IEEE 8th International Conference on Computer and Information Technology Workshops

Applying Particle Swarm Optimization to Prioritizing Test Cases for Embedded Real Time Software Retesting

Khin Haymar Saw Hla,
Department of Computer Engineering,
Korea Aerospace University, Korea
haymar@kau.ac.kr

YoungSik Choi
Department of Computer Engineering,
Korea Aerospace University, Korea
choimail@kau.ac.kr

Jong Sou Park
Department of Computer Engineering,
Korea Aerospace University, Korea
jspark@kau.ac.kr

Abstract

In recent years, complex embedded systems are used in every device that is infiltrating our daily lives. Since most of the embedded systems are multi-tasking real time systems, the task interleaving issues, dead lines and other factors needs software units retesting to follow the subsequence changes. Regression testing is used for the software maintenance that revalidates the old functionality of the software unit. Testing is one of the most complex and time-consuming activities, in which running of all combination of test cases in test suite may require a large amount of efforts. Test case prioritization techniques can take advantage that orders test cases, which attempts to increase effectiveness in regression testing. This paper proposes to use particle swarm optimization (PSO) algorithm to prioritize the test cases automatically based on the modified software units. Regarding to the recent investigations, PSO is a multi-object optimization technique that can find out the best positions of the objects. The goal is to prioritize the test cases to the new best order, based on modified software components, so that test cases, which have new higher priority, can be selected in the regression testing process. The empirical results show that by using the PSO algorithm, the test cases can be prioritized in the test suites with their new best positions effectively and efficiently.

Keywords: embedded system, real time software system, test case coverage, evolutionary structural testing, regression testing and particle swarm optimization

1. Introduction

Recent advances in software technology make it desirable to use complex embedded systems in the industrial products. Most of the embedded systems use multi-tasking real time software systems, in which tasks are competing for resources due to concurrent execution, which causes race conditions and interleaving of task statements. Therefore, task interleaving issues, dead lines and other factors needs software units retesting to follow the subsequence changes. Whenever software is altered there is a need to verify the alternations and to ensure that the changes have not corrupted other functionality of the software. Regression testing is used as a process of retesting test cases to follow system changes. Software engineers often save the test suites they develop for their software so that they can reuse those test suites later as the software evolves. For efficient regression testing, the goal is to choose test cases from test suite in order to establish the correctness of the modification. Such test suite reuse, in the form of regression testing, is as much as one-half of the cost of software maintenance [1, 6, 8]. Running all of the test cases in a test suite, however, can require a large amount of effort. For this reason, researchers have considered various techniques for reducing the cost of regression testing. Test case prioritization techniques attempt to increase their effectiveness in regression testing. These techniques let testers order their test cases so that those test cases with the highest priority are executed earlier in the regression testing process. This paper proposes to use the particle swarm optimization technique to prioritize the test cases to the best new positions based on software unit modifications. PSO is a population based stochastic optimization technique developed by Kennedy and Eberhart in 1995 [2]. It could be implemented and applied easily to solve various functional optimization problems. As an algorithm, the main strength of PSO is its fast convergence, which compares favorably with many global optimization algorithms [6, 12]. The goal is to prioritize the test cases to the new best positions based on modified software units to spend as little resource on retesting as

978-0-7695-3242-4/08 $25.00 © 2008 IEEE
DOI 10.1109/CIT.2008.Workshops.104

possible. The empirical results show that by using the PSO algorithm, the best positions of the test cases can be placed for the real time software retesting.

The rest of the paper is organized as follows: Section 2 introduces the overview of test case prioritization techniques. Section 3 describes the particle swarm optimization techniques and the detailed algorithm, which is used for the prioritization of high coverage test cases for retesting. Section 4 explains the empirical results of the proposed algorithm. Section 5 discusses the previous works on software retesting and finally, section 6 outlines the conclusions and future work.

2. Test Case Prioritization

In the case of regression testing, test case prioritization techniques can use information gathered in previous runs of existing test cases to help prioritize the test cases for subsequent runs. In general, test case prioritization, given program *P* and test suite *T*, test cases in *T* are prioritized with the intent of finding an ordering of test cases that will be useful over a succession of subsequent modified version of *P*. In version specific test case prioritization, given program *P* and test suite *T*, prioritize the test cases in *T* with the intent of finding an ordering that will be useful on a specific version *P′* of *P*. For given any prioritization goal, various prioritization techniques are applied to test suite with the aim of meeting that goal. In this study, we focus on the prioritization of test suite based on the test case coverage.

