A
Dissertation
on

"Application of River Formation Dynamics in Search Based Software Engineering"

Submitted in Partial Fulfillment of the Requirement
For the Award of Degree of

MASTER OF TECHNOLOGY
in
Computer Science and Engineering
Delhi Technological University, Delhi

SUBMITTED BY

YOGITA KHATRI
2K14/CSE/21

Under the Guidance of

Mrs. ABHILASHA SHARMA and Dr. AKSHI KUMAR

Assistant Professor(s)
Department of Computer Science and Engineering
Delhi Technological University

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
DELHI TECHNOLOGICAL UNIVERSITY
2014-16

Certificate

This is to certify that the work contained in this dissertation entitled "**Application of River Formation Dynamics in Search Based Software Engineering**" submitted in the partial fulfillment, for the award of degree of M.Tech in Computer Science and Engineering at **DELHI TECHNOLOGICAL UNIVERSITY** by **YOGITA KHATRI, Roll No. 2K14/CSE/21,** is carried out by her under my supervision. The matter embodied in this project work has not been submitted earlier for the award of any degree or diploma in any university/institution to the best of our knowledge and belief.

(Mrs. ABHILASHA SHARMA)
Project Guide
Assistant Professor
Department of Computer Science and Engineering
Delhi Technological University

Certificate

This is to certify that the work contained in this dissertation entitled "**Application of River Formation Dynamics in Search Based Software Engineering**" submitted in the partial fulfillment, for the award of degree of M.Tech in Computer Science and Engineering at **DELHI TECHNOLOGICAL UNIVERSITY** by **YOGITA KHATRI, Roll No. 2K14/CSE/21,** is carried out by her under my supervision. The matter embodied in this project work has not been submitted earlier for the award of any degree or diploma in any university/institution to the best of our knowledge and belief.

(Dr. AKSHI KUMAR)
Project Co-Guide
Assistant Professor
Department of Computer Science and Engineering
Delhi Technological University

Acknowledgement

I thank the almighty god and my parents, who are the most graceful and merciful, for their blessing that contributed to the successful completion of this project.

I take this opportunity to express a deep sense of gratitude towards my guide **Mrs. ABHILASHA SHARMA** and co-guide **Dr. AKSHI KUMAR**, for providing excellent guidance, encouragement and inspiration throughout the project work. Without their invaluable guidance, this work would never have been a successful one.

I am also grateful to Dr. O.P. Verma, HOD, Computer Science and Engineering Department, DTU for his immense support. I would also like to acknowledge Delhi Technological University library and staff for providing the right academic resources and environment for this work to be carried out.

I would also like to thank all my classmates for their valuable suggestions and helpful discussions.

YOGITA KHATRI

(2K14/CSE/21)

Abstract

Search Based Software Engineering (SBSE) is an emerging field, involves applying search based techniques to address the various problems related to different domains of software engineering.

It is specially excelled in providing an ideally balanced solution to a software engineering problem involving several competing goals and provides optimal solution to give better prospects over tools and techniques required to grow a productive, proficient and cohesive approach. Software testing is the area which is benefitted the most. Although there are many testing related problems which are solved by search based techniques, but automatic test data generation is the prime activity, capturing researcher's attention these days. Test path generation and prioritization is one of the important activities to reduce test effort. But it hardly gets importance in the existing literature as only few researchers have attempted to generate test sequences using different search based algorithms (Ant Colony Optimization, Genetic Algorithm, Tabu Search, Cuckoo Search, Firefly Algorithm), each having their own merits and demerits. Ideally none of them is perfect as far as complete path coverage and redundancy is concerned.

This dissertation aims at carrying the SWOT (strengths, weaknesses, opportunities, threats) analysis of the area i.e. search based software engineering and improving the efficiency of software testing process by generating the optimal test sequences in the control flow graph (CFG) of the program under test (PUT) by using a novel swarm intelligence method called River Formation Dynamics(RFD), inspired by a natural phenomena of how drops transformed into river and then river into sea. It provides full path coverage with zero redundancy in transitions from one node to another. It also tries to prioritize the paths based on their strength, calculated in terms of their traversal by the drops.

Keyword: Cyclometic Complexity, Control Flow Graph, River Formation Dynamics, Test Sequence, Program Under Test

List of Figure(s)

List of Table(s)

List of Abbreviation(s)

ACO	*Ant Colony Optimization*
CFG	*Control Flow Graph*
FA	*Firefly Algorithm*
GA	*Genetic Algorithm*
PUT	*Program Under Test*
RFD	*River Formation Dynamics*
SBSE	*Search Based Software Engineering*
SBST	*Search Based Software Testing*
SE	*Software Engineering*
SWOT	*Strengths Weaknesses Opportunities Threats*

Content(s)

Chapter 1 Introduction and Outline

This Chapter briefly introduces the research work proposed in this thesis. Section 1.1 provides an overview of the research undertaken. Section 1.2 clears the motivation behind the research work. Section 1.3 sets out the research objectives. Section 1.4 illustrates the proposed approach. Section 1.5 presents an outline of this thesis describing the organization of the remaining chapters. Finally, Section 1.6 gives the summary of the chapter.

1.1. Introduction

Search Based Software Engineering is an emerging field basically involves amalgamation of software engineering, operation research and metaheuristic techniques. Different phases of software engineering are complex in nature and consist of numerous problems. Although, we have many classical approaches to solve these problems like linear programming, dynamic programming, but these approaches do not work well with NP-hard problems as they cover maximum range of SE problems. However, it has been found by many researchers that SBSE often gives satisfactory results when used on such problems. They may not be able to give a global optimal solution but can provide a list of near optimal solutions. It provides an efficient way to automate software engineering tasks by applying meta-heuristic search algorithms.

It is specially excelled in providing an ideally balanced solution to a SE problem involving several competing goals. It caters to the problem of SE and provides optimal solution to give better prospects over tools and techniques required to grow a productive, proficient and cohesive approach.

After doing literature survey, it has been observed that software testing is the major area, explored the most, in SBSE, more formally known by the phrase search based software testing (SBST). It involves the application of various search based techniques like hill climbing, simulated annealing, tabu search, genetic algorithm, ant colony optimization etc. to address the testing related problems.

To discover maximum faults in minimum time is the major aim in software testing. It is the most important and a very crucial phase of software development life cycle to ensure software validity and quality. It is a very rigorous process and by far the most costly and time consuming activity and almost account for 50% of the total effort. Therefore, it becomes a major challenge for the researcher to optimize the software testing process.

Many researchers have tried automating various testing activities like test data generation, test case selection, test case prioritization, testing effort estimation etc. to reduce the testing effort as much as possible. Out of all testing activities, test data generation has got the highest researcher's attention whereas test sequence generation which is also a significant activity in cutting down the testing effort, hardly gets importance. Only few researchers have attempted to generate test sequences using different search based algorithms like ant colony optimization, genetic algorithm, tabu search, cuckoo search, firefly algorithm, having their own advantages and disadvantages. Ideally, none of them is perfect as far as coverage and redundancy is concerned.

Therefore, this thesis summarizes in and out of the area i.e. search based software engineering and focuses on coming up with a new approach for test sequence generation and prioritization, that should provide complete path coverage with no redundancy, details of which is explained in the later chapters. The remainder of this chapter clears the motivation behind carrying the research work, sets out the research objectives, describes the main contributions of the research work, and presents an outline of this thesis.

1.2. Motivation

Optimizing the testing effort is one of the prime concerns of the researchers working in this field. A lot of researchers have attempted to automate different testing activities by applying search based techniques. Test sequence generation, which is a significant activity in cutting down the testing effort, hardly gets importance as compared to other testing activities in the existing literature. Only few researchers have attempted to generate test sequences using different search based algorithms like ant colony optimization, genetic algorithm, tabu search, cuckoo search, firefly algorithm, having their own merits and demerits. But none of them is ideal as far as coverage and redundancy is concerned. This inspired us to come up with a new approach for path generation and prioritization, which should be efficient in terms of both the parameters i.e. coverage and redundancy.

1.3. Research Objective(s)

Statement of Research Question

"Can intelligent River Formation Dynamics algorithm be used to optimize the process of Test Sequence Generation in Search Based Software Engineering?"

Consequently, the four main objectives of the research work taken are:

- To carry out the SWOT analysis of Search Based Software Engineering,
- To develop an efficient approach for optimizing the task of test sequence generation via River Formation Dynamics algorithm.
- To prioritize the test paths to observe critical/error prone paths to be tested first.
- To compare and analyze the proposed approach with respect to other existing approaches used for test sequence generation.

1.4. Proposed Framework

To reduce the testing effort, an efficient approach for test sequence generation and prioritization is proposed via river formation dynamics. Firstly, it selects the program for test sequence generation, followed by the calculation of adjacency matrix and cyclometic complexity. Then after, the basic scheme of RFD is extended for optimal test sequence generation. Independent paths are then obtained from the last test sequence generated and the paths are prioritized based on the strength value of their respective edges.

1.5. Organization of Thesis

This thesis is structured into 6 chapters followed by references and appendix.

Chapter 1 presents the research problem, research objectives, justifying the need for carrying out the research work and outlines the main contributions arising from the work undertaken.

Chapter 2 provides the essential background and context for this thesis.

Chapter 3 provides the details of the SBSE

Chapter 4 provides the details of the proposed approach employed.

Chapter 5 describes the experimental results obtained for the programs undertaken. It also presents the comparative analysis with other existing techniques.

Chapter 6 presents the conclusions based on the contributions made by this thesis and provides insight in to the future work.

1.6. Chapter Summary

This chapter has laid the foundations for this thesis. It briefly introduced the research problem, research objectives and the proposed solution framework. A justification for the research problem is outlined, together with an explanation of the research methodology used. The next chapter examines the pertinent literature most relevant to this research.

Chapter 2 Literature Review

The focus of this chapter is to review the prominent and relevant research that has been undertaken related to the proposed approach. Section 2.1 discusses the history of SBSE. This is followed by section 2.2 which elaborates the application areas of SBSE. The Issues and challenges are presented in section 2.3. Section 2.4 briefs the history of SBST along with the details of various earlier attempts to test sequence generation. Finally section 2.5 discusses the various problem areas where RFD has been applied till now.

2.1. History of SBSE

Although the title SBSE (Search Based Software Engineering) was first introduced by Harman and Jones in 2001 [1] but originally an attempt for optimization to a software engineering problem dates back to 1976 and was reported by Miller and Spooner. They have applied it in software testing. With the passage of time , the interest of researchers towards SBSE increased rapidly and since then the significant work has been reported in the fields of requirement engineering[3,4,5], management[6,7,8], testing[9,11,13,25,27,29], maintenance[10,12], design[12,14,15] etc. thus, covering entire life cycle of software engineering as demonstrated in Fig. 2.1.

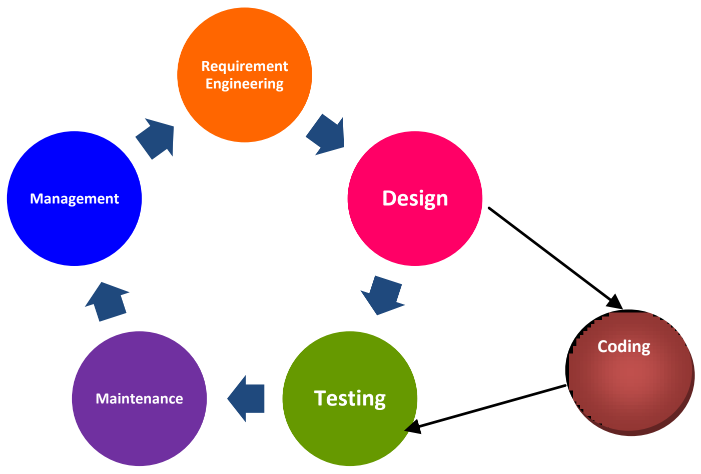


Fig. 2.1 SBSE Scope

A broad range of optimization algorithms have been applied in SBSE. The most commonly exercised are hill climbing, simulated annealing, tabu search, genetic algorithm and genetic programming.

Proportional rate of published papers in different domains of software engineering and yearly publication graph of SBSE has been represented in Fig. 2.2 and Fig. 2.3 respectively. It can be observed from the figure that testing covers 54%, maintenance covers 11%, design covers 10%, management covers 10%, verification covers 4%, requirement covers 3%, general issues covers 4% and other fields covers 4% of the SBSE's work. Thus, so far the major exploration has been done in software testing as reflected from Fig. 2.2. Fig. 2.3 reflects that there is a sharp increase in the SBSE research after 2001. 2010 is the year in which maximum SBSE's publications have been reported.

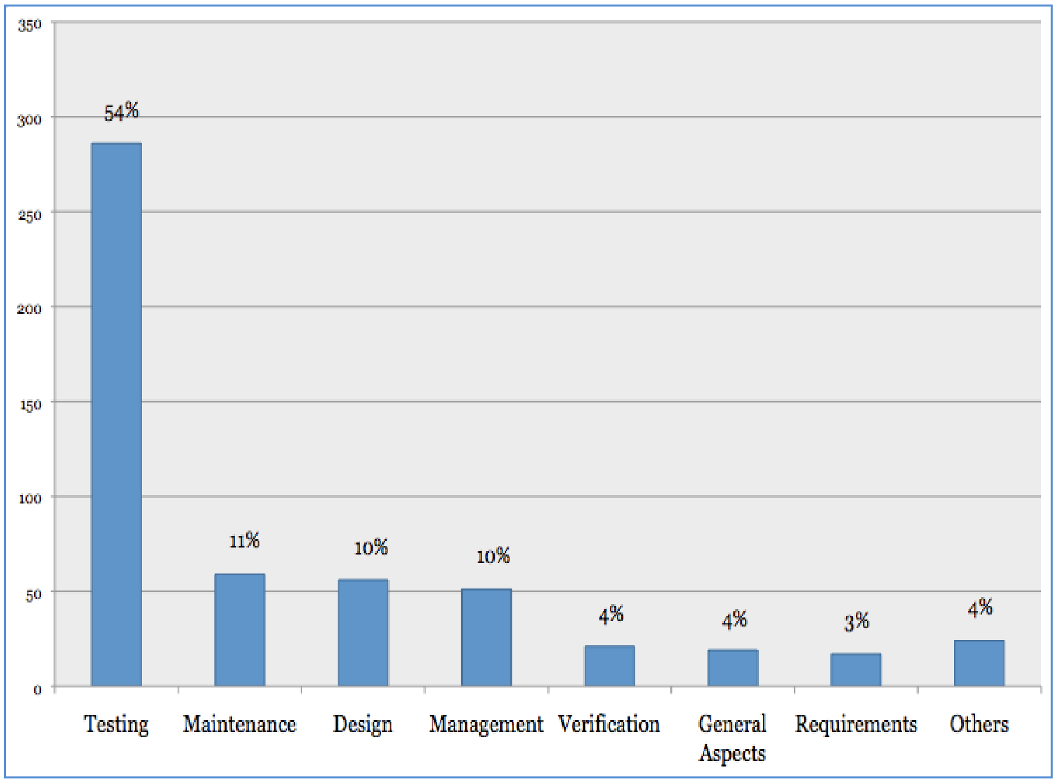


Fig. 2.2 SBSE Application Domain Publication Rate (Source: SBSE Repository [23])

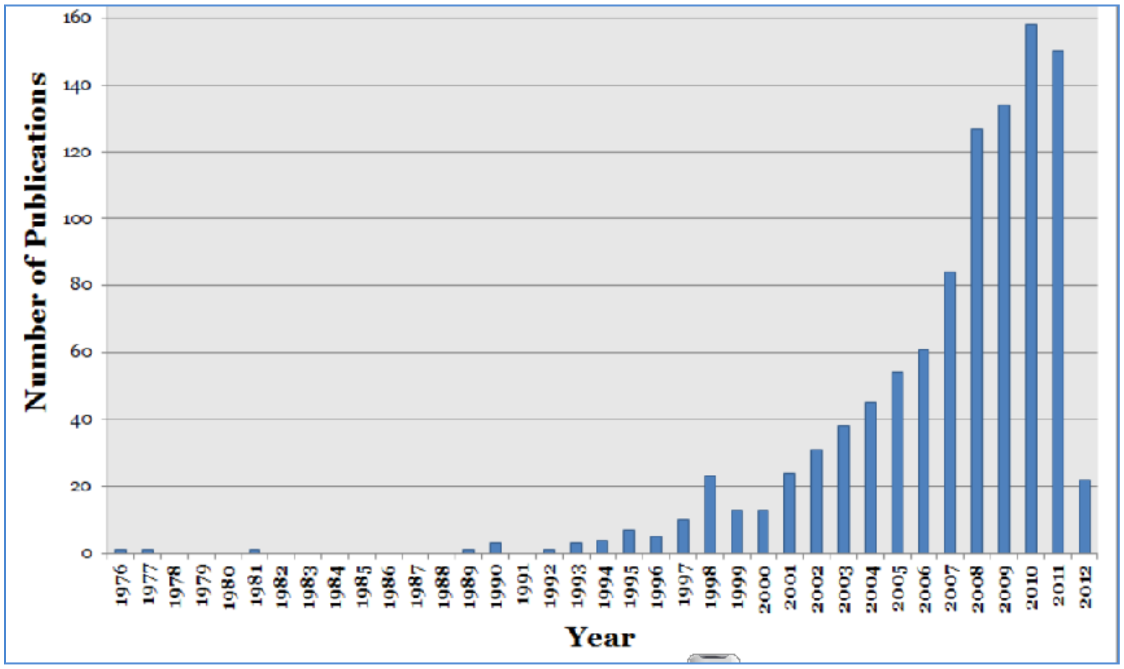


Fig. 2.3 Yearly SBSE Publication Rate (Source : SBSE Repository [23])

2.2. Applications of SBSE

SBSE has a very broad application spectrum. It has been successfully applied to solve the problems related to almost all phases of software engineering. Following are the some of the main application areas where SBSE has excelled.

