Test Case Prioritization in a Specification-based Testing Environment

Gary Yu-Hsin Chen
Department of Industrial & Systems Engineering, Chung Yuan Christian University, Chung Li, Taiwan
Email: yuhsin@cycu.edu.tw

Pei-Qi Wang
Department of Industrial & Systems Engineering, Chung Yuan Christian University, Chung Li, Taiwan
Email: ty880721@yahoo.com.tw

***Abstract*—The topic of test case prioritization has been researched extensively in the past decade. However, current researches carried out on test case prioritization are mainly concerned with independent test cases in a structural testing environment. In a specification-based testing environment, however, some test cases are inter-case dependent and must follow certain sequences of execution. The objective of this research is to propose "prioritizing factors" that better reflect the real-world scenario for test case prioritization in the specification-based environment: (1) requirement severity score and (2) inter-case dependency, and to optimize the test case arrangement through the application of meta-heuristics. The inter-case dependency can be formulated as a sequential ordering problem (SOP), a NP-complete problem for which the precedence relationship exists. Two meta-heuristics, the Genetic Algorithm and Ant Colony Optimization, are used to prioritize the test cases.**

***Index Terms*—Specification-based testing, test case prioritization, inter-case dependency, Maximum Partial Ordering/Arbitrary Insertion, Ant Colony Optimization, Genetic Algorithm**

I. Introduction

Test cases hold an important role to determine the success of a software application. Based on a study by the National Institute of Standards and Technology (NIST) in 2002 [1] , it is found that software defects cost the U.S. economy $59.5 Billion annually. Software testing is one of the major activities performed in the software development life cycle to avoid such scenarios from happening. Although software is playing an increasingly important role in today's systems, large or small [2], the software quality assurance is still more of an art than a science [3]. Essentially, the software testing is the "gate keeping" stage necessary to ensure that the quality of software has met customers' expectations.

The landscape of software testing has expanded since the Myer's trailblazing work dated back in 1979, *The Art of Software Testing* [4]. The software testing has evolved into several categories based on their unique characteristics and usage such as the structural (white-box) and specification-based (black-box) testing. Structural testing approach let the developers to have access to the software source code or work on the software directly. On the other hand, the specification-based approach treats the software under test (SUT) as a "black box". Black box software testing is a method where software testers responsible for testing the software do not have the knowledge of the software's internal structure. The idea is to let the software testers independently test the software as if they themselves are the users, and verify whether the software output matches their expectation.

Organization with a group of dedicated testing staff typically creates test cases for specification-based testing. This method allows software testers to start testing immediately with a relatively short ramp-up time. Furthermore, they would also view the software under test more objectively by avoiding the emotion attachment to the "labor of love"—or recently known as the "IKEA" effect [5, 6].

To test the software functionalities, regardless of structural or specification-based testing, the software testers typically would design and execute a list of test cases. A test case is a detailed step-by-step procedure which examines some aspects of the software, including inputs and outputs, the expected results and other relevant elements [7]. A good test case must be able to yield some information about the software under test [8].

The Institute of Electrical and Electronics Engineers (IEEE) provides the guidelines for designing test cases in the standard IEEE 829-2008 where the following sections should be included: test case specification identifier, test items, input specifications, output specifications, environmental needs, special procedural requirements, and inter-case dependencies [9]. A group of test cases is collectively referred to as a test suite, which examines all aspects (behaviors and operations) of a particular software program. Formally, the test suite and its test cases are defined in a document called the software test plan (STP).

Test cases, which examine software based on a set of customer's requirements, are generated from software deliverables at the requirement analysis stage. Software deliverables specifically consist of statement of work (SOW), consortia specifications, and software requirement specification (SRS). By definition, "a software deliverable is a project result that is delivered to the customers at the end of some major phase such as specification or design" [10].

In Figure 1, the precedence relationships of various deliverables are depicted. At the beginning of the project, the requirements must be identified and specified. There are several ways this can be done—through the SOW provided by customers or consortia specifications, among others. It really depends on whether the customer is known or the general public. A SOW is an official document from customers that states what are to be implemented for a software project—acting as a document for software vendor's bidding and outlining the general software functionalities. Another source of requirements also can be acquired at the external specification from consortia such as World Wide Web Consortium (W3C) or The 3rd Generation Partnership Project (3GPP) for wireless technologies. General requirements from those documents are in turn translated into a set of system requirements.

Since a system is complex, the system is typically broken into smaller components; each component has its own requirement specification called "Software Requirement Specification" or SRS. Based on the SRS, developers then can develop designs and generate the document, software design document (SDD), while software testers create STP before executing test cases. All those deliverables are linked through the requirement traceability: the detailed requirements from the SRS can be traced back to the SOW, while designs (SDD) and testing (STP) must refer to detailed requirements in the SRS. The relationship of requirements is specified in a document called the traceability matrix.

