VOL. 10, NO. 19, OCTOBER 2015 ISSN 1819-6608
ARPN Journal of Engineering and Applied Sciences

©2006-2015 Asian Research Publishing Network (ARPN). All rights reserved.

www.arpnjournals.com

TUNING OF CUCKOO SEARCH BASED STRATEGY FOR T-WAY TESTING

Abdullah B. Nasser, Abdul Rahman A. Alsewari and Kamal Z. Zamli
Faculty of Computer Systems and Software Engineering, Universiti Malaysia Pahang, Kuantan, Pahang, Malaysia
E-Mail: pcc14003@stdmail.ump.edu.my

ABSTRACT

Cuckoo Search Strategy (CSS) is the newly developed strategy based on the Cuckoo Search Algorithm. In order to achieve best performance, a number of parameters in the Cuckoo Search Algorithm needs to be tuned namely the nest size, the elitism probability, and the repetition. This paper describes the tuning process for Cuckoo Search Algorithm involving t-way testing, that is, by taking the standard covering array involving CA (N, 2, 46). Our initial experiment results using obtained range of parameter values of CSS demonstration that CSS able to give sufficiently competitive results compared to existing work.

Keywords: problem T-way testing, cuckoo search algorithm, combinatorial problem, tuning, parameters.

INTRODUCTION

Optimization problem (OP) relates to search of optimal solution for a large search space. OP have been utilized in many various domains and become such Software Engineering, optimal design in electronic, chemical mechanical, and civil. Meta-heuristics excel in this respect. In the last 40 years, many useful meta-heuristic algorithms have been developed in the literature (e.g. Tabu Search (Nurmela, 2004), Simulated Annealing (Stardom, 2001), Genetic algorithm(Stardom, 2001), Ant Colony Algorithm (ACA) , Particle Swarm Optimization (Ahmed, Zamli *et al.* 2012), Harmony Search (Alsewari and Zamli, 2012) and Late Acceptance Hill Climbing (Nasser, Alsariera *et al.* 2014), to name a few). Meta-heuristica are part of stochastic algorithms that efficiently explore the search space by trial and error. Unlike conventional algorithms, most of metaheuristic algorithms consider as population-based algorithms whereas finding a solution starts from many positions of solution space. Therefore, each member of population is a candidate to be the best solution.

Concerning t-way testing, much efforts have been focused on adopting the meta-heuristics algorithms as the basis for the generation strategy. In the literature, many meta-heuristic-based strategies have been developed including that strategies based on Tabu Search, strategies based on SA, strategies based on GA, strategies based on ACA, strategies based on PSO, and strategies based on HS , to name a few (Nurmela, 2004),(Stardom, 2001),(Ahmed, Zamli *et al.* 2012),(Alsewari and Zamli, 2012).

The performance of these a fore mentioned algorithm is subjected to their tuning. For example, PSO requires the tuning of cognitive parameters (C1 and C2), intertial weight (w) as well as population size and interation. GA requires tuning mutation rate, Crossover rate and Population size while HS require population size, harmony memory size, improvisation, pitch adjustment rate and harmony memory consideration rate. SA require initial temperature, cooling rate, epoch length (or the number of state transitions under each temperature), and and stopping condition. The main challenge of these algorithms is the user choices for initial parameter values. For instance, an algorithm with good parameter values may achieve better solution than the algorithm with poor parameter values.

Cuckoo Search (CS) is one of the recent meta-heuristic algorithms. CS requires the tuning of elitism probability, repetition and nest size (Yang and Deb, 2009). In CS, the user choices can affect the performance of algorithm because these choices use for generating new solution and affect the way in which how the algorithm distribute the execution time effort globally by explores the search space and, locally, how exploits the most promising regions (Yang, Deb *et al.* 2013).

Complementing the existing work, we opt to adopt the Cuckoo Search algorithm as part of our research work to generate t-way test suite. This paper describes the tuning of cuckoo search for t-way testing. The rest of this paper is organized as follows. Section 2 gives an overview of t-way testing strategies. Section 3 introduces our proposed strategy. Section 4 present the CSS parameter tuning process. Section 5 highlights the experimental results and discussion. Lastly, section 6 gives the conclusion and future work.