1) *Software Project Cost Estimation* : Many Search based algorithms like hill climbing, genetic algorithm etc. have been applied for predicting the software project cost and the result obtained are accurate and close to reality.
2) *Project Planning*: Search based techniques have been successfully applied to handle the problem of staff allocation to work packages. Different researchers have tried with applying different metaheuristic algorithms like simulated annealing, genetic algorithm, hill climbing etc. taking either real world data or synthetic data into consideration. The results obtained are quite satisfactory and far better than other classical approaches.
3) *Requirement Engineering:* Numerous search based techniques like genetic algorithm, simulated annealing, greedy algorithm etc have been applied to obtain an optimal set of requirement that establishes equilibrium between customer expectations, resources in hand and requirement interdependencies. Moreover some researchers have also attempted

to prioritize the set of requirements to determine the order in which customer requirements need to be satisfied.

4) *Design:* Much work has been reported regarding establishing a balance between cohesion and coupling. But more formally, this work comes under the title of reverse engineering. A Little work has been witnessed at design stage. Lutz [14] considered the problem of hierarchical disintegration of software. He used theoretical fact fitness and awarded more fitness value to hierarchies that present the software design in more understandable way. Not much work has been done in this area but there is a great scope in using the information theory as a basis for constructing a fitness function, because in SE, theoretical information is found in abundance.
5) *Test Data Generation*: This activity comes under the area of testing. Testing is the most prominent area where search based techniques has been applied and showcased remarkable improvements as compared to other classical approaches. For this task, the most commonly used algorithms are genetic algorithm, tabu search and ant colony optimization.
6) *Test case Selection and Prioritization*: Many researchers attempted for test case selection and prioritization problem employing different search based techniques like hill climbing, greedy algorithm, genetic algorithm etc. and have obtained satisfactory results.

2.3. Issues and Challenges in SBSE

Although the success rate of SBSE is very high but still there are some challenges that need to be resolved. Following are some of the major challenges faced in SBSE.

1) *Deciding the right stopping rule:* Two stopping rules have been used till now to terminate the search

   a) By Budget Constraint

   b) By Some Stipulated Time

   These rules are not sufficient as sometimes they give poor results. However in case of GA (since it is population based), third rule arises, which says, stop the search, when all individuals in the population have similar chromosome. But this leads to a problem of how to assess the similarity of individuals. This limitation is splattered as a challenge for SBSE researchers to come out with such metrics.

2) *Selection of Appropriate Optimization Algorithm for a Specific SBSE Problem:* It has been observed that the researchers are randomly choosing the optimization algorithm and comparing their work with random search as standard. This approach is acceptable for new domains of SE which has not yet been explored by SBSE. It is a big challenge to characterize the complexity of SE problem and to analyze them in comparison with that search algorithm which already produced good results for the similar problem space. This characterization will help in selection of suitable search algorithm for the problem in hand, so that already existing results can be improved.

3) *Achieving Human Competitive Results:* One of the challenge of SBSE is that the outcomes produced by automatic computation are not human competitive. But there are fields likes test data generation and modularization, where it would be possible to beat the human competitors. Researchers believe that growing interest in this area will definitely achieve human competitive results in near future.

4) *Choosing Suitable Parameter Setting*: Sometimes the fruitful deployment of search algorithm on a given problem bank on discovery of relevant/desired parameter settings, but the charge of finding such parameters setting is a big challenge as the cost of finding them can be huge.

After analyzing the field of SBSE deeply, we realized that there is a major Scope in SBST. With passage of time more and more researchers started focusing on automation of different testing activities using search based techniques. The next section will provide the detail of what all has been done in SBST and where further improvement is required.

2.4. History of SBST

Software testing is a very vast, expensive and time consuming process. Automating the software test activities in an optimized manner is a major concern of researchers working in this domain, thus ensuring completeness and correctness of the testing process with optimized efforts. Many metaheuristic approaches have been used for solving the different problems of software engineering. The application of these metaheuristic techniques in the field of software engineering is better known by the term Search Based Software Engineering (SBSE) [1] and when applied in the field of software testing, better known by the phrase called Search Based Software Testing. Software testing is benefitted the most by these search techniques as stated earlier in chapter 1.

The very first attempt of applying search based techniques in software testing dates back to 1976. It was reported by David Miller and Spooner, who had applied genetic algorithm for the automatic test data generation for floating types input. With passage of time, more and more researchers started focusing on SBST by exploring many other search based algorithms like random search, local search (hill climbing, simulated annealing and tabu search), genetic programming, ant colony optimization, particle swarm optimization, etc.

A huge rise has been witnessed in search based software testing in recent years. Table 2-1 provides the details of year wise publications in SBST and the corresponding Fig. 2.4 provides an insight in to the increasing interest of researchers towards SBST.

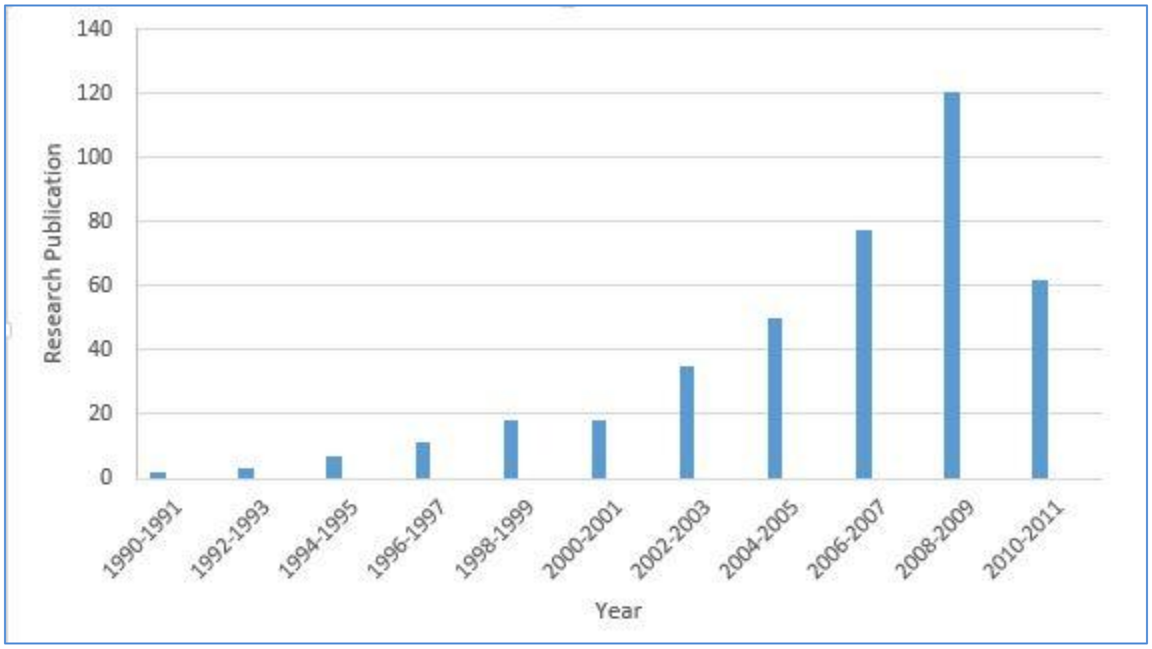
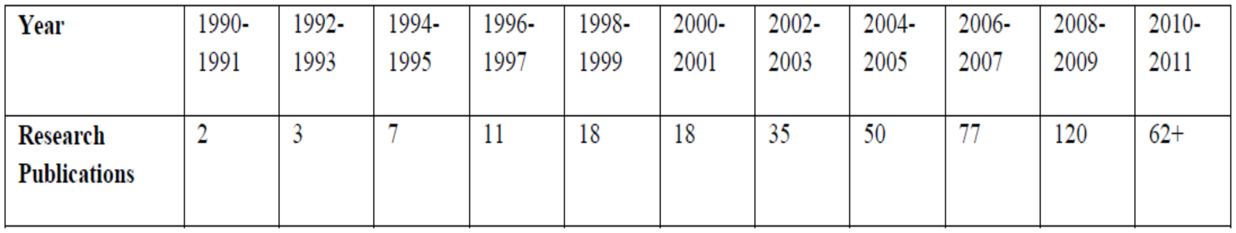


Fig. 2.4 Publication Rate in SBST

Table 2-1 Year Wise Publication in SBST [25]



Year	1990-1991	1992-1993	1994-1995	1996-1997	1998-1999	2000-2001	2002-2003	2004-2005	2006-2007	2008-2009	2010-2011
Research Publications	2	3	7	11	18	18	35	50	77	120	62+

In the literature, the use of ACO in testing has been witnessed the most as compared to other algorithms and test data generation seems the prime activity, catching the researcher's attention these days in software testing. For example, in [41], Ghiduk uses ACO based approach for data flow testing and in [42], Li and Peng Lam uses ACO based approach for generating test sequences from a state diagram for the software under test. On the other hand, C. Peng Lam [36], Fu Bo[37] and Praveen Ranjan Srivastava et al. [39] applied ant colony optimization for test data generation. In [38], Praveen Ranjan Srivastava et al. used a new algorithm called intelligent water drop for test data generation. Jin-Cherng Lin & Pu-Lin Yeh [34] and Dr.Vrlur [35] proposed two different approaches for test data generation for path testing using genetic algorithm. Thus, the automatic test data generation is the major testing activity, the researchers are focusing on these days using various SBSE techniques. However, optimal test sequence generation has received less importance in the existing literature. But one needs to understand that testing efforts can be minimized by generating optimal test sequences.

Moving in this direction, in 2003, Diaz et al. [48] attempted to produce test sequences having high branch coverage as their objective. They applied tabu search on the control flow graph of the software under test. It is one of the famous search technique which is based on the principle of searching and evaluating the next K neighbor around the current solution and learning the best one. They got success in obtaining test sequences having high branch coverage but does not ensures complete path coverage, multiple condition coverage and loop coverage.

In 2008, Doungsa-ard et al. [49] implemented a genetic algorithm based approach for optimal test sequence generation. They succeeded in maximizing the software coverage but still lacked in complete edge/transition coverage.

In 2009, Srivastava et al. [40] came up with an ACO based approach to produce optimal path suit. They worked on the CFG of the software under test. They basically tried to imitate the ant's behavior i.e. how they discover their path from source to destination by depositing pheromone on the path they visit. The selection of paths in ACO depends upon stochastic probability theory and the probability calculation is done on the basis of pheromone level on each edge as well as some heuristic information available on each edge. The path having the highest probability is chosen by the ant. They also tried to prioritize the paths based on the pheromone value and heuristic information available on it. However, they succeeded in full transition/edge coverage but faced redundancy in edge traversal.

In 2012, Srivastava et al. [50] presented an intelligent cuckoo search based approach for generating and prioritizing optimal test sequences. They tried to cover up all loopholes in the previous work by generating optimized test sequences with full edge coverage. Cuckoo search make use of very few parameters and promises that only better egg is carried on to the next iteration, which actually avoids getting stuck in local optima. They tested their approach on real time examples and compared their approach with other search based technique like ACO and GA and discovered that the results obtained are more optimized in terms of coverage and edge redundancy. But node redundancy is still a challenge to resolve.

In 2012, Srivastava et al. [51] extended the firefly algorithm for generating optimized test paths with full coverage. They applied this algorithm on state transition diagram and control flow graph. Their simulation results showcased that the test paths obtained are optimal and also below the number of total independent paths. On comparison with ACO, they discovered that their approach is better and exhibits very small redundancy in path coverage, especially when the numbers of nodes are large and cyclometic complexity is high.

The following table 2-2 depicts the summary of some previous prominent attempts to test sequence generation.

Table 2-2 Summary of Some Previous Attempts to Test Sequence Generation

Author	Algorithms	Inputs	Outcomes	limitations
Diaz et al. (2003) [48]	Tabu Search	CFG	Achieved high branch coverage	Does not ensure complete path coverage & loop coverage.
Doungsa-ard et al. (2008) [49]	Genetic Algorithm	State Transition Diagram	Maximized software coverage	Lacked in complete edge/ transition coverage
Srivastava et al. (2009) [40]	ACO	CFG	Achieved full transition coverage	Faced redundancy in edge traversal
Srivastava et al. (2012) [50]	Cuckoo Search	CFG	Achieved full transition coverage with no edge redundancy	Lacked in node redundancy
Srivastava et al. (2012) [51]	Firefly Algorithm	CFG & State Transition Diagram	Achieved full path coverage	Obtained small redundancy specially when number of nodes are large and cyclometic complexity is high

Research Gaps

After performing literature review the following research gaps are identified.

- Many researchers working in the Domain of SBST attempted for automatic test data generation, but the automatic generation of test data for string and pointer data types is still to be taken care off.

- Although a few techniques for test sequence generation exist, each having their own merits and demerits, but none of them is ideal in terms of complete path coverage and zero redundancy. It’s still remains a challenging task.
- Asymmetry in availability of software and tools, as many of the tools are not open source, limiting the researcher’s capabilities for further optimization.
- Lacked in standardized language for modeling CFG as different programmers can have different CFG for one particular program, resulting in different outputs.
- Coding phase of software development life cycle is yet to be explored in SBSE.

We considered the problem of optimal test sequence generation and proposed an approach for it by bringing a novel swarm intelligence algorithm called RFD into the picture. In the next section we are providing the details of various complex problems that are resolved by RFD till now.

2.5. River Formation Dynamics in Action

RFD [43] is one of the upcoming heuristic optimization technique from the swarm intelligence group, based on a nature’s phenomena of how water drops transform into river and river in turn in to sea by selecting the most favorable path based on the altitudes of the places through which it flow. It constitutes of two main processes of eroding the ground and depositing the soil along the path it traverse. When a drop move, it always move from high altitude place, let say A to a low altitude place , let say B , eroding the place A and depositing at place B , thereby decreasing the altitude of place A and increasing the altitude of place B. Higher the downward slope between these two places, more will be the erosion. The entire process is propagated in the same way, till the drop finds the shortest path to sea. New drops are placed at the origin to construct paths and to strengthen the erosion of the propitious paths. After few iterations, favorable paths are found between origin and the destination Fig. 2.5 represents the basic RFD scheme.

```
initializeDrops()
initializeNodes()
while (not allDropsFollowTheSamePath()) and (not otherEndingCondition())
   moveDrops()
   erodePaths()
   depositSediments()
   analyzePaths()
end while
```

Fig. 2.5 River Formation Dynamics Algorithm

In [43], they applied RFD to solve travelling salesman problem taking 20 and 30 nodes in a randomly generated graph and compare their results with ACO. They observed that initially ACO provide the better solution in short duration where as the results obtained through RFD are not good initially, but after some time, RFD surpasses ACO in terms of quality of solution obtained. Thus convergence rate of drops is slower as compared to convergence rate of ants, but as far as quality is concerned, RFD is much better than ACO.

Redlarski et al. [52] have used RFD in mobile Robot navigation i.e. how a mobile robot searches the shortest path to a given destination in an environment with obstacle. They made certain changes in the basic RFD algorithm to suit the problem in hand. They conducted several test on different landscape using RFD and measured the computation time and compared it with Dijkstra's algorithm and observed that RFD is faster than it.

In [45], Rabanal et al. applied RFD to solve the problem of finding minimum spanning tree in a variable cost graph. They considered variable cost graph in which some cost function and transformation function assigned to each edge. They took three randomly generated graphs with hundred, two hundred and three hundred nodes and applied RFD and ACO to find the minimum spanning tree. RFD's results were found to be 4 to 8 % better than ACO's results. Thus, RFD provides deeper exploration of the graph.

