Generating t-way Test Suite in the Presence of Constraints

*AbdulRahman Al-Sewari[#1], Kamal Z. Zamli[#2], and Basem Al-Kazemi[#3]

[#1,#2]*Faculty of Computer Systems and Software Engineering, Universiti Malaysia Pahang, 26300 Kuantan, Pahang, Malaysia*

[#3]*College of Computer and Information Systems, Umm Al-Qura University, Makkah, Kingdom of Saudi Arabia*

*Corresponding Author: [#1]alsewari@ump.edu.my

Abstract—**Interaction (t-way) testing is a common sampling strategy to minimize combinatorial test data from large configuration space based on the defined interaction strength (t). Here, all t-way strategies generate the t-way test suite with the aim to cover every possible combination produced by the interacting parameters (or also known as tuples). In many systems under test (SUT), there are some known combinations that are impossible to occur based on the requirements set to the system. These combinations (termed constraints) have to be excluded from the final test suite. This paper describes the generation of t-way test suite using the Late Acceptance Hill Climbing based Strategy (LAHC) in the presence of constraints. Our benchmarking results have been promising as LAHC gives competitive results in many constraints configurations considered.**

I. Introduction

Given potentially large possible input parameters, exhaustive testing of any typical software is practically impossible. As such, many sampling based strategies (such as random testing[1], each-choice and base-choice[2], and anti-random[3]) have been proposed in the literature to help test engineers select a subset of test cases (i.e. from the exhaustive testing) that would maximize the probability of fault detection. Although useful, the aforementioned strategies are not designed to tackle faults due to interaction. As such, their applicability is deemed limited to certain types of faults.

Addressing these issues, researchers have turned into t-way strategies [4] whereby t indicates the interaction strength. Here, all t-way strategies generate the t-way test suite with the aim to cover every possible combination produced by the interacting parameters (or also known as tuples). In many systems under test (SUT), there are some known combinations that are impossible to occur based on the requirements set to the system. These combinations (termed constraints) have to be excluded from the final test suite.

While many t-way strategies have been proposed in literature for the past 20 years (e.g. GTWay[5, 6], MIPOG[7-10], TConfig [11, 12] and TCG[13]), few strategies have sufficiently considered constraints during test generation process. In fact, HSS [14], PICT [15], mAETG[16], SA[16] and TestCover [17] are amongst the few known t-way strategies that address constraints issues.

Complementing existing work, this paper describes a novel strategy, called LAHC, based on Late Acceptance Hill Climbing Algorithm that is capable of generating the t-way test suite in the presence of constraints. The main contribution of the work is that it is the first constraints supported t-way strategy that is developed based on the Late Acceptance Hill Climbing Algorithm. Our benchmarking results have been promising as LAHC gives competitive results in many constraints configurations considered.

The rest of the paper is organized as follows. Section II highlights covering array notation. Section III gives the problem definition model. Section IV describes the relevant related work. Section V highlights the general Late Acceptance Hill Climbing. Section VI elaborates on our strategy developed based on the Late Acceptance Hill Climbing Algorithm. Section VII describes our evaluation experiments. Finally, Section VIII gives our conclusion and future work.

II. Covering Array Notations

Mathematically, t-way interaction test suite can be abstracted using the covering array (CA) notations. Normally, the CA has four parameters; N, t, p, and v (i.e., CA (N, t, v^p). Here, the symbols p, v, and t are used to refer to number of parameters, values, and interaction strength for the CA, respectively. For example, CA (9, 2, 3^4) represents a test suite consisting of 9×4 arrays (i.e., the rows represent the size of test cases (N), and the column represents the parameter (p)). In this case, the test suite also covers two-way interaction for a system with four 3-value parameters.

Similar to CA, mixed covering array (MCA) has three parameters; N, t, and Configuration (C) (i.e., MCA (N, t, C)). In addition to N and t that carry the same meaning as in CA, MCA adopts a new symbol, C. Consistent with earlier given notations, C represents the parameters and values of each configuration in the following format: $v_1^{\ p1}\ v_2^{\ p2} \ldots\ v_n^{\ pn}$ indicating that there are p1 parameters with v_1 values, p_2 parameters with v_2 values, and so on. For example, MCA (1265, 4, $10^2\ 4^1\ 3^2\ 2^7$) indicates the test size of 1265 that covers four-way interaction. Here, the configuration takes 12 parameters: two 10-value parameters, one 4-value parameter, two 3-value parameters, and seven 2-value parameters.

