Test Case Prioritization Using Metaheuristic Search Techniques

***Mukesh Mann[1], Pradeep Tomar[2], Om Prakash Sangwan[3]**

1 Department of Computer Science and Engineering, School of Engineering and Technology, BML Munjal University, Gurgaon, Haryana, India
2 Department of Computer Science & Engineering, Schools of Information & Communication Technology, Gautam Buddha University, India
3 Department of Computer Science & Engineering, Guru Jambheshwar University of Science and Technology, India

ID # (2858)
Received: 27/12/2016
In-revised: 06/04/2017
Correspondent Author:
Mukesh Mann
E-mail: mukesh.gbu@gmail.com

KEYWORDS

Particle Swarm Optimization (PSO); Cuscuta Search Algorithm (CSA); Genetic Algorithm (GA); The Prioritized order by average faults found per minute algorithm (AF/M); Software Design Life Cycle (SDLC).

ABSTRACT

In this paper, Artificial Particle Swarm Optimization (PSO) inspired by real Swarm social–psychological tendency is used to solve time constraint prioritization problem-the techniques to prioritize the test cases that finds faults as early as possible, or maximize the rate of fault detection in the suite. The proposed technique is compared with three searches based metaheuristic approaches–(1) an ant-colony optimization approach, (2) Cuscuta search algorithm and (3) Hybrid Particle Swarm Optimization algorithm and two evolutionary metaheuristic- (1) Multi-Criteria Genetic algorithm technique which the fitness is APFD and (2) Multi-Criteria Genetic algorithm technique which the fitness is the proposed fitness multiplied by APFD and with five other non-search based prioritization techniques- (1) optimal, (2) random, (3) reverse, (4) untreated and (5) average faults found per minute algorithm based ordering. We investigate whether the proposed PSO metaheuristic outperforms existing prioritizing techniques in terms of APFD Score.

تقنية تحديد أولويات حالة الاختبار بإستعمال تقنيات الميتاهوريستيك

موكيش مان [1]، براديب تومار [2]، أوم براكاش سانغوان [3] .

1 قسم علوم الحاسوب والهندسة، كلية الهندسة والتكنولوجيا، جامعة BML منجال، جورجاون، هاريانا، الهند
2 قسم علوم الحاسوب والهندسة، مدارس تكنولوجيا المعلومات والاتصالات، جامعة غوتام بوذا، الهند
3 قسم علوم الحاسوب والهندسة، جامعة جورو جامبشوار للعلوم والتكنولوجيا، الهند

رقم المسودة: (2858)
تاريخ استلام المسودة: 2016/12/27
تاريخ المسودة المُعَدّلة: 2017/04/06
الباحث المُرَاسِل: موكيش مان
بريد الكتروني: mukesh.gbu@gmail.com

الكلمات الدالة

Particle Swarm Optimization); (CSA); Genetic Algorithm (GA); The Prioritized order by average faults found per minute algorithm (AF/M); Software Design Life Cycle (SDLC).

المُستلخص

في هذه الورقة، تستخدم تهيئة سرب الجسيمات الاصطناعي المستوحاة من النزعة الاجتماعية والنفسية للسرب الحقيقي لحل مشكلة تحديد أولويات القيد للوقت. تستعمل التقنيات لتحديد أولويات حالات الاختبار التي تجد أخطاء في أقرب وقت ممكن، أو تعظيم معدل الكشف عن خطأ في المجموعة. قورنت التقنية المقترحة مع ثلاث عمليات بحث تعتمد على مقاربات ميتاهوريستيك: (1) نهج التهيئة الأمثل لمستعمرة النمل، (2) خوارزمية البحث كوزكوتا و (3) خوارزمية تحسين سرب الجسيمات الهجين بالإضافة الى خوارزميتين متطورتين (1) متعددة المعايير الجينية (2) تقنية الخوارزمية الجينية المتعددة المعايير وفيهاتكون النمذجة ا لرياضية مضروبة في .. بالإضافة الى خمسة تقنيات ترتيب الأولويات أخرى غير القائمة على البحث: (1) الأمثل، (2) عشوائي، (3) عكس، (4) غير المعالجة و (5) معدل الأخطاء التي وجدت في خوارزمية الترتيب.في هذه الورقة نحقق في ما إذا كان الميتاميولوجي PSO المقترح يتفوق على تقنيات تحديد الأولويات الحالية من حيث نقاط AFPD.

Introduction

Life cycle models are used to develop a software system from scratch and to assure that the developed software works efficiently as per requirement, SDLC include various stages that vary from one SDLC model to other SDLC model, but the most common phase in all models is of testing which assures that the developed software is error free. Software testing tries to detect errors as early as possible because the cost of removing an error become high if detected in later stages of development. The test suites which are used to uncover errors in software undergo frequent reuse and updation as the software evolves with time. This results in large size test suites that may contain redundant test cases. The time and resources required to execute these ever increasing test suite is a critical factor in deciding the cost of a software system. Thus it becomes necessary to minimize the test suite by eliminating the redundant test cases. i.e. test case minimization, to develop techniques to select test cases which are important to identify the parts of the system under test(suite) that have been modified, and the techniques to prioritize the test cases that finds faults as early as possible, or maximize the rate of fault detection in the suite.

A Software system undergoes different changes while fixing a detected/known fault or adding or deleting a new requirement(s). These changes must be validated i.e. software must be retested in order to accomplish the changes. Thus the validation aims: (1) that the new added/ deleted requirement have been implemented correctly; (2) to ensure that the reflection of the new added/ deleted requirement does not affect the previous functionalities (3) checking of those parts of the software that have not been checked before.

The process of retesting the software system to ensure that the modified program still works correctly with all previous and new test cases used to test original program and new modified program respectively so that the modified software still meets its requirements is called regression testing.

A timeline example in figure1 describes the life of a software system. Although the execution time of regression testing is a considerable fraction of the timeline but unfortunately the regression testing cannot always be completed because of the frequent changes and updates to a system. When a program is modified/updated, it is almost impossible to guarantee that the changes work correctly and that the unmodified modules of the program have not been affected by the modification(s). Still it is extremely necessary to perform exhaustive regression testing as a small change in one module can reflect a bigger change(s) in other module(s). Thus Regression testing has emerged as a great subject of research among software testing community. Three core areas in regression testing that attracts most researchers are: (1) test case minimization; (2) test case selection; (3) test case Prioritization.

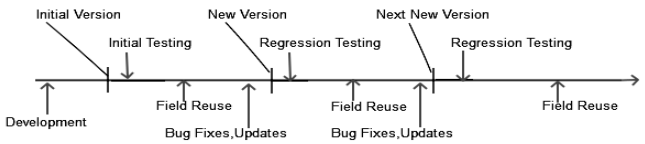


Fig. 1 Journey of a Software System [13].