CUCKOO SEARCH

In 2009, Xin-She Yang developed a new metaheuristic algorithm for solving global optimization problems, called cuckoo search (CS). Cuckoo search inspire from brood parasitic behaviour for some birds such as the Ani and Guira cuckoos (Yang and Deb, 2009). Based on a comparative study, the authors show that CS is a promising algorithm and it is better than the most efficient algorithm in optimization problems such as GA, PSO (Yang and Deb, 2014), (Yang and Press, 2010).

Cuckoo behaviour

Cuckoo is fascinating and interesting species of bird. Besides the fact that cuckoo is beautiful bird, cuckoo also has aggressive reproduction strategy, which made them one of the most attractive and fascinating birds. Some species of the cuckoo, such as the Ani and Guira cuckoos, cannot complete their life cycle without obligate parasitism. The breeding parasite behavior of the cuckoo appears by lay their eggs in the nests of other host birds. If the properties of cuckoo eggs have developed well enough, then the eggs will have a great opportunity to survive. In order to increase the hatching probability of their own eggs, they remover the eggs of the host bird, also, they can mimic the external color of host eggs (see Figure-1). Furthermore, the cuckoo eggs often hatch earlier than the host eggs, because the cuckoo chooses the nest where the host eggs recently are laid, this lead to the first cuckoo chicks instinctively will evict the host eggs out of the nest to increase their share of food.

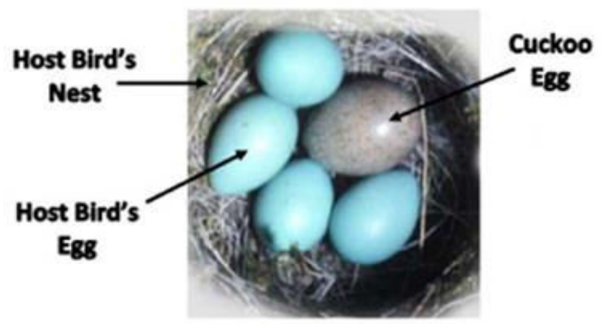


Figure-1. Cuckoo breeding parasite behavior.

Cuckoo search algorithm

Cuckoo Search algorithm works as follows. At the start, the algorithm generates randomly initial nests. Each egg in a nest represents a solution. In each generation of the algorithm, there are two operations are performed. First, generate a new nest by performing a Levy flight from the current nest, and then the new nest is evaluated. The new nest will be chosen as a current nest if the new nest is better than current nest. The second part of the Cuckoo Search algorithm discovers and removes the worse nests with probability *pa*. For simplification purpose, Cuckoo Search relies upon three idealized rules:

- Each cuckoo chooses a nest randomly to lays eggs.
- The number of available host nests is fixed, and nests with high quality of eggs will carry over to the next generations.
- The number of available host nests is fixed, and the egg laid by a cuckoo is discovered by the host bird with a probability $pa \in [0, 1]$. In this case, the host bird can either get rid of the egg, or simply abandon the nest and build a completely new nest.

Based on these aforementioned three rules, Cuckoo Search algorithm can be summarized as shown in Algorithm 1.

The main advantage of Cuckoo Search algorithm is the fact that it is a straightforward algorithms to implement and depends only on a few numbers of parameters. Often, the CS algorithm needs a small population size to achieve good results. The core part of the CS algorithm is generating new solution using levy Equation 1, where each position of the cuckoo is updated.