To cater for constraints covering array (CCA) or mixed-constraints covering array (MCCA); a new variable called forbidden (*F*) interaction is introduced to represent the set of disallowed interactions (i.e., CA (*N, t,* v^p*, F*) or MCCA (*N, t, C, F*)). Here, *F* takes the following format $\{F_{a,b}\}$ where a indicates the p_{th} parameter and b indicates the v_{th} value are within the list of constraints. For example, consider CCA (10, 2, 3^3, *F*) where $F = \{F_{1,1}, F_{3,1}\}$. In this case, the CCA indicates the test size of 10 for pairwise interaction of three 3-value parameters with constraints pair interaction elements from parameter 1 and value 1, as well as parameter 3 and value 1.

III. Problem Definition Model

To illustrate the problem of t-way testing and constraints, a simplified pizza ordering system will be elaborated based on the example in [14]. The simplified pizza system takes five parameters as follows (see Table I).

TABLE I
SIMPLIFIED PIZZA ORDERING SYSTEM

Pizza Type (P1)	Crust (P2)	Toppings (P3)	Size (P4)	Delivery (P5)
Vegetarian Cheese	Thin Crust	Roasted Chicken	Large	Eat In
Meat Lover	Extra Thick	Ground Beef	Medium	Take Away
		Mushroom	Small	

Here, Pizza Type, Toppings and Size take 3 possible values whilst Crust and Delivery take 2 possible values. Exhaustive testing of all possible interactions for the aforementioned pizza ordering system requires 2 × 2 × 3 × 3 × 2 = 72 test cases. Now, pairwise (2-way) interactions can be tested using 9 test cases as shown in Table II.

TABLE II
PAIRWISE TEST SUITE FOR CA (N, 2, 2^3 3^2)

No	Pizza Type	Crust	Toppings	Size	Delivery
1	Vegetarian Cheese	Thin Crust	Roasted Chicken	Small	Take Away
2	Meat Lover	Extra Thick	Mushroom	Small	Eat In
3	Vegetarian Cheese	Extra Thick	Ground Beef	Large	Take Away
4	Meat Lover	Thin Crust	Ground Beef	Medium	Eat In
5	Vegetarian Cheese	Thin Crust	Mushroom	Large	Eat In
6	Meat Lover	Extra Thick	Roasted Chicken	Medium	Take Away
7	Vegetarian Cheese	Thin Crust	Mushroom	Medium	Take Away
8	Meat Lover	Extra Thick	Roasted Chicken	Large	Eat In
9	Meat Lover	Thin Crust	Ground Beef	Small	Eat In

Referring to Table II, it can be deduced that each 2-way interaction between parameters are covered at most once (indicating that the given result is the most optimal one). Nonetheless, there exist a number of constraints. The pair interactions between Pizza Type (Vegetarian) and Topping (Ground Beef, Roasted Chicken) are impossible, hence, must be forbidden. By the same token, pair interaction between Pizza Type (Meat Lover) and Toppings (Mushroom) is also forbidden. Using the mixed-constraints covering array notation discussed earlier, the system configuration can be formally expressed as CA (N, 2, 2^3 3^2, F), where F= $\{(F_{1,1},F_{3,1}), (F_{1,1},F_{3,2}), (F_{1,2},F_{3,3})\}$. Considering these constraints, the correct representation of CA is given in Table III.

TABLE IIII
PAIRWISE TEST SUITE FOR CA (N, 2, 2^3 3^2, F), WHERE F= $\{(F_{1,1},F_{3,1}), (F_{1,1},F_{3,2}), (F_{1,2},F_{3,3})\}$

No	Pizza Type	Crust	Toppings	Size	Delivery
1	Vegetarian Cheese	Extra Thick	Mushroom	Medium	Take Away
2	Meat Lover	Thin Crust	Roasted Chicken	Large	Take Away
3	Meat Lover	Extra Thick	Ground Beef	Medium	Eat In
4	Vegetarian Cheese	Thin Crust	Mushroom	Small	Eat In
5	Meat Lover	Extra Thick	Roasted Chicken	Medium	Eat In
6	Meat Lover	Thin Crust	Ground Beef	Medium	Take Away
7	Vegetarian Cheese	Extra Thick	Mushroom	Large	Eat In
8	Meat Lover	Extra Thick	Roasted Chicken	Small	Take Away
9	Meat Lover	Thin Crust	Ground Beef	Large	Take Away

The results in Table III faithfully forbid the given constraints. Hence, the two way interactions (Pizza Type <<>> Topping) featuring Vegetarian Cheese will only cover Mushroom whilst the two way interactions featuring Meat Lover can take both Roasted Chicken as well as Ground Beef.

