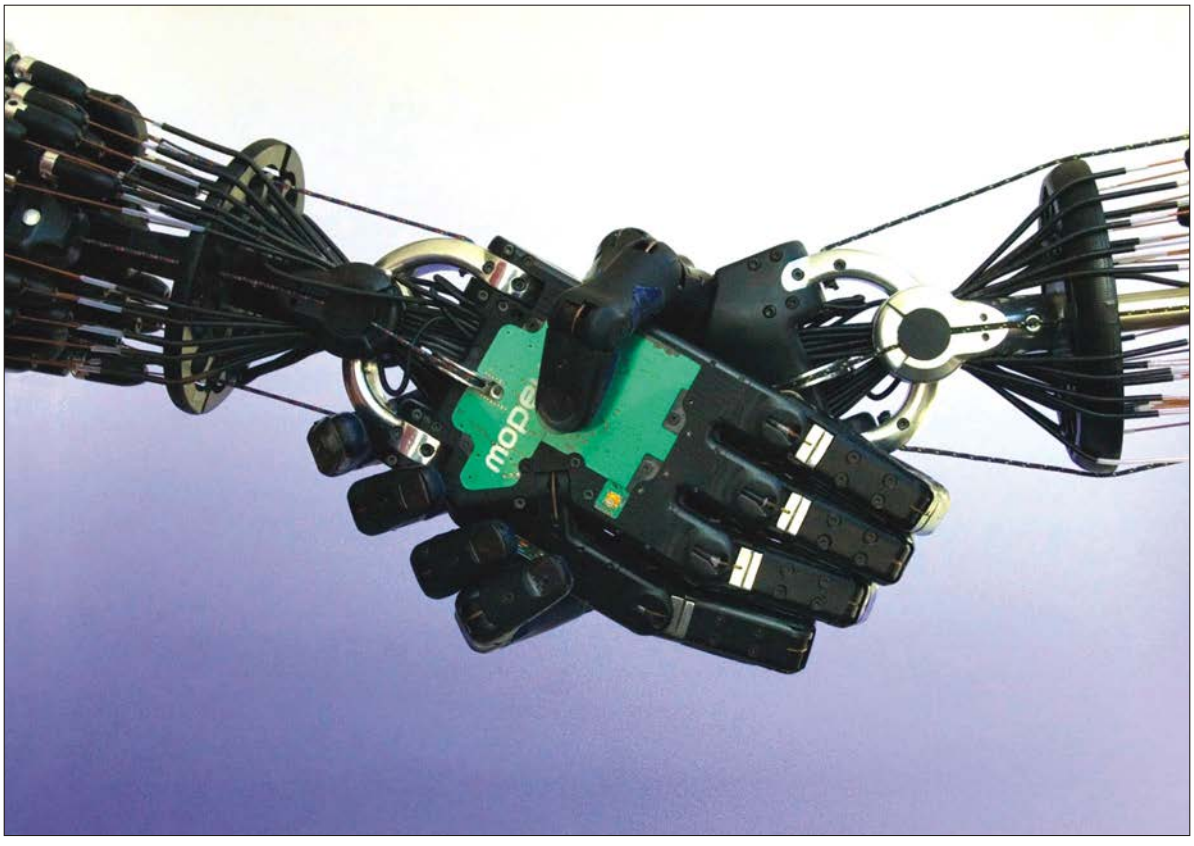

# Journal of Artificial Intelligence

ISSN 1994-5450

science alert

ANSInet
an open access publisher
http://ansinet.com

Journal of Artificial Intelligence 4 (2): 156-165, 2011
ISSN 1994-5450 / DOI: 10.3923/jai.2011.156.165


# The Development of a Particle Swarm Based Optimization Strategy for Pairwise Testing

Bestoun S. Ahmed and Kamal Z. Zamli
School of Electrical and Electronic Engineering, Universiti Sains Malaysia, Engineering Campus, 14300 Nibong Tebal, Seberang Perai Selatan, Pulau Pinang, Malaysia

*Corresponding Author: Kamal Z. Zamli, School of Electrical and Electronic Engineering, Universiti Sains Malaysia, Engineering Campus, 14300 Nibong Tebal, Seberang Perai Selatan, Pulau Pinang, Malaysia Tel: 006-045995843 Fax: 006-5941023*