$$x_i^{(t+1)} = x_i^{(t)} + \alpha \oplus Lévy\ (\lambda) \qquad (1)$$

Here, $\alpha>0$ is the step size which should be related to the problem of interest and $Lévy \sim u = t^{-\lambda}$. Equation (1) is considered as a generic equation to update cuckoo's position using Lévy flights. The Lévy flight essentially is a random walk where the next step is based on the current location, and step lengths have a certain probability distribution that is heavy-tailed (Lévy distribution) (Yang, 2010).

```
Algorithm 1: Cuckoo Search Algorithm
Objective function f(x), x = (x1, ..., xd) ;
Initial a population of n host nests $x_i$ (i = 1, 2, ..., n);
While (t <MaxGeneration) or (stop criterion)
            Get a cuckoo (say i) randomly by Lévyflights;
            Evaluate its quality/fitness Fi;
            Choose a nest among n (say j) randomly;
            IF (Fi > Fj)
              Replace j by the new solution;
            End if
            Abandon a fraction (pa) of worse nests and build_
            new ones at new.
            Keep the best solutions;
            Rank the solutions and find the current best;
End while
Postprocess results and visualization;
End-Procedure
```

Figure-2. Cuckoo search algorithm.

As with other meta-heuristics algorithms, Cuckoo Search algorithm provides two search capabilities: global search, which allows the algorithm to jump out of local optimum, and local search by intensify search around the current best, are controlled by pa probability. If pa = 0.25, the local search takes about 25% and global search takes about 75% of the total search time(Yang and Deb, 2014). Local search and global search capabilities combined with search using levy flight make CS explore the search space more efficiently. Currently, researches on cuckoo search is very active and its applications have been proven successful in many areas such as machine learning area (Vázquez, 2011), the field of truss optimization problems (Gandomi, Talatahari *et al.* 2013), clustering of web results (Cobos, Muñoz-Collazos *et al.* 2014), nurse scheduling problems (Tein and Ramli, 2010), generating test data generation (Perumal, Ungati *et al.* 2011), generating independent paths for software testing (Srivastava, Khandelwal *et al.* 2012). In this paper, we investigate the use of CS algorithm for t-way test suite generation.

T-WAY TESTING AND COVERING ARRAY

In this section, we give a brief definition of t-way testing and covering array. T-way testing is a very efficient technique to generate minimum test cases that can be used for interaction fault detection. The motivation behind the use of t-way testing is that not every parameter contributes to every fault.

To illustrate the concept of t-way testing in test suite reduction, consider a simple application on several software and hardware configurations. This application has three components: operating system, processor, and RAM. Each component have their associated values as shown in Table-1. To test the system exhaustively, there is a need to consider 12 tests. Here, adding the number of parameters exponentially increases the size of combination space exponentially. Thus, testing all interactions of parameters values is generally impractical owing to the large combination space and resource constraints. Therefore, a sampling strategy is needed to select a subset of the combination space for overall testing.

T-way testing samples the large combination space by generate test cases such that the required interaction strength t is covered at least once (where t indicates to the strength of coverage) (Williams and Probert, 2002),(Burnstein, 2003),(Bell, 2006),(Bell and Vouk, 2005). By using two-way testing, in our example as shown in Table 2, only 6 test cases can cover all pair of input parameter values at least one time. Here, the reduction is from 12 to 6 (50%).

Table-1. Software application options.

Operating System	RAM	PC Processor
Windows 7	1GB	Intel core i3
Windows 8	2GB	Intel core i5
		Intel core i7

Table-2. The minimal test suite generate using Tow-way.

No	Operating System	RAM	PC Processor
1	Windows 7	2GB	Intel core i7
2	Windows 8	2GB	Intel core i5
3	Windows 7	1GB	Intel core i5
4	Windows 8	1GB	Intel core i7
5	Windows 8	1GB	Intel core i3
6	Windows 7	2GB	Intel core i3

Concerning Covering Array (CA), there are many mathematical notations and symbols have been developed to express t-way combinatorial testing. CA is one of the common structures that describe and formulate interaction testing. CA has been utilized in different research areas, such as material design (Cawse, 2003), medicine and agriculture (Hedayat, Sloane *et al.* 1999), and industrial processes (Shasha, Kouranov *et al.* 2001), however CA has been used significantly in the area of software testing with major application (Burr and Young, 1998; Shasha, Kouranov *et al.* 2001).