IV. Related Work

In general, existing t-way strategies can be categorized into two categories based on the dominant approaches, that is, algebraic approaches or computational approaches respectively [18].

Algebraic approaches construct test sets using pre-defined rules or mathematical function [18]. Often, the computations involved in algebraic approaches are typically lightweight, and in some cases, algebraic approaches can produce the most optimal test sets. However, the applicability of algebraic approaches is often restricted to small configurations [18, 19]. Orthogonal Arrays (OA) [20, 21], MOA [1] and TConfig [22] are typical example of the strategies that are based on algebraic approach.

Unlike algebraic approaches, computational approaches often rely on the generation of the all pair combinations. Based on all pair combinations, the computational approaches iteratively search the combinations space to generate the required test case until all pairs have been covered. In this manner, computational approaches can ideally be applicable even in large system configurations. However, in the case where the number of pairs to be considered is significantly large, adopting computational approaches can be expensive due to the need to consider explicit enumeration from all the combination space. Example strategies that adopt this approach includes An Automatic Efficient Test Generator (AETG) [23, 24], its variant (mAETG) [25], PICT [8, 36], IPOG [18], Jenny[26], TVG [27, 28], IRPS [29], GA [30], ACA [30], and SA [31].

Although useful, much of the aforementioned strategies do not provide the support for constraints. Hence, in line with the scope of the paper, what follows is the review of strategies that address the problem of constraints.

Automatic Efficient Test Generator (or AETG) [23, 24] and employ a greedy search algorithm based on 2-way interaction pairing in order to generate the final test suite. In this manner, the generated test case is highly non-deterministic. A number of variations of AETG have been developed over the years, including AETGm [30] and mAETG_SAT [34]. Unlike AETG and AETGm, mAETG_SAT provides the support for constraints through its forbidden tuple implementation.

PICT [8, 36] generates all specified interaction tuples and randomly selects their corresponding interaction combinations to form the test cases as part of the complete test suite. In case a particular test case matches a specified constraint, PICT randomly generates a new combination for covering the interaction tuples. Due to its random behaviour, PICT tends to give a non-optimal test size as compared to other strategies.

TestCover [36] is a commercial t-way strategy implementation. No implementation details have been published in the literature apart from a list of benchmark configurations on constraints that can be obtained from its website.

SA [11, 13] relies on a large random search space for generating a *t*-way test suite. Using probability-based transformation equations, SA adopts binary search algorithm to find the best test case per iteration to be added to the final test suite. SA addresses constraints support through its variant strategy, called SA_SAT [34].

HSS [14] is perhaps the most recent t-way strategy that addresses the constraints problem for the t-way test suite generation. Based on the Harmony Search Algorithm, HSS adopts two probability values (i.e. the considering rate and pitch adjustment rate). Here, global search is iteratively performed by randomizing values in the Harmony memory whereby the local best value can be selected given a considering rate probability. Here, local best value can be considered for improvements for further improvements in the local search (i.e. with pitch adjustment probability). At each iteration, the best value will be added to the final test suite (provided that they do not cover constraints) until all the required interactions are covered.

V. LATE ACCEPTANCE HILL CLIMBING ALGORITHM

Late Acceptance Hill Climbing Algorithm is started from a randomly generated potential solutions captured in to the LAHC memory (in the form of list with fixed length). LAHC then generates a current neighbour to be compared one-value-at-a-time with the corresponding value from the LAHC memory. LAHC also maintains the previous cost function in the memory to allow selection of the best fit value. Ideally, the candidate cost is compared with the selected i_{th} cost from the memory. If the cost is not worse, the candidate will be accepted (as the current local best). Upon acceptance, the cost of the new current solution will be made to replace the original i_{th} cost from the memory. Here, the list keeps the fitness array F_a of length L_{fa} ($F_a = \{f_0, f_1, f_2..f_{Lfa-1}\}$). The position *v*, at the i_{th} iteration can be calculated via:

$$v = \text{i mod } L_{fa} \qquad (1)$$

where mod represents the remainder of the integer division

Assuming minimization problem, the final acceptance condition at i_{th} iteration can be expressed as:

$$C_i \leq C_{i-Lfa} \quad \text{or} \quad C_i \leq C_{i-1} \qquad (2)$$

where C_i = the candidate cost; C_{i-1} = the current cost; C_{i-Lfa} = the cost of the current L_{fa} iteration before