## ABSTRACT

Pairwise testing strategies are used to select test cases from a large search space considering the interactions of test input parameters in order to minimize the test suite size. We normally want that all 2-way interactions of parameters' values occur in the test suit at least once. Due to the large and complex search space in the interaction problems, different techniques have been used to deal with this search space. Artificial intelligent techniques have been regarded as being especially adequate search strategies, since they are able to deal with search for optimization. Two of the well known algorithms are Genetic Algorithm (GA) and Ant Colony Algorithm (ACA). However, other heuristic search techniques have started to compete with GA and ACA such as Particle Swarm Optimization (PSO) in the context of algorithm simplicity and performance. This study presents the development of a new pairwise test data generation strategy based on PSO, called Pairwise Particle Swarm-based Test Generator (PPSTG). In doing so, this study also highlights PPSTG design as well as compares its performance in terms of test size against other existing strategies. PPSTG serves as our research vehicle to investigate the effectiveness of PSO for pairwise test data generation. The experimental results and comparisons of our strategy showed that our strategy can generate comparable results as far as the size of the test suite is concerned.

**Key words:** Software testing, interaction testing, combinatorial testing, test design techniques, testing processes, artificial intelligence, meta-heuristics

## INTRODUCTION

Although desirable as an activity to ensure quality and conformance, complete and exhaustive software testing is practically impossible (Nie and Leung, 2011; Roongruangsuwan and Daengdej, 2010). As a result, many sampling strategies have been proposed in the literature including equivalent partitioning, boundary value and cause and effect analysis. Using these strategies, test data can be sampled accordingly to minimize the test data for consideration and hence reduce testing efforts and cost. Although useful, the effectiveness of these strategies can sometimes be questionable especially in the case where there are significant interactions between system variables. Often, these (traditional) sampling strategies are sufficiently lacking for detecting bugs due to interaction.

Mathematically, the representation of the interaction test suites took different forms. Several objects have been defined with the effective properties of the covering array. The notation CA (N;t,k,v) represent an N×k array on v values such that every N×t sub-array contains all ordered

subsets from v values of size t = 2 at least one time (Hartman and Raskin, 2004). When the number of parameter values are varied, Mixed Covering Array MCA (N;t,k,(v1,v2,... $v_k$) notation can be adopted (Colbourn *et al.*, 2006). Both CA and MCA notation can further be simplified as CA (N;t,$v^k$) or MCA (N;t, $v^{1...}v^k$) (Yilmaz *et al.*, 2004). By using the covering array notation in the software testing, each test suite is a construction of N×k array where N is the number of test cases. As a result, each test case represents a single setting of the system under test. In pairwise interaction testing, we normally want that all pair combinations of parameter values occur in the test suite at least once (Klaib *et al.*, 2008; Kuhn and Reilly, 2002; Kuhn *et al.*, 2004).

To address the construction of the interaction test suites, significant efforts in the literature are now being focused on pairwise testing strategies (Bryce and Colbourn, 2007; Czerwonka, 2006; Klaib *et al.*, 2010, 2008). Pairwise testing strategies aim is to construct the pairwise interaction test suites by searching for more interactions coverage to optimize the size of the test suites. Due to the complexity of search space, different techniques have been used to deal with this search space for generating pairwise test cases. Searching for an optimum set of pairwise test cases can be a painstakingly difficult endeavor. Therefore, different pairwise strategies have been proposed to construct near optimal sets of test data. There are two main approaches for constructing pairwise test suite: the algebraic and computational approach (Lei *et al.*, 2007; Lei and Tai, 1998).

Algebraic approach, in some cases, can produce the most optimal test suite. The approach often based on the extensions of the mathematical functions for constructing the Orthogonal Array (OA) (Cheng, 1980). Orthogonal arrays are often too restrictive since it requires the parameters and values to be uniform. However, in general, computations involved in algebraic approaches are lightweight (Lei *et al.*, 2008). Nonetheless, algebraic approaches often impose restrictions on the system configurations to which they can be applied.

In contrast to algebraic approach, computational approach often relies on the generation of all pair interaction possibilities then checking the coverage of each proposed test case with the generated pair interactions. In such a checking process, there is significant searching efforts required in the combinatorial space in order to generate the required test suite until all pair interactions have been covered. There are two efforts for constructing test cases in this approach: one-test-at-a-time or one-parameter-at-a-time. In case of one-test-at-a-time, the algorithm generates one test case and checks its coverage with the pair interactions. Automatic Efficient Test Generator (AETG) (Cohen *et al.*, 1997) and its variant (mAETG) (Cohen *et al.*, 2007), are well-known examples in this approach. In contrast, strategies with one-parameter-at-a-time fashion construct the test case by adding one parameter at a time and check for its coverage with the pair interactions. In Parameter Order (IPO) (Lei and Tai, 1998) is a well-known example in this approach.