In this paper we primarily focus on test case prioritization which is to identify the best order in which a given test cases under some test suite must be executed so as to maximize the rate of fault detection. Different Artificial intelligence techniques are available to solve optimization problems [25, 27, 14, 26, 45]. We proposed Particle Swarm optimization as an effective method to obtain best test case execution order and to maximize the rate of fault detection.

ammonia and nitrite exposure atvarious salinities.

Related Works

Various approaches and techniques have been proposed during the last few years which use different techniques and criteria to prioritize given test cases [9, 16, 6, 12, 19, 39]. The most popular unit framework called JUnit is applied to coverage-based prioritization techniques [16]. The prioritized execution of JUnit test cases results in a higher value of APFD than untreated ordering. In Controlled environment, the effectiveness of the prioritization can be measured by executing the test cases according to the fault detection rate [35].

The hybrid approach [6] that integrates coverage based prioritization with distribution based prioritization is based on the observation that basic coverage maximization performs much better than repeated coverage maximization. With repeated coverage maximization there has been a repetition of prioritized test case as it starts again from zero percent after reaching 100% whereas basic coverage maximization stops prioritizing the test cases once 100% coverage is reached.

History based approach [20] is used to prioritize test cases that are prior selected by RTS technique(s). The execution history of each test cases is carefully noted and last recently used (LRU) prioritization called HTC with a value close to zero is used for prioritization that acts as competitive in several constrained testing environment. If number of test cases selected by RTS technique is too high than test cases may undergo an additional prioritization.

Mirarab and Tahvildari [39] introduced a probabilistic approach to prioritization problem which utilizes Bayesian Networks (BN) and a feedback mechanism. The fault finding probability of each test case is predicted using BN and the performance of BN under several different realizations and under mutation technique which can provide a large number of faults is evaluated. The study concluded that most of the other prioritization techniques significantly outperforms well when used in conjunction with feedback [37].

During last few years artificial intelligence emerged as a great subject of interest in software testing [4, 32, 33, 29]. Different AI techniques are emerging as a tool to solve various optimization problems in software testing, such as Ant colony optimization (ACO) [28, 20] which is inspired by foraging behavior of ant colonies, and target discrete optimization problems [28] and combinatorial optimization problems like classic travelling salesman problem, data mining, telecommunication networks, vehicle routing [20, 5, 8, 10, 15, 23, 31, 44], Artificial Bee Colony Optimization (ABC) based on reaction–diffusion equations which results in collective exploration and exploitation of food source (i.e. candidate solutions) by swarm [22], Genetic Algorithms (GA) which is a heuristic search algorithms that is inspired by natural biological evolution of species and solve a variety of optimization problems to improve the quality of the search.

Various NP-hard combinatorial problems such as NP Knapsack [25] and Traveling Salesman Problem [27] in which optimal path to reach from source to destination with less time and cost have been solved successfully using PSO. It has been classified as an optimization approach and used to solve routing optimization [26] problems where the optimal paths for traffic/ packets are searched. PSO has been successfully implemented to solve various Scheduling problems such as job-scheduling [45], task scheduling problems in distributed systems [22], Test pattern generation for circuits [14] and software faults detection [38] are some other applications which are solved using PSO.

In the following sections we discuss various test case prioritization techniques. First we propose a Particle Swarm Optimization based approach for test case prioritization and then its result of prioritization of test cases in a test suite is compared in terms of average percentage of fault detected (APFD Score). The APFD score of the proposed PSO algorithm is compared with the existing work using two case studies- case study 1 and case study 2. With the case study 1, we demonstrate how the proposed algorithm works and how it is compared with the existing work in test case prioritization especially in genetic and Cuscuta search algorithm metaheuristic. The case study 2 is used to compare the proposed algorithm with three search based algorithms- (i) Ant colony based algorithm (ii) Cuscuta search algorithm and (iii) proposed PSO algorithm.

Each algorithm is first discussed briefly and then compare with the proposed PSO algorithm. Throughout the paper we are interested in the following research question

Q) Which algorithm(s)/technique(s) is/are most effective in solving the test case prioritization problem in software testing?

Particle Swarm Optimization: Swarm Intelligence

What is intelligence? There has been a long debate to find the definition of intelligence which is still in its premature stage. Some scholars define

it as the ability to learn in complex situations, to make thought and reason, to bring out profit from experience. Intelligence is more than memory or learning. One definition defines intelligence as "Adaptively variable behavior during the lifetime of an individual" [42].

Kennedy et al., 1995 [18] discovered a unique pattern that direct the ability of birds to fly synchronously within the space and to suddenly change their direction by regrouping themselves without breaking the pattern. This behavior reflects a social corporation between birds within a flock. With this each particle's goal is to reach a global position by cooperating other particle to search their best position within the flock

Particle swarm optimization is a population-based search algorithm that simulates the social behavior of bird, fish and other particles within a swarm. In PSO individuals are called as potential solutions which flown through a hyper-dimensional search space. The changes in the position of each particle within the search space are governed by social –psychological tendency of each particle to follow the most successful particle around them i.e. change in position is influenced by its own experience and knowledge of its neighbors within the swarm. The particles change its velocity while

$\vec{x}^{\,i}$ = particle's position in hyperspace, at time step i

$\vec{v}^{\,i}$ = particle's velocity in hyperspace, at time step i

$$\vec{x}^{\,i}(t+1) = \vec{x}^{\,i}(t) + \vec{v}^{\,i}(t+1)$$

$$\vec{x}^{\,i}(t) = (x^{i1,}(t), x^{i2}(t), x^{i3}, \ldots\ldots\ldots\ldots\ldots x^{in}(t))$$

$$\vec{v}^{\,i}(t) = = (v^{i1,}(t), v^{i2}(t), v^{i3}(t), v^{i4}(t), \ldots\ldots\ldots v^{in}(t))$$

changing its position. The Velocity vector drives the optimization process and reflects the social exchange information.

Where x^{i1} (t),x^{i2} (t),x^{i3} (t),x^{i4} (t),,……. x^{in} (t)) are the coordinates of particle i at time step With this fundamental principle particles flown through the search space and swap their position until they get an optimal solution. Each particle keeps track of its current coordinates in the solution space which are associated with the best solution (fitness: "pfBest") that has achieved so far by that particle. This value is called personal best fitness (pfBest) of particle at the ith position. The current fitness at the position i is calculated using the given objective function and it is compared with the pfBest i.e.