Basically, CA has four variables N, t, v, and P, where N refers to optimal test cases, p refers to system's components or parameters, v refers to number of components' value, and t refers to interaction strength. For example, covering array of CA (6; 2, 24) consist 6 rows, represent the size of test cases, and 4 columns, represents the parameters, each parameter has 2 values. If the number of values is not equal and each parameter has different number of values, then test suite is called uniform Mixed Coverage Array MCA(N, t, v1 p1 v2 p2 v3 p3 .....vj pj). For example MCA (12, 3, 23 31) consisting three parameters have 2 values and one parameter has 3 values.

T-WAY AS OPTIMIZATION PROBLEM

From optimization perspective, the t-way problem can be expressed mathematically as follows:

$$\text{Maximize } f(x) = \sum_{1}^{N} x_i \quad (2)$$

$$\text{subject to } x \in x_i \ , i = 1,2,\ldots,N \quad (3)$$

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),...,x_i (K)} for discrete decision variables (x_i (1)< x_i (2)<...< x_i (K)); N is the number of decision parameters; and K is the number of possible values for the discrete variables.

PARAMETER TUNING

While the number of parameters in CS is fewer than GA and PSO (Yang, 2010), the behaviour of Cuckoo Search Strategy is still largely determined by population size nest size, elitism factor pa, and iteration number N. Tuning of these parameters is important in order to ensure the best performance.

In our case, we opt to adopt a well-known covering arrays CA (N; 2, 46) as our case study. The reason for adopting this configuration stemmed from the fact that the same covering array has been used for tuning in many other related works. To achieve statistical significance, CCS is executed twenty times with every parameter value, and the average is taken from the reading results. we have tried different values for pa (0.01, 0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.4, and 0.5). Table 3 shows the average values and the best values of the test suite size after 20 executions.

Referring to Table 3, it is observed that using a higher probability pa can result in better solution. For pa < 0.2, the proposed strategy gives poor results. However, for pa => 0.25, the proposed strategy obtains good results. From our simulations, the best test suite size obtained when the probability between 0.25 and 0.4 and after 0.4. In term of nest size and repetition, results have shown that there is a positive correlation between the sizes of population (or nest) and the size of test suite. By increasing the number of nests from 1 to 30, the performance of CSS strategy is improved. Also, when the repetition value increases, the best size is also improved. The best test suite size obtained is when the repetition value varies from 100 to 300.

In summary, the CSS obtains the optimal test suite when probability pa = 0.25, nest size between the range of (30 and 100), and repetition between the range of (100 and 500).

COMPARATIVE EXPERIMENTS

To investigate the performance of the tuned CSS, we opt to undertake the experiments as discussed in (Nurmela, 2004),(Stardom, 2001),(Ahmed, Zamli *et al.* 2012),(Alsewari and Zamli, 2012). Here, we segregate the strategies into pure computational based strategies and meta-heuristics based strategies in order to ensure meaningful comparison. Based on our tuning, we have adopted pa=0.25, iteration = 300, and nest size = 30. The complete results are depicted in Table-3.

Table-3. Best and Average Test Suite for CA (N; 2, 4^6) with variation of values for *pa,* Nest size (*nest_size*) and Repetition (*N*).