More recently, there have been a number of attempts to use Artificial Intelligence (AI) based strategies for pairwise testing as a variant of computational approach since they are able to optimize the search process. One reason for adopting artificial intelligence based strategies is to ensure near optimal solution for every configuration. Two of the most common implemented artificial intelligence based strategies are based on Genetic Algorithm (GA) (Shiba *et al.*, 2004) and Ant Colony Algorithm (ACA) (Shiba *et al.*, 2004). Despite giving good results (Shiba *et al.*, 2004) both strategies appear to suffer from heavyweight computational process rendering difficulties to support high order parameters and values. For this, the search for other lightweight artificial intelligence strategies is an essential demand.

However, other heuristic search techniques have started to compete with GA and ACA such as Particle Swarm Optimization (PSO). Similar to its other AI counter parts, PSO has proven its success in many research areas but with lighter computation mainly due to its algorithmic simplicity (Clerc and Kennedy, 2002; Jie *et al*., 2008; Liang *et al*., 2006; Wachowiak *et al*., 2004). Concerning its application in software testing, the advantages of PSO over GA have been demonstrated by Windisch *et al*. (2007) for structural testing. This raises the question of how PSO compares against other artificial intelligent based pairwise strategies, especially GA and ACA.

To address the aforementioned comparison, we present a new test generation strategy for pairwise combinatorial testing based on Particle Swarm Optimization, called Pairwise Particle Swarm Test Generator (PPSTG). PPSTG serves as our research vehicle to investigate the effectiveness of PSO for pairwise test data generation.

## EXISTING ARTIFICIAL INTELLIGENT BASED PAIRWISE STRATEGIES

**Genetic algorithm based strategies:** Existing Genetic Algorithm (GA) (Shiba *et al*., 2004) base strategy is based on AETG strategy (Cohen *et al*., 1997). In GA, every possible solution (test case here) is treated as a chromosome, a number (m) of test cases candidate (chromosomes) is generated randomly. These candidates pass through an evaluation loop and uniform crossover exchanges processes between each two chromosomes and then the best number of chromosomes will be selected from the candidates' pool. The fitness function is used to estimate the goodness of a candidate solution. The fitness function for a test case has been defined as the number of new interaction elements that can cover. An initial random population is generated and then the GA goes to the evaluation loop. Figure 1 shows the outline of the GA algorithm.

**Ant colony based strategy (ACA):** ACA algorithm was motivated by the actions of natural ant colonies in selecting the best path between their colony and food location (Shiba *et al*., 2004). The strategy minimizes test suite size by combining test cases with an interaction coverage guarantee. Fundamentally, the ACA algorithm's procedure is based on the following generalizations:

```
Input: A test set;
Output: A test case;
Begin
Create the initial population P of candidates.
Evaluate P.
While (stopping condition is not met) {
    Select Elite consisting of Ò best individuals from P
    Apply selection to individuals in P to create P_mating,
     consisting of (m- Ò) individuals.
   Crossover and Mutate  P_mating
   Copy the all individuals of P_mating to P
   Replace the worst (m- Ò) individuals in P
   Evaluate P
  If (stagnation condition is met) Mutate P massively
}
Return the best test case found
End
```

Fig. 1: AETG-GA test generation algorithm (Shiba *et al*., 2004)

- Each possible path from the start to end node is assigned to be a solution candidate
- When arriving at the end node, the quality of the current path is evaluated by measuring total quality of pheromone which is placed by the ant at every node in the current path
- A decision among different edges is taken based on a larger amount of pheromone which will be the shortest or optimum path to the target

Figure 2 represents the search space in ACA strategy. Each node represents one parameter and the arrow represents the possible values. The first parameter $f_1$ has three values $\{V_{11}, V_{12}, V_{13}\}$ and the second parameter $f_2$ also has three values $\{V_{21}, V_{22}, V_{23}\}$ while the third parameter $f_3$ has two values $\{V_{31}, V_{12}\}$ and so on, until reaching the last parameter $f_n$ that has two values $\{V_{n1}, V_{n2}\}$.