Pseudo- code for pfBest and pBest

Initialize randomly pfBest to some random value
for each particle i
{
fitness =given objective function;
if (fitness > pfBest[i])
{
pfBest[i]= fitness;
pBest[i]= x[i]; /* assign current position x[i] to personal best position "pBest"
}

Each particle in its neighbored keep track of the best value obtained so far (fitness) by any other particle. This best value is called global best fitness (gfbest) of the particle at the ith position. A simple comparison between particle current fitness and gfbest is as follow
Pseudo- code for gfBest and gBest

Initialize randomly gfBest to some random value
for each particle i
{
fitness=given objective function;
if (fitness > gfBest[i])
{
gfBest[i]=fitness;
gBest[i]=x[i]; /* global best position "gBest" is the position x[i] of current particle
}

Each particle is accelerated towards its pBest and the gBest locations, using a random weighted (ρ1, ρ2) acceleration at each time step as shown in figure 2.

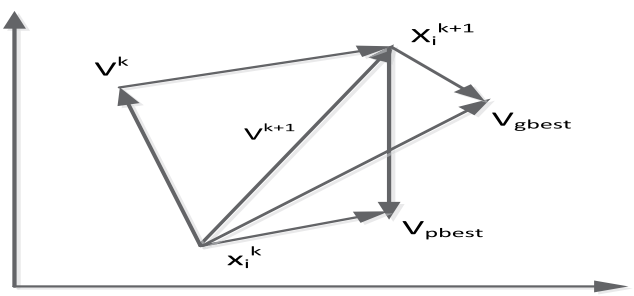


Fig. 2 Modification in position while searching by pso: where, xik : current position, xk+1: modified position, vk: current velocity, vk+1: modified velocity, vpbest velocity based on pBest and vgbest : velocity based on gBest).

The modification of the particle's velocities after comparing its pfBest and gfBest can be mathematically modeled according to the following equation.

$$\vec{v}^{\,i\,k+1} = \vec{v}^{\,i}\ (t) + \rho 1(\vec{x}^{\,pbest_i} - \vec{x}^{\,i\,k}(t))) + \rho 2(\vec{x}^{\,gbest_i} - \vec{x}^{\,i\,k}(t))$$

Where
The random numbers ρ1and ρ2are defined as
ρ1= r1c1 and ρ2= r2c2, where r1, r2 U (0, 1), and c1 and c2 are acceleration constants
ρ1, ρ2 affects velocities and position of particles and it has been asserted that

1. if c1 + c2≤ 4 avoid over the flow of velocities position [17], whereas
2. If c1 + c2 > 4 makes velocities and positions explode toward infinity

Each particle is moved to a new position according to the following equation

$$\vec{x}^{\,i\,k}(t+1) = \vec{x}^{\,i}\ (t) + \vec{v}^{\,i}\ (t+1)$$

In this way particle exchanges their position until they get an optimal point in search space.

Defining Software Test Case Prioritization (Tcp) Problem

Test case prioritization is a technique of ordering given test cases based on some defined prior criteria such that the test cases which find more number of defects/faults are given higher priority i.e. executed first than the others. With this the tester gets an opportunity to prioritize test cases based on some criteria such as maximum code coverage, maximum defect/faults coverage with minimum test suite execution time. In this way testers can save precious time and budget during regression testing. More formally, Prioritization problem is defined as follows [11].

Definition 1. (Test Case Prioritization Problem). Given: A test suite, T, the set permutations of T called PT and a function from PT to real numbers.f:PT→R Problem: To find T' ∈PT such that (∀(T^'')(T^''∈PT) T''≠ T')[F(T')≥f(T'')]

Ideally, the function f should be a mapping from tests to their fault detection capability. But one can know the defect finding capability of a test only after its execution. In practice, a function f is taken, which can substitute fault detection capability of tests. Thus a good choice is of structural coverage.

Informally, Prioritization is the process of scheduling test cases in order to meet some performance goal. We define a test suite T as a tuple of test cases Ti from i=1 to n as (T1, T,……Tn). The goal is to execute Ti in order to meet some performance goal. With Knapsack problem, the minimum time in which we can prioritize the test cases is the maximum output of knapsack, i.e. Test cases are knapsack items, having a total maximum capacity equal to a total number of faults to be covered. The numbers of faults covered by each test case represent its weight and the total time to execute a test case to find the particular number of faults represents the time to put the item (test case) into the knapsack. The knapsack 0/1 algorithm outputs prioritized list in minimum ejection time [30].

0/1 knapsack in terms of test suite prioritization is defined as [1]

Definition 2 (Test Case Prioritization Problem): Maximize: ci xi

Subject to: min (ti, xi), xi = 0 or 1

Where, ci is fault coverage, ti is execution time of test case Ti. Thus, the 0/1 knapsack problem is an NP-complete problem [9]. All NP-complete problems are NP-hard. In this paper, we use particle swarm optimization metaheuristic for solving this hard combinatorial optimization problem.

Modeling Tcp Using Particle Swarm Optimization (Pso)

In order to model the intelligent behavior of proposed PSO algorithm (Algorithm 1), we make the following assumption.

1. The position of each particle (test case) is equivalent to its fault finding capacity.
2. The velocity of each particle is equivalent to the execution time of each test case.
3. Each particle is in search of global maxima/ position in search space and once this global position is found, it stops its search process. Here the global

position is achieved when a particle by exchanging its position covers all the faults in the program.
4. A particle exchanges its position only if shifting to the new position makes the current fitness of particle more fit.
5. When a particle has to choose between two positions, the particle will choose that position which is having more current fitness in search space.
Where fitness in search space is calculated as
Fitness[i] = number of faults found/ total execution time.

Experimental Design

In order to explain and compare the proposed PSO algorithm we have taken two case studies. Case study 1 [2] and case Study 2 [40]. Case study 1 helps us to understand how the proposed PSO algorithm works and is compared with the work done by other researchers on test case prioritization problem. Case study 2 is used to compare the effectiveness of proposed algorithm w.r.t three search based algorithms- (i) ant colony based algorithm (ii) Cuscuta search algorithm, (iii) proposed PSO algorithm and with other traditional techniques such as Random, Reverse and Untreated.
In order to evaluate the performance of various test case prioritization schemes, prior knowledge of faults within the given program is assumed along with execution time to run the test cases. Test suite can be evaluated empirically based on average percentage of fault detected (APFD, for short) over the life time of the test suite. A higher preference will be given to the prioritization scheme having higher APFD value. APFD [24] is defined as

$$APFD = [1- \Sigma^{gi}=1 \text{ reveal}(i,T)/ng] + 1/2n \quad (1)$$

Where, T = test suite, g = number of faults in the program under test, n = number of test cases, reveal(i, T) = position of the first test in T that exposes fault i. Another method to calculate APFD is to find the area under the curve that represents the weighted percentage of faults undetected over the corresponding fraction of the test suite [11].