Venkateswarlu and Rao [53] extended RFD for carrying out data aggregation in wireless sensor network (WSN). They incorporated hop count distance between the node and the base station and residual energy as the node's parameter. Their ultimate goal is to increase the age of the network by saving the energy. They showed a relation between RFD and data aggregation process. As in RFD, the drops tend to find the way to sea, similarly, here sensor nodes, responsible for collecting the data about their environment; tend to send their data to the base station. They search for the shortest path to the base station using hop count and the residual energy. The entire procedure is divided in to two stages. Firstly, in the initialization stage, the sensor nodes are deployed and hop count is calculated by sending Request/Reply packets. In the second stage, aggregate node is selected first employing RFD mechanism and then each sensor node sends their data to the base station using aggregate nodes. The proposed approach is compared with other existing state of the art techniques. It is observed that RFD's complexity of maintenance is less than others.

Dholakiya et al. [54] employed RFD for location area management in GSM. Their goal is to cut down the location management cost. They assumed that the number of location areas corresponds to the number of drops, the number of cells corresponds to the number of nodes and altitude of nodes is replaced with cell's call-to-mobility ratio and distance between two nodes is replaced with cost of merging two cells. With implemented their approach on a cellular network with 7, 10 and 12 cells and compared their results with Location Area (LA) scheme and Always Update (AU) scheme. It has been observed that as the number of cells in an area increases, the RFD provides better results as compared to LA and AU schemes. Thus the proposed approach employing RFD cut downs the location management cost significantly.

The following table 2-3 depicts the summary of RFD in action.

Table 2-3 RFD in Action

Author	Year	Problem	Outcomes
P. Rabanal, I. Rodríguez, and F. Rubio [43]	2007	Travelling Salesman Problem	Quality of solution obtained is better than ACO
P. Rabanal, I. Rodríguez, and F. Rubio [45]	2008	Finding Minimum Spanning Tree	Results are 4 to 8% better than ACO's results
Redlarski et al.[52]	2013	Mobile Robot Navigation	RFD is better than Dijkstra's algorithm in finding the shortest path to a given destination
Venkateswarlu and Rao [53]	2015	Data Aggregation in Wireless Sensor Network	Increased network life by saving energy
Dholakiya et al. [54]	2015	Location Management in GSM	Obtained reduction in location management cost as compared to other state of the art techniques

Chapter 3 Search Based Software Engineering

This chapter will provide a deep insight into the field of SBSE. Section 3.1 gives brief introduction of SBSE. Section 3.2 presents the elements of SBSE. Section 3.2 provides a gist of most common optimization algorithms used in SBSE. Section 3.3 presents the SWOT analysis of SBSE beneficial to address and urge the researchers to make significant improvements in this domain. This is followed by section 3.4 explaining about search based software testing. Section 3.5 describes the types of testing explored in SBST. Finally, Section 3.6 highlights the various testing activities in SBST.

3.1. SBSE: An Introduction

Search Based Software Engineering basically involves amalgamation of software engineering, operation research and metaheuristic techniques. It is specially excelled in providing an ideally balanced solution to a SE problem involving several competing goals. It caters to the problem of SE and provides optimal solution to give better prospects over tools and techniques required to grow a productive, proficient and cohesive approach.

Different phases of software engineering are complex in nature and consist of numerous problems. Following are some of the problems of SE:

- What should be the optimal set of requirements which establishes equilibrium between software development budget and customer satisfaction?
- What should be the best way of distribution of resources to a software development project?
- What should be the optimum alignment of test cases (test case prioritization) to be executed for regression testing?
- What should be the minimum set of test cases that will award complete branch coverage?

Although, we have many classical approaches to solve these problems like linear programming, dynamic programming, but these approaches do not work well with NP-hard problems as they cover maximum range of SE problems. However, it has been found by many researchers that SBSE often gives satisfactory results when used on such problems. They may not be able to give a global optimal solution but can provide a list of near optimal solutions. It provides an efficient way to automate software engineering tasks by applying meta-heuristic search algorithms. For applying any search algorithm to a problem, firstly, we reformulate the given problem as a search problem and then select the appropriate fitness function for guiding the search. Since all the above mentioned problems belong to different domain of software engineering and do not have any thing in common but from SBSE point of view all are essentially the optimization problems. SBSE approach has especially excelled where it is required to obtain an ideal balance between competing factors like completion of goals vs. money spend. In case of such problems there can’t be a single ideal solution rather there can be many near optimal solutions that contain a perfect mix of competing factors.

3.2. Elements of SBSE

In order to apply SBSE in any domain, following four key elements [1, 2] must be satisfied.

1) *Large Search Space*: SBSE problem must have a big search space. If small, then no need to apply this.

2) *Low Computational Complexity*: Fitness function must be easy and fast to compute, so that overall complexity of the algorithm should be low.

3) *Approximate Continuity*: There must be a well defined objective for every SBSE problem that needs to be optimized.

4) *No Known Optimal Solution*: No need to apply SBSE technique, if an ideal solution to a problem is discovered previously.

3.3. Most Common Optimization Algorithms used in SBSE

A wide range of optimization algorithms are used in SBSE. But most commonly used are hill climbing, simulated annealing, genetic algorithm, genetic programming, tabu search. Apart from these, researchers are using nature inspired algorithms like ant colony optimization, particle swarm optimization, bat algorithm, etc. now days. Now we will provide the details of some of the most common algorithms used in SBSE.

- *Hill Climbing*: It begins with initially randomly chosen climb point. In every iteration the fitness of the current solution is compared with a set of neighbors. If fitness of any neighbor exceeds the fitness of the current solution, then it climbs to that neighbor and starts the same process again. If no neighbor has the fitness value more than the current solution, then it means maxima has been found, so just stop the search. The maxima obtained may be a local maxima (as can be seen in Fig. 3.2) and not the global maxima. Therefore one can restart the search staring from the new climb point (as can be seen in Fig. 3.3). Despite of local maxima, hill climbing is relatively fast and easy to implement and suits to the most of the problems found in software engineering. Fig. 3.1 shows the pseudo code for the hill climbing

```
Select a starting solution $s \in S$
Repeat
    Select $s' \in N(s)$ such that $fit(s') > fit(s)$ according to ascent strategy
    $s \leftarrow s'$
Until $fit(s) \geq fit(s'), \forall s' \in N(s)$
```

Fig. 3.1 Pseudo Code for Hill Climbing

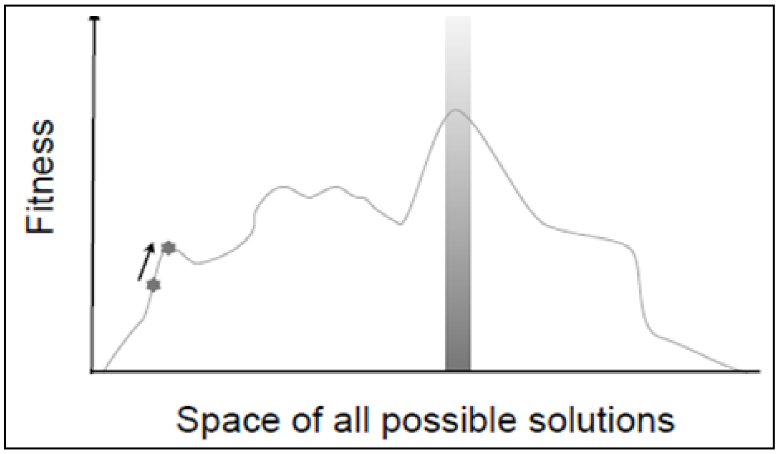


Fig. 3.2 Move to a Local Optimum

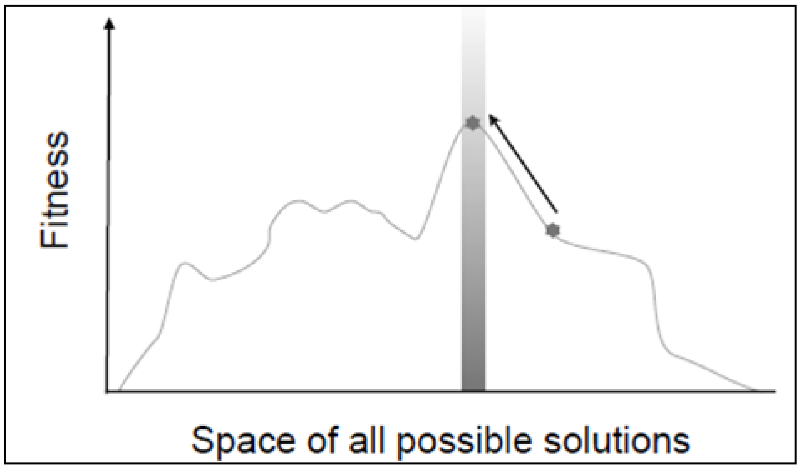


Fig. 3.3 A Restart from a New Climb Point to a Global Optimum

- *Simulated Annealing*: Simulated Annealing is also like hill climbing, but unlike in hill climbing, it can make a move to a poor solution in the beginning to escape local maxima. The whole algorithm is controlled by a parameter called temperature. Initially the temperature is high, as result of it, the search can take a poor move that means it can select a solution with less fitness value as compared to the current solution, just in hope of achieving global optimum (as can be seen in Fig. 3.4). As temperature reduces, the chance of making a poor move also reduces. Eventually when freezing point is reached, it simply starts behaving like hill climbing i.e. no more poor moves. Fig. 3.5 shows the pseudo code for the simulated annealing.

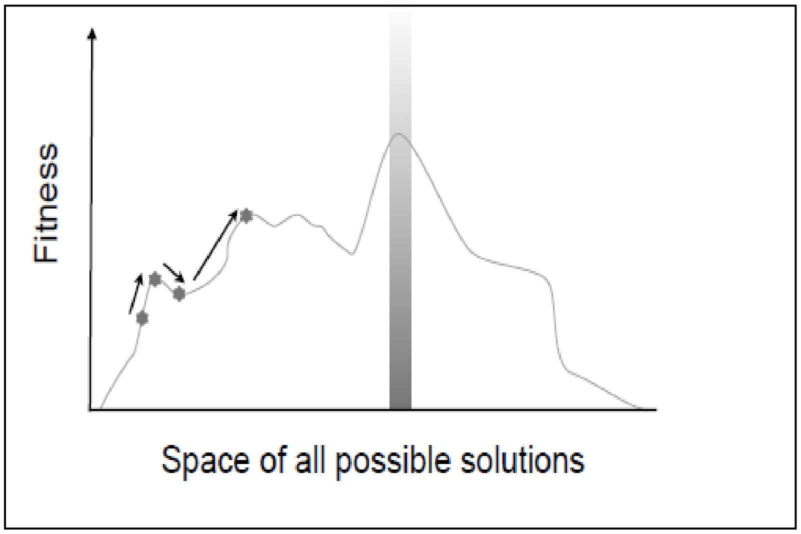


Fig. 3.4 Moves in Simulated Annealing

- *Genetic Algorithm*: Hill climbing and simulated annealing are local searches as they deal with one candidate solution at a time and makes moves on the basis of fitness of its neighbors. Genetic algorithm are global searches as many points are explored simultaneously (as can be seen in Fig. 3.7), thus offering more robustness to local optima. Fig. 3.6 shows the pseudo code for the genetic algorithm. First set of candidate solutions i.e. the population is generally taken randomly. One can also seed the first population based on the problem in hand. After this, the fitness of all individuals is calculated and the some individuals are selected as per the selection mechanism, to go forward in the next stage of cross over followed by mutation.

```
Select a starting solution $s \in S$
Select an initial temperature $t > 0$
Repeat
    $it \leftarrow 0$
    Repeat
        Select $s' \in N(s)$ at random
        $\Delta e \leftarrow fit(s) - fit(s')$
        If $\Delta e < 0$
            $s \leftarrow s'$
        Else
            Generate random number $r$, $0 \leq r < 1$
            If $r < e^{-\frac{\delta}{t}}$ Then $s \leftarrow s'$
        End If
        $it \leftarrow it + 1$
    Until $it = num\_solns$
    Decrease $t$ according to cooling schedule
Until Stopping Condition Reached
```

Fig. 3.5 Pseudo code for Simulated Annealing [24]

```
Set generation number, m:= 0
Choose the initial population of candidate solutions, P(0)
Evaluate the fitness for each individual of P(0), F(Pi(0))
loop
•   Recombine: P(m) := R(P(m))
•   Mutate : P(m) := M(P(m))
•   Evaluate: F(P(m))
•   Select: P(m+ 1) := S(P(m))
•   m := m + 1
•   exit when goal or stopping condition is satisfied
End loop;
```

Fig. 3.6 Pseudo Code for Genetic Algorithm [28]

There are many selection mechanisms like Roulette wheel selection, Rank selection, steady state selection, elitism, etc. One can choose specific selection method based on the problem in hand. After selection, cross over operation is performed which involves breaking and merging of elements of individuals to form two new offspring individuals. A variety of cross over operators can be used like one-point cross over, multi-point cross over and uniform cross over. After this the elements of newly generated individuals are mutated with the aim of redirecting the search in a new location of the search space. When binary representation is used for each chromosome, then mutation simply involves flipping if bits. After mutation, new generation of population is ready and the whole process of selection, cross over and mutation is repeated. This loop continues until either global solution is obtained or resources are exhausted.

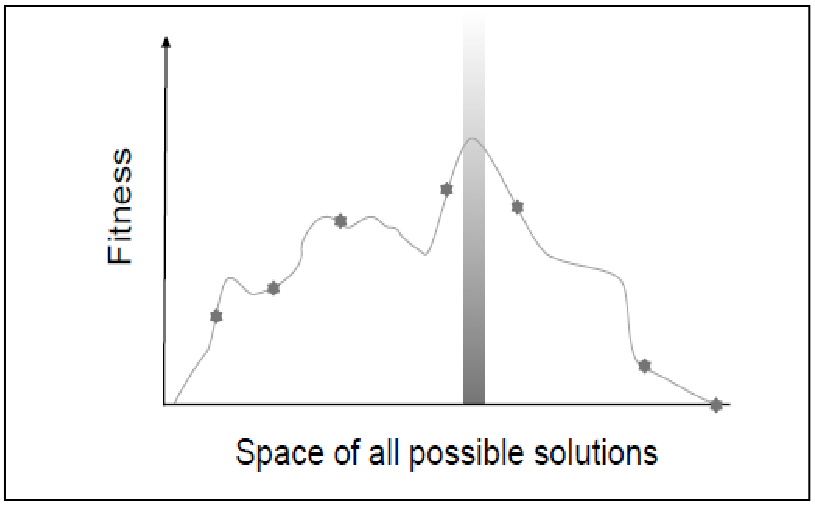


Fig. 3.7 Moves in Genetic Algorithm

3.4. SWOT Analysis of SBSE

SWOT analysis of SBSE represents a layout/framework to assess the internal and external elements that can have an impact on the vitality of the project, place or person. Its four factors are:

1) Strengths: Internal factors and resources that accelerates the success of SBSE.
2) Weaknesses: Internal factors and resources that give rise to the failure of SBSE.

3) Opportunities: External factors that can boost the performance rate of SBSE.

4) Threats: External factors that could imperil the SBSE.

Strengths

Strengths of SBSE are its vigorous characteristics that make it more advantageous over other approaches. These are as follows:

1) *Broad Domain*: SBSE has a very broad application domain. It has been implemented almost on all phases of software development, thus covering requirement analysis, design, testing, project management, and refactoring. This approach views every problem related to above fields as a search based optimization problem and remarkable improvements have been observed as compared to existing techniques.

2) *Readymade Fitness Function:* In order to apply the search based technique to a software engineering problem, one of the essential element required is the exigency of fitness function to lead the search. Many problems in software engineering have a wide variety of software matrices associated with them which can be used as a fitness function .No extra effort required to construct the fitness function. Undeviatingly, any one of the associated software matrices can be used as fitness function.

3) *Re-Coalescence*: It bridges the gap between unrelated areas of software engineering for example, regression testing and requirement analysis are two unrelated techniques of software engineering. But both can be seen as similar problem species with respect to SBSE. The task of selection and prioritization works on similar lines for requirements as well as for test cases of regression testing. Although, the purpose of both are different, regression testing is targeted towards selecting and prioritizing test cases to have code coverage to attain high fitness where as requirement analysis is targeted towards selecting and prioritizing requirements to cover customer expectation. Such relationships give rise to new dimensions for fusion of unrelated research areas.

4) *Good Provider for Solution to NP-Hard Problems*: For the problems which are characterizes by a set of competing parameters and there exist a tradeoff between them, SBSE techniques are found to be ideal for finding out the near optimal solution

representing an ideal balance between the competing factors. For example, the expenditure vs. the realization of goals.

5) *Fast and Efficient*: Most of the SBSE problems are favored by hill climbing or simulated annealing. Since they are fast and easy to implement and thus are very efficient. Sometimes they may lead to a local optima, but can always be restarted multiple times. For problems that require quick solution, HC is a good choice as it provides a palpation of landscape structure. By applying multi-objective search algorithms we can obtain many good solutions that show potential tradeoffs between competing targets specially when time or resources are short.