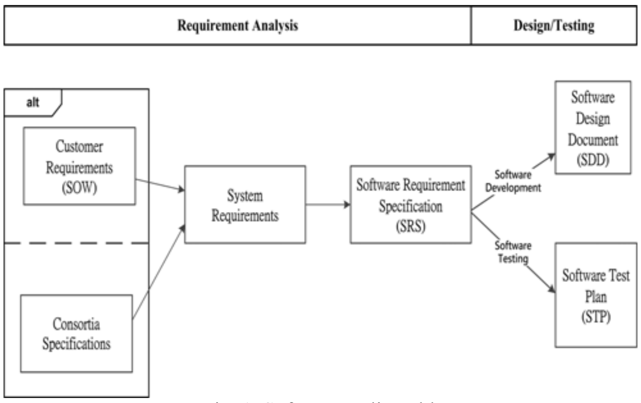


Fig. 1. Software Deliverables

Because the software systems have been becoming large and complex in today's environment, numerous test cases are created to cover those functionalities. How to prioritize those test cases in order to meet the deadline requirements becomes a difficult yet essential task. Traditionally, the planning for the execution of software test suite is performed manually between the software test engineers and project managers. Considerations for arranging the test cases include the test case prioritization and dependencies among test cases. The manual approach works sufficiently for a small test suite but not for a sophisticated software system that calls for hundreds of test cases and more. Thus, test case prioritization techniques for automating the process have been researched, aiming at prioritizing test cases according to some criteria.

The concept of Test case prioritization has been proposed for the past ten years; however, researches into this field mainly concentrate on the structural testing. On the other hand, test case prioritization on the specification-based testing has received a little or no attention although most testers conduct the specification based testing in the software industry [11]. Furthermore, current researches on test case prioritization have assumed the test cases to be completely independent from each other. In reality, many test cases are dependent on other test cases and thus inter-case dependencies should be explored [12].

In this paper, the research considers the test case prioritization from the perspective of specification-based testing. Several considerations are covered: the relationship between requirements and test case prioritization, the metrics for measuring the efficiency of the test case prioritization, and inter-case dependency. By incorporating those factors, we believe it better reflects the true world scenario.

The rest of the paper is organized as follows. Background information on test case prioritization is discussed in section 2. Section 3 presents the prioritizing factors that impact the test case prioritization. The methodology is outlined in Section 4. In section 5 the experimental setup is discussed. In section 6 we present the results and discuss the findings. Finally, the conclusion and some future work directions are mentioned in section 7.

II. Background: Test case Prioritization

Arranging test cases based on certain criteria have been discussed in the literature. Rothermel [13] have coined the word "*test case prioritization problem*" or TCP and given it the formal definition:

Given: *T, a test suite, PT, the set of permutations of T, and f, a function from PT to the real numbers.*

Problem: *Find* $T' \in PT$ *such that* $(\forall T'')(T'' \in PT)(T'' \neq T')[f(T') \geq (T'')]$

In the definition, *PT* represents the set of all possible prioritizations (orderings) of *T*, and *f* is a function that, applied to any such ordering, yields an award value for that ordering.

Several techniques have been developed by Rothermel et. al [13] to prioritize the execution of existing test cases by exposing faults early in the regression testing process. They have also developed a weighted average of the percentage of faults detected, or APFD, which corresponds to the function *f* in the definition above. Because APFD is developed with the number of faults known in advance, it may not be practical for the black box testing environment [14]. Instead, another metric based on the run time execution and test history is proposed.

Li et al.[15] study five search algorithms for regression test case prioritization, which include a Greedy Algorithm, an Additional Greedy Algorithm, a 2-Optimal Algorithm, a Hill Climbing, and a Genetic Algorithm (GAs). The research concludes that the Greedy Algorithm performs worse than the other algorithms, and meta-heuristic algorithms like GAs generate quite encouraging results. Additionally, they also propose three new metrics for the test case prioritization: average percentage block coverage (APBC), average percentage decision coverage (APDC), and average percentage statement coverage (APSC).

Other than those search algorithms mentioned above, more recent meta-heuristics such as Particle Swarm Optimization (PSO) has been proposed to prioritize the test cases. Hla et. al apply the PSO to "prioritize the test cases to the new best positions based on modified software units to spend as little resource on retesting as possible." [16]

In addition to the algorithmic approaches, model-based techniques for TCP are on the rise based on a current study [17]. Zhou, Okamura and Dohi [18] have applied Markov chain Monte Carlo random testing to create the sequence of test cases due to its effectiveness in the framework of random testing. Rajarathinam and Natarajan [19] prioritize test cases by using the trace event techniques. Furthermore, latest TCP discoveries for handling multiple versions of software are addressed in [20], while a latest TCP investigation for GUI applications is conducted by Sun, et al. [21].

Among the papers discussed above, most of them measure the success based on APFD or code/statement coverage in the structural testing environment. The researchers of this study propose to prioritize the test cases based on the inter-case dependency and test case severity in the specification-based testing environment.

III. Prioritizing Factors

The research is conducted under the specification-based testing environment. We generate use cases and later a software requirement specification through analyzing the interactions between users and software applications. Based on the software requirement specification we create test cases without understanding the internal structure of software applications. Two factors of test cases are considered: (1) requirement severity score and (2) inter-case dependency for each test case. The description of each factor is described in the following sections.

A. Test Case Requirement Severity Score

Many researches base test case severity on fault severities or number of faults. However, fault severities or number of faults can only be obtained in the white-box testing environment. Average percentage of fault detection (APFD) suffers from the circularity issue, "if all the faults are presumed in a software application, why those test cases are still needed?" [22] Instead, we measure the test case severities based on the impacts of customer's requirements in the specification-based environment since each test case must be mapped to requirements. Therefore, the impacts of requirements are closely related to the test cases− a highly important requirement may have much higher chance of jeopardizing the software application compared to a less important requirement.