The complete pseudo code for LAHC can be summarised in Fig. 1.

```
Produce an initial solution s
Calculate initial cost function C(s)
Specify $L_{fa}$
for all k ϵ {0...$L_{fa}$-1}
  begin
    s=random(s)
    $f_k$ := C(s)
  end
Assign the initial iteration I:= 0;
While not a chosen stopping condition is met
  Construct a candidate solution s*
  Evaluate its cost function C(s*)
  v :=I mod $L_{fa}$
  if C(s*)≤ $f_v$ or C(s*)≤ C(s)
  then accept candidate (s :=s*)
  Insert cost value into the list $f_v$:= C(s)
  Increment the iteration I:= I+1
end while
```

Fig. 1 General Purpose LAHC Algorithm

VI. ADAPTING LATE ACCEPTANCE HILL CLIMBING ALGORITHM FOR T-WAY TEST GENERATION

The optimization problem of concerned can be specified using on Equation 3 and 4.

$$\text{Maximize } f(x) = \sum_{1}^{N} x_i \qquad (3)$$

$$\text{Subject to } x \in x_i \ , i = 1,2, \ldots, N \qquad (4)$$

where $f(x)$ is an objective function capturing the weight of the test case in terms of the number of covered interactions; x is the set of each decision variable x_i ; x_i is the set of possible range of values for each decision variable, that is, $x_i = \{x_i(1), x_i(2), \ldots, x_i(K)\}$ for discrete $_{\text{decision}}$ variables $(x_i(1) < x_i(2) < \ldots < x_i(K))$; N is the number of decision parameters; and K is the number of possible values for the discrete variables.

Addressing the aforementioned optimization problem, our LAHC strategy works as follows.

A. Parameter Initialization

Firstly, the LAHC accepts the input parameters and their corresponding values. Then, the LAHC generates the interactions list *IL* containing all interactions tuple combinations for each pair which later forms the objective function. Apart from accepting input parameters and their values, LAHC also needs to initialize the size and values of L_{fa} as well as the number of iteration, M.

Owing to the need to generate a population of interaction test cases as opposed to single optimization problem, there is a need to modify the structure of L_{fa} as well as to add the number of iteration into the original LAHC. In this case, it is proposed that L_{fa} keeps both the cost function value as well as its corresponding candidate solution. Here, when LAHC decides to accept the solution in L_{fa}, it can immediately use that solution as the basis for the next neighbourhood solution (i.e. for local search).

As the name suggests, M specifies the number of iteration for improving L_{fa}. Here, the value of M must be greater than or equal to the size of L_{fa} ($M \geq L_{fa}$ size), that is, to ensure that all the values in L_{fa} are visited at least once (see equation 1).

B. Diversification and Intensification

To achieve optimal solution, there is a need for sufficiently elaborate local and global search via exploiting the diversification and intensification property of the algorithm of interest.

Within the general purpose LAHC algorithm, diversification for global search is appropriately addressed by the generation of random initial solution within the L_{fa} list. However, the intensification element within the local search is missing.

Addressing this intensification issue, there is a need for a good perturbation function which can "slightly" modify the current local best solution to get better solution (see Figure 2). For instance, consider a solution candidate, $S(x^{new})$:

$$S(x^{new}) = (x_{1,}^{new}\ x_2^{new}, \ldots x_i^{new}, \ldots . x_N^{new}), (Cs)$$

```
function Pertubate (solution s)
    begin
      for all i ϵ {0...length (s)}
      begin
        with probability, $P_{change}$= 0.5
        begin
          with probability, $P_{direction}$ = 0.5
            if ($P_{direction}$<0.5) // move down
              begin
                if  $x_i$ = max value range
                    $x_i := x_i - 1$
                else
                    $x_i := x_i + 1$
              end
            else  //move up
              begin
                if  $x_i$ = min value range
                    $x_i := x_i + 1$
                else
                    $x_i := x_i - 1$
              end
          end
          update s($x_i$)
        end
      end
      return (s);
    end
```