2.1. Test Case Coverage

Coverage is an important aspect of the software testing, which is the way to show how well tests have covered a special aspect of the system [11]. Test case encompasses a set of test inputs, execution conditions, and expected results developed for a particular objective, such as to exercise a particular program path or to verify compliance with a specific requirement.

2.1.1. Statement coverage prioritization. Statement coverage is coverage metric that fines out for any test case, which statements in the program were covered by that test case.

Figure 1 depicts a procedure, which has the modified statements and the statement coverage of that procedure achieved by three test cases from the test suite. Applied in this case, the best test case can be achieved in terms of the total number of statements covered, by counting the number of statements covered by each test case within the software unit such as (3, 2, 1).

```
Procedure P
1. statement1
2. while (condition1) do
3.     if (condition2) then
4.         statement2
       else
5.         statement3
       end if
6.         statement4
   end while
           7. statement5
```

	Statement Coverage		
	Test case 1	Test case 2	Test case3
1	x	x	x
2	x	x	x
3		x	x
4		x	
5			x
6			x
7		x	x

Figure 1. Procedure *P* and statement coverage of *P* achieved by three test cases

2.1.2 Branch coverage prioritization. Branch coverage uses test coverage measured in terms of program branches rather than statements. It is coverage of each possible overall outcome of the possibly compound conditions in a predicate [10].

	Branch Coverage		
	Test Case 1	Test Case 2	Test Case 3
Entry	x	x	x
2. True		x	x
2. False	x		x
3. True		x	
3. False			x

Figure 2. Branch coverage of *P* achieved by three test cases

Figure 2 shows that in branch coverage, each 'if' or 'while' statement must be covered such that at least 'true' and 'false' must be covered once. It measures the number of branches of the control flow graph of the function under test that are traversed.

2.1.3. Function coverage prioritization. To accommodate functions that contain no branches, each function entry is treated as a branch, and regards that branch as covered by each test case that causes the function to be invoked.

2.1.4. Code coverage prioritization. Code coverage measures whether those statements in a software unit have been executed through a test run and those, which have not.

3. Particle Swarm Optimization Technique in Test Case Prioritization

This study applies the particle swarm optimization technique to prioritize the test cases to the new best positions based on the fitness of the test case coverage.

3.1. Overview of Particle Swarm Optimization

Particle swarm optimization (PSO) is a promising new optimization technique, which models the set of potential problem solutions as a swarm of particles moving about in a virtual search space. PSO model starts with the random initialization of a population of particles in the search space. It converges to the global best solution by simply adjusting the trajectory of each individual particle towards its own best location and towards the best particle of the entire swarm at each generation.

The position vector and the rate of position change vector, velocity of the particle in the d-dimensional search space is represented as $X_i = (x_{i1},x_{i2},x_{i3},\ldots,x_{ik})$ and $V_i = (v_{i1},v_{i2},v_{i3},\ldots,v_{ik})$ respectively. According to a defined fitness function F for the required scenario, by assuming the best position of each particle, which corresponds to the best fitness value obtained by that particle at a selected time is $F_i = (f_{i1},f_{i2},f_{i3},\ldots,f_{ik})$, and the fittest particle found at the selected time is $F_g = (f_{g1},f_{g2},\ldots,f_{gk})$. When a dimension d of the i^{th} particle's position is updated to a new value in the k^{th} iteration, the velocity v of the particle decides the new value of x for each dimension d. The new velocities and the positions of the particles for the next fitness evaluation can be evaluated with the following equations:

$$v_{ik+1} = c_0 . v_{ik} + c_1 . rand . (f_{ik}-x_{ik}) + c_2 . rand . (f_{gk}-x_{ik}) \quad (1)$$

Where c_0, c_1 and c_2 are weighting factor, constants that say how much the particle is directed towards good position, the "*rand*"s are uniformly distributed random numbers in the range [0,1]. When the velocity is updated in the k^{th} iteration, the velocity v of the dimension d of the particle decides where the particle moves next.

$$x_{ik+1}=x_{ik}+v_{ik+1} \quad (2)$$

The termination criterion can be either a specific fitness value, the achievement of a maximum number of iterations or the general convergence of the swarm itself.