6) *Malleability through Parallelism***:** Malleability is a big milestone in software engineering. But it can be achieved because of the implicit parallel nature of SBSE techniques. For example, hill climbing can be done in parallel, each commencing with a new climb point. Genetic Algorithms can also be operated in parallel, since it's a population based technique so fitness for each candidate or individual can be calculated in parallel. Therefore, this inherent parallel nature of SBSE techniques unveils a solution to address the issue of SE malleability.

7) *Feedback and Insight:* One of the root causes of failure in software engineering is poor understanding of customer requirements, making false assumptions, thus leading to improper specification. SBSE can handle this problem as the search is guided by automatic fitness function in comparison to human based search, leads to exciting results with no deviation.

Weaknesses

Weaknesses of SBSE are its fragile characteristics that place it at a competitive disadvantage. These are as follows:

1) *Unfit Stopping Rule*: Two stopping rules have been used till now to terminate the search

   a) By Budget Constraint

   b) By Some Stipulated Time

These rules are not sufficient as sometimes they give poor results. However in case of GA (since it is population based), third rule arises, which says, stop the search, when all individuals in the population have similar chromosome. But this leads to a problem of how to assess the similarity of individuals. This limitation is splattered as a challenge for SBSE researchers to come out with such metrics. Thus we are lacking with the right stopping rule.

2) *Opting Appropriate Optimization Algorithm for a Specific SBSE Problem:* It has been observed that the researchers are randomly choosing the optimization algorithm and comparing their work with random search as standard. This approach is acceptable for new domains of SE which has not yet been explored by SBSE. But the necessity is to characterize the complexity of SE problem and comparatively analyze them with that search algorithm which already produced good results for the similar problem space. This characterization will help in selection of suitable search algorithm for the problem in hand, so that already existing results can be improved.

3) *Achieving Human Competitive Results:* One of the weaknesses of SBSE is that the outcomes produced by automatic computation are not human competitive. But there are fields likes test data generation and modularization, where it would be possible to beat the human competitors. Researchers believe that growing interest in this area will definitely achieve human competitive results in near future.

4) *Choosing Suitable Parameter Setting*: Sometimes the fruitful deployment of search algorithm on a given problem bank on discovery of relevant/desired parameter settings, but the charge of finding such parameters can be huge. Sound experimental design can cut-down the cost of parameter tuning in some cases [15].

Opportunities

There are certain areas under software engineering or search based approaches that are not fully excavated but on scouting leads to astonishing future prospects [24]. Following are the favorable conditions/situations leading towards attainment of SBSE goals:

1) *Theoretical Fact Fitness*: This fitness function is formulated based upon theoretical facts and is first used by Lutz [14] for the problem of hierarchical disintegration of software. It

gives more fitness value to hierarchies that present the software design in more understandable way. Not much work has been done in this area but there is a great scope in using the information theory as a basis for constructing a fitness function, because in SE, theoretical information is found in abundance.

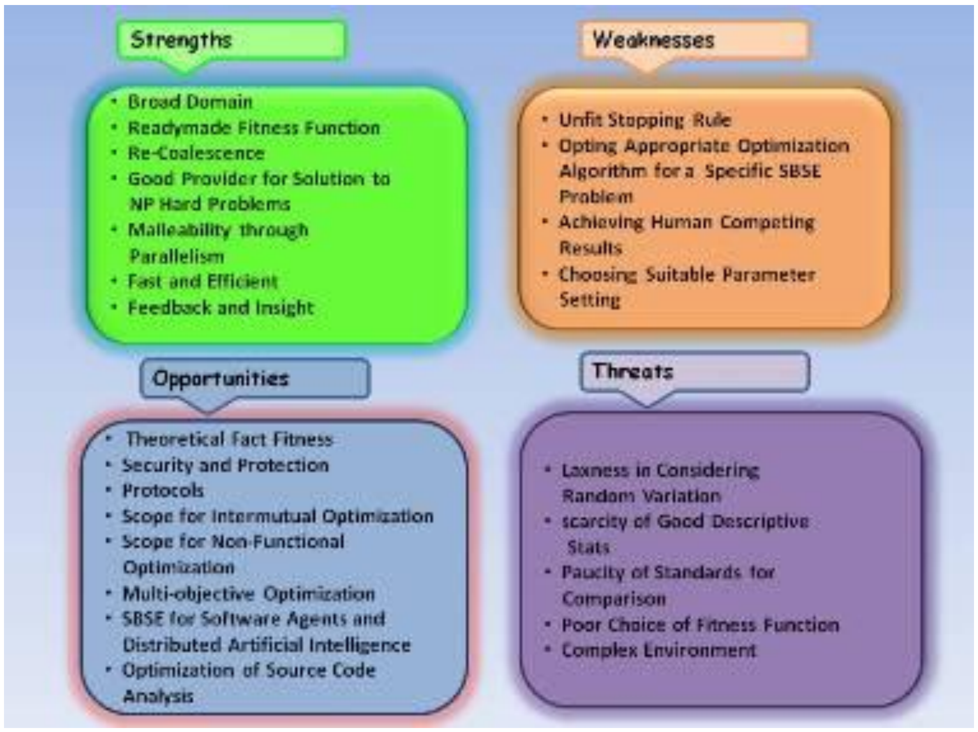

Strengths
Broad Domain
Readymade Fitness Function
Re-Coalescence
Good Provider for Solution to NP Hard Problems
Malleability through Parallelism
Fast and Efficient
Feedback and Insight
Weaknesses
Unfit Stopping Rule
Opting Appropriate Optimization Algorithm for a Specific SBSE Problem
Achieving Human Competing Results
Choosing Suitable Parameter Setting
Opportunities
Theoretical Fact Fitness
Security and Protection
Protocols
Scope for Intermutual Optimization
Scope for Non-Functional Optimization
Multi-objective Optimization
SBSE for Software Agents and Distributed Artificial Intelligence
Optimization of Source Code Analysis
Threats
Laxness in Considering Random Variation
scarcity of Good Descriptive Stats
Paucity of Standards for Comparison
Poor Choice of Fitness Function
Complex Environment

Fig. 3.8 SWOT Analysis of SBSE

2) *Security and Protection*: Although it is one of the very important areas of SBSE application, but limited by the search of a suitable fitness function to handle a security problem.

3) *Protocols*: Correctness of communication protocol can be checked in terms of its efficiency and security using SBSE techniques as first demonstrated by Alba and Troya [16]. They applied genetic algorithm for testing the accuracy of communication protocols. Later in 2000 and 2001, Clark and Jacob [17, 18] used genetic algorithm and simulated annealing for obtaining optimized tradeoffs between efficiency, security and cost of BAN (Burrows, Abadi, and Needham) protocols. In 2008, Ferreira et al [19] proposed particle swarm optimization technique to find network protocol errors in

concurrent system. The research could be extended in handling the problems related to this domain using SBSE techniques.

4) *Scope of SBSE in Distributed Artificial Intelligence and Software Agents*: The researchers believe that there is a great scope of SBSE application in the field of distributed artificial intelligence and software agents. A very few work has been done so far in this area. The nature of multi-agent system resembles very closely with the nature of SBSE problem. There is a population of individuals in an agent based system that communicates with each other, exchange information, leading to solve a common objective. The evolutionary algorithms like genetic algorithm, genetic programming seems a perfect choice for these agent based system. Latest work in this area [20] visualizes how agents can be tested using search based algorithms.

5) *Scope for Intermutual Optimization*: Intermutual optimization means where fitness function is based on human judgment. Outside the domain of SBSE application, huge amount of work on fitness function can be found based on human evaluation. In SE, we can make use of intermutual optimization in design problem where the constraints are subjective or not well defined. One of the possibilities is to make use of a search based technique to delve the implicit assumption in human judgment of solutions. The only problem with this approach is that, in the course of every iteration, human judgment for fitness evaluation has to be considered that can be time consuming and may create fatigue also.

6) *Scope for Non-functional Optimization*: There are some non-functional attributes like temperature, heat dissipation, power consumption that never get importance as software paradigms. But researchers have shown that these are emerging and augmented SE fields for non functional optimization over which SBSE techniques can be applied.

7) *Optimization of Source Code Analysis:* One of the important future prospects for SBSE is the optimization of source code analysis which has been an upcoming area but merely few papers [21, 22, 47] pop up to take a hand in source code based SBSE. With the execution of SBSE practices, it will be possible to explore for striking features and

produce a probabilistic source code analyses resulting in a very rigid optimization model of analysis.

8) *Multi-Objective Optimization***:** There are some areas where problems are multi-objective in nature , like in project planning prime concern is to complete the project well in time and with lesser cost. Optimization is required for both. This multi-objective optimization is called pareto optimality. Recent researches on SBSE have shifted to multi-objective codification from single objective codification, using pareto optimality optimization technique. There is a great scope for this new technique as many areas in SE have multi-objective problem domains.

Threats

Threats to SBSE are potential risks that are involved in design and execution phase. These threats may limit the ability to yield the reliable results or their generalization to a bigger population than the sample instances taken in the experiment. Some of the validity threats are as follows.

1) *Laxness in Considering Random Variation:* Search based algorithms are generally involved with generating random number. For example, the starting point of a hill climbing search is generally selected randomly and the first population for the genetic algorithm is usually generated randomly. Therefore, sometimes even iteratively running the algorithms can lead to a bad starting point selection and tend towards failure.

2) *Scarcity of Good Descriptive Stats***:** Since we have to run the algorithms several times, the data collected from them at the end must be accumulated and presented in a meaning full way to interpret the conclusions. For example, one can make use of function like standard deviation, min-max range for the outcome observed from the experiment. However, they are not of much significance and are actually lacking in good statistical measures and hypothesis that is based on the traits of data under evaluation.

3) *Paucity of Standards for Comparison:* In SBSE, whenever a new algorithm is applied to a particular problem instance, the result of the algorithm is compared with a benchmark. Most of the time random search is used as a baseline, but it’s an incorrect practice, rather the standards should be the representative of best solution known so far.

4) *Poor Choice of Fitness Function:* Many software matrices can be used as fitness function directly. Searching a solution from the search space is totally directed by the fitness function chosen as per the objective to meet. However, if not selected intelligently, can lead to a threat to its validity.

5) *Complex Environment:* The environment under which the software is developed is usually complex, because of technical and social issues involved. So when formulating a model for software related problem by simplifying some measures or making certain assumptions, one must account for its implication on practical application. Ignoring these factor can lead to validity threats.

Fig. 3.8 summarizes all the points related to strengths, weaknesses, opportunities and threats of SBSE. After analyzing the field of SBSE deeply, we realized that major work in SBSE has been done in the area of software testing. With passage of time more and more researchers started focusing on automating different testing activities using search based techniques. The next section will brief about SBST.

3.5. Search Based Software Testing

Search based software testing involves the application of various search based techniques like hill climbing, simulated annealing, tabu search, genetic algorithm , ant colony optimization etc. to address the testing related problems. To discover maximum faults in minimum time is the major objective in software testing. It is a significant and very crucial phase of software development life cycle to ensure software validity and quality. It is by far the most costly and exhaustive activity and almost accounts for 50% of the total effort [29]. Therefore, it becomes a major challenge for the researchers to optimize the software testing process.

Software testing is broadly classified into white box testing (also known as glass testing or structured testing) and black box testing (also known as opaque testing or functional testing). White box testing primarily emphasizes on internal structure of the program under test (PUT) where as black box testing ensures the functionality of the PUT i.e. it test the outputs based on the given set of inputs without having knowledge of internal flow/structure of the program. The most common white box testing techniques are:

- Control Flow/ Code Coverage Testing
- Data Flow Testing
- Mutation Testing

In general, code coverage testing consists of statement coverage, branch coverage, condition coverage, path coverage and function coverage.

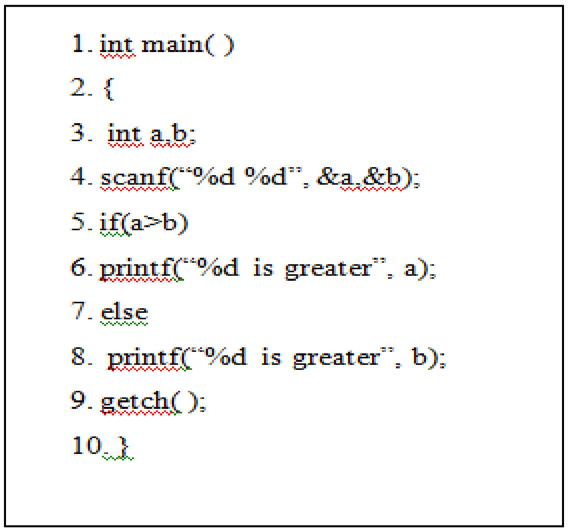
- Statement coverage ensures that each statement must be executed at least once.
- Branch coverage ensures that each edge in the control flow graph (CFG) must be traversed at least once.
- Decision coverage ensures that all edges emanating out from a decision point must be covered.
- Functional coverage ensures that all functions in the PUT must be executed at least once.
- Condition coverage ensures that for every individual condition, both true and false part must be executed at least once.
- Path coverage ensures that all independent paths in the CFG must be executed once.

In our research, we are focusing on code coverage testing and more specifically on path coverage which is explained next.

Path Testing

Path testing is a kind of structural testing which ensures that all paths in the CFG of the PUT must be executed at least once. McCabe cyclometic complexity provides the measure for the number of independent paths that can be found in a program. Path coverage ensures 100% statement coverage & branch coverage. Let us consider the following sample program written in C. The corresponding CFG will be shown in the Fig. 3.9. The cyclometic complexity (CC) is calculated as

$$\boxed{\text{CC} = \text{No. of predicate Nodes} + 1}$$



```
1. int main( )
2. {
3.  int a,b;
4. scanf("%d %d", &a,&b);
5. if(a>b)
6. printf("%d  is  greater", a);
7. else
8.  printf("%d  is  greater", b);
9. getch( );
10. }
```

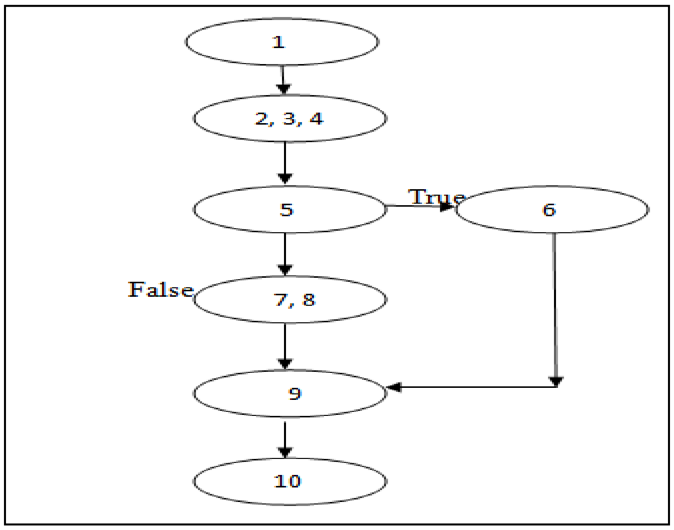


Fig. 3.9 Control Flow Graph for a Sample Program

Thus, there is only one decision node in the CFG, therefore CC= 2, which means, that there are only two independent paths in the CFG.

Path 1= 1-2-3-4-5-6-9-10

Path 2= 1-2-3-4-5-7-8-9-10

But generating control flow paths is one of the most demanding and challenging task in software testing [30,31]. Numerous methods are known for generating the control flow paths like brute force, symmetric matrix algorithm [32, 33], constraint based heuristics etc. But none of them is ideal in terms of providing an optimal solution for path generation. Infact, there can be large number of paths in a small program resulting in a large test suit to be executed. However, there can be many paths which are not really needed, as adding no value to testing. Thus, it becomes important to generate as well as select/prioritize the effective paths

3.6. Types of Testing Excavated in SBST

The various types of testing that have been successfully explored using search based techniques are

- Structural Testing
- Functional Testing
- Non Functional Testing
- Mutation Testing

- Integration Testing
- Robustness Testing
- Stress Testing

3.7. Testing Activities in SBST

Following are some of the testing activities which are explored the most using search based techniques.

- Discovery of Faults in Software
- Test Data Generation
- Test Case Generation
- Test Case Selection
- Test Path Generation
- Test Effort Estimation

Chapter 4 Proposed Work

This chapter illustrates the novel technique that constitutes the proposed approach to address the problem of test sequence generation and prioritization. Section 4.1 briefs the basic RFD scheme. Section 4.2 gives an overview of the research undertaken. Section 4.3 describes the tools and technology required for the proposed approach. Section 4.4 illustrates the process of test sequence generation and prioritization using RFD. Finally section 4.5 presents the detailed flowchart of the proposed approach.