Fig. 2 Probabilistic Pertubation Function

If x_i range values is {0, 1, 2, 3, 4, 5}, and the new x_i^{new}in the L_{fa} has the value of {3} then this value can be moved to the neighbouring value {4}. To ensure that only slight modification is done, we introduce two probability value called P_{change} and $P_{direction}$ respectively. Here if both P_{change} =0.5 and $P_{direction}$ = 0.5, there is only 50% chance of x_i^{new} be changed or remained. Now, if $x_i^{new}$is going to be changed, it can have equal chance of changing either in the lower, upper, or combination of both directions.

At first sight, the approach of adopting two probabilistic values in LAHC appears similar to HSS. A closer look reveals some fundamental differences. Firstly, in HSS, the two probabilistic values are used to decide whether or not to use a random value or a value from memory for improvisation as well as whether or not to do pitch adjustment of the current values. Within LAHC, P_{change} is used to decide whether to pertubate the current values much like the pitch adjustment in HSS, however, the use $P_{direction}$ is completely different. In LAHC, $P_{direction}$ is used to decide on the direction of the neighbourhood search and not on the use of a random value or any existing value from memory.

TABLE IV
COMPARISON IN TERMS OF THE TEST SUITE SIZE FOR NINE SYSTEM CONFIGURATIONS IN THE PRESENCE OF CONSTRAINTS

NO	CCA	LAHC	HSS	SA_SAT	mATEG_SAT	PICT	TestCover
1	**CCA(N, $2,3^3$,F{})**	**9**	**9**	**9**	**9**	10	**9**
	F={($F_{2,3}$,$F_{3,1}$),($F_{2,2}$,$F_{3,1}$),($F_{1,1}$,$F_{3,2}$),($F_{1,3}$,$F_{2,4}$)($F_{1,3}$,$F_{3,3}$),($F_{1,3}$,$F_{2,3}$,$F_{3,3}$)}	**10**	**10**	**10**	**10**	10	**10**
2	**CCA(N, $2,4^3$,F{})**	**16**	**16**	**16**	**16**	17	**16**
	F={($F_{1,1}$,$F_{2,2}$),($F_{1,3}$,$F_{3,4}$),($F_{1,4}$,$F_{2,4}$,$F_{3,1}$)($F_{1,3}$,$F_{2,2}$)}	17	**16**	17	17	19	17
3	**CCA(N, $2,5^3$,F{})**	**25**	**25**	**25**	**25**	26	**25**
	F={($F_{1,2}$,$F_{2,2}$),($F_{1,5}$,$F_{3,3}$),($F_{1,5}$,$F_{3,5}$),($F_{1,5}$,$F_{2,4}$,$F_{3,2}$),($F_{1,5}$,$F_{2,3}$),($F_{1,2}$,$F_{2,4}$)}	**25**	26	26	26	27	30
4	**CCA(N, $2,6^3$,F{})**	38	**36**	**36**	37	39	**36**
	F={($F_{1,4}$,$F_{2,6}$),($F_{2,4}$,$F_{3,5}$),($F_{1,3}$,$F_{2,1}$),($F_{2,2}$,$F_{3,3}$),($F_{1,4}$,$F_{3,2}$),($F_{2,4}$,$F_{3,2}$),($F_{1,6}$,$F_{2,5}$,$F_{3,5}$)}	**36**	**36**	**36**	37	39	38
5	**CCA(N, $2,7^3$,F{})**	51	**49**	**49**	52	55	**49**
	F={($F_{2,1}$,$F_{3,6}$),($F_{1,6}$,$F_{2,6}$,$F_{3,4}$),($F_{1,5}$,$F_{3,1}$),($F_{1,7}$,$F_{2,5}$),($F_{1,2}$,$F_{2,5}$),($F_{1,7}$,$F_{2,4}$)}	53	**51**	52	52	56	54
6	**CCA(N, $3,5^4$,F{})**	145	138	**127**	143	151	NS
	F={($F_{1,4}$,$F_{3,3}$,$F_{4,2}$),($F_{2,2}$,$F_{4,4}$),($F_{1,3}$,$F_{2,4}$),($F_{1,2}$,$F_{3,4}$)}	141	139	140	**138**	143	NS
7	**CCA(N, $3,6^4$,F{})**	253	240	**222**	247	260	NS
	F={($F_{1,5}$,$F_{4,3}$),($F_{3,4}$,$F_{4,2}$),($F_{2,3}$,$F_{4,3}$),($F_{2,2}$,$F_{3,3}$)}	245	**238**	251	241	250	NS
8	**CCA(N, $3,7^4$,F{})**	409	377	**351**	395	413	NS
	F={($F_{2,3}$,$F_{3,7}$),($F_{2,6}$,$F_{3,7}$),($F_{2,5}$,$F_{3,3}$),($F_{4,2}$,$F_{4,6}$)($F_{3,3}$,$F_{4,5}$),($F_{1,3}$,$F_{3,7}$)}	395	**377**	438	383	401	NS
9	**CCA(N, $4,3^5$,F{})**	94	**89**	NS	NS	NS	NS
	F={($F_{1,2}$,$F_{2,2}$,$F_{3,2}$,$F_{4,2}$),($F_{2,1}$,$F_{3,1}$,$F_{4,1}$,$F_{5,1}$)}	**93**	97	NS	NS	NS	NS