3.2. Applied PSO in Test Case Prioritization

To apply the PSO algorithm successfully to the test case prioritization problem, we need to map to the problem solution into the particle space. The problem statement is as follows:

Given that let $U=\{u_1,u_2, \ldots..u_n\}$ be the set of software units of the real time application, $S=\{s_1,s_2,\ldots,s_n\}$ be the set of modified statements of the program unit u_i, $C=\{c_1,c_2,\ldots..,c_n\}$ be the set of compound conditions, let $P=\{p_1,p_2,\ldots..,p_n\}$ be the set of procedures in the software unit and let m be the limited execution time for the given software unit. Let $T=\{t_1,t_2,\ldots.,t_n\}$ be a test suite that contains a set of test cases, let PT be the set of permutations of T and F is a function from T to find out the most feasible test cases to retest. Problem statement is to find $T' \in PT$ such that $\forall T'$, $T'' \in PT$, $T'' \neq T'$ *and* $F(T') \geq F(T'')$. In this definition, PT represents the set of the most feasible test cases of T, and F is a function that applied to any such test cases, yields an optimize value for that test suite. Let f_s be the fitness value of the total statement coverage within limited time m, where $(0 \leq f_s \leq 1)$ and Fs be the utility function that finds out the maximum statement coverage. $\forall$ Test cases $t_i \in T$, where $i \in (1,n)$, Fs finds out the maximum statement coverage value for the statements $s_i \in S$, which are modified parts of the program unit $u'_i \in U$. f_s is represented as:

$$f_s = \arg\max Fs(u'_i,T,S,m) \quad (3)$$

Let f_b be the fitness value of total branch coverage, where $(0 \leq f_b \leq 1)$ and Fb be the utility function that finds out the maximum branch coverage. $\forall$ Test cases $t_i \in T$ where i $\in (1,n)$, Fb finds out the maximum branch coverage value for the compound condition $c_i \in C$ of the software unit $u'_i \in U$, f_b is represented as:

$$f_b = \arg\max Fb(u'_i,T,C,m) \quad (4)$$

Let f_p be the fitness value of functional coverage, where $(0 \leq f_p \leq 1)$ and Fp be the utility function that finds out the maximum functional coverage.$\forall$ Test cases $t_i \in T$ where i $\in (1,n)$, Fp finds out the maximum functional coverage value of the function $p_i \in P$ of the software unit $u'_i \in U$. f_p is represented as:

$$f_p = \arg\max Fp(u'_i,T,P,m) \quad (5)$$

Let f_u be the fitness value of total code coverage, where $(0 \leq f_u \leq 1)$ and Fu be the utility function that finds out the maximum code coverage. $\forall$ Test cases $t_i \in T$ where i $\in (1,n)$ of the source code $u'_i \in$, Fu find out the maximum code coverage value for the specified test case given as:

$$f_u = \arg\max Fu(u'_i,T,U,m) \quad (6)$$

Let test suite T be the search space of the PSO and f be the fitness value of the test cases, we will find the new best fitness value f of the test cases $t_i \in T$ where $i \in (1,n)$ for modified software units $u'_i \in U$ based on the fitness function F. Let E be the expected value of the fitness values, F is represented as:

$$F(f) = E(f_s \wedge f_b \wedge f_p \wedge f_u) \quad (7)$$

Let existing priorities of the test cases represent as velocities and fitness of the test cases represent as

positions, these values are used as the initialization parameters in this system. Based on these parameters, new priorities of the test cases will be carried out by using PSO algorithm. The pseudo code for the test case prioritization based on PSO method is illustrated in the following algorithm.

