Distance-integrated Combinatorial Testing

Eun-Hye Choi*, Cyrille Artho*†, Takashi Kitamura*, Osamu Mizuno‡, Akihisa Yamada*§
* National Institute of Advanced Industrial Science and Technology (AIST), Ikeda, Japan
Email: {e.choi, t.kitamura}@aist.go.jp
† KTH Royal Institute of Technology, Stockholm, Sweden. Email: artho@kth.se
‡ Kyoto Institute of Technology, Kyoto, Japan. Email: o-mizuno@kit.ac.jp
§ University of Innsbruck, Innsbruck, Austria. Email: akihisa.yamada@uibk.ac.at

***Abstract*—This paper proposes a novel approach to combinatorial test generation, which achieves an increase of not only the number of new combinations but also the *distance* between test cases. We applied our distance-integrated approach to a state-of-the-art greedy algorithm for traditional combinatorial test generation by using two distance metrics, Hamming distance, and a modified chi-square distance. Experimental results using numerous benchmark models show that combinatorial test suites generated by our approach using both distance metrics can improve interaction coverage for higher interaction strengths with low computational overhead.**

***Keywords*-Combinatorial testing; t-way test generation; t-way coverage; Interaction strength; Hamming distance; Chi-square distance.**

I. Introduction

Testing is an activity to ensure the reliability of systems by actually executing the system against a certain set of test cases, called a *test suite*. In practice, resources for testing are limited, and exhaustive testing is almost always infeasible. Therefore, as a measure to select appropriate test cases, the notion of *coverage criterion* has been widely used; it is sometimes even required by safety standards, as in the automotive [2] and avionics [1] industries. Hence, one of the central research objectives in software testing is to develop test case generation techniques to comply various coverage criteria.

Combinatorial t-way testing [28], [34]—here t is a small number called the *interaction strength*—is a well-known black-box testing technique based on a coverage criterion called *t-way coverage*, which measures how many of the all possible interactions of t parameters are tested. Based on the observation that most system failures are caused by only a few parameters [30], [42], t-way testing aims at ensuring the quality of software testing by stipulating to test all t-way parameter interactions at least once. In principle, by t-way testing one can detect any defects triggered by the interaction of up to t parameters. Various algorithms [5], [16], [19] for generating small t-way test suites (i. e., a test suite that ensures 100 % t-way coverage) have been developed so far.

Two same-sized t-way test suites may have different t'-way coverage for $t' > t$. For example, 2-way test suites $\mathcal{T}_1$ in Table III and $\mathcal{T}_2$ in Table IV have different 3-way coverage: 69.81 % for $\mathcal{T}_1$ and 73.58 % for $\mathcal{T}_2$. In the real world, the size of test suites is often limited by budget; it may happen that 3-way testing is not admissible. Hence, a test suite with full t-way coverage and higher t'-way coverage can potentially detect more failures with higher interaction strength t' [9].

To our knowledge, Chen and Zhang [9] proposed the only method to tackle this problem. Their approach, which we call the *Enumerating Choice (EC)* approach in this paper, takes a t-way test suite as input and replaces "don't-care" parameter-values (that do not affect t-way coverage) with values that cover as many $(t + 1)$-tuples of parameter-values as possible. This technique indeed improves $(t + 1)$-way coverage but at the expense of a high computation overhead; the number of t-way tuples increases exponentially w. r. t. strength t.

In this paper, we propose a novel approach which generates t-way test suites and at the same time achieves higher $t'(> t)$-way coverage. In order to improve t'-way coverage at a reasonable computational overhead, we do not directly enumerate t'-way tuples; instead, based on the observation in *Adaptive Random Testing (ART)* [11], [10], which integrates the notion of *distance* into random testing, we propose *Distance-Integrated COmbinatorial Testing (DICOT)*. As in traditional greedy t-way test suite generation algorithms, DICOT generates test cases to cover as many t-way parameter-value combinations as possible, but it further tries to maximize the distance between test cases.

As the distance metric, we first investigate *Hamming distance* [22], a traditional metric that has been already used in existing testing approaches [8], [33]. Since the computation time of Hamming distance is quadratic in the length of the input, we also investigate a modified *chi-square distance* [35] to improve efficiency, as its computation cost is linear.

The question is whether considering distance can improve $t'(\geq t)$-way coverage or not. To experimentally investigate this question, we implemented DICOT in a greedy t-way test generation algorithm based on PICT [16]. Through experiments on numerous benchmarks including large-sized real applications from literatures [19], [38], we observe that DICOT achieves higher $t'(\geq t)$-way coverage compared to our implementation of the ART algorithm [10], and with lower computation overhead compared to our implementation of the EC approach [9].

The rest of this paper is organized as follows: Section II describes preliminaries. Section III presents our approach DICOT. Section IV shows experimental results. Section V describes related work and Section VI concludes.

TABLE I
An example SUT model.

Parameter	*Values*	*Constraint*
CPU	Intel, AMD	(OS=Mac → ¬(CPU=AMD))
Net	Wifi, LAN	∧ (Browser=IE → OS=Win)
OS	Win, Linux, Mac	∧ (Browser=Safari → OS=Mac)
Browser	IE, Firefox, Safari, Chrome	

TABLE II
An example of all possible pairs of parameter-values.

Param. pairs	*Parameter-value pairs*
(C,N)	(I,W), (I,L), (A,W), (A,L)
(C,O)	(I,W), (I,L), (I,M), (A,W), (A,L)
(C,B)	(I,I), (I,F), (I,S), (I,C), (A,I), (A,F), (A,C)
(N,O)	(W,W), (W,L), (W,M), (L,W), (L,L), (L,M)
(N,B)	(W,I), (W,F), (W,S), (W,C), (L,I), (L,F), (L,S), (L,C)
(O,B)	(W,I), (W,F), (W,C), (L,F), (L,C), (M,F), (M,S), (M,C)

II. Preliminaries

A. Combinatorial t-way testing

A *system under test (SUT)* for combinatorial testing (CT) is modeled from parameters whose associated value domains are finite. For instance, the SUT model shown in Table I, has four parameters (*CPU, Net, OS, Browser*); the first two parameters have two possible values and the others have three and four possibilities. Constraints among parameter-values express when some parameter-value combinations cannot occur. For example, currently, *Mac* does not support *AMD*, *IE* is available only for *Win*, and *Safari* is available only for *Mac*.

More rigorously, a model of an SUT is defined as follows:

Definition 1 (SUT model)**.** *An* SUT model *is a triple $\langle P, V, \phi \rangle$, where*

- *P is a finite set of parameters $p_1, \ldots, p_{|P|}$,*
- *V is a family that assigns a finite value domain V_i for each parameter p_i ($1 \le i \le |P|$), and*
- *ϕ is a constraint on parameter-value combinations.*

A *test case* is a value assignment for the parameters that satisfies the SUT constraint. For example, a 4-tuple (*Intel, Wifi, Win, IE*) is a test case for our example SUT model. We call a sequence of test cases a *test suite*.

Combinatorial *t-way testing* (e. g., *pairwise*, when $t = 2$) is a CT technique to test all t-way parameter interactions at least once.

Definition 2 (t-way test)**.** *Let $\langle P, V, \phi \rangle$ be an SUT model. We say that a tuple of t ($1 \le t \le |P|$) parameter-values is* possible *iff it does not contradict the SUT constraint ϕ. A* t-way test suite *for the SUT model is a test suite that covers all possible t-tuples of parameter-values in the SUT model.*

Example 1. *Consider the SUT model in Table I and $t = 2$. There exist* 38 *possible t-tuples (pairs) of parameter-values, (Intel, Wifi), …, (Mac, Chrome), as shown in Table II. The test suites $\mathcal{T}_1$ in Table III and $\mathcal{T}_2$ in Table IV are 2-way (pairwise) test suites, since each of them covers all the possible parameter-value pairs in Table II.*

Many algorithms to efficiently construct small t-way test suites have been proposed so far. Approaches to generate t-way test suites for SUT models with constraints include greedy algorithms (e. g., AETG [15], PICT [16] and ACTS [4]), heuristic search (e. g., CASA [19], HHSA [26], and TCA [32]), and SAT-based approaches (e. g., Calot [41]).

In this paper, we are also interested in t'-tuples with $t' > t$, where not all possible t'-tuples are covered. The coverage of possible t'-tuples is called *t'-way coverage*.

Definition 3 (t-way coverage)**.** t-way coverage*, denoted by $C_t(\mathcal{T}, \mathcal{S})$, of a test suite $\mathcal{T}$ for an SUT model $\mathcal{S}$ is defined as*

$$\frac{\textit{Number of t-tuples of parameter-values covered by } \mathcal{T}}{\textit{Number of all possible t-tuples of parameter-values in } \mathcal{S}}.$$

To evaluate coverage growth (i. e., how quickly a test suite obtains t-way coverage), we also use the metric called *APCC (Average Percentage of Covering-array Coverage)* [36], which measures the area under the curve when t-way coverage is on the y-axis and the index of test cases is on the x-axis; higher APCC implies faster growth of and higher t-way coverage.