Validation Of Proposed Pso Algorithm Using Case Study 1

We have taken case study 1 that depend on table 1 which shows the test cases as the test cases represent the possible paths in control flow graph of the source code. Each test case covers a number of conditions, multiple conditions and statement conditions [2].
The test case fault matrix for case study 1 is represented in table 2. The test case fault matrix represents the number of faults identified by a test case and the execution time taken by a test case. The APFD metric is calculated for the proposed PSO approach and compared with other related approaches.
The following mapping is considered between the proposed algorithm's variables and given prioritization problem.
1. We assume that we had prior information about the original test suite T= {t1,t2……tn}and corresponding fault coverage and the total execution time of each test case as shown in table 1 & table 2.
2. A number of particles in search space are equal to a number of test cases.
Let us consider case study 1, we start with test case1 (T1) out of nine available test cases. T1 is positioned at its corresponding fault suite i.e. T1 is positioned at (f1, f2, f3, f5), T2 at positioned at (f1, f2) an so on. The optimal point of each test case is achieved when it found all faults i.e. gfBest is equal to one. Now the particle (test case T1) will search in its domain (nine test cases) the particle having best fitness to combine with. The particle can come to the position of other particle which is having maximum fitness as per the current position of the particle. The current fitness of particle (T1)= count (f1, f2, f3, f5) ÷ total faults .i.e. pfBest (T1) = 4/5=.0.8, i.e.
Position swapped by T1 in each move is as follow
1. Move 1: T1= 2,4,7,9. I.e. pfBest(T1) = 4/5= 0.8
Search the best position to move current position of T1 as:
2. gfBest[T1]= 0/5=0 , as there are no unique faults other than the faults covered in T1
3. gfBest[T2]=0/5, as fault covered by T2={ f1,f2} so there are no unique faults other than the faults covered in T1.
4. gfBest[T3]=0/5=0; as fault covered by T3={

f1,f3,f5} so there are no unique faults other than the faults covered in T1

5. gfBest[T4]=1/5=0.2 as fault covered by T4={ f1,f4,f5} so there is one fault(f4) that is unique fault other than the faults covered in T1. In the similar manner the gfBest for other particle are

6. gfBest[T5]=0/5=0;

7. gfBest[T6]=0/5=0;

8. gfBest[T7]= 0/5=0;

9. gfBest[T8]=1/5=0.2;

10. gfBest[T9]= 0/5=0;

The maximum fitness (gfBest) out all particles is of particle T4 and T8. But as both is having the same gfBest, so the particle T1 will choose the particle T8 as its lBestT8> lBestT5.

The final move sequence is shown in figure 3.

Thus the new position of particle become equal to {f1, f2, f3, f4, f5} i.e. the new pfBest [new] =5/5=1; which is equal to gfbest. When pfBest [new] = gfbest, the algorithm stops otherwise it continues to repeat the same process for each particle.

In a similar manner the different moves by all particles are presented in table 3.

The PSO ordering is obtained from the table 3 on the basis of execution time. The test case having minimum execution time is set at higher priority followed by next higher execution time.

Thus we obtained the PSO order for case study 1 as T6,T8,T1,T4,T3,T2,T9,T5,T7.

Algorithm 1 Proposed PSO algorithm

Input: Test case-fault matrix
Output:-Prioritize test suite T with* its *Average Percentage of Fault Detected (APFD) Score

```
Step 1. Begin
Step 2. Count total number of faults (TF) in given test prioritization problem.
Step 3. Set the number of particles= Number of Test cases.
Step 4. Set gfbest=1;
Step 5. Set V_max;          /* set maximum velocity to avoid overflow condition
Step 6. Set X_min=0;
Step 7. /*Initialize each particle position x[i] with its fault into search space and assign its
    initial velocities v[i]
       for (i=0; i< number of particles; i++)
        x[i]=fault number identified by ith test case
        v[i]= execution time of ith test case
Step 8. for( i=0; i< number of particles; i++)
          /*Initialize each particle pfBest
           pfBest[i]= number of faults in x[i] ÷ TF ;
 Step 9.  for( i=0; i< number of particles; i++)
           Array=i;
          do {
          fitness[i]= count of x[i] ÷ TF       /* count total no of faults in [i]
         pfBest[i]=fitness;
        for( j=0; i< number of particles; j++){
        /* return gfBest
        gfbest[j]= total unique faults in x[j] which are not in x[i] ÷ TF }
        for( k=0; i< number of particles; k++){
         /* return index of max gfBest
        P= max( gfbest[k], gfbest[k+1], gfbest[k+2],……. gfbest[n]) }
       X[new] = U(x[i], x[p])
/* new position is union of previous position fault's + new position fault's
pfBest [new]= count of x[new] ÷ TF
if (pfBest[i]< pfBest[new]){
x[i]= X[new] ;              /* update new position
v[i]= v[i] + v[p];          /* update new velocity
}
If (v[i]> v_max){
Set v[i]= v_max
}
If (v[i]< v_min){
Set v[i]= v_min }
Array= array+ P;
while (pfBest[i]! = gfBest)
Print " test sequence for the particle move is " Array ";
Step 10. Find sort _ascend (v[i])   /*Arrange Test cases in order of increasing execution time
Step 11. Print "The Prioritization order will be the corresponding array index Seq_arry[n] in order of sort
_ascend v[i]".
end
```

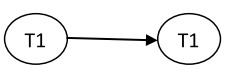


Fig. 3 Total number of moves by particle (T1)

Calculating Average Percentage Of Faults Detected (Apfd)

APFD depends on two things (i) calculation of the total percentage of test suite executed and (ii) number of fault detected by each percentage of test suite executed. In this example we elaborate this calculation w.r.t proposed PSO Algorithm

The PSO ordering is TC ={T6,T8,T1,T4,T3,T2,T9,T5,T7} i.e. a total of 9 test cases in test suite TC. Thus if we execute this sequence then percentage of test suite executed is calculated as

(i) T6= (1/9)*100= 11.11%. Also, for execution of T8 it is necessary to execute T6 first i.e.

(ii) T6, T8= (2/9)*100= 22.22%. Also, for execution of T1 it is necessary to execute T6, T8 first.

(iii) T6, T8, T1 = (3/9)*100= 33.33%. The rest are calculated in similar manner as

(iv) T6, T8, T1, T4= (4/9)*100= 44.44%.

(v) T6, T8, T1, T4, T3= (5/9)*100= 55.55%.

(vi) T6, T8, T1, T4, T3, T2= (6/9)*100= 66.66%.