Combining several studies on requirements and defects reporting [23, 24], we categorize requirements into four priority levels to be applied to functional as well as non-functional requirements. For each priority level, a number is assigned ranging from 1 to 4, with category 1 being core requirements and category 4 being optional requirements. Please see Table 1 for detailed information. Additionally, each requirement is assigned a severity score; a test case may have several requirements associated with it. A test case may have a combined requirement severity score. We may assume that test case and requirements can be in either one-to-many (1-to-N) or many-to-one relationship (N-to-1).

TABLE 1.
DESCRIPTION OF REQUIREMENT PRIORITY LEVELS

Level Description	Priority Level
Showstopper: The system must provide this feature to be functional.	1
Critical: The system may function but would cause severe inconveniences and challenges unacceptable to users	2
Medium: The feature may improve the usage of system and give users more incentives to use the system.	3
Low: A cosmetic feature usually related to customers' preferences and usability.	4

We evaluate the importance of each test case based on its requirement severity score. For example, a test case *A* may cover three requirements with the severity levels of 2 each; the other test case *B* may cover only two requirements with the severity levels of 1 and 4. In order to compare the relative importance of each test case, the researchers calculate the test case's overall requirement severity score through the "maximum element method" [25]:

$$S_j = \sum_{i=1}^{m} n_i(t_j^*)(K+1)^{m-i} \quad (j=1,2,\ldots n) \qquad (1)$$

$$\textbf{s.t.}\ K = Max(k_1, k_2, \ldots, k_n) \qquad (2)$$

$$\sum_{i=1}^{m} n_i(t_j^*) \le K \quad (j=1,2,\ldots n) \qquad (3)$$

where S_j represents the requirement severity score, *n* the number of test cases, *m* the severity levels; t_j* the requirement severity level(s) for the test case *j*; $n_i(t_j^*)$ the different requirement severities for test case *j*; *K* the maximum number of requirements covered by a single test case in the test suite. To illustrate the application of the aforementioned method, suppose there are only two test cases, *A* and *B*, in a test suite. Test case *A* contains three requirements with the same requirement severity level of 2; test case *B* covers two requirements with different requirement severity levels of 1 and 4. *K* equals to 3 because the test case with the most requirements is test case *A*—with three requirements. The severity score of each test case is calculated as follows:

$$A = S_1 = (0)(3+1)^{4-1} + (3)(3+1)^{4-2} + (0)(3+1)^{4-3} + (0)(3+1)^{4-4} = 48$$

$$B = S_2 = (1)(3+1)^{4-1} + (0)(3+1)^{4-2} + (0)(3+1)^{4-3} + (1)(3+1)^{4-4} = 65$$

Throughout the calculations, test case *B* appears to be more urgent in terms of its severity score. In reality, it does make sense to execute the test case with the *showstopper* requirements as soon as possible even if other requirements in the same test case are less important.

B. Inter-case Dependencies

For this prioritizing factor the following questions may be encountered, "What tests have to be executed before this one, why, and what if the program fails them?" [26, 27] *Inter-case dependency* is an integral part of the test specification that the Institute of Electrical and Electronics Engineers (IEEE) lists in the IEEE Standard 829-2008 [9]. For more supporting evidence, Onoma et al.[28] states that dependencies among the test cases require them to be execute in a specific sequence. For example, to test the robustness of software, the so-called stress or disaster recovery testing must be performed. Usually this type of testing is not performed until other test cases have been executed. In another scenario, if a start-up test case that involves the software installation and the system power up fails, the sequential test cases in the test suite cannot be executed. That is, if the system cannot successfully execute the core functionalities, the execution of entire test suite is suspended. For example, the execution of some test cases depends on the output from the test cases executed before them. Table 2 provides the description of different dependency levels among test cases.

TABLE 2.
DESCRIPTION OF DEPENDENCY LEVELS

Dependency Level Description	Levels
Extreme Dependency: The test case pair is extremely related and one must be executed right after another.	4
High Dependency: The test case pair is closely dependent.	3
Medium Dependency: The test case is relatively close.	2
Low Dependency: The test case is not particularly related.	1
No Dependency: The test cases are independent from each other.	0
Precedence Constraint Dependency: The precedence constraint exists between two test cases – test case a must be executed ahead of test case b.	-1

C. Normalization of Test case Severity and Inter-case Dependency Scores

The formula of ranging scaling for normalizing the scores of test case severity and inter-case dependency is shown as follows:

$$f(x) = C\left(1 - \frac{x-A}{B-A}\right) + D\left(\frac{x-A}{B-A}\right) \qquad (4)$$

where [*A*,*B*] is the range to be linearly transformed to [*C*,*D*]. In this research we intend to use the scale with the range of [0,100] for both scores. For example, the score of inter-case dependency, 16, is transformed to the value of 50.

D. Mathematical Formulation of Prioritization Factors

The mathematical formulation for prioritizing factors, inter-case dependency and test case severity, is based on the sequential ordering problem (SOP) [29]. SOP is a NP-complete problem, whose objective is to find the minimum cost on a Hamiltonian path subject to the precedence constraints among the nodes. SOP has many practical applications ranging from freight transportation [30] and crane scheduling in port terminals [31] to helicopter scheduling [32] and automotive paint shops [33].