N	Nest size	*pa*																	
		0.01		0.05		0.1		0.15		0.2		0.25		0.3		0.4		0.5	
		Avg	Best	Avg	Best	Avg	Best	Avg	Avg	Avg	Best	Avg	Best	Avg	Best	Avg	Best	Avg	Best
10	5	32.8	29	28.85	27	28.3	26	26.95	25	26.6	25	26.4	25	26.55	25	25.55	24	25.85	24
	10	27.65	26	26.95	24	26.65	25	25.8	24	25.55	24	25.35	24	25.05	24	24.6	23	24.75	23
	20	26.55	25	25.65	24	25.4	24	25.1	24	24.6	23	24.45	23	24.5	23	24.55	23	24.05	23
	30	25.7	24	25.3	24	24.85	23	24.8	23	24.4	22	24.55	23	24.05	23	24.2	23	23.95	23
	50	25.1	23	24.85	23	24.7	23	24.05	23	24.15	22	24.2	22	23.85	22	24.65	22	23.9	22
	100	24.25	23	24.3	23	23.85	21	24.25	22	24.45	22	23.9	22	24.15	23	23.85	23	23.35	22
	200	24.00	23	23.75	22	23.95	22	23.9	23	23.9	22	23.7	22	23.6	22	24.05	22	23.55	22
	300	23.85	23	23.6	22	23.75	22	23.55	22	23.95	23	23.95	22	23.5	22	23.55	22	23.55	22
20	5	30.1	27	26.55	23	26.4	25	25.4	24	25.0	24	25.15	23	24.2	23	24.45	23	24.8	23
	10	26.8	25	25.55	24	25.15	24	24.6	23	24.4	23	24.9	24	24.05	23	24.0	22	24.3	23
	20	24.9	24	24.75	24	24.5	23	24.05	23	23.85	23	23.55	22	24.3	22	23.9	22	23.7	22
	30	24.8	23	24.15	23	24.35	23	23.65	22	23.9	23	24.05	23	23.8	23	23.95	22	23.6	23
	50	24.45	23	24.0	23	23.65	22	23.8	22	24.0	23	23.5	22	23.9	23	23.45	22	23.8	22
	100	24.0	23	24.0	23	23.95	22	24.05	23	23.4	22	24.0	23	23.8	22	24.05	22	23.6	21
	200	23.75	22	24.05	22	23.8	22	23.65	22	23.45	22	23.95	23	23.3	22	23.9	23	23.75	22
	300	23.65	22	23.6	22	23.35	22	23.55	22	23.9	22	23.5	22	23.65	22	23.75	22	23.35	22
50	5	29.85	23	24.9	22	25.05	24	24.1	23	24.05	22	23.75	22	23.85	22	24.15	22	24.25	22
	10	24.8	23	24.75	23	24.05	23	24.05	23	24.05	23	23.7	22	23.6	22	23.9	22	23.45	22
	20	24.05	23	23.8	22	23.85	23	23.5	22	23.8	22	23.45	22	23.3	22	23.35	23	23.55	22
	30	24.05	23	23.65	22	23.8	23	23.7	23	23.75	22	24.1	23	23.65	22	23.9	22	23.9	23
	50	24.1	22	23.8	22	23.75	22	23.65	22	23.85	21	23.65	23	23.75	22	24.1	23	23.8	22
	100	23.85	23	24.0	23	23.75	22	23.2	22	23.75	22	23.85	22	23.65	22	23.7	22	23.65	23
	200	23.4	22	23.4	22	23.7	22	23.6	22	23.75	22	23.4	22	23.4	22	23.75	22	23.85	22
	300	23.3	22	23.7	22	23.4	22	23.75	22	23.6	22	24.05	22	24.05	22	23.7	21	23.45	22
100	5	28.0	24	24.0	22	24.25	23	23.65	22	23.8	23	23.8	22	23.75	22	23.7	22	23.7	22
	10	24.2	23	24.0	22	24.1	23	24.0	22	23.85	23	23.7	23	23.8	23	23.95	23	23.75	22
	20	23.75	22	23.45	21	23.6	23	23.7	22	23.95	22	23.8	23	23.6	22	23.95	23	23.8	22
	30	23.8	22	23.95	22	23.5	22	23.3	22	23.7	22	24.45	22	23.85	22	23.65	22	23.8	23
	50	24.0	22	23.5	21	23.4	22	23.9	22	23.95	23	23.75	22	23.45	21	23.45	22	23.6	22
	100	23.3	22	23.55	22	23.4	22	24.1	23	23.95	23	23.9	22	23.9	22	23.55	22	23.65	23
	200	24.0	23	23.65	22	23.9	22	23.8	22	23.55	21	23.75	22	23.75	22	23.3	22	23.6	22
	300	23.75	22	23.5	21	23.45	22	24.0	22	23.8	22	23.95	22	24.15	22	23.7	22	23.6	22
200	5	27.25	23	23.9	22	23.7	22	23.75	22	23.65	22	23.55	22	23.75	23	23.6	22	23.35	22
	10	23.8	22	23.75	22	23.75	22	23.95	23	23.85	22	24.1	23	23.65	22	23.8	23	23.8	22
	20	23.7	22	24.0	22	23.4	22	23.6	22	23.55	22	23.9	23	24.15	23	23.85	23	24.15	23
	30	23.6	22	23.7	23	23.7	22	23.55	22	23.45	21	23.4	22	23.85	22	23.15	22	23.7	22
	50	23.35	22	23.7	22	23.7	22	23.45	23	23.95	23	23.65	22	24.2	22	23.3	22	23.45	22
	100	23.95	22	23.8	22	24.1	22	23.5	21	23.9	22	23.1	22	23.75	22	23.85	22	23.65	22
	200	23.9	22	23.25	22	23.8	22	23.6	22	23.75	22	23.95	23	23.9	22	23.95	22	23.55	22
	300	23.3	22	23.75	22	23.6	22	23.7	22	23.8	22	23.7	22	23.8	22	23.8	22	23.75	22
300	5	27.75	24	23.85	22	23.95	23	24.3	23	23.85	22	23.95	22	23.45	22	23.7	22	24.0	22
	10	23.85	22	23.65	22	23.65	22	23.7	22	23.75	21	23.55	22	24.1	22	23.8	21	24.0	22
	20	23.6	22	23.75	22	23.55	22	23.7	22	23.35	22	23.6	22	23.6	22	23.95	23	23.65	21
	30	23.6	22	23.35	22	23.65	22	23.6	22	23.65	22	23.7	22	23.75	22	23.6	23	23.85	22
	50	23.7	22	23.75	22	23.75	23	23.45	22	23.9	22	23.8	23	23.9	22	23.5	22	23.5	22
	100	23.45	22	23.7	23	23.7	22	23.9	22	23.45	21	23.85	22	24.1	22	23.7	22	23.75	22

