AUSTRALIAN JOURNAL OF BASIC AND APPLIED SCIENCES

ISSN:1991-8178 EISSN: 2309-8414
Journal home page: www.ajbasweb.com

A Review of Artificial Intelligence Strategies in Covering Array Construction

Sughan Nair, Adzhar Kamaludin and Kamal Z. Zamli

Faculty of Computer Systems & Software Engineering, Universiti Malaysia Pahang, Lebuhraya Tun Razak, 26300 Gambang, Pahang, Malaysia

Address For Correspondence:
Sughan Nair, A Review of Artificial Intelligence Strategies in Covering Array Construction.
E-mail: mcs14001@stdmail.ump.edu.my

ARTICLE INFO

Article history:
Received 10 December 2015
Accepted 22 January 2016
Available online 30 January 2016

Keywords:
Software Testing; Artificial Intelligence; Covering Array

ABSTRACT

Background: Software systems are getting larger in size and functionality. Exhaustive software testing is becoming nearly impossible with larger systems. **Objective:** Researchers are focusing on methods and strategies to optimize software testing process by applying computational based strategies as well as Artificial Intelligence (AI) based strategy. **Results:** This paper reviews the AI based strategies and its effectiveness in being solution for this optimization problem compared to computational based tools and strategies. **Conclusion:** write the main conclusion for your paper**.**

INTRODUCTION

Design of Experiment (DOE) has been known to be effective for software configuration testing (Hoskins, D.S., 2005). Here, each component of the system is called a "factor," and each test case is called an "experimental run." An experimental run represents a test case to comprehend the system's components, where each component is represented by its valid numeric value or configuration (Chan, F.T.,). When the system is tested exhaustively, the "full factorial" design of the experiment is used. However, when the system is large and the full factorial design is not desirable, the "fractional factorial" design is used to reduce the experimental run to a subset of the full factorial design. The fractional factorial design is used with systems of numeric factors; conversely, systems with categorical factors cannot use this method for experiments (Montgomery, D.C., 2006). The D-Optimality design, on the other hand, has been used with systems, including categorical factors, to reduce the experimental run by selecting a subset of runs from the full factorial. Instead of a purely random subsets selection of experimental runs from the full factorial design, the use of the D-Optimality design method in experiments leads to the production of experimental runs that are closer to full factorial design (Hoskins, D.S., 2005).

Recently, an alternative design based on Covering Array (CA) has been used for the approximation of full factorial design (Hoskins, D.S., 2005). Compared with D-Optimality, empirical evidence demonstrates that CA produces better results than the full factorial approximation experiments (Hoskins, D.S., 2005; Hoskins, D.S., 2004). In such a design, each t-set of factors (or system components) is covered by a set of experimental runs (at least once) to form a CA. Motivated by the effectiveness of CAs, a number of recent studies have focused on the construction of CAs for configuration testing using t-way strategies, where t signifies the interaction strength of the component. These strategies aim to optimally reduce the number of test cases (i.e., the number of rows in the CA) by ensuring that each test case greedily covers the required t-interactions (or t-set of factors) at least once for a typically large space of possible test values. Considering CA construction as an NP hard computational optimization problem (Yilmaz, C., 2004) many strategies based on AI have been developed in the literature

To Cite This Article: Sughan Nair, Adzhar Kamaludin and Kamal Z. Zamli., A Review of Artificial Intelligence Strategies in Covering Array Construction. ***Aust. J. Basic & Appl. Sci.,*** *10(3): 19-24, 2016*

including Genetic Algorithm (GA) (Chen, X., 2009; Shiba, T., 2010), Ant Colony (ACA) (Chen, X., 2009; Wang, Z.Y., 2008; Cohen, M.B., 2008), Particle Swarm Optimization (PSO) (Ahmed, B.S., 2011, Ahmed, B.S. 2012) and more. Several AI based strategies are being reviewed in this article.

Literature:

This section of the article presents an overview of previous works on related field that provide the necessary background for the purpose of this review.