Algorithm: Test Case Prioritization based on PSO

Begin
1. For i=1 to number of test cases Do
2. Initialize the velocity and positions of the test case t_i with existing priority and fitness value
3. End
4. Do
5. For i=1 to number of test cases Do
6. For j=1 to number of parameters in test case t_i Do
7. Calculate coverage values of parameter p_j according to the equation (3)(4)(5)and(6)
8. Calculate fitness value f according to the equation (7)
9. End for
10. If $f \geq$ best fitness of the history of test cases f_{best} then
11. $f_{best} = f$
12. End for
13. For i=1 to number of test cases Do
14. Calculate velocity of test case t_i according equation (1)
15. Update position of test case t_i according equation (2)
16. End for
17. until (Maximum Test suites exhaust)

End

Figure 3. Test case prioritization algorithms

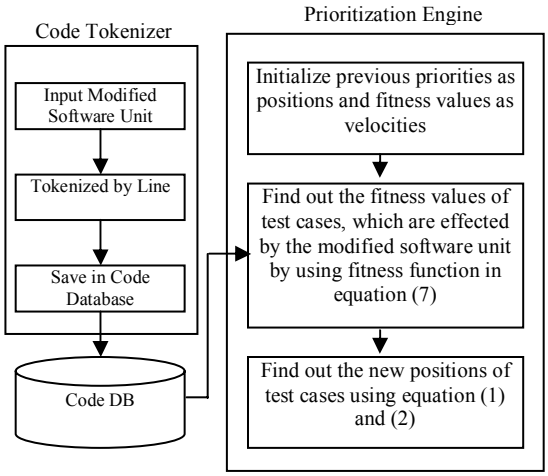

Figure 4. Detailed workflow

In figure 4, the detailed workflow of the scheme is shown. It consists of two major components such as code tokenizer and prioritization engine. The code tokenizer uses the modified software units as an input to the system. It tokenizes the source codes line by line and saves in the code database. Prioritization engine is used to find out the new positions of the test cases. The prioritization engine first gets the priorities of the test cases and their fitness values of the previous run. It only needs to take into account for the test cases, which are affected by modified version of software units. Coverage values are found out for the test cases that are affected by the modified software units by using the utility functions given in (3), (4), (5) and (6). Then, new fitness values are calculated based on the fitness function given in (7). Finally, the new positions of the test cases are carried out according to the rate of position change vector (v) (see equation (1) and (2)).

4. Experiment

We illustrate effectiveness of the PSO algorithm by running 20 test cases from the JUnit test suite. JUnit testing involves Java classes that contain one or more test methods and group into test suites. The composite behavior of JUnit tests allows assembling collection of test cases and automatically performing regression test to the entire test suite. Table 1 and 2 describe the parameters used for the realization of PSO based test case prioritization algorithm.

Table 1. Coverage and execution time of test cases

No.	Test Cases	Coverage Unit	Execution Time
t1	TestOfLogin	2	1.25
t2	CalMidpointBoundry	1	0.25
t3	TestForMinMax	3	0.75
t4	SearchArraySizeone	3	0.5
t5	FilterTest	3	0.5
t6	RegexFilterTest	2	0.625
t7	TestBoundry	3	0.125
t8	TestIndexWith3Nodes	2	0.625
t9	TestLoginAction	3	1
t10	TestIndexWith4Nodes	2	1
t11	TestAssertIdentical	4	1.125
t12	TestSearchPoint	4	0.875
t13	AssertUnwantedPattern	2	0.875
t14	SearchEmptyArray	3	0.375
t15	TestGetValue	1	0.25
t16	TestStoreValue	2	0.25
t17	TestAction	2	1.25
t18	TestSearchArray	4	0.125
t19	TestResponse	3	0.625
t20	TestClockTellsTime	4	0.875