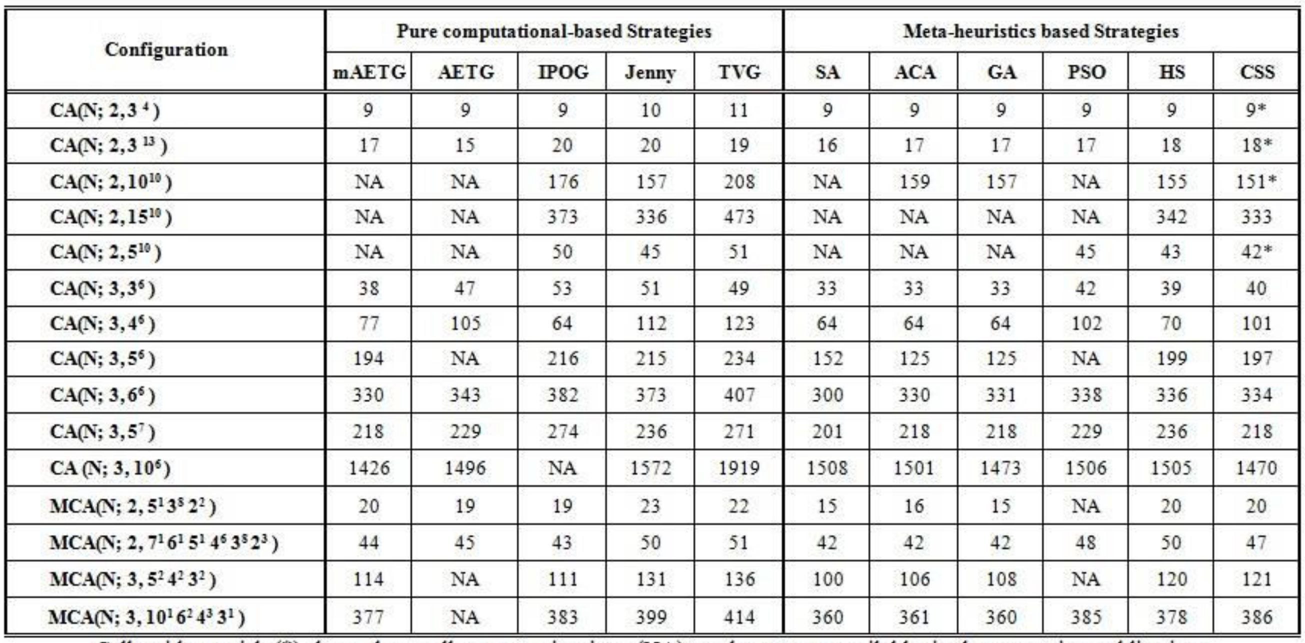
	200	23.35	22	23.6	22	23.95	22	23.8	22	23.65	22	23.95	22	23.7	23	23.6	22	23.05	23
	300	23.55	21	23.75	22	23.8	22	23.6	22	23.8	22	23.8	22	23.55	22	23.6	22	23.55	22