A. Combinatorial Testing:

Today, software systems are built using multiple components. These components often have interactions amongst them to execute a system's function. Often, system faults are caused by unexpected interactions amongst these components (Williams, A.W. and R.L. Probert, 2001). Exhaustive testing, i.e. testing all possible combinations is reasonable for a small system but the number of combinations grow larger with larger systems.

Exhaustive testing is impossible for large and complex systems due to factors including time, cost, and resource constraints. In that case, one approach used is to guarantee that we test all pairs of interactions or all n-way interactions (Cohen *et al*, 2003). For this reason, there is a need for an intelligent sampling strategy that can select a subset of inputs as test data from an inherently large search space. Recently, the widespread use of t-way strategy (where t indicates the interaction strength) has provided a dramatic solution to such a problem. Dalal *et al*. (1999) present empirical results that shows the pairwise interactions testing finds large number of existing faults in a software system. Burr *et al*. (1998) provides more empirical results to support the effectiveness of this type of test coverage. If restricted to pairwise coverage, it is not guaranteed that faults that occur with three or four way interactions can be found (Cohen, M.B., 2003).

Many AI based search strategies (e.g. based on genetic algorithm (GA), Ant Colony algorithm (ACA), Particle Swarm Optimization, as well as Harmony Search Algorithm) have been developed to optimize CA construction.

B. Covering Array:

Covering arrays (CA) are combinatorial designs useful in "pairwise testing" or "t-wise testing" of software system. Since exhaustive testing is impractical in most large systems, CA has been proposed and empirically studied for application to software interaction testing to find errors coming from all t-wise (or smaller) interactions of parameter values, whilst reducing the number of tests needed to be carried out (Hoskins, D.S., *et al*., 2005; Burr, K. and W. Young, 1998). CA has been introduced to complement Orthogonal Array (OA) limitations. Since, OA requires the component values to be uniform; it has been shown to be too restrictive (Wang, Z.Y., *et al*., 2008). CA emerged to overcome this limitation of OA (Shiba, T., *et al*., 2004). Compared with other methods, empirical evidence demonstrates that CA produces better results than full factorial approximation experiments (McCaffrey, 2010; Sthamer, H., 1995). These strategies aim to optimally reduce the number of test cases (i.e., the number of rows in the CA) by ensuring that each test case greedily covers the required t-interactions (or t-set of factors) at least once for a typically large space of possible test values.

A covering array, CA λ (N; *t, k, v*), is an $N \times k$ array for which every $N \times t$ sub array has the property that every *t*-tuple appears as a row at least λ times. Here *t* is the strength, *k* is the number of factors (degree), and *v* is the number of symbols for each factor (order). When λ is 1, every t-way interaction is covered at least once; this is the case of most interest, and we often omit λ when it is 1. The covering array is optimal if it contains the minimum possible number of rows. The size of such a covering array is the covering array number:
CAN (t, k, v)

C. Covering Array NP Optimization Problem:

CA construction has been considered as NP hard computational optimization problem in determining covering array number (Yilmaz, C., 2004; Torres-Jimenez, J., E. Rodriguez-Tello, 2010). Some other applications related to the CA construction problem arise in experimental design where it is absolutely necessary to test the interaction of all combinations of t parameters, and hence that all such selections are covered by columns of the array (Chateauneuf, M.A. 2002).

Addressing the problem of finding the covering array number in reasonable time has been the focus of much research. Amongst the approximate methods that have been developed for constructing covering arrays are recursive methods, algebraic methods, greedy methods and meta-heuristic methods. More and more researches are focused on developing AI techniques in this literature. This is because AI-based strategies naturally excel in dealing with combinatorial optimization problem and outperformed other strategies. AI-based strategies for constructing CA include Genetic Algorithm (GA), Ant Colony (ACA), Simulated Annealing (SA), Particle Swarm Optimization (PSO), Harmony Search Algorithm (HS) and more.