(vii) T6, T8, T1, T4, T3, T2, T9 = (7/9)*100= 77.77%.

(viii) T6, T8, T1, T4, T3, T2, T9, T5= (8/9)*100=88.88%.

(ix) T6, T8, T1, T4, T3, T2, T9, T5, T7= (9/9)*100=100.00%.

Now we calculate a number of faults detected for each percentage of test suite execution. In the case of PSO, For 11.11% test suite execution, a number of participating test cases are {T6} only, which covers {f1,f3, f5 } faults out of total five faults, Thus the number of faults detected by executing 11.11% of test suite in case of PSO is 3/5= 0.6. For 22.22 % test suite execution, number of participating test cases are {T6,T8} only, which covers {f1,f2,f3,f4, f5 } faults out of total five faults. Thus a number of faults detected by executing 22.22% of test suite in case of PSO are 5/5= 1. In similar manner number of fault detected for 33.33%, 44.44%, 55.55%, 66.66%, 77.77%, 88.88% and 100% are 1,1,1,1,1,1,1 respectively

The proposed PSO Ordering is shown using the figure 4.

Thus if we consider the case of PSO ordering the APFD is calculated by finding the area under the curve given in figure 4, which comes out 89.99% (APFD).

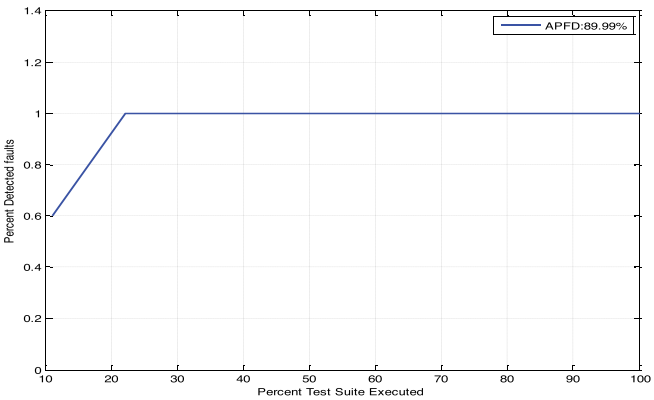


Fig. 4 APFD Score for Proposed PSO [case study 1].

The APFD score can also be calculated using the formula 1 as discussed above

APFD (PSO ordering) = [(1-((1+2+1+2+1)/9*5))+ (1/2*9)]*100 = 89.99%

Both methods results in same APFD.

Table 1. Test cases and its coverage items.

Test case no	*Condition Coverage (CC)*	*Multiple Condition Coverage (MC)*	*Statement Coverage (SC)*
1	4	3	18
2	5	4	18
3	5	4	19
4	4	2	17
5	3	2	19

6	4	3	16
7	5	4	16
8	5	4	17
9	3	1	18

Table 2. Test case- fault matrix for case study

Test Case/Fault	*F1*	*F2*	*F3*	*F4*	*F5*	*NO OF FAULTS*	*EXECUTION TIME*
T1	*	*	*		*	4	11.5
T2	*	*				2	11.5
T3	*		*		*	3	12.33
T4	*			*	*	3	10.66
T5	*					1	15
T6	*		*		*	3	8.33
T7	*					1	15
T8	*	*		*		3	10
T9			*			1	11
Severity Value (s_i)	6	8	5	4	9		

Table 3. Moves sequences by PSO

Particle	***Initial Position***	***Moves***				***Final Position***	***Execution _Time***
T1	1,2,3,5	Move	T1	T8		1,2,3,4,5	21.5
		Time	11.5	10			
		Position	1,2,3,5	1,2,4			
T2	1,2	Move	T2	T6	T8	1,2,3,4,5	29.83
		Time	11.5	8.33	10		
		Position	1,2	1,3,5	1,2,4		
T3	1,3,5	Move	T3	T8		1,2,3,4,5	22.33
		Time	12.33	10			
		Position	1,3,5	1,2,4			

T4	1,4,5	Move	T4	T1		1,2,3,4,5	22.16
		Time	10.66	11.5			
		Position	1,4,5	1,2,3,5			
T5	1	Move	T5	T1	T8	1,2,3,4,5	36.5
		Time	15	11.5	10		
		Position	1	1,2,3,5	1,2,4		
T6	1,3,5	Move	T6	T8		1,2,3,4,5	18.33
		Time	8.33	10			
		Position	1,3,5	1,2,4			
T7	1	Move	T7	T1	T8	1,2,3,4,5	36.5
		Time	15	11.5	10		
		Position	1	1,2,3,5	1,2,4		
T8	1,2,4	Move	T8	T6		1,2,3,4,5	18.33
		Time	10	8.33			
		Position	1,2,4	1,2,3,5			
T9	3	Move	T9	T1	T8	1,2,3,4,5	32.5
		Time	11	11.5	10		
		Position	3	1,2,3,5	11,2,4		
Total Execution Time							**237.98**

Comparison With Different Ordering

We compare the result of proposed PSO algorithm with No order, Random order, Reverse order, optimal order and various other existing work as shown in table 4.

The various approaches and their prioritization order as mentioned in table 4 are compared by calculating their average percentage of faults detected (APFD).

The proposed algorithm show better APFD score for six techniques and it has near optimal score (89.99%), which has slightly lower AFFD (2.22% lower) than the optimal (92.22%) and M-C- GA- FF*APFD (92.22%) score. Thus we are close to the optimal value and the algorithm itself shows a different approach for test case prioritization.

Table 4. Comparison between various algorithms [case study 1]

S.No	*Test Case Prioritization Technique*	*APFD*
1	Reverse order	12.22%
2	Random order	41.11%
3	The Prioritized order by average faults found per minute algorithm (AF/M) *[41]*	87.77%

4	The proposed Multi-Criteria Genetic algorithm technique with the proposed fitness (MC-GA-PF) *[2]*	85.56%
5	Untreated order	87.78%
6	The proposed Multi-Criteria Genetic algorithm technique which the fitness is APFD (M-C-GA=APFD) *[2]*	92.22%
7	The proposed Multi-Criteria Genetic algorithm technique which the fitness is the proposed fitness multiplied by APFD [M-C- GA- F_{F*APFD}] *[2]*	92.22%
8	Hybrid Particle Swarm Optimization for Regression Testing (H-PSO) *[3]*	75.6
9	The optimal order	92.22%
10	The Cuscuta Search Ordering *[30*]	89.99%
11	The Proposed PSO Ordering	89.99%

Comparison Of Proposed Pso Algorithm With Search Based Algorithms Using Case Study 2

In this section we compare proposed PSO algorithm with three different searches based algorithm namely Ant colony based algorithm, Cuscuta search algorithm and Proposed PSO algorithm. Beside the mentioned search algorithms we also compare the effectiveness of these search based algorithms w.r.t Optimal, Random, Reverse and original ordering. Table 5 & Table 6 shows case study 2 which is similar to case study 1 except that here we do not know about Condition Coverage (CC), Multiple Condition Coverage (MC), Statement Coverage (SC) and the Severity Value (si). We are given only the test case, its corresponding revealed fault(s) and execution time for each test case.
First we briefly introduce the fundamental of all search based algorithms taken in this case study and then one by one we search for the following research questions

1) Are search based techniques more effective than traditional techniques for test case prioritization (in terms of APFD)?