As far as the authors are aware of, this may be the first application of SOP on the test case prioritization. Similar to the SOP, the precedence constraints in this problem are imposed for the inter-case dependency for the test case prioritization, and the objective is to find the minimized cost from a group of viable solutions. Furthermore, we also consider the test case severity, which does not have its counterpart in the SOP.

minimize

$$TCP_{totalScore} = \sum_{k=1}^{n-1} d(C_{\pi(k)}, C\pi_{(k+1)}) + TS_{C\pi(k+1)} \qquad (5)$$

subject to if $d(C_i, C_j) = -1$ (6)
for $i = T(k), j = T(l)$
then $l < k$

where
n : number of test cases

π : test case sequence in the test suite
k : a particular test case in the test suite
d : inter-case dependency
TS: test case severity
C_i : test cases ($i = 1, 2, \ldots, n$)
C_j : test cases ($j = 1, 2, \ldots, n$)
$\pi\ (k)$: the k^{th} position in the test case sequence
$\pi\ (l)$: the l^{th} position in the test case sequence
$C_{\pi(k)}$: the test case at the k^{th} position of the test case sequence π
w_1: weight assigned to test case severity
w_2: weight assigned to inter-case dependency

If the precedence constraint exists between test cases—for example, test case i cannot be executed ahead of test case j— then the dependency is *-1* between test case i and j. If the test case i is at the k^{th} position of the sequence and test case j is at the l^{th} position of the sequence, the test case i cannot be executed before the test case j, then the l^{th} position of test case j must be in front of test case i (at k^{th} position) or $l < k$.

IV. METHODOLOGY

Two approaches are taken to solve the test case prioritization in the specification-based environment: (1) ant colony optimization (ACO) and (2) genetic algorithm (GA). The ant colony optimization is based on the foraging behaviors of ants while the genetic algorithm is inspired by the natural chromosomal crossover and genetic mutation. The ant colony optimization is based on the constructive ACO algorithm [34] while the GA is based on the [29]. Both meta-heuristics are incorporated with the Maximum Partial Order (MPO) and Arbitrary Insertion (AI) mechanism[29] for finding the minimized test case prioritization based on the test case severity and inter-case dependency. Henceforth, those meta-heuristics are referred to as ACO MPO/AI and GA MPO/AI. The maximum partial order/arbitrary insertion are described in greater details and a small example is given in the following sub-sections.

A. Maximum Partial Ordering/Arbitrary Insertion

The maximum partial order aims at finding the longest partial order that consists of the commonalities of two parents, two solution entities which may be ants in the ACO or individuals in the GA. Then through the arbitrary insertion the remaining test cases are inserted into the partial order graph. The major steps of the maximum partial order are displayed in Fig. 2.

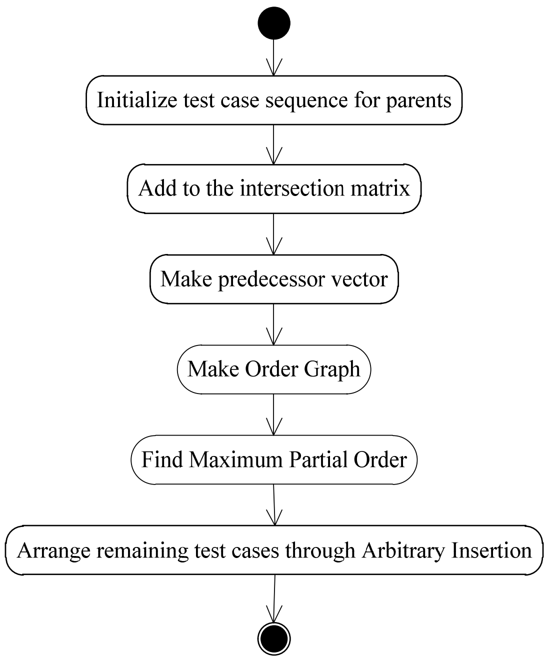

Initialize test case sequence for parents
Add to the intersection matrix
Make predecessor vector
Make Order Graph
Find Maximum Partial Order
Arrange remaining test cases through Arbitrary Insertion

Fig. 2. Maximum partial order/arbitrary insertion process

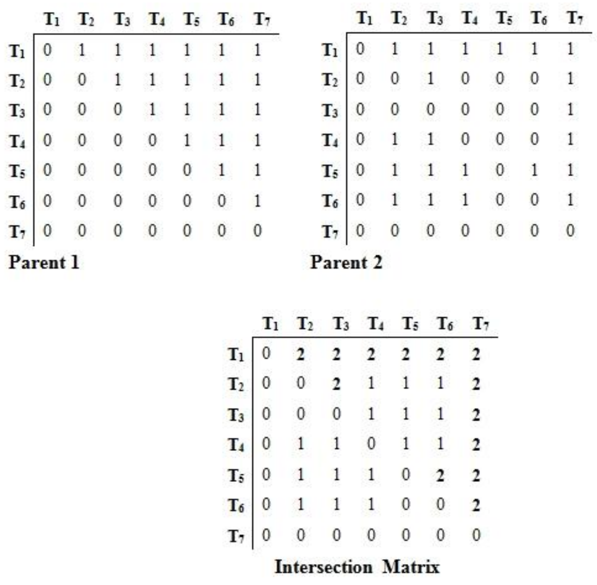
B. Initializing the Test Case Sequence

The test case sequences are initially randomly generated for each "parent", which refers to the solution entities of the meta-heuristics. In the small example, the solution entity consists of seven test cases in various sequential orders.

Parent 1: $T_1T_2T_3T_4T_5T_6T_7$
Parent 2: $T_1T_5T_6T_4T_2T_3T_7$

C. Adding to the Intersection Matrix

The inter-case dependencies among the test cases are represented by the n × n matrices. If T_i is followed by T_j, the entry in the matrix is incremented by 1 and 0, otherwise. Both matrices are added together to form the intersection matrix.



