and ultimately improve their test design and execution approach.

6. RELATED WORK

Many CTD tools are listed in [17], e.g., [3,5,10,15]. To the best of our knowledge, almost none is providing visualization of models and test plans (with the exceptions described below). We are aware of only three works that suggest the use of visualizations related to combinatorial models and test plans. A short paper by Lopez-Herrejon and Egyed [16] suggested the use of visualizations related to combinatorial models and test plans, specifically for pairwise test plans in the context of software product lines. This paper did not present implementation and evaluation. The NIST Combinatorial Coverage Measurement Tool [14] creates a graphical view for a given test set, showing for each level of value coverage, the percentage of parameter combinations that reached that coverage level. The view is accumulative and does not refer to individual parameter combinations. Finally, the commercial CTD tool Hexawise [8] includes a value-level coverage matrix and has a slider one can use to see how the coverage increases as one adds tests from the test plan. The tool does not show a matrix at the parameters level and does not include any additional visualizations.

Save of the above, to the best of our knowledge, our work is the first to present, implement, and evaluate interactive visualizations for combinatorial models and test plans.

7. CONCLUSION AND FUTURE WORK

We presented visualizations based on matrices, graphs, and treemaps, as a means to assist in comprehension of combinatorial models and test plans and in prioritizing a test plan. We integrated the visualizations into IBM FOCUS, a commercial industry-strength CTD tool, using a non-trivial symbolic implementation, and evaluated it via a user survey and two real-world industrial projects. The evaluation shows that the visualizations can help testers better tune, trust, and improve their test design and execution approach.

Matrices, graphs, and treemaps are useful if they are not too large and complex; otherwise, additional techniques should be used to help the viewer in exploring them, such as fisheye graph views (see, e.g., [22]). Our use of interactive drill-down and focus, in all three visualization forms, which follows the overview first, zoom and filter, details-on-demand paradigm [20], is one means to address this challenge. In terms of performance, our use of BDD-based data structures and algorithms makes the computation of the data for the visualizations scale well: in all cases we encountered the computation took no more than a few seconds; we did not receive any scale complaints from the users of IBM FOCUS.

In the future, we plan an extended and more systematic user evaluation. We would also like to expand the interactivity of our visualizations by enabling on-the-fly updates to the views following changes in the underlying data layer, e.g., when choosing different subsets of the test plan for execution. Finally, we plan to investigate the use of visualization to help IBM FOCUS users compare between different versions of a combinatorial model and between alternative combinatorial test plans.

8. ACKNOWLEDGMENTS

We thank Itai Segall for suggesting the use of visualizations to improve the usability of IBM FOCUS. We thank Aya Chayat and Shiri Ladelsky for implementing the visualizations as part of an undergraduate project class at TAU. Part of this work was done while SM was on sabbatical as visiting scientist at MIT CSAIL. This research was done under the terms of a joint study agreement between IBM Corporation Inc (via the IBM Research Lab - Haifa) and Tel Aviv University. Additionally, part of the research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 610802.

9. REFERENCES

[1] R. E. Bryant. Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Trans. on Comp.*, 1986.

[2] K. Burroughs, A. Jain, and R. Erickson. Improved quality of protocol testing through techniques of experimental design. In *SUPERCOMM/ICC*, 1994.

[3] D. M. Cohen, S. R. Dalal, M. L. Fredman, and G. C. Patton. The AETG System: An Approach to Testing Based on Combinatorial Design. *IEEE Trans. on Softw. Eng.*, 1997.

[4] M. B. Cohen, J. Snyder, and G. Rothermel. Testing across configurations: implications for combinatorial testing. *SIGSOFT Softw. Eng. Notes*, 2006.

[5] J. Czerwonka. Pairwise Testing in Real World. In *PNSQC*, 2006.

[6] 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 *ICSE*, 1999.

[7] M. Grindal, B. Lindström, J. Offutt, and S. F. Andler. An evaluation of combination strategies for test case selection. *Empirical Softw. Eng.*, 2006.

[8] Hexawise. https://hexawise.com/.

[9] IBM Functional Coverage Unified Solution (IBM FOCUS). http://researcher.watson.ibm.com/researcher/view_group.php?id=1871.

[10] Jenny website. http://burtleburtle.net/bob/math/jenny.html.

[11] Java universal network/graph framework (JUNG). http://jung.sourceforge.net/.

[12] D. R. Kuhn, R. N. Kacker, and Y. Lei. *Introduction to Combinatorial Testing*. Chapman & Hall/CRC, 2013.

[13] D. R. Kuhn, D. R. Wallace, and A. M. Gallo. Software Fault Interactions and Implications for Software Testing. *IEEE Trans. on Softw. Eng.*, 2004.

[14] R. Kuhn. Combinatorial coverage measurement tool, user guide. Technical report, NIST, 2011.

[15] R. Kuhn, Y. Lei, and R. Kacker. Practical Combinatorial Testing: Beyond Pairwise. *IT Professional*, 2008.

[16] R. E. Lopez-Herrejon and A. Egyed. Towards interactive visualization support for pairwise testing software product lines. In *2013 First IEEE Working Conference on Software Visualization (VISSOFT)*, pages 1–4, 2013.

[17] Pairwise testing website. http://www.pairwise.org/tools.asp.

[18] I. Segall, R. Tzoref-Brill, and E. Farchi. Using Binary Decision Diagrams for Combinatorial Test Design. In *ISSTA*, 2011.

[19] B. Shneiderman. Tree visualization with tree-maps: 2-d space-filling approach. *ACM Trans. Graph.*, 11(1), 1992.

[20] B. Shneiderman. The eyes have it: a task by data type taxonomy for information visualizations. In *Visual Languages, 1996. Proceedings., IEEE Symposium on*, 1996.

[21] K. Tai and Y. Lie. A Test Generation Strategy for Pairwise Testing. *IEEE Trans. on Softw. Eng.*, 2002.

[22] C. Tominski, J. Abello, F. van Ham, and H. Schumann. Fisheye tree views and lenses for graph visualization. In *10th Int. Conf. on Inf. Vis. (IV)*, pages 17–24. IEEE Computer Society, 2006.

[23] treemaplib. https://github.com/smurf667/treemaplib.

[24] A. W. Williams. Determination of test configurations for pair-wise interaction coverage. In *TestCom*, 2000.

[25] P. Wojciak and R. Tzoref-Brill. System Level Combinatorial Testing in Practice – The Concurrent Maintenance Case Study. In *ICST*, 2014.