AI Strategies In Ca Construction:

A. Genetic Algorithm for Pairwise Test Sets (GAPTS)[8]:

All the GA based strategies rely on chromosomes of individuals with fitness value which indicates the effectiveness in solving the problem. Chromosomes with high fitness values used to produce offspring solutions which have the same or increased fitness value. Eventually, the fittest chromosomes will dominate the population when the low fitness level chromosomes are replaced by the offspring.

McCaffrey (2010) proposed Genetic Algorithm for Pairwise Test Sets (GAPTS) as a possible solution for generating pairwise test sets (Ahmed, B.S., 2011). GAPTS is tested against seven benchmark input sets, and the results are compared with other pairwise test generation algorithms. GAPTS uses the population size of 20. This size is relatively small compared to other genetic algorithms. GAPTS's effectiveness did not increase with larger population size with respect to the final pairwise test set size.

GAPTS algorithm uses roulette wheel method in selecting the individuals as basis of offspring production. Being the common selection method in GA, roulette wheel selects individuals for GAPTS by the ratio of the fitness value to the sum of all fitness value in the population. For crossover, GAPTS uses single-point crossover method. To ensure each crossover operation produces two valid chromosome-solutions, GAPTS ensures the crossover point location fell on the test vector boundary. A fixed mutation rate of 0.001 used in GAPTS. Each gene is independently mutated to new legal gene value with probability equals to mutation rate.

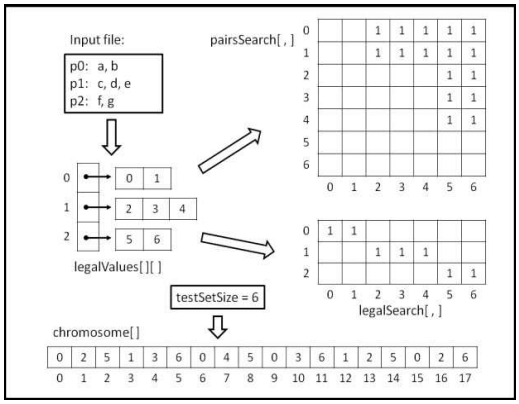

Fig. 1: Principal GAPTS Data Structure.

GAPTS uses two mechanisms to address the population stagnation effect. Form of elitism used to ensure chromosomes with highest fitness values are immune from removal in each generation. A form of immigration used to randomly generate chromosomes to be inserted for every 1000 generation. GAPTS algorithm is implemented in C# programming language by using published results as guidelines. McCaffery's study focus on investigating effectiveness of GAPTS with respect to test set size. GAPTS test results then compared and analyzed with results from other pairwise test set generation tools. Promisingly, GAPTS produce comparable or better results than other tools in 39 out of 40 instances. However GAPTS algorithm required more generation time than other tools. GAPTS took significant time as the input size increased. McCaffrey mentioned in his study that time required by different algorithms is not a significant factor in most software testing scenarios.

*B. Ant Colony System Based Variable Strength Interaction Testing (ACS – VSIT) (*Chen, X., 2009*):*

The VSIT proposed by Chen *et al*. is based on Ant Colony System (ACS), a promising variant of Ant Colony Optimization algorithm (ACO). ACS has advantage over other variants of ACO because its edge selection rule which provides the direct way to the balance between new edge exploration and accumulated knowledge about the problem. Besides that, the global updating rule is applied only to the best solution available. ACS is also capable of applying local pheromone updating rule whilst ants construct a solution.

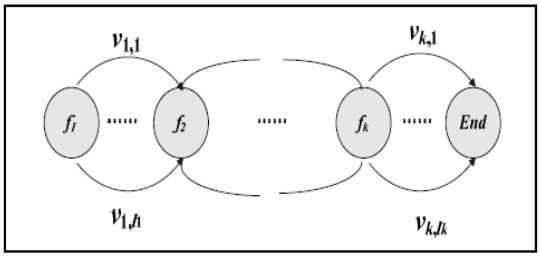

Fig. 2: Graph representation of solution space.