	T1	T2	T3	T4	T5	T6	T7
T1	0	1	1	1	1	1	1
T2	0	0	1	1	1	1	1
T3	0	0	0	1	1	1	1
T4	0	0	0	0	1	1	1
T5	0	0	0	0	0	1	1
T6	0	0	0	0	0	0	1
T7	0	0	0	0	0	0	0

Parent 1

	T1	T2	T3	T4	T5	T6	T7
T1	0	1	1	1	1	1	1
T2	0	0	1	0	0	0	1
T3	0	0	0	0	0	0	1
T4	0	1	1	0	0	0	1
T5	0	1	1	1	0	1	1
T6	0	1	1	1	0	0	1
T7	0	0	0	0	0	0	0

Parent 2

	T1	T2	T3	T4	T5	T6	T7
T1	0	2	2	2	2	2	2
T2	0	0	2	1	1	1	2
T3	0	0	0	1	1	1	2
T4	0	1	1	0	1	1	2
T5	0	1	1	1	0	2	2
T6	0	1	1	1	0	0	2
T7	0	0	0	0	0	0	0

Intersection Matrix

Fig. 3. Resulting Intersection matrix of two parents

D. Making the Predecessor Vector

Based on the previous intersection matrix, if there is an intersection, i.e., 2, in the columns, it is tallied in the predecessor vector table. For example, the number of 2's in column T_3 is 2, representing two predecessors of T_3.

TABLE 3.
PREDECESSOR VECTOR OF TEST CASES

	T_1	T_2	T_3	T_4	T_5	T_6	T_7
No. of Predecessors	0	1	2	1	1	2	6

E. Making Order Graph

The order graph is generated from the predecessor vector table. The order graph contains two columns, Immediate Predecessor and Depth. The Immediate Predecessor column contains the immediate predecessor of T_n while the Depth column contains the order in the sequence.

TABLE 4.
ORDER GRAPH OF TEST CASES

	Immediate Predecessor	Depth
T_1	None	1
T_2	T_1	2
T_3	T_2	3
T_4	T_1	2
T_5	T_1	2
T_6	T_5	3
T_7	T_3	4

F. Finding Maximum Partial Order Graph

Figure 4 depicts the graphical representation of the maximum partial order graph. The maximum partial order graph is generated by first finding a test case, T_n, with the fewest predecessors. Then the T_n is attached to the predecessor with the most ordered predecessors. When all the test cases have been added, the longest path in the graph is the maximum partial order; in this case, the longest path is T_1-T_2-T_3-T_7.

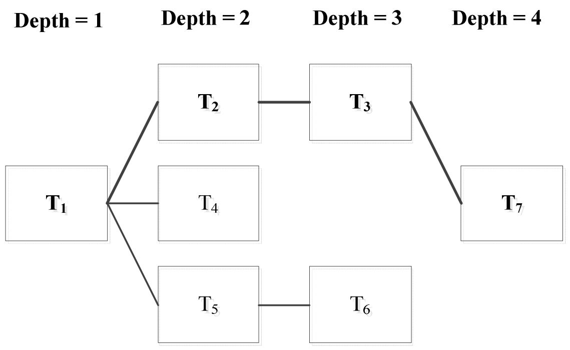


Fig. 4. The maximum partial order graph of test cases

G. Making the Predecessor Vector

After finding the Maximum Partial Order graph, the remaining test cases are arbitrarily inserted in the order graph. To insert a test case in the graph, two cases are considered: (1) minimization of both test case severity and inter-case dependency score, $d(T_i, T_j)$ and (2) if there is a tie in terms of the combined score, the precedence is decided based on the test case severity.

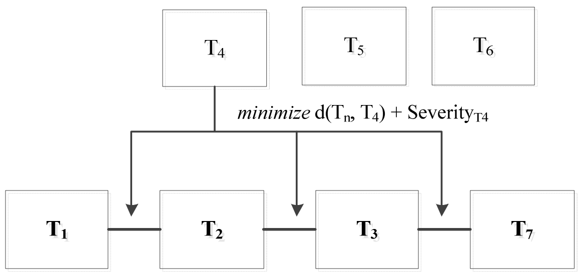


Fig. 5. The arbitrary insertion of test cases

V. Experiment Setup

In this research we have considered five data sets with various test suite sizes: 18, 71, 101, 255, and 380 test cases. The data sets are henceforth referred to as p18, p71, p101, p255 and p380. The values for inter-case dependencies and test case severities are randomly generated; for the larger test suites, p255 and p380, the test case severities are evenly distributed among four quartiles; that is, 25% of the test suite consist of severities in the first quartile (75-100), another 25% in the second quartile (50-75), and so on. Each test suite has been executed five times for each meta-heuristics.

Both ACO MPO/AI and GA MPO/AI codes are programmed in Visual C++ and executed on a computer with CPU Duo P8400 and 2GB memory.

A. Limitations and Constraints

The research is conducted for the specification-based testing, i.e. black-box testing, only. The structural or white-box testing is not within the scope of this research. Additionally, the deadline for the completion of test suite is not considered for this research.