Definition 4 (APCC [36])**.** *The APCC with t, which means the average percentage of t-way coverage, of a test suite $\mathcal{T}$ for an SUT model $\mathcal{S}$ is defined by*

$$A_t(\mathcal{T}, \mathcal{S}) = 1 - \frac{\sum_{1 \le i \le m} I_i}{nm} + \frac{1}{2n}$$

where n denotes the number of test cases, m denotes the number of possible t-tuples of parameter-values in $\mathcal{S}$, and I_i denotes the index of the first test case that covers the parameter-value t-tuple i.

Example 2. *Table III and Table IV show two test suites $\mathcal{T}_1$ and $\mathcal{T}_2$ for the example SUT model of Table I that provide the same* 100% 2*-way coverage but different* 3*-way coverage:* 69.81 % *for $\mathcal{T}_1$ and* 73.58 % *for $\mathcal{T}_2$. APCC with $t = 3$ for $\mathcal{T}_1$ is* 37.30 % *and that for $\mathcal{T}_2$ is* 38.51 %.

B. t-way testing with higher $t'(> t)$-way coverage

Chen and Zhang [9] proposed a metric for t-way testing called *tuple density*, which is defined as the $(t + 1)$-way coverage plus t. Metrics of $t'(> t)$-way coverage of t-way test suites, like tuple density, are important [28], [29], [37] because they distinguish between two t-way test suites with the same t-way coverage from the viewpoint of higher interaction strengths.

Chen and Zhang [9] also proposed a technique to construct t-way test suites with higher tuple density (or equivalently, higher $(t+1)$-way coverage). Their technique works as follows: Given a t-way test suite, it detects "don't-care" values, which are parameter-values of a test case whose assignment does not contribute to the coverage of more t-tuples; then, it computes all the yet-uncovered $(t+1)$-tuples and replaces each don't-care value with another value that covers as many new $(t+1)$-tuples as possible.

TABLE III
Test suite $\mathcal{T}_1$ (by X, CI$_X$).

	C	N	O	B	C_2(%)	C_3(%)
1	I	W	M	C	15.79	7.55
2	I	L	W	I	31.58	15.09
3	A	W	L	F	47.37	22.64
4	A	L	L	C	60.53	30.19
5	I	L	M	F	71.05	37.74
6	A	W	W	I	81.58	45.28
7	I	L	M	S	89.47	50.94
8	I	W	L	F	92.11	56.60
9	I	W	M	S	94.74	60.38
10	A	W	W	F	97.37	64.15
11	A	L	W	C	100.00	69.81

TABLE IV
Test suite $\mathcal{T}_2$ (by DI$_X$).

	C	N	O	B	C_2(%)	C_3(%)
1	I	W	M	C	15.79	7.55
2	A	L	L	F	31.58	15.09
3	I	L	W	I	47.37	22.64
4	A	W	W	C	60.53	30.19
5	I	W	L	F	71.05	37.74
6	I	L	M	S	81.58	45.28
7	I	L	L	C	86.84	52.83
8	A	W	W	I	92.11	58.49
9	I	W	M	S	94.74	62.26
10	A	W	W	F	97.37	67.92
11	I	L	M	F	100.00	73.58

TABLE V
Test suite $\mathcal{T}_3$ (by D).

	C	N	O	B	C_2(%)	C_3(%)
1	I	W	M	C	15.79	7.55
2	A	L	W	F	31.58	15.09
3	I	W	W	F	42.11	22.64
4	I	L	W	C	50.00	30.19
5	I	L	M	F	55.26	37.74
6	I	W	M	F	55.26	39.62
7	A	W	W	C	60.53	47.17
8	I	L	W	F	60.53	47.17
9	A	W	L	F	68.42	54.72
10	I	W	W	C	68.42	54.72
11	A	W	W	F	68.42	54.72

Although their approach simply improves a t-way test suite on $(t+1)$-way coverage, its limitation is that computing all the t'-tuples with higher interaction strength t' is expensive since the number of such t'-tuples increases exponentially with respect to t'.

III. Distance-integrated Combinatorial Test Generation

In this section, we introduce our approach for generating t-way test suites with higher $t'(> t)$-way coverage.

A. Proposed Approach: DICOT

The key concept of our approach is increasing *distance* among test cases when generating t-way test suites. Algorithm 1 describes the pseudo code of our algorithm, which we call DICOT (Distance-Integrated COmbinatorial Testing).

Traditional *one-test-at-a-time* t-way testing algorithms [7] commonly determine each test case (or parameter-value) to cover as many yet uncovered parameter-value t-tuples as possible (Line 3), until all possible parameter-value t-tuples are covered (Line 2). DICOT uses a distance between test cases as a tie breaker when there exist test case candidates (or parameter-value assignment candidates) with the same score; it chooses one that maximizes the distance from previous test cases (Line 4). We will explain the distance metrics we use in Section III-B.

DICOT can generalize existing t-way test generation algorithms by integrating their original test selection strategy and our distance strategy. For instance, AETG [14] (resp. Huang's method [25]) constructs each test case for pairwise testing by first generating r different candidate test cases using a greedy algorithm (resp. randomly) and choosing one that covers the most new parameter-value pairs. DICOT can be easily applied to such algorithms in a way of among r candidate test cases, choosing the one with not only the most new pairs but also the maximum distance.

DICOT can also employ a lot of state-of-the-art t-way test generation algorithms, e. g., PICT [16], ACTS [31], and CASA [19], that do not generate test case candidates but have tie-breakable choices in parameter-value assignments for test case generation. DICOT provides a tie breaker rule of maximizing distance between test cases for the existing tools.

The concept of DICOT, i. e. increasing the distance among test cases, can also be used for prioritizing (sorting) a given t-way test suite while we focus on using this concept in generating a t-way test suite in this paper.

Algorithm 1: Distance-integrated CT generation. (DICOT)

```
Input: SUT model $\mathcal{S}$, Interaction strength $t$
Output: $t$-way test suite $\mathcal{T}$
1 $UC$ = { All possible $t$-tuples of parameter-values in $\mathcal{S}$ };
2 while $UC \neq \emptyset$ do
3     Find test case candidates that maximize the number of
        parameter-value $t$-tuples in $UC$ $\boxed{\textit{CT strategy}}$;
4     Choose $tc$ among the candidates that maximizes the
        distance from previous test cases $\boxed{\textit{Distance strategy}}$;
5     Add $tc$ to $\mathcal{T}$;
6     Remove parameter-value $t$-tuples covered by $tc$ from $UC$;
7 return $\mathcal{T}$;
```

B. Distance Metrics

We use two metrics, (1) the *minimum Hamming distance* and (2) a *modified chi-square distance*, to define the distance between a test case and previous test cases. The minimum Hamming distance is used in distance-based testing [8], and we adopt it. On the other hand, through our knowledge, the use of chi-square distance [35] for test generation is new and motivated for improving efficiency.

1) Minimum Hamming Distance: As the first distance metric, we use the traditional Hamming distance [22]. The Hamming distance between two test cases is the number of parameters whose values are different in test cases. The minimum Hamming distance of a test case and another test suite is the minimum value of Hamming distance between the test case and a test case in the test suite.

Definition 5 (Minimum Hamming Distance)**.** *The* Minimum Hamming distance*, denoted by* $HD(t, \mathcal{T})$*, of a test case t from a test suite* $\mathcal{T}$ *is defined by*

$$HD(t, \mathcal{T}) = \min_{t_j \in \mathcal{T}} \ d(t, t_j)$$

where $d(t_i, t_j)$ *denotes the Hamming distance between the test cases* t_i *and* t_j*, which is the number of parameters assigned different values in test cases* t_i *and* t_j*.*

For example, the minimum Hamming distance of t_3 from previous test cases in $\mathcal{T}_2$ is computed by $HD(t_3, \{t_1, t_2\}) =$

TABLE VI

An example calculation of the modified chi-square distance (CD) for $\mathcal{T}_2$.

	CPU		Net		OS			Browser				
	I	A	W	L	W	L	M	I	F	S	C	CD
U_I	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	$M - 0 = \frac{53}{30}$
$U(t_3, \{t_1, t_2\})$	$\frac{2}{3}$	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{2}{3}$	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{0}{3}$	$\frac{1}{3}$	$M - \frac{1}{5} = \frac{47}{30}$

$\min(d(t_3, t_1), d(t_3, t_2)) = \min(4, 3) = 3$.

Maximizing the minimum Hamming distance for a new test case was also used in adaptive distance-based testing [8]. On the other hand, antirandom testing [33] adopted maximizing the *total* Hamming distance for a new test case. We internally compared using the minimal, maximal, and total Hamming distance, and concluded that maximizing the distance between test cases on the minimum Hamming distance achieves higher interaction coverage compared to maximizing that on the maximum Hamming distance and the total Hamming distance. The cost of computing this metric in generating a test suite $\mathcal{T}$ is $O(|\mathcal{T}|^2)$.

2) Chi-square Distance: For the second distance metric, we modify the well-known χ^2-divergence [35]. In order to spread parameter-values as much as possible, we assume that in an ideal situation, the number of occurrences of a value for each parameter is identical in a test suite. Under this assumption, we define the distance of a test case from a test suite by the difference between the probability distribution for parameter value occurrences when the test case is added to the test suite and the ideal probability distribution.

Employing the distance of probability distributions reduces computational overhead since we can avoid calculating the distance of a test case with all the previous test cases one by one; instead, we calculate the distance of a new distribution after adding a test case to the previous distribution.