2) Which search based technique among (a) Ant colony based algorithm (b) Cuscuta Search algorithm, and (c) Proposed PSO algorithm is more effective in terms of APFD for test case prioritization problem?

For answering the first question we break it into the following sub-questions:

1.1) Is Ant colony based algorithm for test case prioritization outperforms other traditional techniques such as optimal, random, reverse and original ordering?

1.1.2) Is Cuscuta search algorithm for test case prioritization outperforms other traditional techniques such as optimal, random, reverse and original ordering?

1.1.3) Is Proposed PSO algorithm for test case prioritization outperforms other traditional techniques such as optimal, random, reverse and original ordering?

Table 5. Sample test cases vs. faults identified.

Test case/faults	*F1*	*F2*	*F3*	*F4*	*F5*	*F6*	*F7*	*F8*	*F9*	*F10*
T1		*		*			*		*	
T2	*		*							
T3	*				*		*	*		
T4		*		*					*	
T5			*			*				*
T6	*						*			
T7			*			*		*		
T8		*								*

Ant Colony Based Algorithm: A Wild Metaphor

In 1959 Grasse first introduced a term know as Stigmergy to the indirect form of communication among multi-agents evolving as a self-organized single system while modifying their local environment. The studies on ant colonies Stigmergy nature has proved an indirect communication between each ant by depositing a chemical known as pheromone on their path and the ant then tends to follow that path, gradually all ants converge to the trail having a higher concentration on pheromone deposit [7].

Let us demonstrate a simple "shortest bridge experiment" to demonstrate forging behavior of real ants, figure 5 shows the diagrammatic view of setup in which two possible paths from source(nest) to destination(food) are shown. Path A is shorter than path B. Initially as the ant move from the nest foraging for food, it follow a random path but as the time passes we observed that all ants following the path A. This is due to the fact that when initially an ant move from her nest following path A and after collecting its food it will reach to its nest again in less time than an ant which had followed the path B due to the distance parameter and in this way it had deposited a pheromone before than the second ant following the path B. Thus a convergence to the shortest path is achieved.

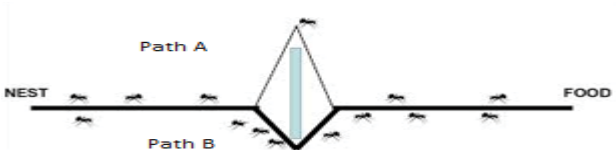


Fig. 5 Two possible paths for foraging

The experiment shows the exploration to exploitation due to pheromone deposition tendency of ants in the real world.

Tour Construction

Initially, an ant is put on the initial node and by using action choice rule, called random proportional rule, to decide which node to visit next. In particular, the probability with which ant k, currently at node I, chooses to go to node j is

$$p_{ij} = \frac{[\tau_{ij}]^{\alpha} \times [\eta_{ij}]^{\beta}}{\sum_{l \in N_i^k} [\tau_{ij}]^{\alpha} \times [\eta_{ij}]^{\beta}} \quad if\ l \in N_i^k$$

where ηij=1/dij is a heuristic value that is available a priori, indicated the visibility of a path for an ant at the current vertex, α and β are two parameters which determine the relative influence of the pheromone trail and the heuristic information, and l€ N_i^kis the feasible neighborhood of ant k when being at node .

The roles of the parameters are indicated in table 7.

Some good parameters setting used in ACO algorithm are shown in table 8. Good parameter values are very important in order to converge algorithm at the optimal point.

Table 7. Role of parameters in ACO probabilistic rule.

Parameter	*Value*	*Biasing*	*Conclusion*
α	Zero	Heuristic biased	the closest node is selected
β	Zero	Pheromone biased	may lead to poor convergence

Table 8. Parameter setting for various aco algorithms [28]

ACO algorithm	*α*	*β*	*Pheromone Evaporation* (ρ)	*Number of ants M*	*Initial Pheromone* t_0
AS	1	2 to 5	0.5	N	m/c^{nn}
EAS	1	2 to 5	0.5	N	$(e+m)/ \rho c^{nn}$
ASrank	1	2 to 5	0.1	N	$0.5r(r-1)/ \rho c^{nn}$
MMAS	1	2 to 5	0.02	N	$1/ \rho c^{nn}$
ACS	-	2 to 5	0.1	10	$1/nc^{nn}$

Cuscuta Search Algorithm

Till now we have strong observations and formulation about the animal's intelligent behavior such as ant colony, bee colony [43]. They involve foraging for food not by simple but by collective intelligence behavior. The plant's foraging has been the least studied area in computational intelligence. Not only animals as described above forage for food intelligently but the same have been done by the plants too.

One such example is the dodder (Cuscuta sp) as shown in figure 6, which attacks its prospective host through some host-plant clue. If the host is found unsuitable the Cuscuta sp. continue its search but once the selection is made the Cuscuta sp. coil around its selected host in a specific manner (anticlockwise) to transfer resources from the host plant. Research has found that Cuscuta sp. seedlings show directed growth toward tomato volatiles experimentally released in the absence of any other plant-derived clue [30]. Furthermore, volatile cues (α-pinene, β-myrcene and β-phellandrene) are used by the dodder to "choose" tomatoes, a preferred host, over non-host wheat.

Fig 6. A Dodder (Cuscuta sp.) (Light yellow) coiling around its host. Cuscuta sp. has the ability to assess its prospective host before coiling around the host plant [21] and thus it does not coil around every host with which it comes in contact. If the prospective host is found to be unsuitable the parasitic plant continues its search for other hosts. Photo courtesy: Mukesh Mann and Om Prakash Sangwan [30].

A key point of observation is that Cuscuta somehow knows its starvation i.e. if the same cues would have been coming from the wheat, the bend will be towards the wheat rather than tomato. Considering this dynamics Cuscuta search algorithm for test case prioritization has been proposed [30].

In order to answer the research questions framed above, we need to (i) compare the APFD Scores of each search based algorithms as discussed

above with other traditional test case prioritization techniques and (ii) compare the APFD Scores of each search based algorithms with each other.