B. Parameter Settings

Both Tables 5 and 6 contains the parameter settings for ACO MPO/AI and GA MPO/AI, respectively. ACO MPO/AI parameters include *M* (the number of ants), *ITERATION* (the number of iterations), α (favoring pheromone information/exploitation), β (favoring unexplored search space), and ρ (the pheromone evaporation rate). On the other hand, GA MPO/AI include *GEN* (number of generations), *POPSIZE* (population size), and *PARENTS* (number of parents). All the parameter settings are determined empirically. In the future, more systematical approaches can be taken to obtain the precise values for parameters.

TABLE 5.
ACO MPO/AI PARAMETER SETTINGS

M	*ITERATION*	α	β	ρ
5	100	1	1	0.6

TABLE 6.
GA MPO/AI PARAMETER SETTINGS

GEN	*POPSIZE*	*PARENTS*
100	100	2

VI. Results and Discussions

The results obtained after executing the meta-heuristics against the test suites are displayed in Table 7. The second column, Precedence Constraints, indicates the number of precedence constraint relationships that must not be violated during the execution. The fifth column, Best Found Score, contains the best found scores for the combination of inter-case dependency and test case severity. Notice that the values are all negative—it is related to converting the inter-case dependency and severity scores for minimization. The last two columns indicate the deviations of ACO and GA from the best found scores, respectively.

As shown in Table 5, the performance of ACO MPO/AI and GA MPO/AI are virtually similar in terms of the combined inter-case dependency and severity scores, with the exception of test suite p255, where the ACO MPO/AI performed slightly worse. The results of Wilcoxon signed rank test performed on the runs of p255 for both ACO and GA indicate that ACO and GA are not significantly different from each other with the *p*-value of 0.042. As far as the execution time is concerned, both meta-heuristics complete the execution of test case prioritization within one minute with GA MPO/AI taking a few seconds less than ACO MPO/AI.

TABLE 7.
PERFORMANCE OF ACO MPO/AI AND GA MPO/AI ON THE COMBINED SEVERITY AND INTER-CASE DEPENDENCY SCORES

Test Suite Name	Preced. Relation.	ACO (sec)	GA (sec)	Best Found Score	ACO Score Dev.	GA Score Dev.
p18	15	0.03	0.015	-1177	0%	0%
p71	17	0.24	0.088	-9931	0%	0%
p101	33	0.487	0.166	-14708	0%	0%
p255	30181	2.01	0.692	-36870	-0.14%	0%
p380	63583	5.49	0.926	-56098	0%	0%

Test case severity is also one of the performance indicators; hence, we expect test cases with higher severities to be executed earlier than those with lower severities. Even though we have to take inter-case dependency— particular precedence constraints— into account, we can still look for the overall trend of the severity distribution. The Mann-Kendall trend analysis shows all the prioritization of test suites follow the downward trend with the *p*-value of 0.000. Figures 6-10 show the downward trends and slopes of the test case prioritization for each test suite performed by the best meta-heuristics between GA and ACO.

Table 8 shows slopes of test case prioritization of both approach, ACO and GA MPO/AI. In addition, by comparing the slopes of test case prioritization between them no significant differences is found. The Wilcoxon signed rank test performed on the largest deviation between ACO and GA fails to reject the null hypothesis of ACO and GA are not statistically different from each other.