We choose χ^2-divergence to simply measure the difference of probability distributions.[1] We define χ^2-divergence between probability distributions for value occurrences U of test suites $\mathcal{T}$ and $\mathcal{T}'$ by

$$\chi^2\ (U(\mathcal{T}) \parallel U(\mathcal{T}')) = \frac{1}{2} \sum_{v \in V_i (1 \le i \le |P|)} \frac{(u_v - u'_v)^2}{u_v + u'_v}$$

where u_v (resp. u'_v) is the occurrence probability for each parameter-value v in the test suite $\mathcal{T}$ (resp. $\mathcal{T}'$). Using the above χ^2-divergence, we define the following *modified chi-square distance* as the distance between test cases.

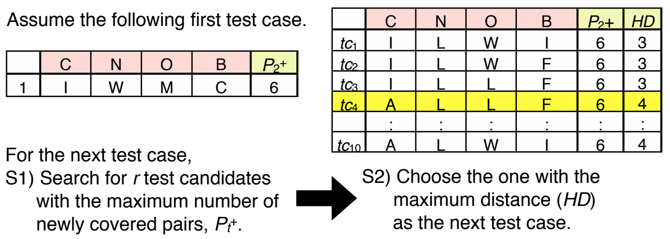
Definition 6 (Modified Chi-square Distance)**.** *We define the* modified chi-square distance*, denoted by $CD(t, \mathcal{T})$, of a test case t from a test suite $\mathcal{T}$ on a given SUT model by*

$$CD(t, \mathcal{T}) = M - \chi^2\ (U(\mathcal{T} \cup \{t\}) \parallel U_I)$$

where

- *$U(\mathcal{T} \cup \{t\})$ denotes the probability distribution for value occurrences of the test suite $\mathcal{T} \cup \{t\}$,*
- *U_I denotes the ideal probability distribution for occurrences of parameter-values in the SUT model. That is, for each parameter $p_i \in P$ and its value $v \in V_i$, we have $u_v = 1/|V_i|$ in U_I, and*
- *M denotes the maximum value of $\chi^2\ (U(\mathcal{T} \cup \{t\}) \parallel U_I)$, i. e. $M = \sum_{1 \le i \le |P|} (|V_i| - 1)/(|V_i| + 1)$.*

[1] To measure the divergence of probability distributions, we can also use other metrics [18], e. g., Kullback-Leibler divergence and Jensen-Shannon divergence, instead of chi-square distance.


Assume the following first test case.

	C	N	O	B	P_2^+
1	I	W	M	C	6

For the next test case,
S1) Search for r test candidates with the maximum number of newly covered pairs, P_t^+.

	C	N	O	B	P_2+	HD
tc_1	I	L	W	I	6	3
tc_2	I	L	W	F	6	3
tc_3	I	L	L	F	6	3
tc_4	A	L	L	F	6	4
	:	:	:	:	:	:
tc_{10}	A	L	W	I	6	4

S2) Choose the one with the maximum distance (HD) as the next test case.

Fig. 1. An example of distance-integrated t-way test generation by DICOT.

For example, the maximum χ^2-divergence for our example SUT model in Table I is $M = \frac{1}{3} + \frac{1}{3} + \frac{1}{2} + \frac{3}{5} = \frac{53}{30}$. The modified chi-square distance of t_3 from its previous test cases in $\mathcal{T}_2$ is obtained by $M - \frac{1}{5} = \frac{47}{30}$, which is computed as shown in Table VI.

The ideal value of χ^2-divergence is 0 and thus the ideal value of the modified chi-square distance is M, the maximum value of χ^2-divergence, by definition. Maximizing the modified chi-square distance (corresponding to minimizing χ^2-divergence) for a new test case makes value occurrences of a generated test suite close to having the ideal parameter-value divergence.

The cost of computing the modified chi-square distance in generating a test suite $\mathcal{T}$ is $O(|\mathcal{T}|)$. Thus, using the modified chi-square distance reduces the computational overhead of our distance strategy.

C. DICOT on an Ideal CT Generation: A Case Study

We first illustrate DICOT using the *ideal* t-way test generation, i. e., repeatedly choosing the *best* test cases among *all* possible test case candidates (w. r. t. newly covered tuples or distances). A more feasible setting is considered in the next section. We compare the following four approaches for our example SUT model to reveal the influence of using different measures for the test case distance.

- X: Generate a test case to maximize the number of new parameter-value t-tuples, denoted by $P_t^+(tc)$, not considering distances nor the $(t+1)$-way coverage. This is the basic form of the traditional one-test-at-a-time t-way test generation.
- DI_X (ours): After generating r test case candidates that maximize $P_t^+(tc)$ (Line 3 in Algorithm 1), choose the one with the *maximum* distance (Line 4 in Algorithm 1).
- CI_X : After generating r test case candidates that maximize $P_t^+(tc)$, choose the one with the *minimum* distance.
- D: Generate a test case with the maximum distance from the previous test suite, not considering $P_t^+(tc)$.

Note that X, $\mathsf{DI_X}$, and D employ the concept of the traditional CT generation, that of our DICOT, and that of test generation focusing on only the distance. Hereafter, we call the distance-focused testing approaches, e. g., D and ART, *DT*.

For the case study, we implemented all the four algorithms in Python. We encoded the problem of finding a test case candidate that maximizes $P_t^+(tc)$ to a *pseudo-Boolean optimization* (*PBO*) problem, and resolve this using an existing PBO solver, Sat4j [44]. (Zhang et al. [43] proposed a similar approach of one-test-at-a-time CT generation using PBO.) Sat4j is also used to find a test case with the maximum distance from the previous test suite for approach D.

Consider the example SUT model in Table I and $t = 2$. Figure 1 illustrates a t-way test generation process by our approach, denoted by $\mathsf{DI_X}$, using the minimum Hamming distance *HD*, assuming that the first test case is (*Intel, Wifi, Mac, Chrome*) and $r = 10$. The first test case candidate is tc_1=(*Intel, LAN, Win, IE*) which has the maximum $P_2^+(tc_1) = 6$, and the Hamming distance of candidate tc_1 from the previous test case t_1 is 3. In the same way, we find other test case candidates, calculate their distance from the previous test suite, and choose the one with the maximum distance, tc_4 in our example, as the next test case. This process iterates until all possible parameter-value pairs in Table II are covered. $\mathcal{T}_2$ in Table IV is generated by $\mathsf{DI_X}$.

Conversely to our approach, $\mathsf{CI_X}$ chooses a test case with the minimum distance among test case candidates. $\mathcal{T}_1$ in Table III is generated by $\mathsf{CI_X}$ for our example model. We can see that $\mathcal{T}_2$ by our $\mathsf{DI_X}$ and $\mathcal{T}_3$ by $\mathsf{CI_X}$ are the same-sized pairwise test suites, but $\mathsf{DI_X}$ achieves better 3-way coverage compared to $\mathsf{CI_X}$: 73.58 % for $\mathcal{T}_2$ and 69.81 % for $\mathcal{T}_1$, which is a relative improvement of 5.4 %.

On the other hand, approach X represents a typical CT construction that does not consider the distance; it corresponds to selecting a test case with a random distance. This means that in the worst case, X corresponds to $\mathsf{CI_X}$. For our running example, X generates the same test suite $\mathcal{T}_1$ as $\mathsf{CI_X}$ does, and hence $\mathsf{DI_X}$ obtains better 3-way coverage compared to X.

Conversely, approach D generates test cases where not P_t^+ but only the test case distance is considered. For example, $\mathcal{T}_3$ in Table V is generated by D when its termination condition is the number of test cases (11). We see that our $\mathsf{DI_X}$ achieves both higher 2-way and 3-way coverages compared to D.

As a result of this case study, we observe that DICOT has the ability to improve interaction coverage with higher interaction strength compared to the approach considering only combinatorial coverage or only distances. We present more experimental results and an analysis of the effectiveness and efficiency of our approach using large benchmark SUT models in Section IV.

D. DICOT on a Greedy CT Generation

Since the ideal combinatorial test generation strategy is not scalable for large SUT models, in this section we integrate our approach DICOT into an existing greedy t-way test generation algorithm. As explained in Section III-A, DICOT can also employ other state-of-the-art t-way test generation algorithms, which obtain as many uncovered parameter-value combinations as possible in various heuristic and greedy ways.

Algorithm 2: Distance-integrated pairwise test generation based on the PICT algorithm. (DC)

```
Input: SUT model $\mathcal{S}$
Output: Pairwise test suite $\mathcal{T}$
1  $UC$ = { All possible pairs of parameter-values in $\mathcal{S}$ };
2  while $UC \neq \emptyset$ do
3      while unassigned parameter exists for the next test case tc do
4          if no parameter is assigned then
5              Choose a parameter pair with the most parameter-value pairs in UC;
6              Choose a parameter-value pair p of the parameter pair that maximizes the distance from previous test cases $\boxed{Distance\ strategy}$;
7              Assign the parameter-value pair p to tc;
8              Remove p from UC;
9          else if $UC \neq \emptyset$ then
10             List parameter-value pairs in UC that can be assigned to tc and cover the maximum number of new parameter-value pairs;
11             Choose any candidate pair p that maximizes the distance from previous test cases $\boxed{Distance\ strategy}$;
12             Assign the parameter-value pair p to tc;
13             Remove parameter-value pairs covered by the assignment of p from UC;
14         else
15             Assign to unassigned parameters in tc values that do not violate SUT constraints and maximize the distance from previous test cases $\boxed{Distance\ strategy}$;
16     Add tc to $\mathcal{T}$;
17 return $\mathcal{T}$;
```

Algorithm 2, which we call DC, describes the pseudo code of the proposed algorithm which applies DICOT to a pairwise test generation algorithm that is based on the combinatorial test generation strategy by PICT [16].