In order to measure performance of CSS against existing t-way strategies. Table 4 shows a comparison between CSS and existing strategies. From Table 4,CSS outperforms existing strategy in a number of the configurations (as marked with *). Even when CSS is not the best, the obtained results are still sufficiently competitive.

CONCLUDING REMARKS

In this paper, we have introduced a new t - way testing based on cuckoo search, called cuckoo search strategy (CSS) and demonstrate its tuning. Our initial experiment has been encouraging as we manage to outperform existing strategies in a number of configurations. For future work, we plan to adopt cuckoo search for highly configurable systems with constraints. We are also currently improving CSS to support both sequence and sequence-less t-way testing.

Table-4. Benchmarking CSS with existing strategies.



Configuration	Pure computational-based Strategies					Meta-heuristics based Strategies					
	mAETG	AETG	IPOG	Jenny	TVG	SA	ACA	GA	PSO	HS	CSS
CA(N; 2,3 ^{4})	9	9	9	10	11	9	9	9	9	9	9*
CA(N; 2,3 ^{13})	17	15	20	20	19	16	17	17	17	18	18*
CA(N; 2,10^{10})	NA	NA	176	157	208	NA	159	157	NA	155	151*
CA(N; 2,15^{10})	NA	NA	373	336	473	NA	NA	NA	NA	342	333
CA(N; 2,5^{10})	NA	NA	50	45	51	NA	NA	NA	45	43	42*
CA(N; 3,3^{6})	38	47	53	51	49	33	33	33	42	39	40
CA(N; 3,4^{6})	77	105	64	112	123	64	64	64	102	70	101
CA(N; 3,5^{6})	194	NA	216	215	234	152	125	125	NA	199	197
CA(N; 3,6^{6})	330	343	382	373	407	300	330	331	338	336	334
CA(N; 3,5^{7})	218	229	274	236	271	201	218	218	229	236	218
CA (N; 3,10^{6})	1426	1496	NA	1572	1919	1508	1501	1473	1506	1505	1470
MCA(N; 2, $5^{1} 3^{8} 2^{2}$)	20	19	19	23	22	15	16	15	NA	20	20
MCA(N; 2, $7^{1} 6^{1} 5^{1} 4^{6} 3^{8} 2^{3}$)	44	45	43	50	51	42	42	42	48	50	47
MCA(N; 3, $5^{2} 4^{2} 3^{2}$)	114	NA	111	131	136	100	106	108	NA	120	121
MCA(N; 3, $10^{1} 6^{2} 4^{3} 3^{1}$)	377	NA	383	399	414	360	361	360	385	378	386

Cells with asterisk (*) show the smallest test suite size, (NA) results are not available in the respective publications

ACKNOWLEDGEMENTS

The work reported in this paper is funded by the generous Science Fund – "Constraints T-Way Testing Strategy with Modified Condition Decision Coverage" from the MOSTI, Malaysia. Abdullah B. Nasser is the recipient of the Graduate Research Scheme (GRS) from Universiti Malaysia Pahang

REFERENCES

[1] Ahmed B. S., Zamli K. Z. and Lim C. P. 2012. Constructing a t-way interaction test suite using the particle swarm optimization approach. International Journal of Innovative Computing, Information and Control, Vol. 8, No. 1, pp. 431-452.