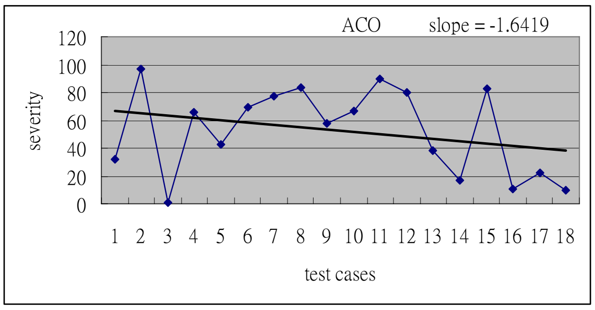


Fig. 6. Trend analysis of test case severity for 18 test cases

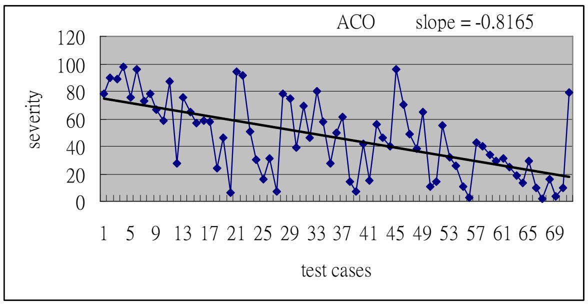


Fig. 7. Trend analysis of test case severity for 71 test cases

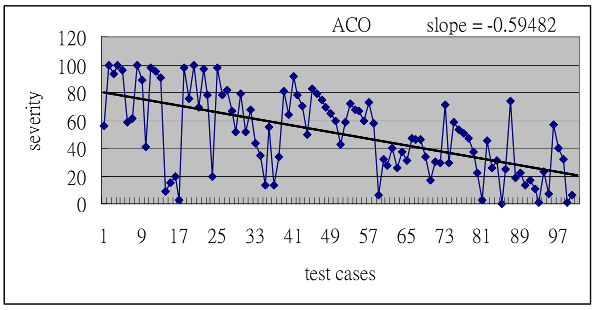


Fig. 8. Trend analysis of test case severity for 101 test cases

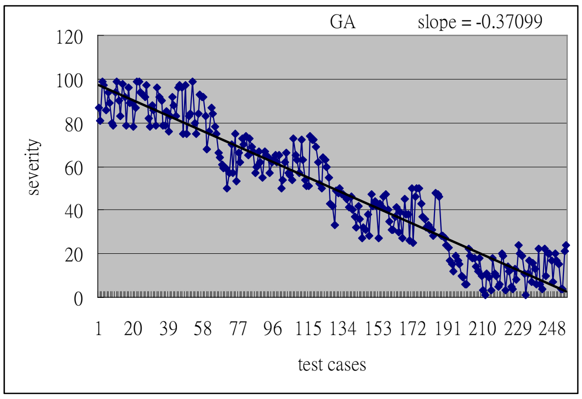


Fig. 9. Trend analysis of test case severity for 255 test cases

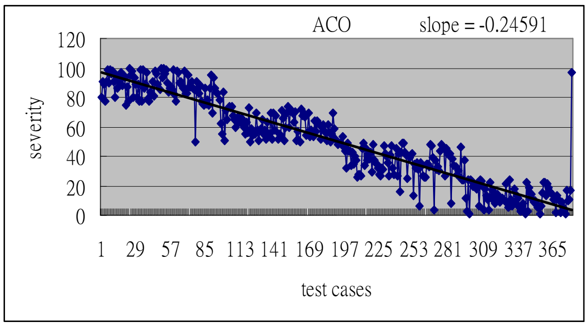


Fig. 10 Trend analysis of test case severity for 380 test cases

TABLE 8.
PERFORMANCE OF ACO MPO/AI AND GA MPO/AI ON THE TREND ANALYSIS

Test Suite Name	Best Found Slope	ACO Slope Deviation	GA Slope Deviation
p18	-1.6419	0%	-0.50%
p71	-0.8165	0%	-6.43%
p101	-0.59482	0%	-0.08%
p255	-0.37099	-0.04%	0%
p380	-0.24591	0%	-0.14%

VII. Conclusion

In the past decade, the research topic, test case prioritization, has been widely discussed and researched. By prioritizing the test cases, it may effectively improve the requirement coverage and software quality. At current time, many researchers focus on the structural testing. Even though structural testing has been applied to the early test stages of unit and integration testing, the later test stages of specification-based testing shall not be ignored. If the software is delivered to the customers without going through the specification-based testing—which may uncover errors not detected in the structural testing—defects uncovered by customers may deal a severe blow to customer's confidence with the product and the company's reputation. However, few researches are conducted on the topic of specification-based testing. Thus, the research considers the severity and inter-case dependency for evaluating test case prioritization under the specification-based testing environment. Based on the research result and discussion, the research fills the void of TCP with inter-case dependency under the specification-based environment, which has not yet been actively researched.

In the research, we have considered two meta-heuristics, ACO and GA incorporated with the MPO/AI operator, which have shown to be viable approaches to prioritize test cases in the specification-based environment.

Because of the resource and time constraints, we recommend the following for future work:

(1) Applying the proposed method on more sophisticated and complex software

(2) The research measures the effectiveness of test case prioritization on time. In the future, other criteria such as cost may be considered.

(3) Other multi-objective methods may be considered and incorporated with existing methods.

Acknowledgment

The authors would like to thank the anonymous referees for their times and efforts to improve the quality of this paper. This work was supported in part by a grant from National Science Council in Taiwan, NSC 100-2221-E-033-030.

References

[1] M. Newman. (2002, December 13th). *Software Errors Cost U.S. Economy $59.5 Billion Annually*. Available: http://www.abeacha.com/NIST_press_release_bugs_cost.htm

[2] M. Xie, Q. P. Hu, Y. P. Wu, and S. H. Ng, "A study of the Modeling and Analysis of Software Fault-detection and Fault-correction Processes," *Quality and Reliability Engineering International,* pp. 459-470, 2007.

[3] S. Wang, Y. Wu, M. Lu, and H. Li, "Discrete Nonhomogeneous Poisson Process Software Reliability Growth models based on test coverage," *Quality and Reliability Engineering International,* pp. 103-112, 2013.

[4] G. J. Myers, *The art of software testing*. New York: Wiley, 1979.

[5] M. I. Norton, D. Mochon, and D. Ariely, "The IKEA effect: When labor leads to love," *Journal of Consumer Psychology,* vol. 22, pp. 453-460, Jul 2012.

[6] O. F. Shmueli, Lior; and Pliskin, Nava, "OVER-REQUIREMENT IN SOFTWARE DEVELOPMENT: AN EMPIRICAL INVESTIGATION OF THE 'IKEA' EFFECT," in *ECIS 2012 Proceedings*, 2012, p. Paper 85.

[7] (n.d., December 21th, 2011). *Test Case*. Available: http://www.businessdictionary.com/definition/test-case.html

[8] C. Kaner, "What Is a Good Test Case?," *STAR East,* p. 16, May 2003 2003.

[9] IEEE, "IEEE Standard for Software Test Documentation," ed. New York, NY: The Institute of Electrical and Electronics Engineers, 2008.

[10] I. Sommerville, *Software engineering*, 7th ed. Boston: Pearson/Addison-Wesley, 2004.

[11] C. Kaner, J. Bach, and B. Pettichord, *Lessons learned in software testing : a context-driven approach*. New York: Wiley, 2002.

[12] G. Y. Chen and J. Rogers, "Arranging software test cases through an optimization method," in *PICMET '10 - Portland International Center for Management of Engineering and Technology, Proceedings - Technology Management for Global Economic Growth* Phuket, Thailand, 2010, pp. 1596-1600.