4.1. Basics of River Formation Dynamics

RFD [43] was first introduced by Pablo Rabanal, Ismael Rodriguez, and Fernando Rubio in 2007. They launched it as a novel heuristic algorithm to crack complex /NP hard problems such as dynamic travelling sales man problem [44], minimum spanning tree [45] and optimal quality investment tree [46] and showed improvements over some existing heuristics methods such as ACO. It is one of the upcoming heuristic optimization technique from the swarm intelligence group, based on a nature's phenomena of how water drops transform into river and river in turn in to sea by selecting the most favorable path based on the altitudes of the places through which it flow. It constitutes of two main processes of eroding the ground and depositing the soil along the path it traverse. When a drop move, it always move from high altitude place, let say A to a low altitude place , let say B , eroding the place A and depositing at place B , thereby decreasing the altitude of place A and increasing the altitude of place B. Higher the downward slope between these two places, more will be the erosion. The entire process is propagated in the same way, till the drop finds the shortest path to sea. New drops are placed at the origin to construct paths and to strengthen the erosion of the propitious paths. After few iterations, favorable paths are found between origin and the destination. Fig. 4.1 is showing the flowchart of the basic RFD scheme.

Firstly , all nodes are initialized i.e. the altitude of all nodes are initialized to some equal positive value, while keeping the destination node(representing sea) at zero altitude, which never changes during the execution of the algorithm. After this, we initialize all the drops, i.e. they all are put at the origin. Then there is a loop which will be executed until the end condition met. Generally, the loop will end when all drops encounter the same path or one can also end the loop by putting a count on number of rounds/runs or by the execution time.

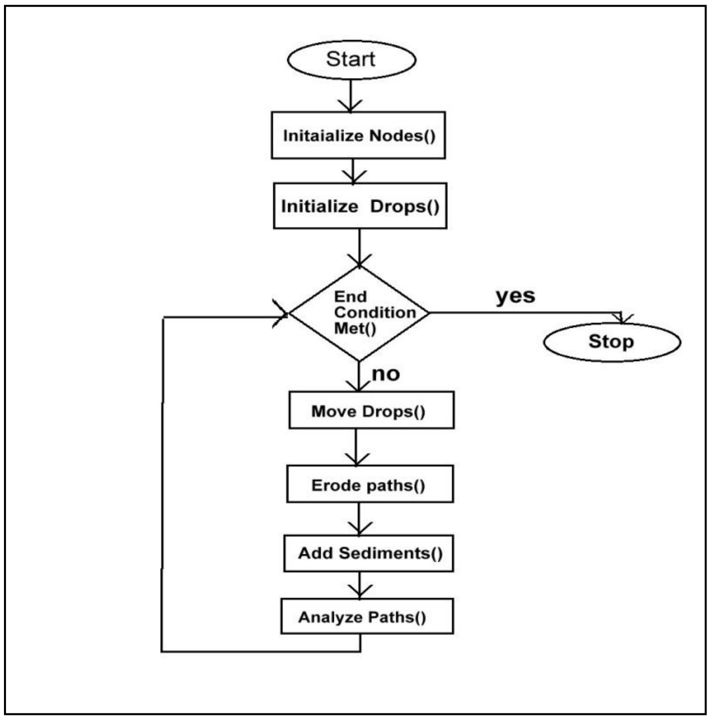


Fig. 4.1 Flow Chart of RFD Scheme

In the very first step of the loop, drops are allowed to move from one node to the other in a random way using the following rule.

$$P_M(i,j) = \left\{ \begin{array}{ll} \left(\text{gradient}(i, j) / \sum_{k \epsilon AM(i)} (\text{gradient}(i, k)\right) & \text{if } j \in A_M(i) \\ 0 & \text{if } j \notin A_M(i) \end{array} \right\}$$

The above rule says, drop M, which is at node ‘i’ will choose next node ‘j’, if probability is maximum on the edge ($i \rightarrow j$), where $A_M(i)$ is the set of adjacent nodes of node ‘i’ . Gradient between two nodes is the difference of their altitudes divided by the distance between them.

gradient(i,j)=(altitude(i)-altitude(j))/Distance(i, j)

Since, all nodes have same altitude in the beginning which leads to Σ gradient (i,k) to zero. Therefore, to deal with flat gradient, probability of drop M, moving on a flat edge is fixed to some non null value. Distance(i, j) denotes the cost/distance between node 'i' and node 'j'. In the next step of the loop body, erosion is carried out along the edge(i→j). As a result of it, the altitude of node 'i' is decreased, based on the gradient between node 'i' and node 'j'. Higher the downward slope more will be the erosion. In case of flat edge erosion will be small. In the next step of the loop the altitudes of all nodes are increased slightly. The reason is to avoid a situation, where after some iteration, all nodes are left with zero altitude. If it happens, it will simply undo all efforts done so far and would lead back to the beginning. But the altitude of destination will never change. It will remain zero always. Lastly, all paths are analyzed and the best one is chosen as an optimal solution.

4.2. Proposed Approach

First It will accepts a program written in either 'C' or 'C++' as input and will do the following task in sequence as shown in Fig. 4.2 also.

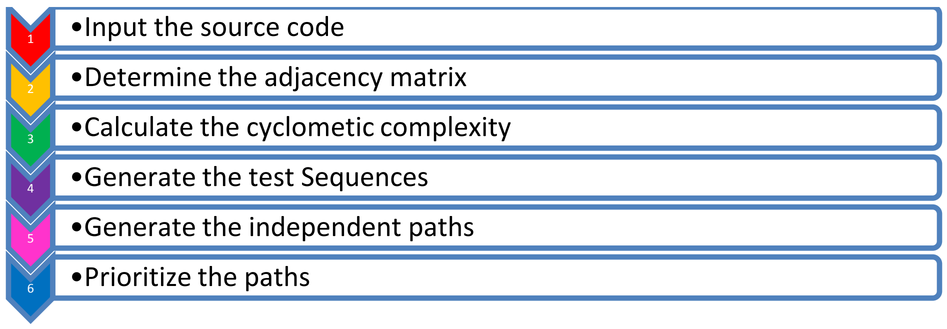

1
•Input the source code
2
•Determine the adjacency matrix
3
•Calculate the cyclometic complexity
4
•Generate the test Sequences
5
•Generate the independent paths
6
•Prioritize the paths

Fig. 4.2 Steps of the Proposed Approach

4.3. Tools/Technology Required

Following tools / technologies shall be used for implementation / simulation scenarios

- Java
- Net Beans
- MS Windows

4.4. Path Generation and Prioritization using RFD

Path testing ensures that all independent paths in a CFG of a program should be executed at least once. It ensures complete statement coverage, branch coverage and path coverage. The total number of independent paths in a program can be found by calculating McCabe cyclometic complexity and can be prioritized using path prioritization techniques. Path prioritization technique provides an ordering mechanism by assigning priority to each path based on test adequacy criteria. The path with the highest priority is most fault revealing and should be tested first because it increases the chance of detecting more errors in early stage with in limited resources.

Our proposed approach generates and prioritizes the paths in a CFG of the PUT. CFG is a kind of directed graph G= {V, E}, where V denotes the set of nodes in the graph and E denotes the set of edges/paths between the nodes. It represents all paths that can be traversed through a program under test.

Following is the description of all parameters used by drop M in the proposed approach.

- Every node ‘i’ in the CFG is assigned with some altitude value.

  alt(i) = some positive value

- Each node ‘i’ maintains its adjacency list $A_M(i)$, which contains the set of nodes which are directly connected to it.

  1. $A_M(i,j) = 1$, means that there is a path between node ‘i’ and node ‘j’.
  2. $A_M(i,j) = 0$, means that there is no path between node ‘i’ and node ‘j’. Adjacency list $A_M(i)$, of each node ‘i’ is further divided into three sets:

  a) $\{ V_M(i,j) \}$, is the set of adjacent nodes with positive gradient. (altitude of node ‘i’ is higher than node ‘j’)

b) { $U_M(i,j)$}, is the set of adjacent nodes with negative gradient. (altitude of node ‘i’ is lower than node ‘j’)

c) { $F_M(i,j)$ }, is the set of adjacent nodes with flat gradient. (altitude of node ‘i’ and node ‘j’ is same)

- To mark the status of a node, color attribute is used. If color(i) = 1 , means drop M has already visited the node ‘i’, where as color(i) = 0, means node ‘i’ not yet traversed by the drop.
- E_v, E_u, E_f are the parameters corresponding to above three sets of neighbors with positive gradient, negative gradient, flat gradient respectively.
- str(i,j) and hv(i,j) denotes the strength and heuristic value of each edge(i→j) respectively.

As compared to basic RFD scheme, where a drop M cannot climb, an improvement has been made, where a drop can make a move to a climbing edge(i→j) , j∊ $U_M(i)$, but with a very low probability, given by

$$P_M(i,j)=\{C/(\text{gradient}(i\ j)/\text{str}(i,j))/\sum_{k\epsilon AM(i)}(\text{gradient}(i,k)/\text{str}(i,k))\}$$

In the calculation of gradient, distance between node ‘i’ and node ‘j’ is considered as unity. And str(i,j) denotes the strength of the edge(i→j),which is set to one initially. As drops traverse it, its strength will increase. When there is a flat edge between node ‘i’ and node ‘j’, then the probability is set to some fixed non null value, say $p_{f..}$ In the normal case, where a drop M, residing at node ‘i’, making a downward move to a node ‘j’, j ∊ $V_M(i)$ the probability is given by

$$P_M(i,j)=\{(\text{gradient}(i\ ,\ j)/\text{str}(i,j))\ /\sum_{k\epsilon AM(i)}(\text{gradient}(i,k)/\text{str}(i,k))\}$$

Similarly the amount of erosion when a drop M moves through an edge (i→j) , depends on which group of neighbors, node ‘j’ belongs.

$$\text{erosion}(i,j) = \begin{cases} Ev*\text{gradient}(i\ ,\ j) & \text{if } j \in V_M(i) \\ E_u/\text{gradient}(i\ ,\ j) & \text{if } j \in U_M(i) \\ E_f & \text{if } j \in F_M(i) \end{cases}$$

Here, we have made a change in the basic RFD scheme, where after erosion, the altitudes of all the nodes are increased slightly. But in our proposed approach we are adding the sediments to only the node ‘j’ along the edge(i→j) rather than adding to all.

Now we present our proposed approach. It will be executed by drops one at a time in sequence.

```
1. Initialization
        1.1 Set the altitude of every node(alt) : For every node in the CFG, initialize alt =
            10000, except the last node, whose altitude is set to zero.
        1.2 Set strength of each edge(str): Initially strength of each edge(i→j) is 1, str(i,j) = 1
        1.3 Set a heuristic value for each edge(hv): Initialize hv(i,j) = 2 , for each edge(i→j)
        1.4 Set probability for flat environment($p_f$): $p_f = 0.1$
        1.5 Set erosion constant for different environment($E_v$, $E_u$, $E_f$) $E_{v=}0.5$,$E_u$=1000, $E_f$=100
        1.6 Initialize the number of drops(nd): let’s take nd=10
        1.7 Declare a list of source nodes(source)
        1.8 Set M=1
2. While (M<=nd)
    2.1 Initialize the adjacency matrix
    2.2 Add first node to the source list ‘source’
    2.3 Initialize the color matrix : for every node in the CFG, set color = 0 (white).
    2.4 While (source list is not empty)
        2.4.1 Remove the first node from the source list and assign it to node ‘i’ .
        2.4.2 While (there is a feasible path to move)
            2.4.2.1     Set color(i) = 1 (black)
            2.4.2.2     Determine adjacency list of i ($A_M$(i)) and if the out degree of
                        node ‘i’ is greater than or equal to 2, then add node ‘i’ in the
                        source list
        2.4.2.3    Calculate probability along each edge from current node ‘i’ to
                   all nodes in the $A_M(i)$ as per the rules described above.
        2.4.2.4    To make a move from current node ‘i’, drop M will use the
                   following rules:

          R1  Choose the path(i→j) with the maximum probability $P_{ij}$. If two or more
              paths have the same probability, then go to rule 2.

          R2  Choose the path(i→j) , if node j is the end node, else choose the
              path(i→j), if node j has the maximum out degree. If two or more
              adjacent nodes have the same out degree then make a random selection.

        2.4.2.5    Erode path(i→j): decrease the altitude of node ‘i’.
                       alt(i) = alt(i)-erosion(i,j)
                   where, erosion(i,j) will be calculated as described earlier in the
                       same section.
                   Deposit sediments: increase the altitude of node ‘j’.
                       alt(j)= alt(j) + erosion(i,j)
        2.4.2.6    Update strength and heuristic value of the path(i→j):
                       str(i,j)=str(i,j) + (1/hv(i,j))
                       hv(i,j)=hv(i,j)*2
        2.4.2.7    Update adjacency list of node ‘i’: remove node ‘j’ from node
                   i’s adjacency list $A_M(i)$ and assign node ‘i’= node ‘j’
        2.4.2.8    If(node ‘i’!= end node) , then go to step 2.4.2
                   Else
                   Record the path and discard it, if redundant and put the drop
                   M on to step 2.4
    2.5 M=M+1 , go to step 2
```

3. Generate independent paths and calculate the strength of each independent path by adding the strength of their respective edges and assign the priority to each path based on their strength.

4. End.

In the above algorithm, a source list is maintained, which is initialized with the first node. First of all, drop M will start traversing from the very first node in the source list and then after, it is removed from that list. During its course of journey, it will choose the nodes on the basis of probability, however if probability happens to be same for two or more nodes, the decision will be taken on the basis of the rules described in the algorithm above. As the drop M moves through the nodes, the strength and the heuristic value of the edges it traverse are updated. The process is repeated till it encounters a dead end (means no feasible path ahead). The moment it encounters an end node, it records the path and discard it, if found redundant. After this, the drop M, again starts the same journey, but this time, starting from the next node in the source list. It will continue to do so until the source list becomes empty. The entire procedure is iterated by the drops, initialized in the beginning. The test sequences obtained, may or may not be the complete paths i.e. starting from the source node and ending at the destination node. Therefore independent paths are generated from these sequences and prioritized on the basis of their strength. Here we have taken10 drops, but ideally we have to continue running in the loop until drops produce the same test sequences in the subsequent runs

4.5. Flow Chart of the Proposed Approach

Fig. 4.3 shows the flow of the steps involved in the proposed approach in detail.