C. Control Loop with Interaction Coverage iteration, M iteration and L_{fa} Memory Update

Using the general Late Acceptance Hill Climbing algorithm with the aforementioned perturbation function, the complete LAHC strategy can be summarised in Fig. 3.

```
Define interactions to cover list, L
Define the constraints list, F
Produce an initial solution s
Calculate initial cost function C(s)
Specify $L_{fa}$, and iteration M
Populate F with constraints
while L is not empty
 begin
   ////////////////// diversification //////////////////
   for all k $\epsilon$ {0...$L_{fa}$-1} do randomize $f_k$ := s,C(s)

   Assign the initial number of iteration I:= 0;
   do until I=M
      Construct a candidate solution s* based on at
      least 1 uncovered pair
      Calculate its cost function C(s*)
      v :=I mod $L_{fa}$
      if C(s*)$\leq f_v$ or C(s*)$\leq$ C(s)
      then accept candidate (s :=s*)

     ////////////////// intensification //////////////////
      s:=pertubate (s)
      if C(s)> $f_{worst}$
       Replace the worst solution in $L_{fa}$ , $f_{worst}$:= s,C(s)
      Increment the number of iteration I:= I+1
   end do
     Pick the best s  from $L_{fa}$ not in F
     If exist best s not violating F
        Add best s  to the final suite
     Reset $L_{fa}$  for the next iteration
 end
```

Fig. 3 LAHC Strategy

Here, the internal M iteration loop will iteratively update L_{fa} with the local best value. Here, an index of the worst solution in L_{fa} is kept internally to facilitate the update of L_{fa}. Upon completion of the M iteration, the local best solution (with the best Cs) will be taken into the final test suite. Here, LAHC maintains the list of constraints as forbidden list in order to make sure that the local best solution does not contain the constraints tuple. If so, new test value will be generated accordingly. The main iteration loop will stop when all the interactions are covered.

VII. EVALUATION EXPERIMENTS

In this section, we benchmark LAHC with other existing strategies that support constraints using the experiments described by Cohen [34]. For our experiments, we have used L_{fa} = 100, P_{change}=0.2, $P_{direction}$=0.5, and M iteration =1000 for all the experiments. Here, we report the best results after 20 runs for statistical significance. Table IV summarizes the results. Here, the best generated results are highlighted in bold font. Entries marked with Not Supported (NS) indicates that the configurations that are not supported by the given strategy implementation.