In the original PICT algorithm, for each test case, first a parameter pair that has the most uncovered possible parameter-value pairs is selected, and one of the parameter-value pairs of the parameter pair is assigned. Next, a parameter pair is assigned one by one to cover the most uncovered parameter-value pairs until all parameters of the test case are assigned.

In DC, when assigning each parameter-value pair, we choose the one that not only covers the most uncovered parameter-value pairs but also maximizes the distance from previous test cases (Lines 6 and 11). When there exist no more uncovered parameter-value pairs that can be assigned to a parameter pair, the original PICT algorithm assigns any already-covered parameter-value pair, but we assign a parameter-value pair that maximizes the distance (Line 15).

IV. Experiments and Results

A. Research Questions

We set up the following four research questions to investigate the effectiveness and the efficiency of our approach.

RQ1. Compared with the traditional t-way test generation, can DICOT deliver higher t'-way coverage with higher interaction strength $t'(> t)$? If so, how big are the improvement and computational overhead?

RQ2. Compared with the Enumerating Choice (EC) approach, how effective and efficient is DICOT w. r. t. $t'(> t)$-way coverage and computational overhead?

RQ3. Compared with the DT approach, how effective and efficient is DICOT w. r. t. the t-way test suite size and $t'(> t)$-way coverage?

RQ4. How different is the performance when using Hamming distance and chi-square distance as a distance metric in DICOT?

DICOT employs another approach for the same purpose of EC and integrates the concept of DT into traditional t-way test generation. We thus explore RQ1–RQ3 to compare DICOT with the traditional t-way test generation, EC, and DT. We also compare two distance metrics used in DICOT by RQ4.

B. Experimental Setting

In order to answer the above research questions, we implemented the following five methods in C:

- CT: A PICT-based pairwise test generation algorithm.
- DC_{CD}: The proposed algorithm DC (Algorithm 2) using the modified chi-square distance.
- DC_{HD}: The proposed algorithm DC (Algorithm 2) using the minimum Hamming distance.
- EC (improved): The original algorithm of Chen and Zhang [9] does not support constraints, since "don't-care" analysis under constraints becomes a hard computational problem. We avoid this problem by integrating the idea of Chen and Zhang into the PICT-based algorithm. The new algorithm, which we call just EC later on, constructs test cases as in CT and additionally tries to increase the number of newly covered 3-tuples of parameter-values. This means that it chooses a parameter-value pair that maximizes P_3^+ among candidates in lines 6, 11, and 15 of Algorithm 2.
- DT: The FSCS-ART-based algorithm [11] that for each test case, first randomly generates a fixed number, r, of test case candidates satisfying SUT constraints and next among the r test case candidates, chooses one with the maximum over all minimal Hamming distances to previous test cases.

We implemented CT by ourselves based on the description of PICT's algorithm [16], since its original implementation was not yet open at the time of our implementation.[2] Unfortunately, from that description, we could not figure out how constraints are handled in PICT. Our implementation naively uses a SAT solver to check if each assignment in test generation satisfies constraints,[3] and is slower than the original PICT, when SUT constraints are considered. For a fair comparison, our naive constraints handling is adopted in the same way for all the five methods. To evaluate the computation overhead, we also use benchmarks where SUT constraints are disregarded. For DT, we set $r = 10$, use the random number generator of the standard C library, and take the average value of 20 runs.[4] For evaluation of t'-way coverage, we also implemented a program to calculate t'-way coverage and APCC for a given SUT model with constraints.

As benchmarks, we collected 55 SUT models. 20 of the models, which are from the work by Segall et al. [38], are for real-life applications for such as banking, health care, and insurance. The other 35 models, which are from the work by Garvin et al. [19], include five models for real applications: spins, spinv, apache, gcc, and bugzilla, and large artificial models whose numbers of parameters are up to around 200. (See Table VIII in Appendix, which shows the size and the numbers of possible pairs and 3-tuples of parameter-values for each benchmark SUT model.)

For the benchmark models, we generated 2-way (i. e., pairwise) test suites by the five methods, and measured their 3-way and 4-way coverages and the corresponding APCCs. We also evaluate the size of the generated test suite, i. e., the number of test cases (denoted by $|\mathcal{T}|$), and the test generation time. The sizes of test suites differ depending on the method. For a fair comparison, we investigate t-way coverage ($2 \le t \le 4$) of the same sized test suites, each of which is achieved by the first m test cases, where m is the minimum size of the test suites generated by the five methods.

Experiments were performed using a computer with Quad-Core Intel Xeon E5 3.7GHz, with 64GB memory running on Mac OS 10.10.5.

C. Results

Table VII summarizes the averages of results,[5] test suite sizes, test generation times, and t-way coverage (C_t) and APCC (A_t) with $2 \le t \le 4$, for all models. Note that we compare C_t and A_t of the truncated test suites with the same size, so that less than 100 % C_2 appears in the table. In addition, the table reports the results for models *without* constraints, i. e., models with their constraints removed.

For 17 models with constraints (and 13 models without constraints), EC could not finish generating test suites in an hour. CT, DC_{CD}, DC_{HD}, and DT can generate test suites for all models, but we could not finish computing 4-way coverage of test suites for 15 models with constraints (and two models without constraints) in one hour. The number of such cases is shown as "NAs" in the table.

[2] PICT is now open at https://github.com/microsoft/pict as of 2015-10-16.

[3] For the SAT solver, we employ PicoSAT [3].

[4] The work on FSCS-ART [11] has shown that failure-detection effectiveness improves as r increases up to about 10, and does not improve much further.

[5] See http://staff.aist.go.jp/e.choi/issre2016/results.html for the detailed results.

TABLE VII
COMPARISON OF THE RESULTS FOR BENCHMARK MODELS WITH/WITHOUT CONSTRAINTS. "N/A" DENOTES THE CASE CANNOT BE OBTAINED DUE TO TIMEOUT CASES.

With Constraints		CT	DC$_{CD}$	DC$_{HD}$	EC	DT
	μ_{all}	**35.42**	36.86	36.25	N/A	83.96
Size $\lvert\mathcal{T}\rvert$	μ_{sub}	31.89	33.21	32.52	**31.42**	65.00
	Wins	**36**	14	20	27	0
	μ_{all}	3.25	3.28	3.47	N/A	**0.61**
Time (s)	μ_{sub}	0.42	0.42	0.45	5.35	**0.13**
	Wins	7	3	2	2	**47**
	NAs	0	0	0	18	0
2-way coverage	μ_{all}	**99.85**	99.70	99.75	N/A	98.17
C$_2$ (%)	μ_{sub}	99.86	99.61	99.70	**99.92**	97.51
	Wins	**35**	16	20	26	1
APCC	μ_{all}	80.78	**82.04**	81.36	N/A	78.58
A$_2$ (%)	μ_{sub}	76.66	**78.29**	77.44	76.73	74.43
	Wins	2	**46**	6	4	0
3-way coverage	μ_{all}	77.30	80.19	**81.37**	N/A	80.12
C$_3$ (%)	μ_{sub}	71.82	74.89	75.78	**76.35**	74.22
	Wins	2	3	**30**	24	2
APCC	μ_{all}	42.47	45.20	**45.76**	N/A	44.14
A$_3$ (%)	μ_{sub}	34.24	36.84	37.10	**37.42**	35.78
	Wins	0	8	**26**	21	1
4-way coverage	μ_{all}	41.89	44.88	**46.47**	N/A	46.34
C$_4$ (%)	μ_{sub}	40.68	43.70	45.12	**45.35**	44.68
	Wins	2	5	15	**20**	6
APCC	μ_{all}	10.50	11.75	**12.22**	N/A	11.97
A$_4$ (%)	μ_{sub}	9.66	10.85	11.19	**11.31**	10.99
	Wins	2	7	16	**19**	4
	NAs	15	15	15	17	15

Without Constraints		CT	DC$_{CD}$	DC$_{HD}$	EC	DT
Size $\lvert\mathcal{T}\rvert$	μ_{all}	**34.35**	36.20	34.91	N/A	57.31
	μ_{sub}	**32.05**	33.89	32.73	32.44	52.08
	Wins	**35**	10	24	18	0
Time (s)	μ_{all}	0.03	0.07	0.30	N/A	**0.02**
	μ_{sub}	**0.01**	0.03	0.09	1.70	**0.01**
	Wins	25	6	6	4	**31**
	NAs	0	0	0	13	0
2-way coverage	μ_{all}	**99.90**	99.50	99.70	N/A	97.96
C$_2$ (%)	μ_{sub}	**99.88**	99.35	99.61	99.84	97.41
	Wins	**35**	16	20	26	1
APCC	μ_{all}	80.00	**80.69**	80.20	N/A	78.58
A$_2$ (%)	μ_{sub}	77.29	**78.04**	77.48	77.22	75.85
	Wins	4	**42**	4	2	4
3-way coverage	μ_{all}	73.49	76.70	76.90	N/A	**77.49**
C$_3$ (%)	μ_{sub}	69.10	72.32	72.44	**73.75**	72.78
	Wins	2	3	**30**	24	2
APCC	μ_{all}	37.68	40.29	40.26	N/A	**40.39**
A$_3$ (%)	μ_{sub}	31.88	34.28	34.18	**35.07**	34.13
	Wins	3	2	1	**27**	**27**
4-way coverage	μ_{all}	41.05	44.07	44.71	N/A	**47.19**
C$_4$ (%)	μ_{sub}	36.44	39.14	39.62	40.85	**41.49**
	Wins	2	5	15	**20**	6
APCC	μ_{all}	11.40	12.58	**12.79**	N/A	12.17
A$_4$ (%)	μ_{sub}	7.43	8.26	8.39	8.79	**8.97**
	Wins	5	4	6	15	**32**
	NAs	2	2	2	**13**	2

We report the average, denoted by μ_{all},[6] for each of the four evaluation metrics by each of the five methods. There are cases where EC could not generate a test suite and cases where we could not compute 4-way coverage for all methods. Hence we also show the average, denoted by μ_{sub}, of results for the subset of models, denoted by *sub*, that all the five methods could handle. We also report the number of "Wins", i. e., how often the method obtains the best result among others. Ties are counted as a win for all tied methods.