Table 2. PSO parameters

Parameter	Value
No. of Particles	20
Acceleration Coefficient c	0,1
Iteration	100

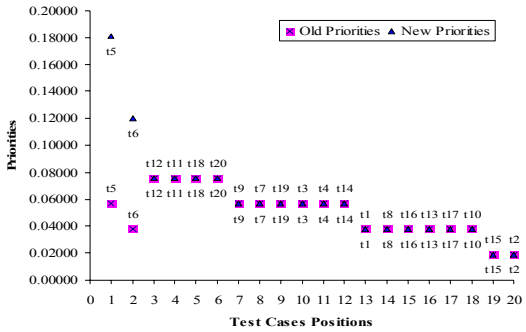

Figure 5. Old and new positions of the test cases

Suppose that test cases are prioritized by using the priorities and fitness values based on the old version of the software units, new fitness positions are searched in the search space based on the modified parts of software units. Figure 5 illustrates the position changes of the test cases, in which old positions are based on the old version of software unit and new positions are based on the modified software units. In the above illustration, the modified software unit affects test cases t5 and t6. The new positions are found out based on the new fitness value of the test cases t5, t6 and previous priorities of the all test cases. Based on the new fitness values of the test cases, position change vector, velocity (v) is found out. Upon changes of the software unit, current positions of the test cases are changed to the new positions based on the rate of velocity. The predicted positions are validated against false positive rate. False positive rate is calculated as follows.

$$\frac{\#\text{ of false positions}}{\#\text{ of runs}} \qquad (8)$$

$$\frac{164}{20\times100} = 8.2\%$$

The experiment is performed over 100 runs. Due to the randomized weighting factors, the false positive rate is 8.2% under 100 runs that shows results are promising.

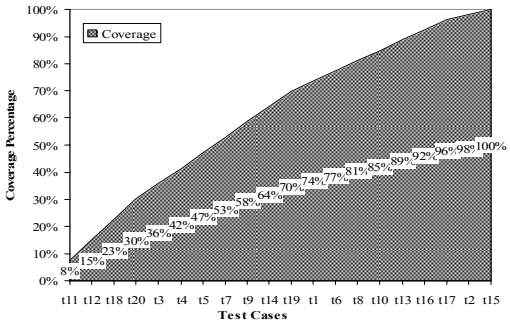

Figure 6. Prioritized test cases over coverage criteria

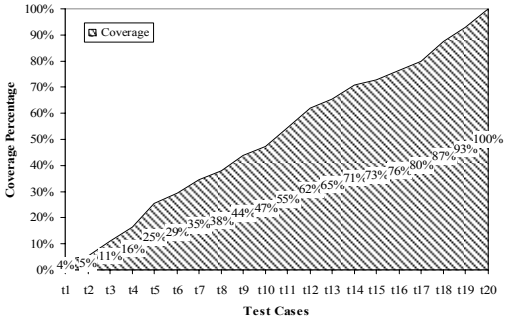

Figure 7. Randomize test cases over coverage criteria

Figure 6 and 7 show the effective used of the test case prioritization. By applying test case prioritization technique based on PSO, the 64% percent of coverage can be achieved after running 10 test cases. If test cases are placed randomly, only 47% percent of the test case coverage can be achieved after running 10 test cases.

The effectiveness of PSO based algorithm is measured by comparing with the greedy search algorithm. Run time complexity of the PSO based algorithm is considerably reduced comparing with greedy algorithm, which is $O(mn^2)$[4]. Consider a software unit containing m statements and test suite containing n test cases. By applying PSO based algorithm, fitness function is considered over the statement, branch and functional coverage, which can be accomplished in $O(mn)$ times for the whole software unit. Suppose that k number of test cases are affected by the modified software unit, which is p percentage of the old version of software unit, prioritization cost will be $O((m*p)k)$, which is less than $O(mn)$. And readjustment must be performed $O(n)$ times. Therefore total prioritization cost will be $O((m*p)kn) < O(mn^2)$.

5. Related Work

Real time systems are most often implemented as a set of task, running on top of a real-time operating system. When developing complex embedded real time applications, the software is often built in the form of subsystems or components, which are later integrated and assembled to a full system. Throughout all stages of development and assembly, software testing is performed. Throughout the entire development cycle and at every level, testing will reveal bugs. Most often, these bugs will be corrected via changes in the source code, leading to a necessity to retest the software. Regression testing is process of retesting, which is required during the software maintenance, where functionality is added or bugs are corrected. Regression testing activities has been estimated to account for as