Figure 2 shows the solution space of the ACS used to generate a single test. ACS simulate the behavior of artificial ants crawling on this graph. Pheromone is a variable associated with each edge, readable and modified by those artificial ants. Chen *et al*. approach adopts one-test-a-time strategy. This strategy initializes and empty the test suite at first, then continues to generate a test, removes the interaction covered by the test and adds it to the test suite. The loop is terminated and test suite returned when all the interactions are covered.

This test suite generation can be summarized as follow. A set of ants are initially put on a node where each ant repeatedly apply the edge selection rule to build a solution. Both the heuristic information and the pheromone information will guide the rule. Local updating rule will be applied by the ants to modify the pheromone amount on the visited edges. Pheromone amount is modified again once all ants have built their solutions. This will be done by applying global updating rule. In Chen's algorithm, heuristic values are decided by considering both the number of interactions and its corresponding strength. One-test-a-time strategy has a characteristic that, earlier generated tests have more interaction coverage. That affects the interaction coverage ability of tests which are generated later. Chen *et al*. adopts tests minimization algorithm to merge some of the tests without compromising the interaction coverage and therefore reduce the final test suite size.

The experimental run of the proposed algorithm is designed in Java. The inputs are obtained from a previous research by Cohen, which is based on Simulated Annealing (SA). The best results out of 20 experimental runs then compared with Cohen's result and some other greedy algorithms such as Density, ParaOrder and tools such as PICT and TVG. Chen's results analyze thoroughly in comparison with all these previous research results. ACS based VSIT nearly resemble Cohen's SA approach results and outperformed the rest of the methods and tools. Although, Cohen's method is proven to yield better results of constructing CA in most of the cases, ACS managed to produce similar results with Cohen's method. Chen *et al*. even records the time taken for ACS VSIT to generate those CA, which put Chen's research in between Cohen's algorithm and the other algorithms and tools that have been analyzed in terms of performance.

C. Particle Swarm Test Generator(PSTG) (Ahmed, B.S., 2011; Ahmed, B.S., 2012):

PSTG is a Particle Swarm Optimization (PSO) based strategy (Ahmed *et al*., 2012). PSO simulates the swarm behavior of flocks of birds or schools of fish. Just like previous strategy, PSO also includes local and global searches to manipulate candidate solutions. Each candidate solutions are known as particle and the whole population is called as a swarm. By recording information about its movement, each particle works in the search space to find a better solution. This information is related to particle of interest, which includes velocity, current position, personal best, local best and global best. PSTG is composed of two main algorithms; a Combination Generator Algorithm and a Test Suite Generator Algorithm.

PSTG combination generator algorithm generates the parameters combinations and the values for each parameter combination. Parameters generated in binary, where 0 indicates exclusion and 1 indicates inclusion of the particular parameter. Interaction elements are generated accordingly by using those parameter combinations. This algorithm then store generated interaction element in an indexed list. Index of each group of the interaction elements is then recorded in indexing record to facilitate searching.

PSTG test suite generator algorithm adopts the discrete version of PSO. Swarm search space initialized as a D-dimensional vector, where each dimension represents a parameter, and contains integer numbers between 0 and number of values of the ith parameter. PSTG also initializes the velocity of each particle. During iteration, particle's velocity is updated to its current position. Velocity will be rounded to the nearest integer by PSTG in case where non-integer velocities are produced. PSTG adopts simplest topology as particles in a swarm matrix array choose the neighbors next to each other.

Ahmed *et al*. compared their experimental results with existing AI based strategies and computational based strategy. By analyzing results from their work, we can observe GA and ACA perform slightly better than AETG, mAETG and PSTG. Due to large search space, SA outperformed all other strategies in most cases. They extend their experiment with PSTG by comparing with computational based strategies and tools such as IPOG, WHITCH, Jenny, TConfig and TVG. PSTG outperforms these computational strategies in most of the configurations. Even when PSTG is not at its best, the test size still fall within the acceptable range. PSTG is proven to be competitive with other strategies in most of the cases.