Candidates' test cases are generated by crawling through the above shown representation. All the ants start from node $f_1$. Each ant moves to the next node by probabilistically selection one value based on the amount of pheromone and heuristic value to guide the ants to each possible branch. When all the ants reach the end point, the candidate's solutions will be evaluated. Then the pheromone will be updated on each edge in the paths during the return back process to the starting node. The process will be repeated, where at every cycle, an extra pheromone will be dropped on each edge. Then the best candidates will be chosen. Figure 3 depicts the outline of the ACA algorithm.

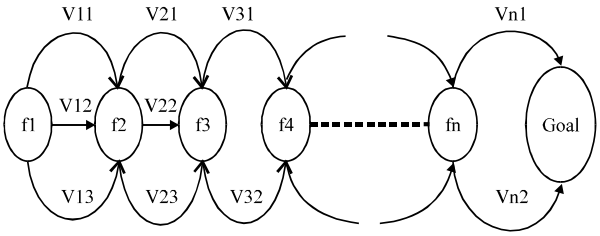


Fig. 2: ACA search space (Shiba *et al.*, 2004)

```
Input: A test set;
Output: A test case;
Begin
Compute local heuristic
Initialize pheromone
While (stopping conditions is not met) {
    For (each ant k) {
        Generate a candidate test case Sk
        Evaluate Sk
        Lay pheromone
    }
    Update pheromone
    If (stagnation condition is met)
    Initialize pheromone
}
Return the best test case found
End.
```

Fig. 3: AETG-ACA test generation algorithm (Shiba *et al.*, 2004)

**PSO and PPSTG strategy:** PSO is a population-based swarm intelligence algorithm introduced by Eberhart and Kennedy (1995) and Kennedy and Eberhart (1995). As a popular optimization method, PSO has been used during the last years since it showed a number of advantages in comparison to other optimization methods. Some of the key advantages are (Sutha and Kamaraj, 2008; Windisch *et al*., 2007):

- PSO has few parameters to regulate and can be easily merged with the environment that needs optimization
- PSO is less sensitive to the nature of the objective function
- PSO does not need the calculation of derivatives that the knowledge of good solutions is kept by all particles and that particle share the information with others in the swarm

Particle swarm optimization tries to manipulate a certain number of candidate solutions at once (Pant *et al*., 2008). The whole population is called swarm and the solution is called particle. Each solution represented by a particle that works in the search space to find a better position or solution of the problem. Hence, each particle has a random position and updates its position iteratively in the hope of finding better solutions. Each particle holds the essential information about its movement. The information are related to the i th particle of interest, including its position currently ($x_i$), its velocity currently ($v_i$), personal best ($pBest_i$), local best ($lBest_i$) and the global best ($gBest_i$).

The manipulation of the particles around the search space is restricted by a certain update and positions rule. It took different forms in the literature. The particles are manipulated according to the following equations (Yang *et al*., 2009; Yap *et al*., 2011):

$$V_{j,d}(t) = w\,V_{j,d}(t-1)+c_1\,r_{j,d}(pBest_{j,d}(t-1)-X_{j,d}(t-1))+c_2 r'_{j,d}(lBest_{j,d}(t-1)-X_{j,d}(t-1)) \quad (1)$$

$$X_{j,d} = X_{j,d}(t-1)+V_{j,d}(t) \quad (2)$$

where, t is iteration number or time, d is the dimension, j the particle index, w is the inertia weight, r and r' are two random factors which are two random real numbers between 0 and 1 and c, c' are acceleration coefficients that are adjusting the weight between components. Here, we have deduced the values for $c_1$ and $c_2$ =1.375 and w = 0.3 based on the existing work by Liang *et al*. (2006) and Windisch *et al*. (2007). Pursuant to such updated rule, each particle updated its velocity for better movement around the search space and the new velocity used to find the new position of the particles depending on a cost factor that controls this movement.

In our PPSTG strategy, we first introduce each particle as a vector. Since each test case has (D) parameters, as a result, the particle or the vector is (D) dimension also. We can illustrate this vector by the notation: $X_j = [X_{j,1}, X_{j,2}, X_{j,d} \ldots\ldots\ldots\ldots, X_{j,D}]$.

In fact, the strategy is a combination of two algorithms: pair generation algorithm and test case generation algorithm. The strategy starts with pair generation algorithm by receiving the parameter values. It will immediately manipulate all parameter values. Then, the algorithm generate a list named Ps that contains all pair interactions of parameter values that are not been covered yet. The illustration example in Figure 4 shows the pair generation algorithm.