Figure 2 presents the box plots for the results of C$_t$ and APCC$_t$ with $2 \leq t \leq 4$ by CT, DC$_{CD}$, DC$_{HD}$, EC, and DT for all models. Figure 3 presents the box plots for test suite sizes and test generation times for all models, and also for the models where all constraints are removed. Each box plot shows the mean (triangle in the box), median (thick horizontal line), the first/third quartiles (hinges), and highest/lowest values within 1.5 × the inter-quartile range of the hinge (whiskers). "Wins" and "NAs" for each method are also attached to the box plot.

RQ1. Can DICOT deliver higher $t'(> t)$-way coverage? If so, how big is the improvement and the computational overhead, compared with traditional t-way test generation?
Ans. *Yes. We observe that DICOT improves 3-way and 4-way coverage over traditional 2-way test generation CT which does not consider distances or the higher-way coverage. Compared to CT, the test suites by DICOT also achieve higher APCC, i. e., can quickly obtain higher t-way coverage for all $2 \leq t \leq 4$ with small computation overhead.*

From Figure 2 and Figure 3, we conclude that both DC$_{HD}$ and DC$_{CD}$ obtain higher 3-way and 4-way coverage compared to CT while the sizes of test suites are not significantly affected. In detail, Table VII shows that our DC$_{HD}$ (resp. DC$_{CD}$) improves 3-way coverage by 5.26 % (resp. 3.74 %), APCC for $t = 3$ by 7.74 % (resp. 6.45 %), 4-way coverage by 10.94 % (resp. 7.14 %), and APCC for $t = 4$ by 16.38 % (resp. 11.97 %) over CT on average for the given benchmarks. This improvement is not minor; for example, for benchmark Apache, 3-way coverage over CT is improved by 4.69 % and 2.58 %, resp., using DC$_{HD}$ and DC$_{CD}$, which indicates that they cover 347,053 and 190,750 more 3-tuples of parameter-values compared to CT in the first 34 test cases. We confirmed that the improvements by DC$_{HD}$ and DC$_{CD}$ over CT for 3-way and 4-way coverage are all significant with $p < 0.01$ by the Wilcoxon signed-rank test [40].

As for the sizes of test suites, DC$_{HD}$ and DC$_{CD}$ generated 0.63 and 1.44 more test cases than CT in average, but the tendency is unclear as sometimes (e. g., benchmarks 9 and 18) DC$_{HD}$ generates smaller test suites than CT.

RQ2. How effective and efficient is DICOT compared with the EC approach?
Ans. *DC$_{HD}$ and DC$_{CD}$ improve $t'(> t)$-way coverage with much smaller test generation time compared to EC.*

[6]We use the *geometric mean* to avoid emphasizing larger benchmarks over smaller ones, which would be the case with the arithmetic mean.

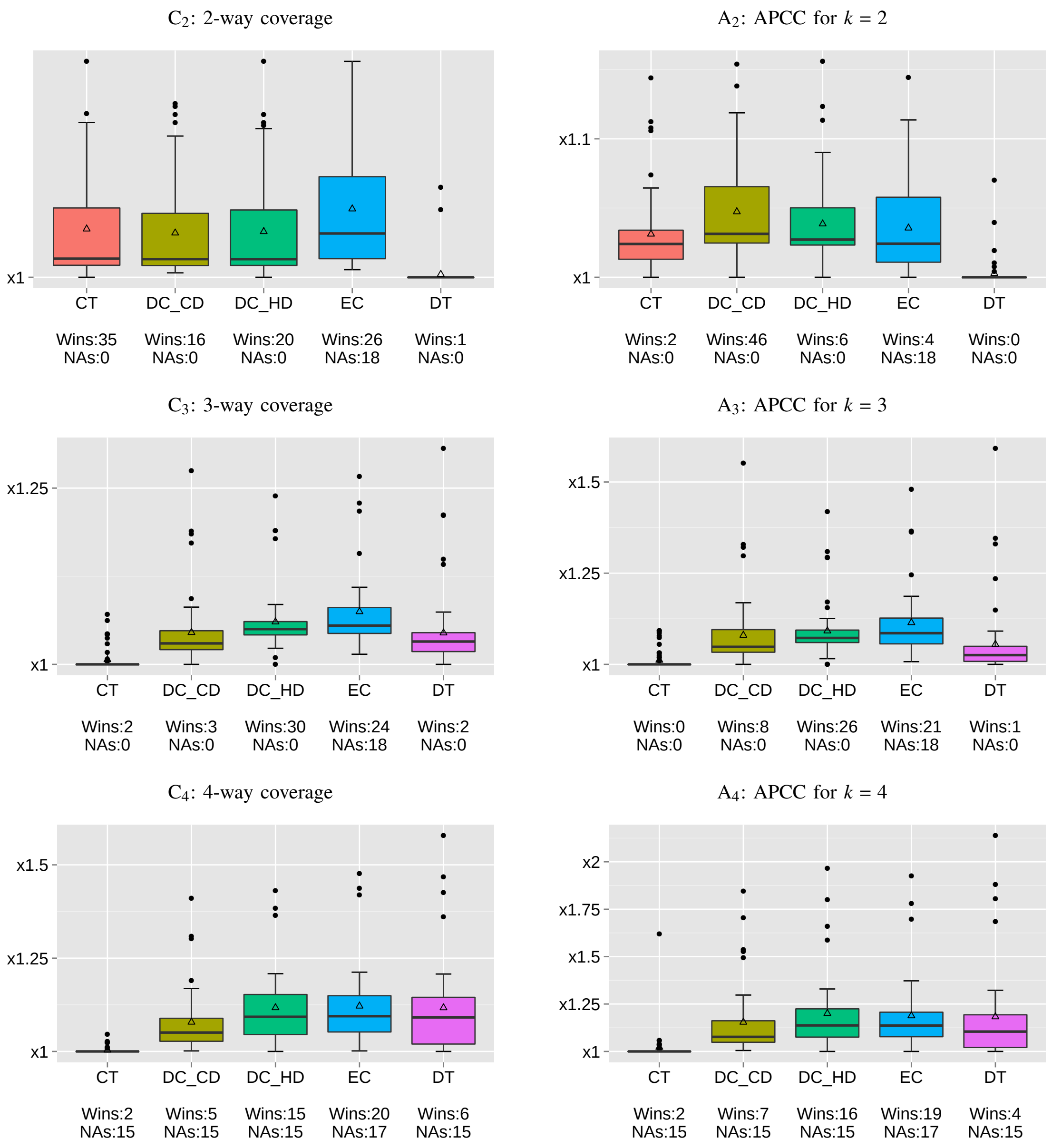


Fig. 2. Comparison of t-way coverage and APCC with $2 \le k \le 4$ of test suites by the five methods for benchmark models. The ratio over the worst results among all methods for each benchmark is plotted.

Experimental results show the efficiency of our approach DC$_{HD}$ and DC$_{CD}$ compared to EC. We observe that EC achieves slightly higher 3-way and 4-way coverage compared to DC$_{HD}$ and DC$_{CD}$, but requires much longer test generation time. For the models that EC could handle, EC improves 3-way coverage by 6.31 % (DC$_{HD}$ by 5.52 %) and 4-way coverage by 11.48 % (DC$_{HD}$ by 10.91 %) over CT. However, EC could not finish test generation for 17 (resp. 13) models among the 55 models with (resp. without) constraints in an hour. In addition, the test generation time for *sub* with (resp. without) constraints for EC was more than 11 (resp. 18) times of those for DC$_{HD}$ and DC$_{CD}$ on average.

RQ3. How effective and efficient is DICOT compared with the DT approach?
Ans. *Compared to DT, our approach, especially DC$_{HD}$, effectively generates small-sized t-way test suites that improve $t'(> t)$-way coverage for SUT models with constraints.*

Experimental results indicate that DT requires extremely large test suites to satisfy 100 % 2-way coverage although it generates the test suites very fast compared to other methods. In detail, DT generates 131.16 % (127.78 %) more test cases for pairwise tests with 82.42 % (81.40 %) less times over DC$_{HD}$ (DC$_{CD}$) on average.