Summing up, LAHC appears to perform well on all the given configurations. In fact, LAHC gives the most optimal test cases for two cases. The first case involve the configuration with CCA(N,2,5^3,F{}) where F={($F_{1,2}$,$F_{2,2}$),($F_{1,5}$,$F_{3,3}$),($F_{1,5}$,$F_{3,5}$),($F_{1,5}$,$F_{2,4}$,$F_{3,2}$),($F_{1,5}$,$F_{2,3}$),($F_{1,2}$,$F_{2,4}$)}. The second case involves the configuration with CCA(N,4,3^5,F{}) where F={($F_{1,2}$,$F_{2,2}$,$F_{3,2}$,$F_{4,2}$), ($F_{2,1}$,$F_{3,1}$,$F_{4,1}$,$F_{5,1}$)}. For the rest of the configuration, HSS appears to have the best overall results. SA_SAT, mAETG_SAT, and Test Cover also give optimal values for the first two configuration involving CCA(N,2,3^3,F{}) where F={($F_{2,3}$,$F_{3,1}$),($F_{2,2}$,$F_{3,1}$),($F_{1,1}$,$F_{3,2}$),($F_{1,3}$,$F_{2,4}$),($F_{1,3}$,$F_{3,3}$),($F_{1,3}$,$F_{2,3}$,$F_{3,3}$)} and CCA(N,2,4^3,F{}) where F={($F_{1,1}$,$F_{2,2}$),($F_{1,3}$,$F_{3,4}$),

$(F_{1,4},F_{2,4},F_{3,1})(F_{1,3},F_{2,2})\}$. PICT appears to perform poorly on all the given configurations.

VIII. CONCLUSIONS

In short, this paper has elaborated a new strategy, called LAHC, based on Late Acceptance Hill Climbing Algorithm. Our experience with LAHC has been promising. As the scope for future work, we are looking to tune LAHC to get better results. We are also looking to address adopt LAHC for software product line testing.

ACKNOWLEDGMENT

This research work involves collaborative efforts between Universiti Malaysia Pahang and Umm Al-Qura University. The work is funded by grant number 11-INF1674-10 from the Long-Term National Plan for Science, Technology and Innovation (LT-NPSTI), the King Abdul-Aziz City for Science and Technology (KACST), Kingdom of Saudi Arabia. We thank the Innovation Office, Universiti Malaysia Pahang and the Science and Technology Unit at Umm Al-Qura University for their continued logistics support.

REFERENCES

[1] R. Mandl, "Orthogonal Latin Squares: An Application of Experiment Design to Compiler Testing," *Communications of the ACM*, vol. 28, pp. 1054-1058, 1985.

[2] P. E. Ammann and A. J. Offutt., "Using Formal Methods To Derive Test Frames in Category-Partition Testing," in *9th Annual Conference on Computer Assurance (COMPASS'94)*, Gaithersburg MD, 1994, pp. 69-80.

[3] Y. K. Malaiya, "Antirandom Testing: Getting The Most Out of Black-Box Testing," in *6th International Symposium on Software Reliability Engineering*, 1996, pp. 86–95.

[4] K. Z. Zamli, R. R. Othman, M. I. Younis, and M. H. Mohamed Zabil, "Practical Adoptions of T-Way Strategies for Interaction Testing," in *Software Engineering and Computer Systems*. vol. 181, J. M. Zain, W. M. Wan Mohd, and E. El-Qawasmeh, Eds., ed: Springer Berlin Heidelberg, 2011, pp. 1-14.

[5] M. F. J. Klaib, "Development Of An Automated Test Data Generation And Execution Strategy Using Combinatorial Approach," PhD, School of Electrical And Electronics, Universiti Sains Malaysia, 2009.

[6] K. Z. Zamli, M. F. J. Klaib, M. I. Younis, N. A. M. Isa, and R. Abdullah, "Design And Implementation Of A T-Way Test Data Generation Strategy With Automated Execution Tool Support," *Information Sciences*, vol. 181, pp. 1741-1758 2011.

[7] M. I. Younis, "MIPOG: A Parallel T-Way Minimization Strategy For Combinatorial Testing," PhD, School of Electrical And Electronics, Universiti Sains Malaysia, 2010.

[8] M. I. Younis and K. Z. Zamli, "MC-MIPOG: A Parallel T-Way Test Generation Strategy for Multicore Systems," *ETRI Journal*, vol. 32, pp. 73-83, 2010.

[9] M. I. Younis, K. Z. Zamli, and N. A. M. Isa, "MIPOG - Modification Of The IPOG Strategy For T-Way Software Testing," in *Proceeding of The Distributed Frameworks and Applications (DFmA)*, Penang, Malaysia, 2008.