The algorithm iterates until Ps get empty. When a test case is found for the test suite (named Ts) that can cover more pair interaction, the test generation algorithm remove the pair interactions which are covered by this test case, from Ps list. The strategy randomly initializes each particle with

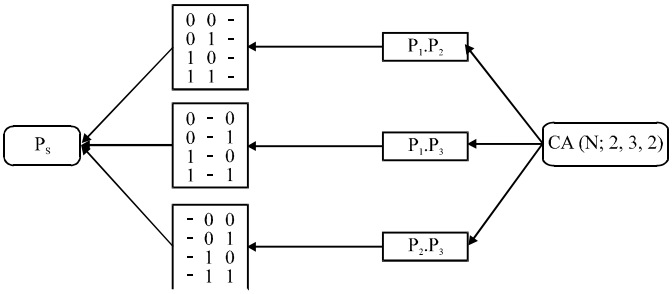


Fig. 4: An illustrative example for pair generation algorithm

```
Input: A test set;
Output: A test case;
Let Ps be a set of all pair interactions of parameter values that are not been covered yet;
Let Ts be a set of candidate tests;
Randomly initialize particles Xi (t) and velocities Vi (t);
While Ps is not empty do {
   Evaluate Xi (t) for its pair interaction coverage with Ps;
   Choose maximum coverage particle to be lBest;
   For a specific number of iterations do {
      Calculate Vi (t+1);
      Move Xi (t) to Xi (t+1) according to Vi (t+1);
      Evaluate Xi (t+1);
      If lBest (t+1) cover bigger pair interactions {
         lBest = lBest (t+1);
         }
      }
   Let gBest be the best test case found;
   gBest = lBest;
   Add gBest to the test set Ts;
   Remove those pair interactions in Ps that covered by Bt;
   }
```

Fig. 5: PPSTG test suite generator algorithm

its associated parameter values. It will compare each particle which represents a test case, with the list of pair interactions element Ps. For each particle, there will be a factor named weight that represents an integer number of covered pair interactions. This weight factor for each particle stored in a vector named cost vector. When the evaluation of all particles finished, the biggest weight and its representative particle will be chosen to be the lBest test case. For the next iteration, the algorithm will update the particles' position according to the update rule. Figure 5 shows the detail steps of the algorithm.

## EXPERIMENTAL RESULTS AND DISCUSSION

To evaluate the performance of our strategy in term of test size against other strategies, we made three stage comparisons with some existing strategies and tools. In the first stage, we compared our strategy with published results from the literature (Cohen *et al.*, 2007; Shiba *et al.*,

2004) for GA, ACA, AETG, mAETG and IPO. In addition, we made comparisons with some existing strategies and tools that were available for implementation. These strategies are IPOG with its tool ACTS (Forbes *et al.*, 2008), Jenny (Grindal *et al.*, 2005), G2Way (Klaib *et al.*, 2008), TConfig (Williams, 2000) and TVG (Nie and Leung, 2011). To ensure fairness, we have downloaded and employed all strategies within our environment which consists of a desktop PC with Windows 7, 2.8 GHz Core 2 Duo CPU, 2 GB of RAM and JSharp installed. In the second stage, we take a uniform binary parameters configuration and vary the parameters from 3 to 15. Finally, in the third stage, we take a uniform of ten parameters configuration and we vary the values from 3 to 10. Here, the rationale for selecting such configurations is to investigate how PPSTG's test size grew with the configurations.

Table 1-3 showed the results obtained for the above-mentioned experiments. As PPSTG generates non-deterministic results, we perform 20 independent runs in order to ensure high statistical confidence. Here, we report the best results of these independent run. Cells marked NA (not available) indicate that the results are unavailable from the literature.