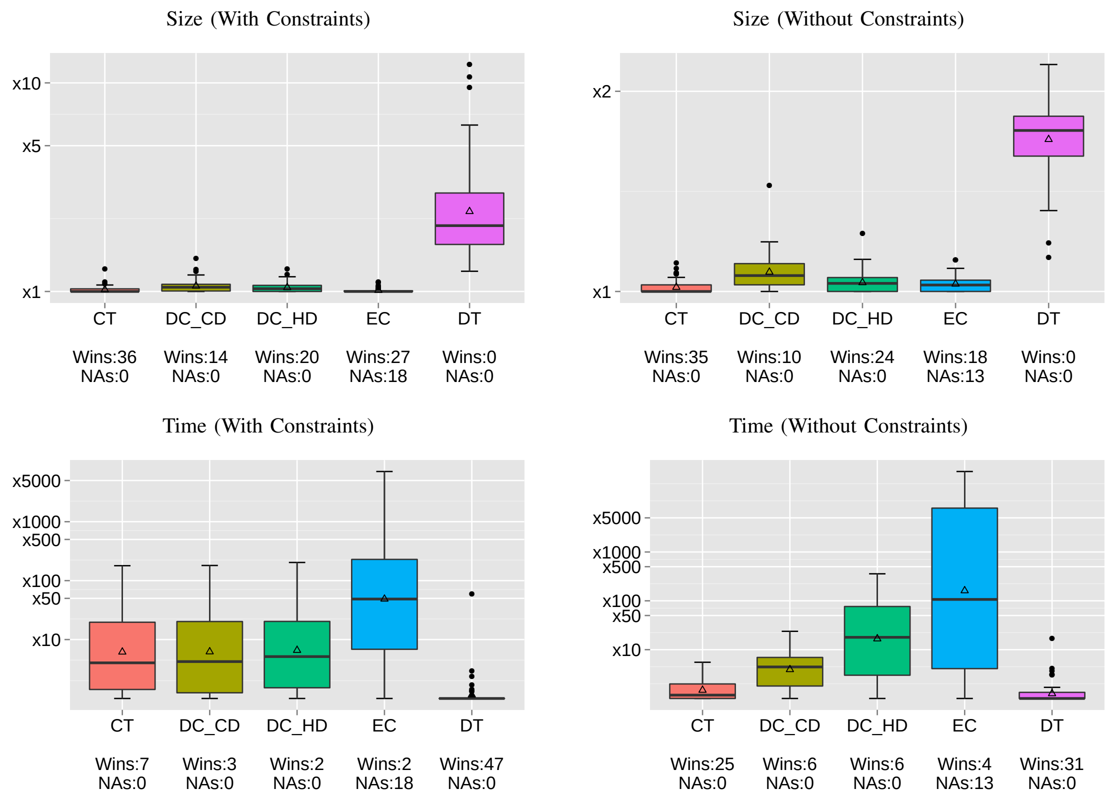


Fig. 3. Comparison of the test suite sizes and generation times for benchmark models with/without constraints. The ratio over the best results among all methods for each benchmark is plotted.

From Table VII and Figure 2, we observe that our approach achieves higher t-way coverage and APCC for $t = 2$ and 3 compared to DT. For $t = 4$, DC$_{HD}$ obtains higher 4-way coverage and APCC but DC$_{CD}$ obtains lower ones. In detail, DT improves the 3-way coverage by 3.65 % (DC$_{HD}$ by 5.26 % and DC$_{CD}$ by 3.74 %), APCC for $t = 3$ by 3.94 % (DC$_{HD}$ by 7.74 % and DC$_{CD}$ by 6.45 %), the 4-way coverage by 10.63 % (DC$_{HD}$ by 10.94 % and DC$_{CD}$ by 7.14 %), and APCC for $t = 4$ by 14.03 % (DC$_{HD}$ by 16.38 and DC$_{CD}$ by 11.97 %) over CT. For the number of Wins, DC$_{HD}$ and DC$_{CD}$ are in total superior to DT. In detail, for 2-way, 3-way, and 4-way coverage, and the relative APCC, DC$_{HD}$ wins 20, 30, 15, 6, 26, and 16 times and DC$_{CD}$ wins 16, 3, 5, 46, 8, and 7 times while DT wins 1, 2, 6, 0, 1, and 4 times. Interestingly, DT obtains much better results w. r. t. higher strength for models without constraints, but does not for models with constraints.

RQ4. How different are the performance when using Hamming distance and chi-square distance in DICOT?
Ans. *Using Hamming distance is slightly better to improve test effectiveness, and using chi-square distance is competitive for reducing computational overhead.*

From the experimental results, we observe that DC$_{HD}$, which uses Hamming distance, achieves avg. 1.18–1.59 % higher 3-way and 4-way coverage over DC$_{CD}$, which uses chi-square distance, while DC$_{CD}$ requires shorter test generation time. For the benchmarks with constraints, DC$_{CD}$ was avg. 5.48 % faster than DC$_{HD}$, although most of the computation time is consumed for constraint handling. For the benchmarks without constraints, DC$_{CD}$ was avg. 76.67 % faster than DC$_{HD}$.

D. Threats to Validity

In order to evaluate the efficiency of the proposed approach, we compared the overhead of computing distance for DICOT and that of enumerating $(t + 1)$-tuples of parameter-values newly covered for EC. In our implementation, each assignment is checked using a SAT solver, and it could take considerable time for large models. One might suspect that the difference of test generation times by DICOT and EC shown in the experimental results could be smaller if constraint checking was not time-consuming.

To assess the validity, we also showed the experimental results for models without constraints. From the results, we can see that the difference of the computational overhead by DICOT and that by EC is more significant. Further studies by integrating our approach to actual combinatorial testing tools could be helpful to reduce this threat.

V. Related Work

There have been a number of techniques and tools that generate t-way test suites, including greedy algorithms [14],

[16], [31], heuristic search [19], [26], [39], and SAT-based approaches [24], [41]. These techniques, however, only ensure 100 % t-way coverage for given t, and do not try to improve $t'(> t)$-way coverage.

Chen and Zhang [9], as far as we know, are the first to focus on the $t'(> t)$-way coverage of t-way testing. In order to achieve higher $t'(> t)$-way coverage, they adopt the intuitive approach that enumerates the number of parameter-value $(t + 1)$-tuples covered by an alternative test case. They showed that using several unconstrained small SUT models whose number of parameters is up to 20, the improvement rate of 3-way coverage by their method is from 2% to 4% over the original pairwise test suites. For the same objective, we adopted increasing the distance between a new test case and the previous test cases. We showed that, using constrained/unconstrained large SUT models whose number of parameters is up to around 200, our approach can improve 3-way coverage by approx. 5% and 4-way coverage by approx. 11% over traditional pairwise test suites, with lower computational overhead.

Our work is inspired by *Adaptive Random Testing (ART)* [11], [10] which takes the notion of distance into account in random testing; e. g., FSCS-ART [11] randomly generates a certain number of test case candidates, and picks one that has the maximum distance from already generated test cases. ART is to improve random testing on failure detection effectiveness [10], but do not guarantee t-way coverage.

Henard et al. [23] pointed out that t-way test generation with higher interaction strength ($t > 2$) is not scalable for large SUT models with constraints even when parameters have only two values (their targets, *Software Product Lines (SPLs)*, can be seen as SUT models whose parameters have only Boolean values). This is because t-way testing has to enumerate all possible t-way combinations, whose number is exponential in t. To overcome the problem, they proposed random-based and search-based algorithms to generate test cases, employing the concept of DT: their algorithms consider the distance between configurations and do not compute t-way combinations. They used the distance metric based on the total Jaccard distance among SPL configurations.

Bryce et al. [8] proposed *adaptive distance-based testing*, which constructs test cases as follows: to generate one test case, it assigns to every parameter (in an arbitrary order) a value that makes the generated test case as distant as possible from previous test cases. They used either the number of new parameter-value t-tuples or the Hamming distance as a distance metric, while we consider both in an integrated way in our approach.

Huang et al. [25] integrated the notion of t-way coverage into ART, and used the number of newly covered parameter-value t-tuples as a distance metric. Their method randomly generates test case candidates, and chooses a test case with the maximum distance, i. e., the maximum number of newly covered parameter-value t-tuples.

The approaches by Bryce et al. [8] and Huang et al. [25] both interpret the number of new parameter-value t-tuples as the test case distance to generate t-way test suites, and do not care about $t'(> t)$-way coverage. On the other hand, we integrate increasing the distance and increasing the new parameter-value t-tuples so as to improve $t'(> t)$-way coverage.

We previously proposed a t-way test generation [12] to construct test suites where higher priority test cases and parameter-values appear early and frequently for SUT models whose parameter-values are prioritized. The previous method considers increasing both the coverage called *weight coverage* and the metric called *KL divergence*. Its concept of integration is similar but the purpose, target SUT models, and the metrics are different from this newly proposed method.

VI. Conclusion and Future Work

In this paper, we proposed a distance-integrated CT construction approach, called DICOT, where we increase not only the number of new combinations of parameter-values but also the distance between test cases. The contribution of this paper is that we propose the first CT generation approach that takes into account both the CT criterion, i. e., the number of new parameter-value t-tuples, and the test case distance, e. g., Hamming distance or a modified chi-square distance, in a hierarchical integration.