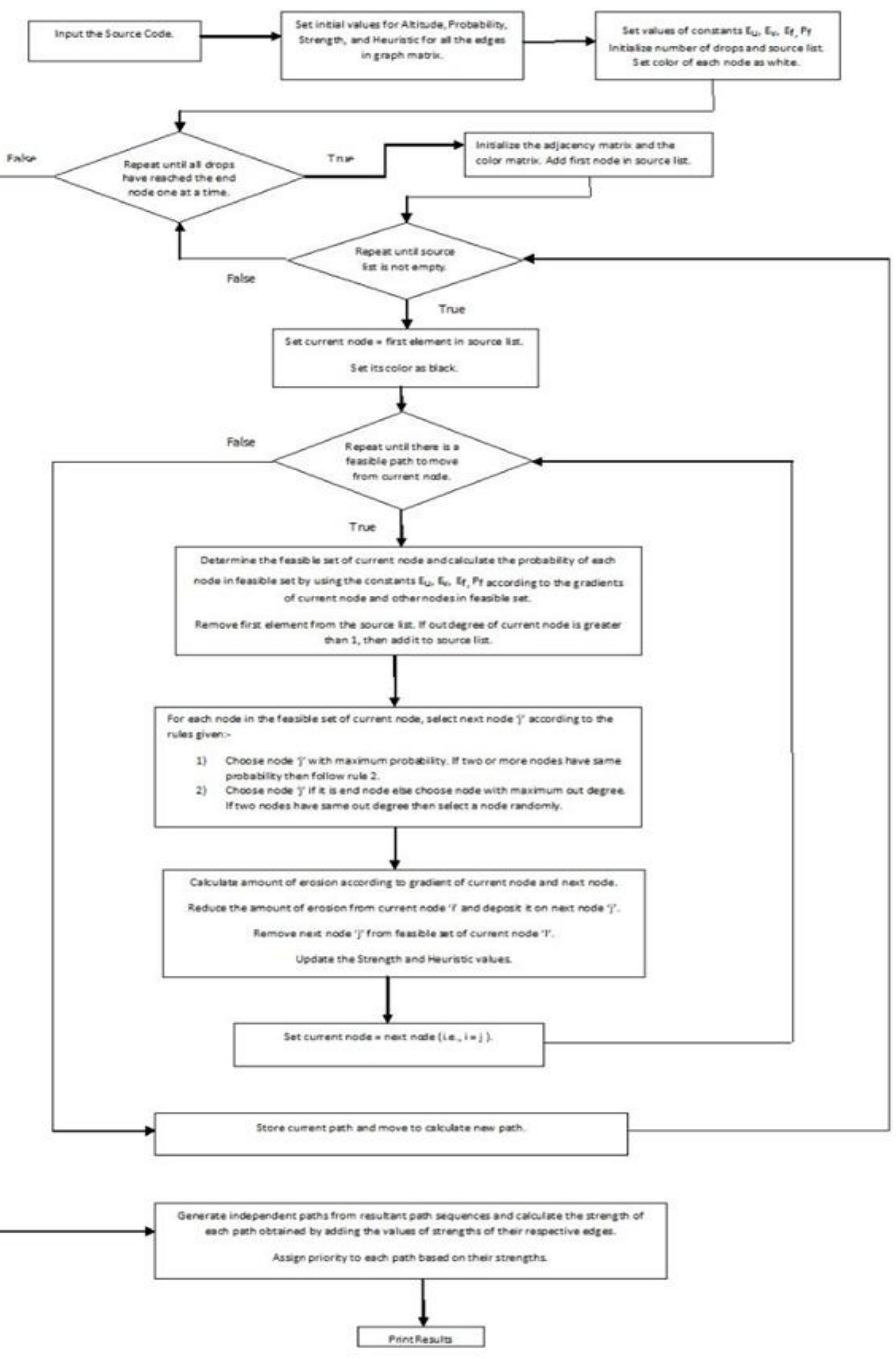

Input the Source Code.
Set initial values for Altitude, Probability, Strength, and Heuristic for all the edges in graph matrix.
Set values of constants E_U, E_V, E_f, P_f Initialize number of drops and source list. Set color of each node as white.
False
Repeat until all drops have reached the end node one at a time.
True
Initialize the adjacency matrix and the color matrix. Add first node in source list.
Repeat until source list is not empty.
False
True
Set current node = first element in source list.
Set its color as black.
False
Repeat until there is a feasible path to move from current node.
True
Determine the feasible set of current node and calculate the probability of each node in feasible set by using the constants E_U, E_V, E_f, P_f according to the gradients of current node and other nodes in feasible set.
Remove first element from the source list. If out degree of current node is greater than 1, then add it to source list.
For each node in the feasible set of current node, select next node 'j' according to the rules given:-
1) Choose node 'j' with maximum probability. If two or more nodes have same probability then follow rule 2.
2) Choose node 'j' if it is end node else choose node with maximum out degree. If two nodes have same out degree then select a node randomly.
Calculate amount of erosion according to gradient of current node and next node.
Reduce the amount of erosion from current node 'i' and deposit it on next node 'j'.
Remove next node 'j' from feasible set of current node 'I'.
Update the Strength and Heuristic values.
Set current node = next node (i.e., i = j).
Store current path and move to calculate new path.
Generate independent paths from resultant path sequences and calculate the strength of each path obtained by adding the values of strengths of their respective edges.
Assign priority to each path based on their strengths.
Print Results

Fig. 4.3 Flow Chart of the Proposed Approach

Chapter 5 Experimental Results and Analysis

This chapter describes the experimental results obtained from the examples taken into consideration. It also presents the analysis of the proposed approach in comparison with state of art techniques.

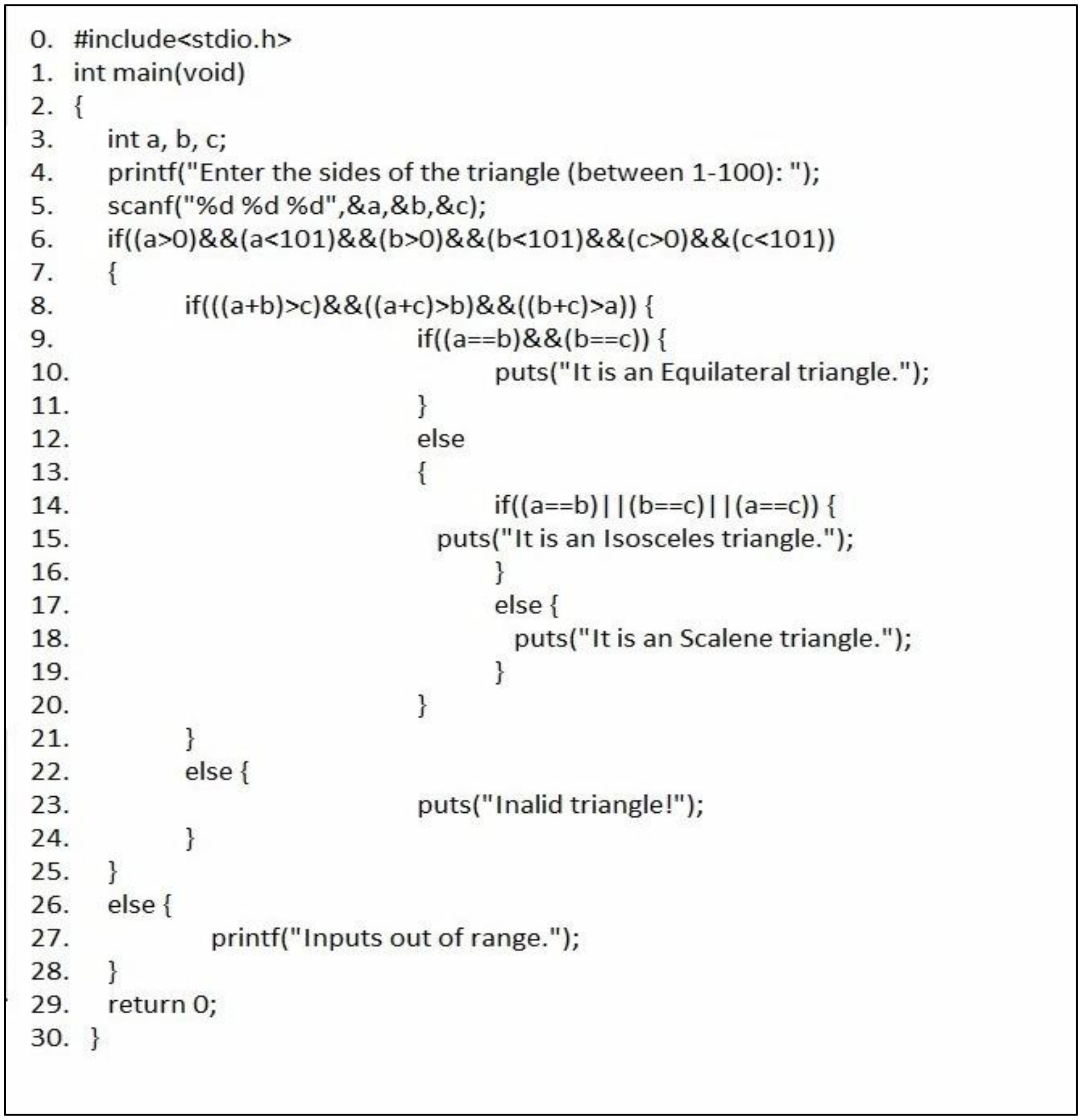
5.1. Testing Scenario A

We implemented our proposed approach using a testing tool called River Formation dynamics Test sequence Generator (RFDTSG), which we have implemented in java and Net beans has been used as IDE. It accepts a program written in either ‘C’ or ‘C++’ as input and does the following task:

- Determine the adjacency matrix.
- Calculate the cyclometic complexity
- Generate the test sequence as described earlier in chapter 3
- Generate independent paths
- Prioritize the paths

Let us consider the following ‘C’ language program for triangle classification shown in Fig. 5.1. The corresponding CFG is shown in Fig. 5.2. Initially we have taken a flat environment, i.e. , every node has the same altitude except the last node, therefore several iterations need to be carried out to transform the flat environment into an optimal environment where a drop should take the best path to traverse from source to the destination. Tables from 5-1 to 5-10 are showing different moves taken by the drops. The number of paths obtained can be less than or equal to the cyclometic complexity of the PUT. Here the cyclometic complexity for the above program is five and each drop encounters five different paths.

Last few subsequent drops (as can be seen in table 5-7 through table 5-10) and drops ahead are producing the same test sequences; it means more or less, further drops will follow the same sequences. So it is not worth continuing, taking more drops. Since the test sequences generated are not the complete path from source to the destination, therefore after that, independent paths needs to be generated from the last test sequences obtained and then, paths are prioritized on the basis of number of times , it is traversed by the drops, i.e. , the path traversed the most will have the highest priority. Table 5-11 is showing all independents paths and their priorities, which is assigned on the basis of their strength and their correlation with the number of predicate nodes. It is evident from table 5-11 that, path 5 is having the highest priority as covering all the decision nodes.



```
0.  #include<stdio.h>
1.  int main(void)
2.  {
3.      int a, b, c;
4.      printf("Enter the sides of the triangle (between 1-100): ");
5.      scanf("%d %d %d",&a,&b,&c);
6.      if((a>0)&&(a<101)&&(b>0)&&(b<101)&&(c>0)&&(c<101))
7.      {
8.              if(((a+b)>c)&&((a+c)>b)&&((b+c)>a)) {
9.                                      if((a==b)&&(b==c)) {
10.                                             puts("It is an Equilateral triangle.");
11.                                     }
12.                                     else
13.                                     {
14.                                             if((a==b)||(b==c)||(a==c)) {
15.                                       puts("It is an Isosceles triangle.");
16.                                             }
17.                                             else {
18.                                               puts("It is an Scalene triangle.");
19.                                             }
20.                                     }
21.             }
22.             else {
23.                                     puts("Inalid triangle!");
24.             }
25.     }
26.     else {
27.             printf("Inputs out of range.");
28.     }
29.     return 0;
30. }
```

Fig. 5.1 'C' Program for Triangle Classification

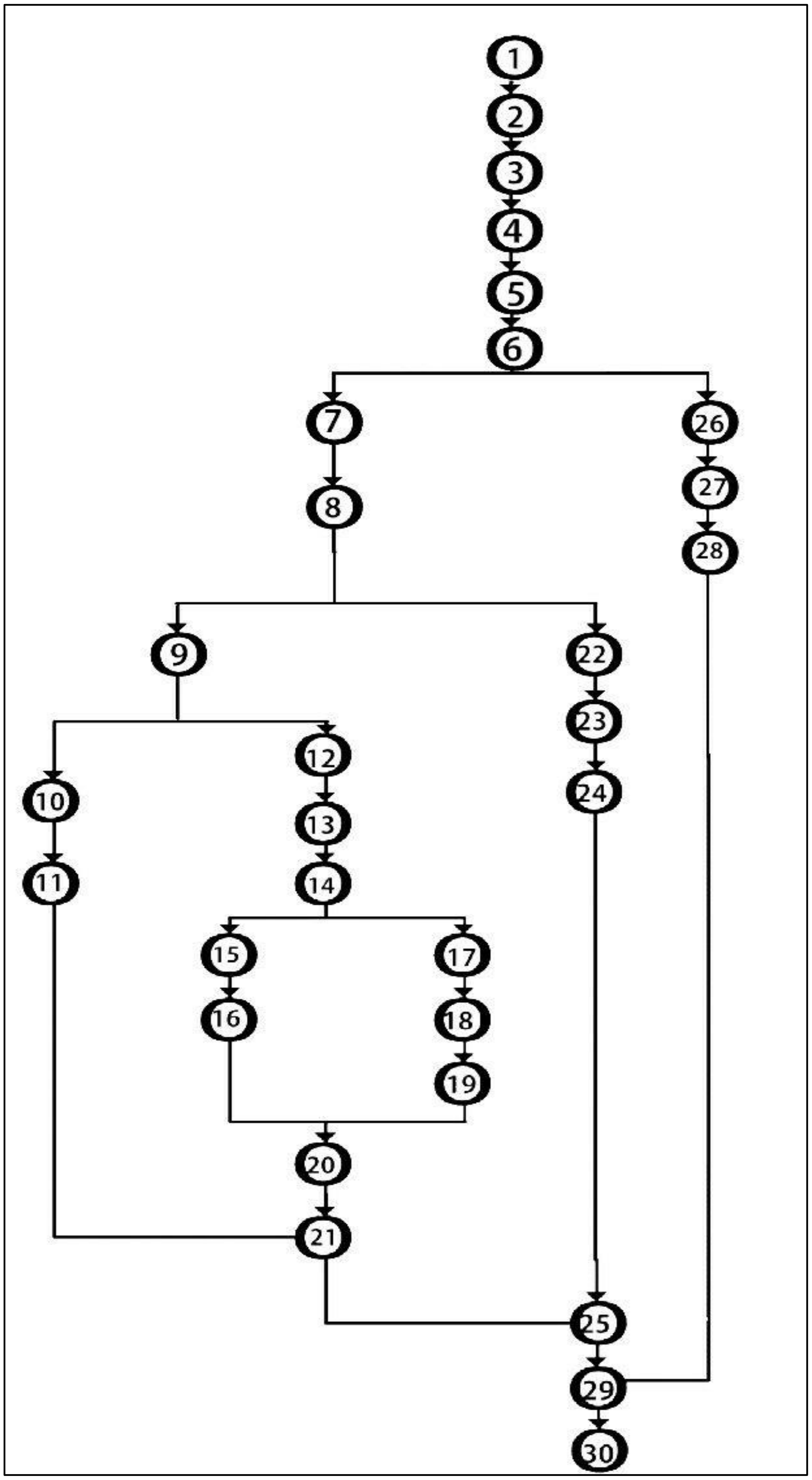


Fig. 5.2 CFG for Triangle Classification

Table 5-1 Test Sequences Obtained by 1st Drop in Test A

Paths	Nodes
Path 1	1-2-3-4-5-6-26-27-28-29-30
Path 2	6-7-8-9-10-11-21-25-29
Path 3	8-22-23-24-25
Path 4	9-12-13-14-17-18-19-20-21
Path 5	14-15-16-20

Table 5-2 Test Sequences Obtained by 2nd Drop in Test A

Paths	Nodes
Path 1	1-2-3-4-5-6-7-8-9-12-13-14-15-16-20-21-25-29-30
Path 2	6-26-27-28-29
Path 3	8-22-23-24-25
Path 4	9-10-11-21
Path 5	14-17-18-19-20

Table 5-3 Test Sequences Obtained by 3rd Drop in Test A

Paths	Nodes
Path 1	1-2-3-4-5-6-26-27-28-29-30
Path 2	6-7-8-22-23-24-25-29
Path 3	8-9-10-11-21-25
Path 4	9-12-13-14-15-16-20-21
Path 5	14-17-18-19-20

Table 5-4 Test Sequence Obtained by 4th Drop in Test A

Paths	Nodes
Path 1	1-2-3-4-5-6-7-8-22-23-24-25-29-30
Path 2	6-26-27-28-29
Path 3	8-9-12-13-14-15-16-20-21-25
Path 4	9-10-11-21
Path 5	14-17-18-19-20

Table 5-5 Test Sequence Obtained by 5th Drop in Test A

Paths	Nodes
Path 1	1-2-3-4-5-6-26-27-28-29-30
Path 2	6-7-8-22-23-24-25-29
Path 3	8-9-10-11-21-25
Path 4	9-12-13-14-15-16-20-21
Path 5	14-17-18-19-20

Table 5-6 Test Sequence Obtained by 6th Drop in Test A

Paths	Nodes
Path 1	1-2-3-4-5-6-7-8-22-23-24-25-29-30
Path 2	6-26-27-28-29
Path 3	8-9-12-13-14-1516-20-21-25
Path 4	9-10-11-21
Path 5	14-17-18-19-20

Table 5-7 Test Sequence Obtained by 7th Drop in Test A

Paths	**Nodes**
Path 1	1-2-3-4-5-6-7-8-9-12-13-14-15-16-20-21-25-29-30
Path 2	6-26-27-28-29
Path 3	8-22-23-24-25
Path 4	9-10-11-21
Path 5	14-17-18-19-20

Table 5-8 Test Sequence Obtained by 8th Drop in Test A

Paths	**Nodes**
Path 1	1-2-3-4-5-6-7-8-9-12-13-14-15-16-20-21-25-29-30
Path 2	6-26-27-28-29
Path 3	8-22-23-24-25
Path 4	9-10-11-21
Path 5	14-17-18-19-20

Table 5-9 Test Sequence Obtained by 9th Drop in Test A

Paths	**Nodes**
Path 1	1-2-3-4-5-6-7-8-9-12-13-14-15-16-20-21-25-29-30
Path 2	6-26-27-28-29
Path 3	8-22-23-24-25
Path 4	9-10-11-21
Path 5	14-17-18-19-20