Table 1: Comparison of PPSTG with some existing published results

| Configurations | TVG | PICT | AETG | mAETG | GA | ACA | CTE-XL | TConfig | IPO | IPOG | Jenny | G2Way | PPSTG |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CA (N;2, $3^3$) | 11 | 10 | NA | NA | NA | NA | 10 | 10 | NA | 11 | 9 | 10 | 9 |
| CA (N;2, $3^4$) | 12 | 13 | 9 | 9 | 9 | 9 | 10 | 10 | 9 | 12 | 13 | 10 | 9 |
| CA (N;2, $3^{13}$) | 20 | 20 | 15 | 17 | 17 | 17 | 21 | 20 | 17 | 12 | 20 | 19 | 17 |
| CA (N;2, $5^{10}$) | 50 | 47 | NA | NA | NA | NA | 50 | 48 | 47 | 50 | 45 | 46 | 45 |
| CA (N;2, $10^{10}$) | 189 | 170 | NA | NA | 157 | 159 | 192 | 170 | 169 | 176 | 157 | 160 | 170 |
| MCA (N;2, $5^1$ $3^8$ $2^2$) | 23 | 21 | 19 | 20 | 15 | 16 | 21 | 22 | NA | 19 | 41 | 23 | 21 |
| MCA (N;2, $6^1$ $5^1$ $4^6$ $3^8$ $2^3$) | 41 | 38 | 34 | 35 | 33 | 32 | 39 | 33 | NA | 36 | 31 | NA | 39 |
| MCA (N;2, $7^1$ $6^1$ $5^1$ $4^6$ $3^8$ $2^3$) | 52 | 46 | 45 | 44 | 42 | 42 | 53 | 49 | NA | 44 | 51 | NA | 49 |
| MCA(N;2, $10^1 9^1 8^1 7^1 6^1 5^1 4^1 3^1 2^1$ | 100 | 101 | NA | NA | NA | NA | 102 | 92 | NA | 91 | 98 | NA | 97 |

[1]TVG: Test vector generator, available at: http://sourceforge.net/projects/tvg, [2]PICT: Private independent combinatorial testing, available at:http://msdn.microsoft.com/cn-us/testingg/bb980925.aspx, [3]AETG: Automatic efficient test generator, [4]mAETG: modified Attomatic efficient test generator, [5]GA: Genetic algorithm, [6]ACA: Ant colony algorithm, [7]CET-XL: Classification-Tree editor extended logis, available http://www.berner-mattner.com/en/berner-mattner-home/products/cte-xl, [8]TConfig: Test configuration, [9]IPO: In-parameter-order, [10]IPOG: In-parameter-order-generator, available at: http://csrc.nist.gov/acts, [11]Jenny, available at: http://burtleburtle.net/bob/math/jenny.html, [12]G2Way: Test generator for pairwise. [13]PPSTG: Pairwise particle swarm-based test generator, [14]NA: Not available, [15]P: System parameters

Table 2: Binary parameters growing comparison of PPSTG with existing tools

| P | TVG | PICT | CTE-XL | TConfig | IPOG | Jenny | PPSTG |
|---|---|---|---|---|---|---|---|
| 3 | 4 | 4 | 6 | 4 | 4 | 5 | 4 |
| 4 | 6 | 5 | 6 | 6 | 6 | 6 | 6 |
| 5 | 6 | 7 | 6 | 6 | 6 | 7 | 6 |
| 6 | 6 | 6 | 8 | 7 | 8 | 8 | 7 |
| 7 | 8 | 7 | 8 | 9 | 8 | 8 | 7 |
| 8 | 8 | 7 | 8 | 9 | 8 | 8 | 8 |
| 9 | 8 | 9 | 9 | 9 | 8 | 8 | 8 |
| 10 | 9 | 9 | 9 | 9 | 10 | 10 | 8 |
| 11 | 9 | 9 | 10 | 9 | 10 | 9 | 9 |
| 12 | 10 | 9 | 10 | 9 | 10 | 10 | 9 |
| 13 | 10 | 9 | 10 | 9 | 10 | 10 | 9 |
| 14 | 10 | 10 | 10 | 9 | 10 | 10 | 9 |
| 15 | 10 | 10 | 10 | 9 | 10 | 10 | 10 |

Table 3: 10 Parameters variable values comparison of PPSTG with existing tools

| V | TVG | PICT | CTE-XL | TConfig | IPOG | Jenny | PPSTG |
|---|---|---|---|---|---|---|---|
| 3 | 18 | 18 | 18 | 17 | 20 | 19 | 17 |
| 4 | 33 | 31 | 33 | 31 | 31 | 30 | 29 |
| 5 | 50 | 47 | 50 | 48 | 50 | 45 | 45 |
| 6 | 72 | 66 | 71 | 64 | 68 | 62 | 62 |
| 7 | 98 | 88 | 97 | 85 | 90 | 83 | 81 |
| 8 | 124 | 112 | 125 | 114 | 117 | 104 | 109 |
| 9 | 152 | 139 | 161 | 139 | 142 | 129 | 139 |
| 10 | 189 | 170 | 192 | 170 | 176 | 157 | 170 |