[2] Alsewari A. R. A. and Zamli K. Z. 2012. Design and implementation of a harmony-search-based variable-strength t-way testing strategy with constraints support. Information and Software Technology, Vol. 54, No. 6, pp. 553-568.

[3] Bell K. Z. 2006. Optimizing effectiveness and efficiency of software testing: A hybrid approach. PhD Dissertation, North Carolina State University.

[4] Bell K. Z. and Vouk M. A. 2005. On effectiveness of pairwise methodology for testing network-centric software. Paper presented at the Information and Communications Technology. Enabling Technologies for the New Knowledge Society: ITI 3rd International Conference on.

[5] Burnstein I. 2003. Practical software testing: a process-oriented approach: Springer.

[6] Burr K. and Young W. 1998. Combinatorial test techniques: Table-based automation, test generation

and code coverage. Paper presented at the Proc. of the Intl. Conf. on Software Testing Analysis & Review.

[7] Cawse J. N. 2003. Experimental design for combinatorial and high throughput materials development: Wiley-Interscience New York.

[8] Cobos C., Muñoz-Collazos H., Urbano-Muñoz R., Mendoza M., León E. and Herrera-Viedma E. 2014. Clustering of Web Search Results based on the Cuckoo Search Algorithm and Balanced Bayesian Information Criterion. Information Sciences.

[9] Gandomi A. H., Talatahari S., Yang X. S. and Deb S. 2013. Design optimization of truss structures using cuckoo search algorithm. The Structural Design of Tall and Special Buildings, Vol. 22, No. 17, pp. 1330-1349.

[10] Hedayat A. S., Sloane N. J. A. and Stufken J. 1999. Orthogonal arrays: theory and applications: Springer.

[11] Nasser A. B., Alsariera Y. A. and Zamlifll K. Z. 2014. Late Acceptance Hill Climbing Based Strategy for Addressing Constraints Within Combinatorial Test Data Generation.

[12] Nurmela K. J. 2004. Upper bounds for covering arrays by tabu search. Discrete applied mathematics, Vol. 138, No. 1, pp.143-152.

[13] Perumal K., Ungati J. M., Kumar G., Jain N., Gaurav R. and Srivastava P. R. 2011. Test data generation: a hybrid approach using cuckoo and tabu Search Swarm, Evolutionary, and Memetic Computing, pp. 46-54.

[14] Shasha D. E., Kouranov A. Y., Lejay L. V., Chou M. F. and Coruzzi G. M. 2001. Using combinatorial design to study regulation by multiple input signals. A tool for parsimony in the post-genomics era. Plant Physiology, Vol. 127, No. 4, pp.1590-1594.

[15] Srivastava P. R., Khandelwal R., Khandelwal S., Kumar S. and Santebennur Ranganatha S. 2012. Automated test data generation using cuckoo search and tabu search (csts) algorithm.

[16] Stardom J. 2001. Metaheuristics and the search for covering and packing arrays. Trent University.

[17] Tein L. H. and Ramli R. 2010. Recent advancements of nurse scheduling models and a potential path. Paper presented at the Proceedings of 6th IMT-GT

[18] Conference on Mathematics, Statistics and its Applications.

[19] Vázquez R. A. 2011. Training spiking neural models using cuckoo search algorithm. IEEE Congress on Evolutionary Computation (CEC).

[20] Williams A. W. and Probert R. L. 2002. Software Components Interaction Testing: Coverage Measurement and Generation of Configurations: University of Ottawa.

[21] Yang X. S. 2010. Nature-inspired metaheuristic algorithms: Luniver press.

[22] Yang X. S. and Deb S. 2009. Cuckoo search via Lévy flights. World Congress on the Nature & Biologically Inspired Computing, NaBIC 2009.

[23] Yang X. S. and Deb S. 2014. Cuckoo search: recent advances and applications. Neural Computing and Applications, Vol. 24, No. 1, pp.169-174.

[24] Yang X. S. and Press L. 2010. Nature-Inspired Metaheuristic Algorithms, Second Edition.

[25] Yang X. S., Deb S. and Fong S. 2013. Metaheuristic algorithms: optimal balance of intensification and diversification.