D. Enhanced leader PSO (ELPSO)(Rezaee Jordehi, A., 2015):

ELPSO is another variant of PSO designed for solving global optimization problem. All PSO based strategies suffers from premature convergence. When solving complex optimization problem, these PSO strategies often get trapped in local optima. The weak exploration capability of PSO algorithms causes problems in optimization, especially multimodal problems.

ELPSO is developed to have a strong explorative and exploitative capability. In conventional PSO, the particles may converge prematurely without exploring the search space because they are attracted to swarm leader. Also in PSO, there is no mechanism to jumping out from local optima, if the particle is trapped in local optimum. ELPSO is designed to solve this two common problems. Main characteristic of ELPSO is to enhance

the swarm leader in every iteration of search. In ELPSO, Swarm leader will undergo a five-staged successive mutation in each iteration. At the end of each mutation, new swarm leader will replace current swarm leader if it has higher fitness value than the current leader. The premature convergence problem is expected to be mitigated since different portion of search space is explored to find the best swarm leader in each iteration. In successive mutation, the swarm leader is only replaced by the better leader. This ensures the whole swarm is attracted towards the region that produced the swarm leader with the higher fitness. This region has more good objective values and leads to achievement of more quality solution.

Five-staged mutations in ELPSO are classified in two groups; short jump mutation and long jump mutation. Whilst short jump mutations help in avoiding premature convergence, high jump mutations ensure the particles are not trapped in local optima. These long jump mutations play role as efficient jump-out mechanism for particles to jump out of the local optima. In ELPSO, mutations in stage 1, 2 and 3 are known as long jump mutations to increase the particles capability to jump out of local optimum after stagnation.

In Jordehi's experiment, ELPSO is validated by comparing with other meta-heuristic optimization algorithms such as conventional PSO, firefly swarm optimization (FSO), gravitational search algorithm (GSA), brainstorm optimization (BSOA), artificial bee colony (ABC), HS and GA. The results, approve the outperformance of ELPSO in terms of accuracy, convergence rate and scalability.

Conclusion:

All the reviewed strategies show the effectiveness of AI based strategies in CA construction. McCaffery's GAPTS is proven to produce better or comparable results than other computational based tools in 39 out of 40 instances. Whilst, ACS-VSIT produces comparable results with Cohen's SA [21] algorithm and outperformed computational based tools as similar to GAPTS. PSTG comparable results with other AI based algorithms and once again produces better results than computational based tools.

Based on these, we can conclude that AI based strategies outperformed computational based tools and strategies in dealing with combinatorial optimization problem. AI based strategies have been proven as better solutions in finding optimized CA number in reasonable time compared to computational based strategies.

However, existing works prove that, major drawbacks of these algorithms are: long generation time due to repeated loop that try to find the best and near optimal solution and these algorithms only focus on sequence-less CA.

This review will be helpful for researchers to derive or manipulate new variant from existing artificial intelligence strategies to overcome the common limitations and drawbacks in generating covering arrays.

ACKNOWLEDGEMENT

We would like to thank University Malaysia Pahang and Faculty of Computer Systems & Software Engineering for providing us a platform for our research. We would also like to extend our sincere gratitude to Ministry of Education, Malaysia for providing Exploratory Research Grant Scheme to support the research financially

REFERENCES

Hoskins, D.S., C.J. Colbourn, D.C. Montgomery, 2005. Software Performance Testing Using Covering Arrays: Efficient Screening Designs with Categorical Factors, in: Proccedings of the 5th International Workshop on Software and Performance, ACM, Palma, Illes Balears, Spain, 131-136.

Chan, F.T., T.Y. Chen, I.K. Mak, Y.T. Yu, Proportional Sampling Strategy: Guidelines for Software Testing Practitioners,

Montgomery, D.C., 2006. Design and Analysis of Experiments, John Wiley & Sons.

Mitchell, T.J., 2000. An Algorithm for the Construction of "D-optimal" Experimental Designs, Technometrics, 42: 48-54.

Hoskins, D.S., R.C. Turban, C.J. Colbourn, 2004. Experimental Designs in Software Engineering: D-Optimal Designs and Covering Arrays, in: Proceedings of the ACM Workshop on Interdisciplinary Software Engineering Research, ACM, Newport Beach, CA, USA, 55 - 66.