From Table 1, our PPSTG strategy generates test suites with satisfactory results in most part of the experiments. The strategy scales well against other strategies. However, when we take SA, GA, ACA, mAETG and AETG into account, clearly GA and ACA generate slightly better size than AETG and mAETG while SA generates the optimum results in most of the cases due to its large random search space. In case of GA and ACA, the design and implementation of both algorithms (GA and ACA) are mainly depending on AETG algorithm. Here, the final test suite is further optimized by an compaction algorithm that merges the test cases whenever possible for optimality (Shiba *et al*., 2004). As a result, it is not clear whether the implemented algorithms performed better than AETG and mAETG as the GA and ACA algorithms are directly implemented in the AETG framework. Taking our PPSTG strategy into consideration, test size is near that of GA and ACA in most cases and performs better than AETG and mAETG in these specific configurations.

Concerning the size of the generated test suite in Table 2 and 3, our PPSTG strategy outperforms other strategies and seems to give an optimal size in most part of the configurations. The closest competitors to our strategy are Jenny and TVG.

## CONCLUSION

This study discusses the development of pairwise test case generation, called PPSTG, by adopting Particle Swarm Optimization approach. Experimental results demonstrate the effectiveness of PPSTG against other existing strategies. We are currently adding general interaction testing support as well as the support for constraints, seeding, variable strength features. Furthermore, we plan to fine-tune the initial variable setting values (i.e., inertial weight and acceleration coefficient) in order to ensure that PPSTG always generate near optimal results in every case.

## ACKNOWLEDGMENT

This study is partially funded by the generous fundamental grants-"Investigating T-Way Test Data Reduction Strategy Using Particle Swarm Optimization Technique" from Ministry of Higher Education (MOHE) and the USM research university grants-"Development of Variable Strength Interaction Testing Strategy for T-Way Test Data Generation". The first author, Bestoun S. Ahmed, is the USM followship recipient.

## REFERENCES

Bryce, R.C. and C.J. Colbourn, 2007. The density algorithm for pairwise interaction testing. Softw. Test. Verification Reliab., 17: 159-182.

Cheng, C.S., 1980. Orthogonal arrays with variable numbers of symbols. Ann. Stat., 8: 447-453.

Clerc, M. and J. Kennedy, 2002. The particle swarm-explosion, stability and convergence in a multidimensional complex space. IEEE Trans. Evol. Computat., 6: 58-73.

Cohen, D.M., S.R. Dalal, M.L. Fredman and G.C. Patton, 1997. The AETG system: An approach to testing based on combinatorial design. IEEE Trans. Software Eng., 23: 437-444.

Cohen, M.B., M.B. Dwyer and J. Shi, 2007. Interaction testing of highly-configurable systems in the presence of constraints. Proceedings of the 2007 International Symposium on Software Testing and Analysis, July 09-12, ACM, London, pp: 129-139.

Colbourn, C.J., S.S. Martirosyan, G.L. Mullen, D. Shasha, G.B. Sherwood and J.L. Yucas, 2006. Products of mixed covering arrays of strength two. J. Combin. Des., 14: 124-138.

Czerwonka, J., 2006. Pairwise testing in real world: Practical extensions to test case generator. Proceedings of the 24th Pacific Northwest Software Quality Conference, (PNSQC'06), IEEE Computer Society, pp: 419-430.

Eberhart, R.C. and J. Kennedy, 1995. A new optimizer using particle swarm theory. Proceedings of the 6th International Symposium on Micro Machine and Human Science, Oct. 4-6, Nagoya, Japan, pp: 39-43.

Forbes, M., J. Lawrence, Y. Lei, R.N. Kacker and D.R. Kuhn, 2008. Refining the in-parameter-order strategy for constructing covering arrays. J. Res. Natl. Inst. Standards Technol., 113: 287-297.

Grindal, M., J. Offutt and S.F. Andler, 2005. Combination testing strategies: A survey. Softw. Test. Verification Reliab., 15: 167-199.

Hartman, A. and L. Raskin, 2004. Problems and algorithms for covering arrays. Discrete Math., 284: 149-156.

Jie, J., J. Zeng, C. Han and Q. Wang, 2008. Knowledge-based cooperative particle swarm optimization. Applied Math. Comput., 205: 861-873.

Kennedy, J. and R. Eberhart, 1995. Particle swarm optimization. Proceedings of the IEEE International Conference on Neural Networks, Nov. 27-Dec. 1, Piscataway, New Jersey, pp: 1942-1948.