Table 5-10 Test Sequence Obtained by 10th Drop in Test A

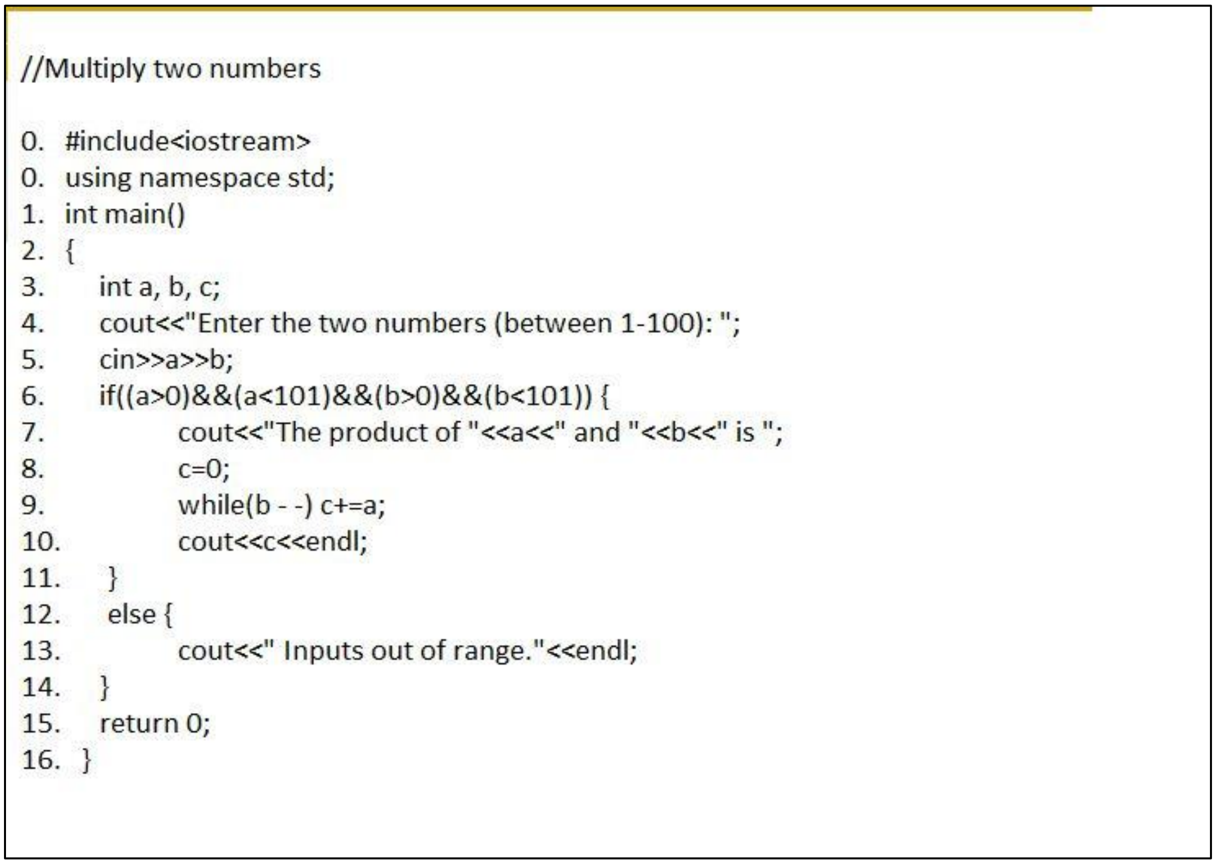
Paths	Nodes
Path 1	1-2-3-4-5-6-7-8-9-12-13-14-15-16-20-21-25-29-30
Path 2	6-26-27-28-29
Path 3	8-22-23-24-25
Path 4	9-10-11-21
Path 5	14-17-18-19-20

Table 5-11 Generated Independent Paths in Test A

Paths	Nodes	Strength	Priority	Number of Predicate Nodes
Path 1	1-2-3-4-5-6-7-8-9-10-11-21-25-29-30	27.98632812500000	3	3
Path 2	1-2-3-4-5-6-26-27-28-29-30	19.99023437500000	5	1
Path 3	1-2-3-4-5-6-7-8-22-23-24-25-29-30	25.98730468750000	4	2
Path 4	1-2-3-4-5-6-7-8-9-12-13-14-15-16-20-21-25-29-30	35.98242187500000	2	4
Path 5	1-2-3-4-5-6-7-8-9-12-13-14-17-18-19-20-21-25-29-30	37.98144531250000	1	4

5.2. Testing Scenario B

Let us consider the following ‘C++’ language program for multiplication of two numbers that contains loop and self loop as shown in Fig. 5.3. The corresponding CFG is shown in Fig. 5.4.



```
//Multiply two numbers

0.  #include<iostream>
0.  using namespace std;
1.  int main()
2.  {
3.      int a, b, c;
4.      cout<<"Enter the two numbers (between 1-100): ";
5.      cin>>a>>b;
6.      if((a>0)&&(a<101)&&(b>0)&&(b<101)) {
7.              cout<<"The product of "<<a<<" and "<<b<<" is ";
8.              c=0;
9.              while(b - -) c+=a;
10.             cout<<c<<endl;
11.      }
12.      else {
13.             cout<<" Inputs out of range."<<endl;
14.     }
15.     return 0;
16. }
```

Fig. 5.3 C++ Program for Multiplication of Two Numbers

Again we have taken a flat environment in the beginning, i.e. , every node has the same altitude except the last node, therefore several iterations need to be carried out to transform the flat environment into an optimal environment where a drop should take the best path to traverse from source to the destination. Tables from 5-12 to 5-21 are showing different moves taken by the drops. The number of paths obtained can be less than or equal to the cyclometic complexity of the PUT. Here the cyclometic complexity for the above program is three and each drop encounters three different paths.

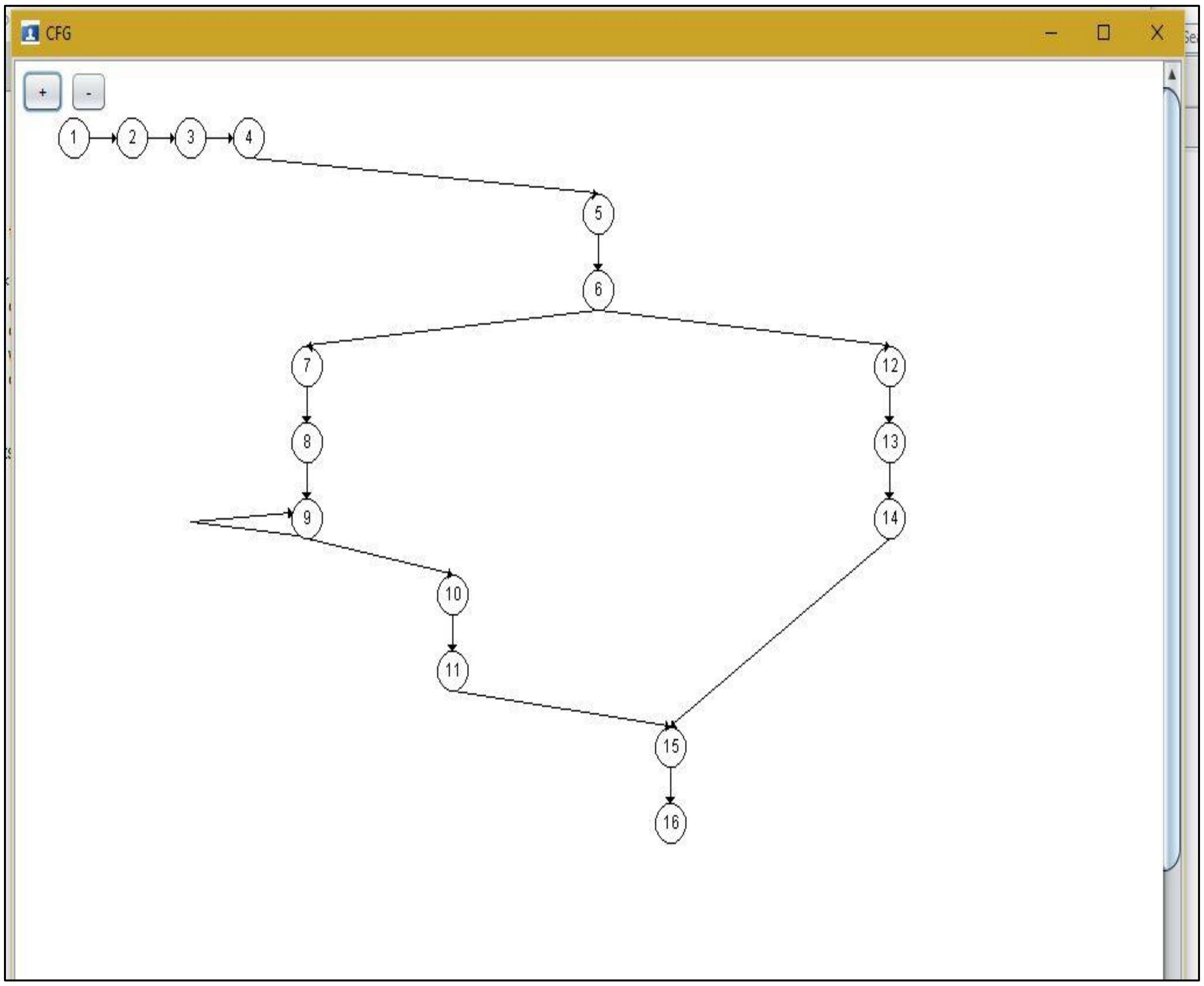


Fig 5.4 CFG for the Program of Multiplication of Two Numbers

Last few subsequent drops (as can be seen in table 5-18 through table 5-21) and drops ahead are producing the same test sequences; it means more or less, further drops will follow the same sequences. So it is not worth continuing, taking more drops. Since the test sequences generated are not the complete path from source to the destination, therefore after that, independent paths needs to be generated from the last test sequences obtained and then, paths are prioritized on the basis of number of times , it is traversed by the drops, i.e. , the path traversed the most will have the highest priority. Table 5-22 is showing all independents paths and their priorities, which is assigned on the basis of their strength and their correlation with the number of predicate nodes. It is evident from table 5-22 that, path 3 is having the highest priority as covering all the decision nodes.

Table 5-12 Test Sequences Generated by 1st Drop in Test B

Paths	Nodes
Path 1	1-2-3-4-13
Path 2	1-2-3-4-5-6-7-8-9-13
Path 3	1-2-3-4-5-6-10-11-4-5-6-7-8-9-13

Table 5-13 Test Sequences Generated by 2nd Drop in Test B

Paths	Nodes
Path 1	1-2-3-4-13
Path 2	1-2-3-4-5-6-7-8-9-13
Path 3	1-2-3-4-5-6-10-11-4-5-6-7-8-9-13

Table 5-14 Test Sequences Generated by 3rd Drop in Test B

Paths	Nodes
Path 1	1-2-3-4-13
Path 2	1-2-3-4-5-6-7-8-9-13
Path 3	1-2-3-4-5-6-10-11-4-5-6-7-8-9-13

Table 5-15 Test Sequences Generated by 4th Drop in Test B

Paths	Nodes
Path 1	1-2-3-4-13
Path 2	1-2-3-4-5-6-7-8-9-13
Path 3	1-2-3-4-5-6-10-11-4-5-6-7-8-9-13

Table 5-16 Test Sequences Generated by 5th Drop in Test B

Paths	Nodes
Path 1	1-2-3-4-13
Path 2	1-2-3-4-5-6-7-8-9-13
Path 3	1-2-3-4-5-6-10-11-4-5-6-7-8-9-13

Table5-17 Test Sequences Generated by 6th Drop in Test B

Paths	Nodes
Path 1	1-2-3-4-13
Path 2	1-2-3-4-5-6-7-8-9-13
Path 3	1-2-3-4-5-6-10-11-4-5-6-7-8-9-13

Table 5-18 Test Sequences Generated by 7th Drop in Test B

Paths	Nodes
Path 1	1-2-3-4-13
Path 2	1-2-3-4-5-6-7-8-9-13
Path 3	1-2-3-4-5-6-10-11-4-5-6-7-8-9-13

Table 5-19 Test Sequences Generated by 8th Drop in Test B

Paths	Nodes
Path 1	1-2-3-4-13
Path 2	1-2-3-4-5-6-7-8-9-13
Path 3	1-2-3-4-5-6-10-11-4-5-6-7-8-9-13

Table 5-20 Test Sequences Generated by 9th Drop in Test B

Paths	**Nodes**
Path 1	1-2-3-4-13
Path 2	1-2-3-4-5-6-7-8-9-13
Path 3	1-2-3-4-5-6-10-11-4-5-6-7-8-9-13

Table 5-21 Test Sequences Generated by 10th Drop in Test B

Paths	**Nodes**
Path 1	1-2-3-4-13
Path 2	1-2-3-4-5-6-7-8-9-13
Path 3	1-2-3-4-5-6-10-11-4-5-6-7-8-9-13

Table 5-22 Generated Independent Paths in Test B

Paths	**Nodes**	**Strength**	**Priority**	**Number of Predicate Nodes**
Path 1	1-2-3-4-5-6-7-8-9-9-10-11-15-16	25.99963378813117	1	2
Path 2	1-2-3-4-5-6-12-13-14-15-16	19.99969482421875	3	1
Path 3	1-2-3-4-5-6-7-8-9-10-11-15-16	23.99963378906250	2	2

5.3. Comparison with State-of-Art Techniques

Since RFD is a gradient version of ACO, therefore first we have compared our proposed approach with ACO [40] and then with firefly algorithm (FA) [51]. Table 5-23 lists the paths generated for the CFG in Fig. 5.5, using ACO. The paths generated in approach [40] has a lot of redundancy, because the path segment 0-1-2-3-4 has been repeated in all the paths as visible in table 5-23, where as in our approach and FA, each transition/edge has been traversed exactly once as can be seen in table 5-24 and 5-25 respectively. Fig. 5.6 shows the comparison of ACO, RFD and FA. This illustrates that RFD is at par with FA and produces better test sequences as compared to ACO with no redundancy.

Fig. 5.7 is demonstrating the effect of increase in number of nodes in CFG on redundancy in path coverage. It is evident that, in case of ACO as number of nodes in the CFG increases, so is the redundancy. FA seems better than ACO but still encountering small amount of redundancy as the number of nodes increases. The situation in RFD is quite different and much better than ACO and FA. The increase in the number of nodes has no effect on the redundancy in test paths

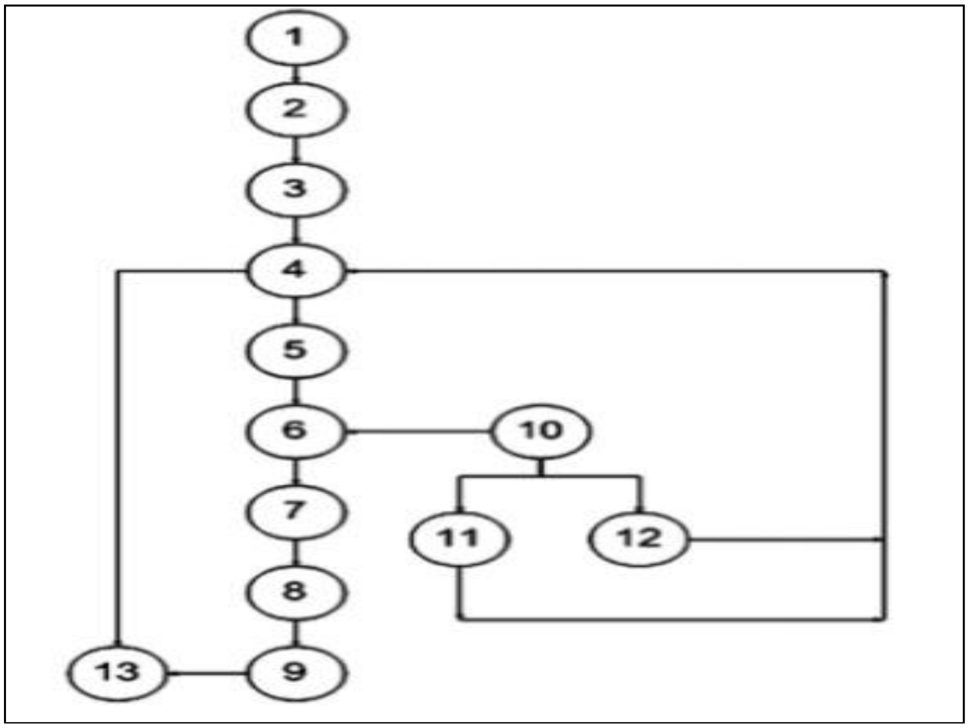


Fig. 5.5 CFG of Binary Search [40]

Table 5-23 Test Sequence Generated Using ACO [40]

Paths	**Nodes**
Path 1	1-2-3-4-13
Path 2	1-2-3-4-5-6-7-8-9-13
Path 3	1-2-3-4-5-6-10-11-4-5-6-7-8-9-13
Path 4	1-2-3-4-5-6-10-12-4-5-6-7-8-9-13