[13] G. Rothermel, R. Untch, and M. Harrold, "Prioritizing test cases for regression testing," *IEEE TRANSACTIONS ON SOFTWARE ENGINEERING,* vol. 27, pp. 929-948, 2001.

[14] B. Qu, C. H. Nie, B. W. Xu, and X. F. Zhang, "Test case prioritization for black box testing," in *IEEE 31st Annual International Computer Software and Applications Conference (COMPSAC 2007)*, 2007.

[15] Z. Li, M. Harman, and R. M. Hierons, "Search algorithms for regression test case prioritization," *IEEE Transactions on Software Engineering,* vol. 33, pp. 225-237, Apr 2007.

[16] K. H. Hla, Y. S. Choi, and J. S. Park, "Applying particle swarm optimization to prioritizing test cases for embedded real time software retesting," in *IEEE 8th International Conference on Computer and Information Technology Workshops*, 2008.

[17] C. Catal and D. Mishra, "Test case prioritization: a systematic mapping study," *Software Quality Journal,* vol. 21, pp. 445-478, Sep 2013.

[18] B. Zhou, H. Okamura, and T. Dohi, "Application of Markov Chain Monte Carlo Random Testing to Test Case Prioritization in Regression Testing," *IEICE Transactions on Information and Systems,* vol. E95D, pp. 2219-2226, Sep 2012.

[19] K. Rajarathinam and S. Natarajan, "Test suite prioritisation using trace events technique," *IET Software,* vol. 7, pp. 85-92, Apr 2013.

[20] C.-T. Lin, C.-D. Chen, C.-S. Tsai, and G. M. Kapfhammer, "History-based test case prioritization with software version awareness," in *18th International Conference on Engineering of Complex Computer Systems, ICECCS 2013, July 17, 2013 - July 19, 2013*, Singapore, Singapore, 2013, pp. 171-172.

[21] W. Sun, Z. Gao, W. Yang, C. Fang, and Z. Chen, "Multi-objective test case prioritization for GUI applications," in *28th Annual ACM Symposium on Applied Computing, SAC 2013, March 18, 2013 - March 22, 2013*, Coimbra, Portugal, 2013, pp. 1074-1079.

[22] M. Last, S. Eyal, and A. Kandel, "Effective black-box testing with genetic algorithms," in *Hardware and Software Verification and Testing*. vol. 3875, S. Ur, E. Bin, and Y. Wolfsthal, Eds., ed Berlin: Springer-Verlag Berlin, 2006, pp. 134-148.

[23] Y.-S. Lee, "An Approach to Generating Test Cases for Non-functional Requirements," Master's degree, Department of Electrical Engineering, National Chung Cheng University, Chia Yi, Taiwan, 2003.

[24] authors. (2000, 12/7). *Bugs and Fixes*. Available: http://www.sqatester.com/bugsfixes/defectparametrs.htm

[25] T.-H. Yu, "Test Case Ordering and Selection for Blackbox Testing," Master's Thesis, Department of Industrial and Systems Engineering, Chung Yuan Christian University, Chung Yuan Christian University, 2012.

[26] IEEE, "IEEE 829-1998 Standard for Software Test Documentation," in *Test Case Specification*, ed. New York: IEEE Service Center, 1998.

[27] C. Kaner, J. L. Falk, and H. Q. Nguyen, *Testing computer software*, 2nd ed. New York: Wiley, 1999.

[28] A. K. Onoma, W. T. Tsai, M. Poonawala, and H. Suganuma, "Regression testing in an industrial environment," *Communications of the ACM,* vol. 41, pp. 81-86, 1998.

[29] S. Chen and S. F. Smith, "Commonality and genetic algorithms," Carnegie Mellon University, Pittsburgh, PA, Technical Report CMU-RI-TR-96-271996.

[30] L. F. Escudero, M. Guignard, and K. Malik, "A Lagrangean relax-and-cut approach for the sequential ordering problem with precedence relationships," *Annals of Operations Research,* vol. 50, pp. 1219-237, 1994.

[31] R. Montemanni, D. H. Smith, A. E. Rizzoli, and L. M. Gambardella, "Sequential Ordering Problems for Crane Scheduling in Port Terminals," *International Journal of Simulation and Process Modelling,* vol. 5, pp. 348–361, 2009.

[32] T. M.T. and P. W.R., "Precedence constrained routing and helicopter scheduling: heuristic design," *Interfaces,* vol. 22, pp. 100-111, 1992.

[33] S. Spieckermann, K. Gutenschwager, and S. Vos, "A sequential ordering problem in automotive paint shops," *International Journal of Production Research,* vol. 42, pp. 1865–1878, 2004.

[34] A. P. Engelbrecht, *Computational intelligence : an introduction*, 2nd ed. Chichester, England ; Hoboken, NJ: John Wiley & Sons, 2007.

Gary Yu-Hsin Chen received his PhD in Industrial and Systems Manufacturing Engineering from the University of Texas at Arlington, Arlington, Texas, USA. He had worked as a senior software development/test engineer in the telecommunications and industrial automation industries in USA. He is currently an assistant professor in the Department of Industrial and Systems Engineering at Chung Yuan Christian University, Taiwan. His research interests are in the fields of software testing/quality assurance, meta-heuristics, facility layout optimization and telecommunications applications. He is a permanent member of Chinese Institute of Industrial Engineers.

Pei-Qi Wang is currently working as a software test engineer for a multi-national hi-tech company in Taiwan. She received her MS degree in Industrial and Systems Engineering from Chung Yuan Christian University, Taiwan, in 2011.