We applied the ant colony optimization [40], Cuscuta search algorithm [30] and proposed PSO algorithm on the case study 2 and APFD Score is calculated. A comparative analysis of APFD Score is shown from figure 7.

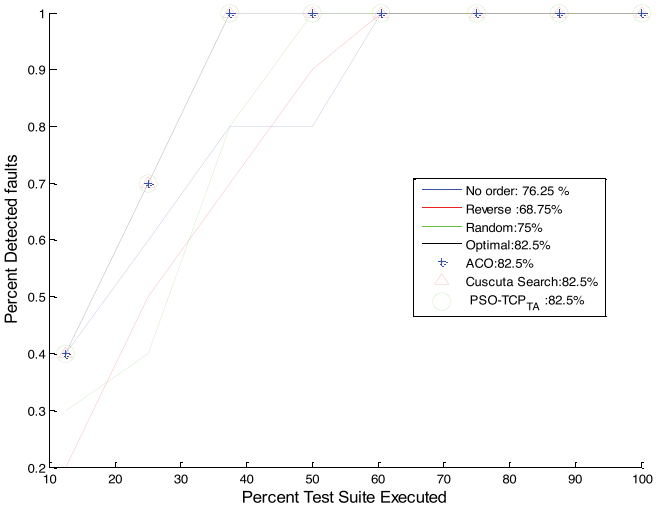
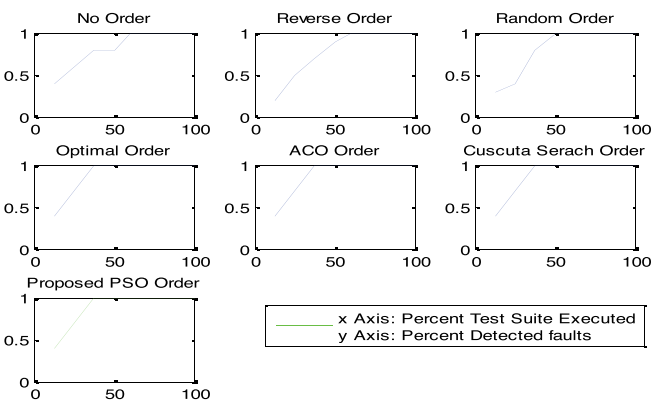


Fig. 7 APFD Comparison for different Prioritization Techniques.

An individual comparison of each prioritizing technique is plotted and the individual AFPD behavior during the lifetime of total percentage execution of test suite is observed as shown in figure 8.



Results obtained by measuring the Average Percentage of Faults Detected (APFD) shows that proposed PSO algorithms show similar results as shown by Cuscuta search, ACO and optimal ordering and all search based algorithms as discussed above shows better performance w.r.t No order, Random order and Reverse order. Figure 8 clearly shows the effectiveness of three search based technique in detecting the average percentage of faults.

All search based algorithms and traditional techniques are compared w.r.t their APFD score as shown in table 9.

The various search based approaches and their prioritization order are compared by calculating their average percentage of faults detected (APFD). From the figure 8, it is clear that APFD Score of the Proposed PSO algorithm is same as that of Cuscuta search, Ant colony based algorithm and optimal ordering. It has been observed that proposed algorithm outperform random ordering, reverse ordering and no ordering for test case prioritization. PSO can be used in large and complex test suite prioritization problems and thus saving the larger amount of time and cost during software development life cycle as compared to smaller ones. With this approach software testers can easily select and prioritize test cases with minimum execution time and a higher percentage of fault detection.

Table 9. Comparison between various Search based algorithms [case study 2]

S.no	*Test Case Prioritization Technique*	*APFD Score*
1	Ant colony based algorithm	82.5
2	Cuscuta search	82.5
3	Optimal ordering	82.5
4	Proposed PSO Algorithm	82.5
5	Random Ordering	75
6	Reverse ordering	68.75
7	Untreated ordering	76.25

Conclusion

In this paper we have designed and implemented an artificial particle swarm optimization algorithm for test case prioritization problem using two case studies. We have compared our approach against four other metaheuristic approaches – Multi-Criteria Genetic algorithms (with different versions), an ant colony optimization approach, Cuscuta search algorithm and Hybrid Particle Swarm Optimization algorithm. Our approach has APFD score of 89.99% which has 2.22% lower AFFD than the optimal (92.22%) and Multi-Criteria Genetic algorithms (92.22%). Our approach outperforms Hybrid Particle Swarm Optimization (75.6% APFD), whereas the APFD score of our proposed approach is same as that of Cuscuta search algorithm and ant-colony optimization approach in both case studies. We have compared our approach with four traditional methods of prioritization- random ordering, reverse ordering, untreated ordering and Prioritized order by average faults found per minute algorithm. Our approach outperforms all the traditional approaches in both case studies.

By observing the good performance of this new paradigm, it is expected that proposed PSO will be used to solve problems in many areas such as machine learning, test case generation and optimization. As a future work, it is planned to study on applying proposed PSO to solve real-world problems and also on improving the performance of the algorithm by introducing new modifications.

Acknowledgement

The authors are very grateful to the anonymous reviewers for providing valuable and detailed comments that have significantly improved the paper.

References

Alspaugh S, Walcott K.R, Belanich M, Kapfhammer G.M, and Soffa M.L(2007) Efficient time-aware prioritization with knapsack solvers. In Proc. Of WEASELTech, 22(1):13-1

Amr Abdel, Fatah Ahmed, Mohamed Shaheen and Essam Kosba (2012) Software testing suite prioritization using multicriteria fitness function. In ICCTA, IEEE, Alexandria, Egypt, pp. 160-166.

Arvinder Kaur and Divya Bhatt (2011) Hybrid Particle Swarm Optimization for regression Testing. Int. Journal of Computer Science and Engineering 3(5):1815-1824.

Briand L. C (2002) On the many ways Software Engineering can benefit from Knowledge Engineering. Proc. 14th SEKE, Italy, 3-6.

Caro G.D, and Dorigo M (1998) AntNet: Distributed Stigmergetic Control for Communications Networks. Journal of Artificial Intelligence Research, 9:317-365

David Leon and Andy Podgurski (2003) A comparison of coverage-based and distribution based techniques for filtering and prioritizing test cases. In Proceedings of the 14th International Symposium on Software Reliability Engineering, ISSRE '03, 442–450

Deneubourg JL, S. Aron S, Goss S and Pasteels JM (1990) The self-organizing exploratory pattern of the argentine ant. Journal of Insect Behavior, 3, 159-168

Dorigo M, Maniezzo V, and Colorni A (1996) Ant system: optimization by a colony of cooperating agents. Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on, 26(1):29-41.