We applied our approach to a traditional greedy algorithm for CT generation and investigated the effectiveness and the efficiency of our approach using a number of practical SUT models with constraints. The experimental results show that our distance-integrated test case generation achieves higher $t'(> t)$-way coverage, and hence, can be effective in detecting failures that are triggered by the interaction of more than t parameters. Moreover, the required computational overhead is smaller than the intuitive approach of Chen and Zhang [9].

Future work includes investigating other distance metrics to determine test case dissimilarity for CT. In this paper, we use two metrics, Hamming distance and a modified chi-square distance, but there are other dissimilarity metrics for binary data [13] or categorical data [6]. Those metrics could be adopted in our approach in order to define the distance of test cases for a discrete and finite CT domain.

Another future work is to investigate the correlation between t'-way coverage and fault detection effectiveness. On the one hand, the effectiveness of t-way testing has been shown by a number of empirical studies [4], [17], [20], [27], [30], [42]. On the other hand, coverage-based software testing could raise an open question whether the coverage is actually useful for real fault detection [21]. Evaluating the improvement of fault detection effectiveness by DICOT is further work.

Acknowledgments

The authors would like to thank Tatsuhiro Tsuchiya and anonymous referees for their helpful comments and suggestions to improve this paper. This work was partly supported by JSPS KAKENHI Grant Number 16K12415.

References

[1] Radio Technical Commission for Aeronautics (RTCA) standards, DO-178B - Software considerations in airborne systems and equipment certification, December 1992.

[2] International Standardization Organization, ISO26262: Road vehicles - Functional safety, November 2011.

[3] A. Biere. Picosat essentials. *Journal on Satisfiability, Boolean Modeling and Computation*, 4(2-4):75–97, 2008.

[4] M. N. Borazjany, L. Yu, Y. Lei, R. Kacker, and R. Kuhn. Combinatorial testing of ACTS: A case study. In *Proc. of the 5th International Conference on Software Testing, Verification and Validation (ICST)*, pages 591–600. IEEE, 2012.

[5] M. N. Borazjany, L. Yu, Y. Lei, R. N. Kacker, and D. R. Kuhn. Combinatorial testing of ACTS: A case study. In *Proc. of the 5th International Conference on Software Testing, Verification and Validation (ICST)*, pages 591–600, 2012.

[6] S. Boriah, V. Chandola, and V. Kumar. Similarity measures for categorical data: A comparative evaluation. In *Proc. of the 8th international conference on data mining (SDM'08)*, pages 243–254. SIAM, 2008.

[7] R. C. Bryce, C. J. Colbourn, and M. B. Cohen. A framework of greedy methods for constructing interaction test suites. In *Proc. of the 27th International Conference on Software Engineering (ICSE)*, pages 146–155. IEEE, 2005.

[8] R. C. Bryce, C. J. Colbourn, and D. R. Kuhn. Finding interaction faults adaptively using distance-based strategies. In *Proc. of the 18th International Conference on Engineering of Computer Based Systems (ECBS)*, pages 4–13. IEEE, 2011.

[9] B. Chen and J. Zhang. Tuple density: a new metric for combinatorial test suites (NIER track). In *Proc. of the 33rd International Conference on Software Engineering (ICSE)*, pages 876–879. IEEE, 2011.

[10] T. Y. Chen, F. C. Kuo, R. G. Merkel, and T. Tse. Adaptive random testing: The art of test case diversity. *Journal of Systems and Software*, 83(1):60–66, 2010.

[11] T. Y. Chen, H. Leung, and I. K. Mak. Adaptive random testing. In *Proc. of the 9th Asian Computing Science Conference (ASIAN), Lecture Notes in Computer Science*, volume 3321, pages 320–329, 2004.

[12] E. Choi, T. Kitamura, C. Artho, A. Yamada, and Y. Oiwa. Priority integration for weighted combinatorial testing. In *Proc. of the 39th Annual Computer Software and Applications Conf. (COMPSAC)*, volume 2, pages 242–247. IEEE, 2015.

[13] S. S. Choi, S. H. Cha, and C. C. Tappert. A survey of binary similarity and distance measures. *Journal of Systemics, Cybernetics and Informatics*, 8(1):43–48, 2010.

[14] D. M. Cohen, S. R. Dalal, M. L. Fredman, and G. C. Patton. The AETG system: An approach to testing based on combinatiorial design. *IEEE Trans. Software Eng.*, 23(7):437–444, 1997.

[15] M. B. Cohen, M. B. Dwyer, and J. Shi. Constructing interaction test suites for highly-configurable systems in the presence of constraints: A greedy approach. *IEEE Trans. Software Eng.*, 34(5):633–650, 2008.

[16] J. Czerwonka. Pairwise testing in the real world: Practical extensions to test-case senarios. In *Proc. of the 24th Pacific Northwest Software Quality Conference*, pages 419–430. Citeseer, 2006.

[17] 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. of the International Conference on Software Engineering (ICSE)*, pages 285–294. IEEE, 1999.

[18] D. M. Endres and J. E. Schindelin. A new metric for probability distributions. *IEEE Trans. Information Theory*, 49(7):1858–1860, 2003.

[19] B. J. Garvin, M. B. Cohen, and M. B. Dwyer. Evaluating improvements to a meta-heuristic search for constrained interaction testing. *Empirical Software Engineering*, 16(1):61–102, 2011.

[20] L. S. G. Ghandehari, M. N. Borazjany, Y. Lei, R. Kacker, and D. R. Kuhn. Applying combinatorial testing to the Siemens suite. In *Proc. of the 6th International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pages 362–371. IEEE, 2013.

[21] A. Groce, M. A. Alipour, and R. Gopinath. Coverage and its discontents. In *Proc. the ACM Symposium on New Ideas, New Paradigms, and Reflections on Programming & Software*, pages 255–268. ACM, 2014.

[22] R. Hamming. Error Detecting and Error Correcting Codes. *Bell System Technical Journal*, 29:147–160, 1950.

[23] C. Henard, M. Papadakis, G. Perrouin, J. Klein, P. Heymans, and Y. Le Traon. Bypassing the combinatorial explosion: Using similarity to generate and prioritize t-wise test configurations for software product lines. *IEEE Trans. Software Eng.*, 40(7):650–670, 2014.

[24] B. Hnich, S. D. Prestwich, E. Selensky, and B. M. Smith. Constraint models for the covering test problem. *Constraints*, 11(2-3):199–219, 2006.

[25] R. Huang, X. Xie, T. Y. Chen, and Y. Lu. Adaptive random test case generation for combinatorial testing. In *Proc. of the 36th Annual Computer Software and Applications Conf. (COMPSAC)*, pages 52–61. IEEE, 2012.

[26] Y. Jia, M. B. Cohen, M. Harman, and J. Petke. Learning combinatorial interaction testing strategies using hyperheuristic search. In *Proc. of the 37th International Conference on Software Engineering (ICSE)*, pages 540–550. IEEE/ACM, 2015.

[27] R. Krishnan, S. M. Krishna, and P. S. Nandhan. Combinatorial testing: learnings from our experience. *ACM SIGSOFT Software Engineering Notes*, 32(3):1–8, 2007.

[28] D. R. Kuhn, R. N. Kacker, and Y. Lei. *Introduction to combinatorial testing*. CRC Press, 2013.

[29] D. R. Kuhn, I. D. Mendoza, R. N. Kacker, and Y. Lei. Combinatorial coverage measurement concepts and applications. In *Proc. of the 6th Software Testing, Verification and Validation Workshops (ICSTW)*, pages 352–361. IEEE, 2013.

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

[31] Y. Lei, R. N. Kacker, D. R. Kuhn, V. Okun, and J. Lawrence. IPOG: A general strategy for t-way software testing. In *Proc. of the 14th International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS)*, pages 549–556. IEEE, 2007.

[32] J. Lin, C. Luo, S. Cai, K. Su, D. Hao, and L. Zhang. TCA: An efficient two-mode meta-heuristic algorithm for combinatorial test generation. In *Proc. of the 30th International Conference on Automated Software Engineering (ASE)*, pages 494–505. ACM/IEEE, 2015.

[33] Y. Malaiya. Antirandom testing: getting the most out of black-box testing. In *Proc. of the 6th International Symposium on Software Reliability Engineering (ISSRE)*, pages 86–95. IEEE, 1995.

[34] C. Nie and H. Leung. A survey of combinatorial testing. *ACM Computing Surveys*, 43(2):11, 2011.

[35] K. Pearson. X. On the criterion that a given system of deviations from the probable in the case of a correlated system of variables is such that it can be reasonably supposed to have arisen from random sampling. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 50(302):157–175, 1900.

[36] J. Petke, M. B. Cohen, M. Harman, and S. Yoo. Practical combinatorial interaction testing: Empirical findings on efficiency and early fault detection. *IEEE Trans. Software Eng.*, 41(9):901–924, 2015.

[37] X. Qu and M. B. Cohen. A study in prioritization for higher strength combinatorial testing. In *Proc. of the 6th Software Testing, Verification and Validation Workshops (ICSTW)*, pages 285–294. IEEE, 2013.

[38] I. Segall, R. Tzoref-Brill, and E. Farchi. Using binary decision diagrams for combinatorial test design. In *Proc. of the 2011 International Symposium on Software Testing and Analysis (ISSTA)*, pages 254–264. ACM, 2011.