[10] M. I. Younis, K. Z. Zamli, and N. A. M. Isa, "A Strategy For Grid Based T-Way Test Data Generation," in *Proceedings the 1st IEEE International Conference on Distributed Frameworks and Application (DFmA '08)*, Penang, Malaysia, 2008, pp. 73-78.

[11] A. W. Williams. (2010, February). *TConfig*. Available: http://www.site.uottawa.ca/~awilliam/

[12] A. W. Williams, "Determination of Test Configurations for Pair-wise Interaction Coverage," in *Proceedings of the 13th International Conference on Testing of Communicating System*, Ottawa, Canada, 2000, pp. 59-74.

[13] Y. W. Tung and W. S. Aldiwan, "Automatic Test Case Generation For The New Generation Mission Software System," in *Proceedings of IEEE Aerospace Conference*, Big Sky, MT, USA, 2000, pp. 431-437.

[14] A. R. A. Alsewari and K. Z. Zamli, "Design and Implementation of a Harmony-Search-based Variable-Strength t-way Testing Strategy with Constraints Support," *Information and Software Technology*, vol. 54, pp. 553-568, 2012.

[15] M. B. Cohen, "Designing Test Suites for Software Interaction Testing (PhD Thesis)," Computer Science, University of Auckland, Auckland, 2004.

[16] M. B. Cohen, M. B. Dwyer, and J. Shi, "Interaction Testing of Highly-Configurable Systems in the Presence of Constraints," in *International Symposium on Software Testing and Analysis (ISSTA2007)*, New York, USA, 2007, pp. 129-139.

[17] G. Sherwood. (2006, January). *Testcover*. Available: http://testcover.com

[18] Y. Lei, R. Kacker, D. R. Kuhn, V. Okun, and J. Lawrence, "IPOG: A General Strategy for T-Way Software Testing," in *Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems*, Tucson, AZ U.S.A, 2007, pp. 549-556.

[19] J. Yan and J. Zhang, "Backtracking Algorithms And Search Heuristics To Generate Test Suites For Combinatorial Testing," in *Proceeding of the 30th Annual International Computer Software and Applications Conference*, 2006, pp. 385-394.

[20] A. Hartman and L. Raskin, "Problems and Algorithms for Covering Arrays," *Discrete Mathematics*, vol. 284, pp. 149-156, July 2004.

[21] A. S. Hedayat, N. J. A. Sloane, and J. Stufken, *Orthogonal Arrays: Theory and Applications*. New York: Springer Verlag, 1999.

[22] A. W. Williams. (2002, 16 June 2010). *TConfig*. Available: http://www.site.uottawa.ca/~awilliam

[23] 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 Transactions on Software Engineering*, vol. 23, pp. 437-444, July 1997.

[24] D. M. Cohen, S. R. Dalal, J. Parelius, G. C. Patton, and N. J. Bellcore, "The Combinatorial Design Approach to Automatic Test Generation," *IEEE software*, vol. 13, pp. 83-88, 1996.

[25] M. B. Cohen, "Designing Test Suites for Software Interaction Testing," Ph.D, Computer Science, University of Auckland, New Zealand, 2004.

[26] D. Pallas. (2003, 16 June 2010). *Jenny*. Available: http://www.burtleburtle.net/bob/math.

[27] J. Arshem. (2010, 16 June). *TVG*. Available: http://sourceforge.net/projects/tvg

[28] T. Yu-Wen and W. S. Aldiwan, "Automating Test Case Generation for the New Generation Mission Software System," in *Proceedings of the IEEE Aerospace Conference*, Big Sky, MT, USA, 2000, pp. 431-437.

[29] M. I. Younis, K. Z. Zamli, and N. A. M. Isa, "IRPS --- An Efficient Test Data Generation Strategy for Pairwise Testing," in *Proceedings of the 12th international conference on Knowledge-Based Intelligent Information and Engineering Systems, Part I*, Zagreb, Croatia, 2008.

[30] T. Shiba, T. Tsuchiya, and T. Kikuno, "Using Artificial Life Techniques to Generate Test Cases for Combinatorial Testing," in *Proceedings of the 28th Annual International Computer Software and Applications Conference*, 2004, pp. 72-77.

[31] J. Yan and J. Zhang, "A Backtracking Search Tool for Constructing Combinatorial Test Suites," *Journal of Systems and Software*, vol. 81, pp. 1681-1693, 2008.