G. Rothermel, R.H. Untch, Chengyun Chu and M.J. Harrold (2001) Prioritizing test cases for regression testing. Software Engineering, IEEE Transactions on software engineering, 27(10):929 –948

Gomez O, Baren B (2005) Omicron ACO: A New Ant Colony Optimization Algorithm. CLEI Electronic Journal, 8(1):1-8

Gregg Rothermel, Roland H. Untch, Chengyun Chu and Mary Jean Harrold (1999) Test Case Prioritization: An Empirical Study. In Proceedings of the International Conference on Software Maintenance, 1-10

Gregg Rothermel and Mary Jean Harrold (1996) Analyzing regression test selection techniques. IEEE Trans. Software Eng., 22(8):529–55.

Hiralal Agrawal, Joseph R. Horgan, Edward W. Krauser and Saul London (1993) Incremental regression testing. In Proceedings of the Conference on Software Maintenance, ICSM '93, pp 348–357

Hou,Y, Zhao C and Liao,Y (2006) A new method of test generation for sequential circuits, in Proceedings, (2006,International Conference on Communications, Circuits and Systems, 2181–2185.

Huaizhong Li and C.Peng Lam (2005) Software Test Data Generation using Ant Colony Optimization. In Proceedings Of World Academy of Science, Engineering And Technology, 1:1-4.

Hyunsook Do, Gregg Rothermel and Alex Kinneer (2004) Empirical studies of test case prioritization in a junit testing environment. In Proceedings of the 15th International Symposium on Software Reliability Engineering, pp 113–124

J. f Kennedy (1998) The Behavior of Particles, in V.W. Porto, N. Saravana, D.Waagen (ed.), proceedings of the 7th Conference on Evolutionary Programming, pp. 581-589

J. Kennedy and R.C. Eberhart (1995) Particle Swarm Optimization, Proceedings of the IEEE International Conference on Neural Networks, 4,. 1942-1948

Jung-Min Kim and Adam Porter (2002) A history-based test prioritization technique for regression testing in resource constrained environments. In Proceedings of the 24th International Conference on Software Engineering 2:119–129

K. Ayari, S. Bouktif and G. Antoniol (2007) Automatic Mutation Test Input Data Generation via Ant Colony. Genetic and Evolutionary Computation Conference, London,1074-1081

Kelly C. K (1990) Plant foraging: a marginal value model and coiling response in Cuscuta subinclusa. Ecology, 71: 1916 – 1925

Kong,X., and Sun, J., Xu,W (2006) Particle swarm algorithm for tasks scheduling in distributed heterogeneous system, In Proceedings of Sixth International Conference on 6: 690–695.

Li L, Ju S and Zhang Y (2008) Improved ant colony optimization for the traveling salesman problem. In Proceedings of 1st International Conference on Intelligent Computation Technology and Automation, 76–80.

Krishnamoorthi R, Sahaaya S.A, andMary A (2009). Regression Test Suite Prioritization using Genetic Algorithms. International Journal of Hybrid Information Technology, 2(3):35-52

Liang,Y, Liu,L.,Wang,D and WU, R (2010) Optimizing Particle Swarm Optimization to Solve Knapsack Problem, ICICA, CICIS, 105:437-443.

Liu, H., Sun, S., and Abraham, A (2006) Particle swarm approach to scheduling work-flow ap p lications in distributed data- intensive computing environments, Proceedings of Sixth International Conference on In Intelligent Systems Design and Applications, 6: 661–666.

Lope, H.S., and Coelho, L.S (2005) Particle Swarm Optimization with the fast local search for the blind traveling salesman problem, Proceedings of Fifth International Conference on hybrid intelligent systems, 5: 245-250.

Marco Dorigo and Thomas Stutzle (2005) Ant Colony Optimization. phi publishers.

Mark Harman (2007) The Current State and Future of Search Based Software Engineering. International Conference on Software Engineering. Future of Software Engineering, IEEE Computer Society press, Washington, DC, USA, 342-357.

Mann Mukesh and Sangwan Om Prakash (2014) Test case prioritization using Cuscuta search. Network Biology, 4(4): 179-192.

Parpinelli RS, Lopes HS and Freitas AA (2002) Data mining with an ant colony optimization algorithm. IEEE Transactions on Evolutionary Computation, 6(4): 321-332

Pedrycz W and Peters, J. F (1998) Computational Intelligence in Software Engineering", World Scientific Publishers.

Phil. McMinn (2004) Search-Based Software Test Data Generation: A Survey, 14: 212-223. Proceedings of Sixth International Conference on Software Testing, Verification and Reliability, 6: 690–695.

Rothermel G, Untch R. H, Chu C, and Harrold M. J (2001) Prioritizing test cases for regression testing. IEEE Transactions on Software Engineering, 27(10):929-948

Yoo, S., and Harman, M (2012). Regression testing minimization, selection and prioritization: a survey. Software Testing Verification and Reliability, 22 (2): 67-120

Runyon J.B, Mescher M.C, and De Moraes C.M (2006) Volatile chemical cues guide host location and host selection by parasitic plants, Science 313:1964–1967

Sebastian Elbaum and Alexey Malishevsky (2002) Test case prioritization: A family of empirical studies. IEEE Transactions on Software Engineering, 28:159–182.

Sheta A (2006) Reliability growth modeling for software fault detection using particle swarm optimization, Proceedings of IEEE Congress on Evolutionary Computation, 3071–3078.

Siavash Mirarab and Ladan Tahvildari (2008) An empirical study on Bayesian network-based approach for test case prioritization. In Proceedings of the International Conference on Software Testing, Verification, and Validation, 278–287

Singh Y, Kaur A, and Suri B (2010) Test Case Prioritization using Ant Colony Optimization ACM SIGSOFT Software Engineering Notes 35:1-7

Srivastava P.Ranjan (2008) Test case prioritization. Journal of Theoretical and Applied Information Technology JATIT, 178-181.

Stenhouse D (1974) The Evolution of Intelligence: A General Theory and Some of Its Implications. Harper & Row, USA

Tereshko V (2000) Reaction-diffusion model of a honeybee colony's foraging behavior. In Schoenauer M.(ed.) Parallel Problem Solving from Nature VI. Lecture Notes in Computer Science, Springer, 1917: 807–816

Zhao P, Zhao P, and Zhang X (2006) New Ant Colony Optimization for the Knapsack Problem. In the Proceedings of the 7th International Conference on Computer-Aided Industrial Design and Conceptual Design,1-3

Zhao, F., Zhang, Q., and Yang, Y (2006) An improved particle swarm optimization based approach for production scheduling problems, In Proceedings of the IEEE International Conference on Mechatronics and Automation, 2279–2283.