[39] T. Shiba, T. Tsuchiya, and T. Kikuno. Using artificial life techniques to generate test cases for combinatorial testing. In *Proc. of the 28th Annual International Computer Software and Applications Conf. (COMPSAC)*, pages 72–77. IEEE, 2004.

[40] F. Wilcoxon. Individual comparisons by ranking methods. *Biometrics Bulletin*, 1(6):80–83, 1945.

[41] A. Yamada, T. Kitamura, C. Artho, E. Choi, Y. Oiwa, and A. Biere. Optimization of combinatorial testing by incremental SAT solving. In *Proc. of the 8th International Conference on Software Testing, Verification and Validation (ICST)*, pages 1–10. IEEE, 2015.

[42] Z. Zhang, X. Liu, and J. Zhang. Combinatorial testing on ID3v2 tags of MP3 files. In *Proc. of the 5th International Conference on Software Testing, Verification and Validation (ICST)*, pages 587–590. IEEE, 2012.

[43] Z. Zhang, J. Yan, Y. Zhao, and J. Zhang. Generating combinatorial test suite using combinatorial optimization. *Journal of Systems and Software*, 2014(98):191–207, 2014.

[44] Sat4j, Available: http://www.sat4j.org/.

Appendix

In Table VIII, we show the sizes and the numbers of possible 2-tuples and 3-tuples of parameter-values for all benchmark SUT models we used for experiments. The size of an SUT model is expressed as $|P|; g_1^{k_1} g_2^{k_2} \ldots g_n^{k_n}$, i. e., for each i there are k_i parameters with g_i values and $|P|$ is the number of parameters. The size of constraint is expressed as $l; h_1^{l_1} h_2^{l_2} \ldots h_m^{l_m}$, i. e., its conjunctive normal form (CNF) has l variables and h_j clauses with l_j literals for each j.

TABLE VIII
Benchmark SUT models.

No.	Name	Model size	Constraint size	# of possible t-tuples t=2	t=3
1	Banking1	5 ; $3^4 4^1$	16 ; 5^{112}	102	324
2	Banking2	15 ; $2^{14} 4^1$	21 ; 2^3	473	4,290
3	CommProtocol	11 ; $2^{10} 7^1$	27 ; $2^{10} 3^{10} 4^{12} 5^{24} 6^{30} 7^{30} 8^{12}$	285	1,650
4	Concurrency	5 ; 2^5	10 ; $2^4 3^1 5^2$	36	55
5	Healthcare1	10 ; $2^6 3^2 5^1 6^1$	18 ; $2^3 3^{18}$	361	2,535
6	Healthcare2	12 ; $2^5 3^6 4^1$	32 ; $2^1 3^6 5^1 8$	466	4,076
7	Healthcare3	29 ; $2^{16} 3^6 4^5 5^1 6^1$	77 ; 2^{31}	3,092	74,274
8	Healthcare4	35 ; $2^{13} 3^{12} 4^6 5^2 6^1 7^1$	99 ; 2^{22}	5,707	191,398
9	Insurance	14 ; $2^6 3^1 5^1 6^2 11^1 13^1 17^1 31^1$	—	4,573	113,592
10	NetworkMgmt	9 ; $2^2 4^1 5^3 10^2 11^1$	54 ; 2^{20}	1,228	15,370
11	ProcessorComm1	15 ; $2^3 3^6 4^6$	48 ; 2^{13}	1,058	14,229
12	ProcessorComm2	25 ; $2^3 3^{12} 4^8 5^2$	81 ; $1^4 2^{121}$	2,525	53,228
13	Services	13 ; $2^3 3^4 5^2 8^2 10^2$	62 ; $3^{386} 4^2$	1,819	30,031
14	Storage1	4 ; $2^1 3^1 4^1 5^1$	14 ; 4^{95}	53	71
15	Storage2	5 ; $3^4 6^1$	—	126	432
16	Storage3	15 ; $2^9 3^1 5^3 6^1 8^1$	48 ; $2^{38} 3^{10}$	1,020	11,840
17	Storage4	20 ; $2^5 3^7 4^1 5^2 6^2 7^1 10^1 13^1$	31 ; 2^{24}	3,491	86,153
18	Storage5	23 ; $2^5 3^8 5^3 6^2 8^1 9^1 10^2 11^1$	106 ; 2^{151}	5,342	157,950
19	SystemMgmt	10 ; $2^5 3^4 5^1$	27 ; $2^{13} 3^4$	310	1,982
20	Telecom	10 ; $2^5 3^1 4^2 5^1 6^1$	29 ; $2^{11} 3^1 4^9$	440	3,431
21	Spins	18 ; $2^{13} 4^5$	38 ; 2^{13}	979	12,835
22	Spinv	55 ; $2^{42} 3^2 4^{11}$	133 ; $2^{47} 3^2$	8,741	369,976
23	Apache	172 ; $2^{158} 3^8 4^4 5^1 6^1$	366 ; $2^3 3^1 4^2 5^1$	66,927	8,085,958
24	Gcc	199 ; $2^{189} 3^{10}$	82 ; $2^{37} 3^3$	82,770	11,131,894
25	Bugzilla	52 ; $2^{49} 3^1 4^2$	108 ; $2^4 3^1$	5,818	202,683
26	bm1	97 ; $2^{86} 3^3 4^1 5^5 6^2$	43 ; $2^{20} 3^3 4^1$	23,876	1,704,243
27	bm2	94 ; $2^{86} 3^3 4^3 5^1 6^1$	40 ; $2^{19} 3^3$	20,331	1,339,412
28	bm3	29 ; $2^{27} 4^2$	14 ; $2^9 3^1$	1,838	34,728
29	bm4	58 ; $2^{51} 3^4 4^2 5^1$	27 ; $2^{15} 3^2$	7,530	298,517
30	bm5	174 ; $2^{155} 3^7 4^3 5^5 6^4$	86 ; $2^{32} 3^6 4^1$	76,259	9,816,481
31	bm6	77 ; $2^{73} 4^3 6^1$	32 ; $2^{26} 3^4$	11,382	559,764
32	bm7	30 ; $2^{29} 3^1$	12 ; $2^{13} 3^2$	1,567	27,669
33	bm8	119 ; $2^{109} 3^2 4^2 5^3 6^3$	55 ; $2^{32} 3^4 4^1$	33,680	2,865,125
34	bm9	61 ; $2^{57} 3^1 4^1 5^1 6^1$	31 ; $2^{30} 3^7$	6,835	259,060
35	bm10	147 ; $2^{130} 3^6 4^5 5^2 6^4$	62 ; $2^{40} 3^7$	52,659	5,619,635
36	bm11	96 ; $2^{84} 3^4 4^2 5^2 6^4$	48 ; $2^{28} 3^4$	23,636	1,676,265
37	bm12	147 ; $2^{136} 3^4 4^3 4^1 6^3$	65 ; $2^{23} 3^4$	49,522	5,131,693
38	bm13	133 ; $2^{124} 3^4 4^1 5^2 6^2$	58 ; $2^{22} 3^4$	38,862	3,564,693
39	bm14	92 ; $2^{81} 3^5 4^3 6^3$	45 ; $2^{13} 3^2$	20,544	1,361,650
40	bm15	58 ; $2^{50} 3^4 4^1 5^2 6^1$	29 ; $2^{20} 3^2$	8,388	349,991
41	bm16	87 ; $2^{81} 3^3 4^2 6^1$	36 ; $2^{30} 3^4$	14,600	815,386
42	bm17	137 ; $2^{128} 3^3 4^2 5^1 6^3$	60 ; $2^{25} 3^4$	43,390	4,199,845
43	bm18	141 ; $2^{127} 3^2 3^3 4^6 6^2$	76 ; $2^{23} 3^4 4^1$	50,128	5,222,583
44	bm19	197 ; $2^{172} 3^9 4^9 5^3 6^4$	95 ; $2^{38} 3^5$	98,778	14,485,184
45	bm20	158 ; $2^{138} 3^4 4^5 5^4 6^7$	80 ; $2^{42} 3^6$	64,620	7,647,389
46	bm21	85 ; $2^{76} 3^3 4^2 5^1 6^3$	37 ; $2^{40} 3^6$	15,442	885,299
47	bm22	79 ; $2^{72} 3^4 4^1 6^2$	32 ; $2^{31} 3^4$	13,405	714,902
48	bm23	27 ; $2^{25} 3^1 6^1$	16 ; $2^{13} 3^2$	1,495	25,363
48	bm24	119 ; $2^{110} 3^2 5^3 6^4$	55 ; $2^{25} 3^4$	34,204	2,932,980
50	bm25	134 ; $2^{118} 3^6 4^2 5^2 6^6$	66 ; $2^{23} 3^3 4^1$	46,968	4,728,180
51	bm26	95 ; $2^{87} 3^1 4^3 5^4$	45 ; $2^{28} 3^4$	20,921	1,396,703
52	bm27	62 ; $2^{55} 3^2 4^2 5^1 6^2$	27 ; $2^{17} 3^3$	9,714	436,049
53	bm28	194 ; $2^{167} 3^{16} 4^2 5^3 6^6$	98 ; $2^{31} 3^6$	96,599	14,013,342
54	bm29	144 ; $2^{134} 3^7 5^3$	62 ; $2^{19} 3^3$	45,839	4,570,949
55	bm30	79 ; $2^{73} 3^3 4^3$	32 ; $2^{20} 3^2$	12,453	640,511