Klaib, M.F.J., K.Z. Zamli, N.A.M. Isa, M.I. Younis and R. Abdullah, 2008. G2Way-a backtracking strategy for pairwise test data generation. Proceedings of the 15th IEEE Conference on Asia-Pacific Software Engineering, Dec. 3-5, IEEE Xplore Press, Beijing, China, pp: 463-470.

Klaib, M.F.J., S. Muthuraman, N. Ahmad and R. Sidek, 2010. Tree based test case generation and cost calculation strategy for uniform parametric pairwise testing. J. Comput. Sci., 6: 542-547.

Kuhn, D.R. and M.J. Reilly, 2002. An investigation of the applicability of design of experiments to software testing. Proceedings of the 27th NASA/IEEE Software Engineering Workshop, Dec. 5-6, IEEE Computer Society, Washington DC., USA., pp: 69-80.

Kuhn, D.R., D.R. Wallace and A.M. Gallo, 2004. Software fault interactions and implications for software testing. IEEE Trans. Software Eng., 30: 418-421.

Lei, Y. and K.C. Tai, 1998. In-parameter-order: A test generation strategy for pairwise testing. Proceedings of the 3rd IEEE International Symposium on High-Assurance Systems Engineering, June 21, Washington DC. USA., pp: 254-261.

Lei, Y., R. Kacker, D.R. Kuhn, V. Okun and J. Lawrence, 2007. IPOG: A general strategy for T-way software testing. Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems, March 26-29, IEEE Computer Society, Tucson, AZ., USA., pp: 549-556.

Lei, Y., R. Kacker, D.R. Kuhn, V. Okun and J. Lawrence, 2008. IPOG/IPOG-D: Efficient test generation for multi-way combinatorial testing. Softw. Test. Verification Reliab., 18: 125-148.

Liang, J.J., A.K. Qin, P.N. Suganthan and S. Baskar, 2006. Comprehensive learning particle swarm optimizer for global optimization of multimodal functions. IEEE Tans. Evol. Comput., 10: 281-295.

Nie, C. and H. Leung, 2011. A survey of combinatorial testing. ACM Comput. Surveys, 43: 1-29.

Pant, M., P. Sharma, T. Radha, R.S. Sangwan and U. Roy, 2008. Nonlinear optimization of enzyme kinetic parameters. J. Boil. Sci., 8: 1322-1327.

Roongruangsuwan, S. and J. Daengdej, 2010. A test case prioritization method with practical weight factors. J. Software Eng., 4: 193-214.

Shiba, T., T. Tsuchiya and T. Kikuno, 2004. Using artificial life techniques to generate test cases for combinatorial testing. Proceedings of the 28th Annual International Conference on Computer Software and Applications, Sept. 28-30, IEEE Computer Society, Washington DC., USA., pp: 72-77.

Sutha, S. and N. Kamaraj, 2008. Particle swarm optimization applications to static security enhancement using multi type facts devices. J. Artif. Intell., 1: 34-43.

Wachowiak, M.P., R. Smolikova, Z. Yufeng, J.M. Zurada and A.S. Elmaghraby, 2004. An approach to multimodal biomedical image registration utilizing particle swarm optimization. IEEE Trans. Evol. Comput., 8: 289-301.

Williams, A.W., 2000. Determination of test configurations for pair-wise interaction coverage. Proceedings of the IFIP TC6/WG6.1 13th International Conference on Testing Communicating Systems: Tools and Techniques, (TCSTT'00), Kluwer, B.V., Deventer, The Netherlands, pp: 57-72.

Windisch, A., S. Wappler and J. Wegener, 2007. Applying particle swarm optimization to software testing. Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation, July 7-11, ACM Press, London, England, pp: 1121-1128.

Yang, L.Y., J.Y. Zhang and W.J. Wang, 2009. Selecting and combining classifiers simultaneously with particle swarm optimization. Inform. Technol. J., 8: 241-245.

Yap, D.F.W., S.P. Koh, S.K. Tiong and S.K. Prajindra, 2011. Particle swarm based artificial immune system for multimodal function optimization and engineering application problem. Trends Applied Sci. Res., 65: 282-293.

Yilmaz, C., M.B. Cohen and A. Porter, 2004. Covering arrays for efficient fault characterization in complex configuration spaces. ACM SIGSOFT Softw. Eng. Notes, 29: 45-54.