much as one-half of the cost of software maintenance. Therefore, regression test selection and test suite minimization techniques [1, 7, 10] are used for reducing the cost of regression testing. Regression test selection and test suite minimization techniques however can have drawbacks since they discard test cases for retesting. Wong et al. [13] proposed test case prioritization techniques that take advantage to regression testing. Rothermel et al. [8, 9] analyzed the prioritization techniques empirically. Analysis results showed that each of prioritization techniques improved the rate of fault detection. Previous work on regression test case prioritization has focused on greedy algorithms. However, it is known that these algorithms may produce suboptimal results because they may construct results that denote only local minima within the search space [4]. By contrast, evolutionary search algorithms such as genetic algorithm aim to avoid such problems. This study focuses on PSO based evolutionary search algorithm that finds out the results based on global optima within the search space. PSO is easy to implement and needs few parameters than genetic algorithm [12]. Also it is based on a simple concept, therefore, the computation time is short and it requires little memory, which makes dealing with real time computation possible. In this work, we focus on the automatic test case prioritization for modified version of software units based on PSO algorithm. PSO based test case prioritization algorithm can reduce the run time complexity comparing with greedy algorithm. Furthermore, it only needs to take into account for the test cases, which are affected by modified version of software units to find out the new best order of test cases to retest. Therefore, this system can take advantage for software retesting of real time systems with minimum effort and time.

6. Conclusions and Future Work

This paper proposed to use PSO search algorithm in test case prioritization on embedded real time software retesting. The proposed algorithm can search the best new positions of the test cases based on the modified parts of the software units. It also finds out the test cases, which have the high coverage features. This study has shown that the PSO based test case prioritization algorithm can discover reasonable quality solution effectively and efficiently. As our future work, we intend to tackle the test case prioritization for embedded system applications by using PSO in a pragmatic way.

7. References

[1]D. Binkley, "Semantics Guided Regression Test Cost Reduction". IEEE Transactions on Software Engineering, 23(8):498-516, August 1997.

[2]J. Kennedy, and R. Eberhart, "Swarm Intelligence". Morgan Kaufmann, CA, 2001.

[3]H. K. N. Leung and L. White, "Insights Into Regression Testing", In Proceedings of the Conference on Software Maintenance, October 1989, pp 60-69.

[4]Z. Li, M. Harman, R.M. Hierons, "Search Algorithms for Regression Test Case Prioritization", IEEE Transactions on Software Engineering, Vol 33, No 4, 2007.

[5]H. Liu, S. Sun and A. Abraham. "Particle Swarm Approach to Scheduling Work-flow Applications in Distributed Data-intensive Computing Environments", In Proceedings of the Sixth International Conference on Intelligent Systems Design and Applications, (ISDA'06), IEEE Computer Society Press, 2006, pp 661–666.

[6]N. Nedjah, L. Mourelle. "Swarm Intelligent Systems, Studies in Computational Intelligence", Springer Verlag, 2006, pp 3–25.

[7]K. Onoma, W.T. Tsai, M. Poonawala, H. Suganuma, "Regression Testing in an Industrial Environment", Communications of the ACM, 41(5), May 1988, pp 81-86.

[8]G. Rothermel, R. H. Untch, C. Chu, M. J. Harold, "Prioritizing Test Cases for Regression Testing", IEEE Transactions on Software Engineering", 2001.

[9] G. Rothermel, R. H. Untch, C. Chu, and M. J. Harrold, "Test case prioritization: An empirical study", In Proceedings of the International Conference on Software Maintenance, August 1999, pp 179-188.

[10]D. Sundmark, A. Petterson, H. Thane, "Regression Testing of Multi-tasking Real Time System a Problem Statement", Work in Progress Session of the 11th IEEE Real Time and Embedded Technology Applications Symposium, San Francisco, USA, 2005.

[11]D. Sundmark, A. Petterson, S. Eldh, M. Ekman and H. Thane, "Efficient System Level Testing of Embedded Real-Time Software", Work in Progress Session of the 17th Eurmicro Conference on Real-Time System, Spain, 2007, pp 53-56.

[12]A. Windisch, S. Wappler and J. Wegener, "Applying Particle Swarm Optimization to Software Testing", In Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation, London, England, 2007, pp1121-1128.

[13] W. E. Wong, J.R. Horgan, S. London, H. Agrawal, "A Study of Effective Regression Testing in Practice", IEEE TENCON Digital Signal Processing Applications Proceedings, 1997.

Acknowledgements

This research was supported by the Advanced Broadcasting Media Technology Research Center (ABRC) in Korea Aerospace University, Korea, under the Gyeonggi Regional Research Center (GRRC) support program supervised by Gyeonggi Province.