Yilmaz, C., M.B. Cohen, A. Porter, 2004. Covering Arrays for Efficient Fault Characterization in Complex Configuration Spaces, ACM SIGSOFT Software Engineering Notes, 29: 45-54.

Chen, X., Q. Gu, A. Li, D. Chen, 2009. Variable Strength Interaction Testing with an Ant Colony System Approach, in: Proceedings of the 16th Asia-Pacific Software Engineering Conference, IEEE Computer Society, 160-167.

McCaffrey, 2010. An Empirical Study of Pairwise Test Set Generation Using a Genetic Algorithm, in: Proceedings of the 7th International Conference on Information Technology, IEEE Computer Society, 992-997.

Sthamer, H., 1995. The Automatic Generation of Software Test Data Using Genetic Algorithms, in, University of Glamorgan,Pontyprid, Wales.

Shiba, T., T. Tsuchiya, T. Kikuno, 2004. Using Artificial Life Techniques to Generate Test Cases for Combinatorial Testing, in: Proceedings of the 28th Annual International Computer Software and Applications Conference, IEEE Computer Society, 72-77-71.

Wang, Z.Y., B.W. Xu, C.H. Nie, 2008. Greedy Heuristic Algorithms to Generate Variable Strength Combinatorial Test Suite, in: Proceedings of The 8th International Conference on Quality Software, IEEE Computer Society, 155-160.

Cohen, M.B., C.J. Colbourn, A.C.H. Ling, 2008. Constructing Strength Three Covering Arrays with Augmented Annealing, Discrete Mathematics, 308: 2709-2722.

Ahmed, B.S., K.Z. Zamli, C.P. Lim, 2011. Constructing a T-Way Interaction Test Suite Using the Particle Swarm Optimization Approach, International Journal of Innovative Computing, Information and Control, 7: 1-10.

Ahmed, B.S.and Kamal Z. Zamli, and C.P. Lim, 2012. Application of Particle Swarm Optimization for Uniform and Variable Strength Covering Array Construction, Applied Soft Computing Journal, Elsevier, 12(4): 1330-1347.

Williams, A.W. and R.L. Probert, 2001. A measure for component interaction test coverage. In Proc. ACS/IEEE Intl. Conf. on Computer Systems and Applications, 301-311.

Cohen, M.B., Peter B. Gibbons, Warwick B. Mugridge, and Charles J. Colbourn. 2003. Constructing test suites for interaction testing. In Proceedings of the 25th International Conference on Software Engineering (ICSE '03). IEEE Computer Society, Washington, DC, USA, 38-48.

Dalal, S.R., A.J.N. Karunanithi, J.M.L. Leaton, G.C.P. Patton, B.M. Horowitz, 1999. Model-based testing in practice. In Proc. of the Intl. Conf. on Software Engineering,(ICSE '99), 285-94, New York.

Burr, K. and W. Young, 1998. Combinatorial test techniques: Table-based automation, test generation and code coverage. In Proc. of the Intl. Conf. on Software Testing Analysis & Review, San Diego.

Torres-Jimenez, J., E. Rodriguez-Tello, 2010. Simulated Annealing for constructing binary covering arrays of variable strength, Evolutionary Computation (CEC), IEEE Congress on, 1(8): 18-23 DOI: 10.1109/CEC.2010.5586148.

Chateauneuf, M.A. and D.L. Kreher, 2002. On the state of strength-three covering arrays, Journal of Combinatorial Design, 10(4): 217–238.

Cohen, M.B., P.B. Gibbons, W.B. Mugridge, C.J. Colbouns, J.S. Collofello, 2003. Variable strength interaction testing of components, In Proceedings of 27th Annual International Computer Software and Applications Conference, 413-418.

Rezaee Jordehi, A., 2015. Enhanced leader PSO (ELPSO): A new PSO variant for solving global optimisation problems, Applied Soft Computing Journal, Elsevier, 26: 401-417.