Table 5-24 Test Sequence Generated Using RFD

Paths	**Nodes**
Path 1	1-2-3-4-5-6-10-11-4-13
Path 2	6-7-8-9-13
Path 3	10-12-4

Table 5-25 Test Sequence Generated Using FA [51]

Paths	**Nodes**
Path 1	1-2-3-4-5-6-10-12-4-13
Path 2	6-7-8-9-13
Path 3	10-11-4

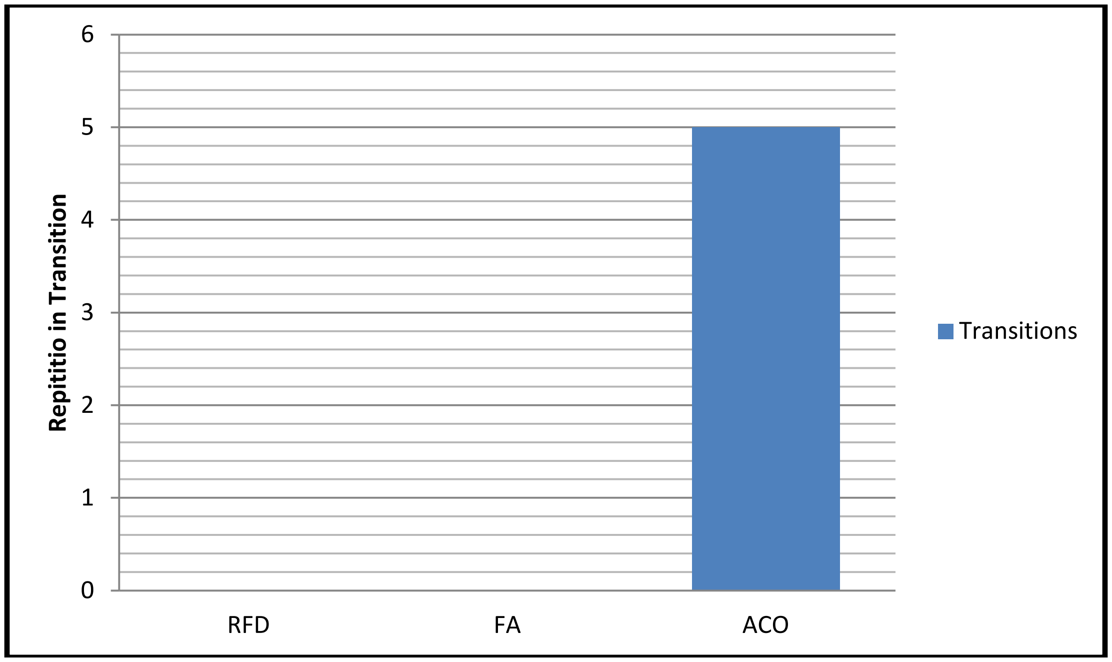


Fig. 5.6 Comparative Analysis of RFD, FA and ACO

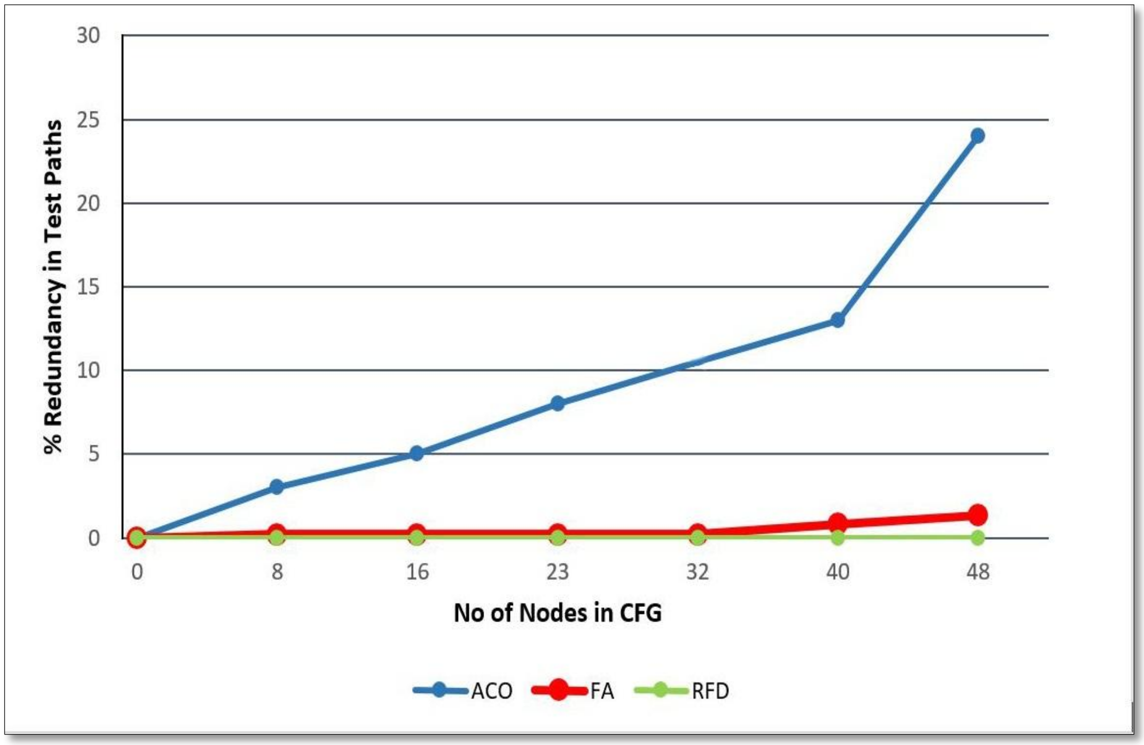


Fig. 5.7 Redundancy vs. Number of Nodes

since each edge is traversed only once. Table 5-26 is showcasing some test cases highlighting the % of redundancy in ACO, FA and RFD with respect to the number of nodes in the CFG. The results of ACO and FA have been taken from [51]. Thus our proposed approach is more efficient as compared to ACO and FA.

Table 5-26 Test Cases for Comparison between ACO [40], FA [51] and RFD

S. No	No. of Nodes	Cylometic Complexity	% Redundancy in path coverage ACO	% Redundancy in path coverage FA	% Redundancy in path coverage RFD
1	16	4	5	0	0
2	23	3	8	0	0
3	40	5	13	0.8	0
4	48	12	24	1.32	0

Chapter 6 Conclusion and Future Scope

This chapter draws conclusions based on the contributions made by this thesis.

6.1. Research Summary

This thesis summarizes the SWOT analysis of SBSE to understand the alphas and omegas of this area and in particular focused on optimization of test sequence generation in the area of SBST via RFD to reduce the test effort It began with discussion of motivations, theoretical framework and research question, importance of the research, aims and outcomes. Vast literature of search based software engineering, search based software testing and river formation dynamics was reviewed and different approaches for test sequence generation were studied resulting in identification of research gaps.

We then proposed a novel approach for test sequence generation using river formation dynamics algorithm. Also, priorities have been assigned to different paths for identification of critical/error prone paths to be tested first. Finally, a comparative analysis of proposed approach with other existing approaches of test sequence generation has been done and significant improvement in minimizing testing effort is observed.
The major contributions of this research are:

i. We carried out SWOT analysis of search based software engineering to urge academicians and researchers to make the significant improvement in this domain.
ii. We proposed a novel approach for test sequence generation and prioritization via a swarm intelligent method called river formation dynamics for reducing the testing effort.

6.2. Conclusion and Future Research Directions

Search Based Software Engineering is an emerging field basically involves amalgamation of software engineering, operation research and metaheuristic techniques. It is specially excelled in providing an ideally balanced solution to a SE problem involving several competing goals. It caters to the problem of SE and provides optimal solution to give better prospects over tools and techniques required to grow a productive, proficient and cohesive approach. We carried out SWOT analysis of SBSE to illustrates all the bane's & boon's within and outside the areas that impact the development and coerce for a universal and proactive approach to SBSE, which will minimizes the inaccuracies and enhance the results.

Software testing is the major area, explored the most, by search based techniques. But test sequence generation which is a significant activity in cutting down the testing effort, hardly gets importance as compared to other testing activities such as automatic test data generation. Only few researchers have attempted to generate test sequences using different search based algorithms like ant colony optimization, genetic algorithm, tabu search, cuckoo search, firefly algorithm, each having their own advantages and disadvantages. Ideally none of them is perfect as far as Coverage and redundancy is concerned. Therefore, as a step forward, we have proposed a new technique for path generation and prioritization, inspired by a swarm intelligence method called River Formation Dynamics.

We have extended the basic RFD scheme for the generation of optimal test sequences for path testing. The proposed technique not only minimizes the redundancy in test sequences, but also successful in finding the error prone/critical paths to be tested first. Experimental results demonstrate that our approach has shown a remarkable improvement over other approaches in [40] and [51] thus, leading to a significant reduction in testing effort. Furthermore the success of our approach depends on various RFD parameters and constants. The optimization of these parameters can be taken up as the future work to fine tune the algorithm to provide better convergence rate and to apply in many other domains.

References

[1] M. Harman and B.F. Jones, "Search-based software engineering", *Journal of Information and Software Technology*, Vol. 43, pp. 833- 839, December 2001.

[2] M Harman and J. Clark, "Metrics are fitness functions too", International Software Metrics Symposium (METRICS 2004), pp. 58-69, 2004.

[3] A. Bagnall, V. Rayward-Smith, and I. Whittley , "The next release problem" , *Information and Software Technology*, Vol. 43, No.14, December 2001.

[4] A. Ngo-The and G. Ruhe, "Optimized resource allocation for software release planning", *IEEE Transactions on Software Engineering*, Vol. 35, No. 1, pp. 109-123, Jan.-Feb. 2009.

[5] Y. Zhang, M. Harman, and S.A. Mansouri, "The multi-objective next release problem", Genetic and Evolutionary Computation Conference (GECCO 2007), ACM, page 1129-1137, 2007.

[6] C. Burgess and M. Leey, "Can genetic programming improve software effort estimation? a comparative Evaluation", *Information and Software Technology*, Vol. 43, No. 14, pp. 863-873, 2001.

[7] J. Dolado, " On the problem of the software cost function", *Information and Software Technology*, Vol. 43, No. 1, pp. 61-72, 2001.

[8] C. Kirsopp, M. Shepperd, and J. Hart, "Search heuristics, case-based reasoning and software project effort prediction", Proceedings of the 4th Annual conference on Genetic and evolutionary computation, (GECCO '02), pp. 1367-1374, 2002.

[9] A. Baresel, H. Sthamer, and M. Schmidt, "Fitness function design to improve evolutionary structural testing", Proceedings of the 4th Annual conference on Genetic and evolutionary computation, (GECCO '02), pp. 1329-1336, 2002.

[10] B. S. Mitchell and S. Mancoridis, "Using heuristic search techniques to extract design abstractions from source code", Proceedings of the 4th Annual conference on Genetic and evolutionary computation, (GECCO '02), pp. 1375-1382, 2002.

[11] S.L. Rhys, S. Poulding, and J.A. Clark, “Using automated search to generate test data for matlab”, Proceedings of the 11th Annual conference on Genetic and evolutionary computation, (GECCO '09), pp. 1697-1704, 2009.

[12] M. Harman, R. Hierons, and M. Proctor, “A new representation and crossover operator for search-based optimization of software modularization”, Proceedings of the Genetic and Evolutionary Computation (GECCO 2002), pp. 1351-1358, July 2002.

[13] M. Claudia Figueiredo, P. Emer, and S. Regina Vergilio, “GPTesT: A testing tool based on genetic Programming”, Proceedings of the 4th Annual conference on Genetic and evolutionary Computation, (GECCO '02), pp. 1343-1350, 2002.

[14] R. Lutz, “Evolving Good Hierarchical Decompositions of Complex Systems”, *Journal of Systems Architecture*, Vol. 47, No. 7, pp. 613– 634, 2001.

[15] S. Poulding, P. Emberson, I.Bate, and J A. Clark, “An efficient experimental methodology for configuring search-based design algorithms”, Proceedings of 10th IEEE High Assurance System Engineering Symposium (HASE’2007), pp. 53–62, 2007.

[16] E. Alba, and J. M. Troya, “ Genetic Algorithms for Protocol Validation”, Proceedings of the 4th International Conference on Parallel Problem Solving from Nature (PPSN ’96), Springer, pp. 870–879,1996.

[17] J. A. Clark and L. Jacob, “Searching for a Solution: Engineering Tradeoffs and the Evolution of Provably Secure Protocols”, Proceedings of the 2000 IEEE Symposium on Security and Privacy (S&P ’00), IEEE, pp.82–95, 2000.

[18] A. Clark and J. L. Jacob, “Protocols are Programs too: the Meta-Heuristic Search for Security Protocols”, *Information and Software Technology*, Vol. 43, No. 14, pp. 891–904, 2001.

[19] M. Ferreira, F. Chicano, E. Alba, and J. A. Gomezpulido, “Detecting Protocol Errors using Particle Swarm Optimization with Java Pathfinder”, Proceedings of the High Performance Computing & Simulation Conference (HPCS ’08), Cyprus, pp. 319–325, 2008.

[20] C. Nguyen, S. Miles, A. Perini, P. Tonella, M. Harman, and M. Luck, “Evolutionary Testing Autonomous Software Agents”, Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems (AAMAS ’09), International Foundation for Autonomous Agents and Multiagent Systems, pp. 521–528.

[21] T. Jiang, N. Gold, M. Harman and Z. Li, “Locating Dependence Structures using Search-based Slicing”, *Information and Software Technology*, Vol. 50, No. 12, pp. 1189–1209, 2007.

[22] T. Jiang, M. Harman and Y. Hassoun, “Analysis of Procedure Splitability”, Proceedings of the 15th Working Conference on Reverse Engineering (WCRE ’08). IEEE, pp. 247-256, 2008.

[23] Y. Zhang, M. Harman, and A. Mansouri, “The SBSE repository: A repository and analysis of authors and research articles on search based software engineering”, crestweb.cs.ucl.ac.uk/resources/sbse repository.

[24] M. Harman, A. Mansouri, and Y. Zhang, “Search based software engineering: A comprehensive analysis and review of trends techniques and applications”, Technical Report TR-09-03, April 2009.

[25] R. Roshan, R. Porwal, C. M. Sharma, “Review of Search based Techniques in Software Testing”, *IJCA*, Vol. 51, No. 6, 2012.

[26] J. Bedi, K. Kaur, “Search Based Software Engineering”, *IJERA*, Vol. 4, No. 4, 2014.

[27] P. Maragathavalli, “Search Based Software Test Data Generation Using Evolutionary Computation”, *IJCSIT*, Vol. 3, No. 1, 2011.

[28] M. Harman, “The Current State and Future of Search Based Software Engineering”, Future of Software Engineering (FOSE'07) 0-7695-2829-5/07, IEEE, 2007.

[29] J. J. Li, “Prioritize code for testing to improve code coverage of complex software”, Proceedings of the 16th IEEE International Symposium on Software Reliability Engineering, ISSRE, Chicago, pp.84-93, 2005.

[30] A. P. Mathur, “Foundation of software Testing”, First Edition, Pearson Education, 2007.

[31] W.E. Howden, “Functional Program Testing and Analysis”, McGraw-Hill, 1987.

[32] V. Bhattacherjee, Suri, P.Mahanti, “Application of Regular Matrix Theory to Software Testing”, *European Journal of scientific Research*, Vol. 12, No. 1, pp. 60-70, 2005.

[33] R. Sedgewick, “Algorithms in java”, Third Edition, Part 5, Graph algorithms, Addison Wesley, 2003.

[34] Jin-Cherng Lin, Pu-Lin Yeh, “Using Genetic Algorithms for Test Case Generation in Path Testing”, Proceedings of the Ninth Asian Test Symposium (ATS'00), pp.241-246,2000.

[35] Dr. V. Rajappa, A. Biradar, and S. Panda, “Efficient Software Test Case Generation Using Genetic Algorithm Based Graph Theory”, First International Conference on Emerging Trends in Engineering and Technology, pp.298-303, 2008.

[36] H. Li, C. Peng Lam. “Software Test Data Generation using Ant Colony Optimization”, Proceedings Of World Academy Of Science, Engineering And Technology, Vol. 1, pp.1-4, 2005.

[37] FU Bo, “Automated software test data generation based on ant colony algorithm”, *Computer Engineering and Applications*, Vol. 43. No. 12, pp. 97-99, 2007.

[38] P. R Srivastava , A. K. Patel ,K. Patel and P. Vijaywargiya, “Test data generations based on Test path Discovery using intelligent Water Drop” , *International journal of applied Metaheuristic computing*, Vol. 3, No. 2, pp. 56-74, 2012.

[39] P.R. Srivastava & K. Baby ,“Automated Software testing using metaheuristics techniques based on Ant colony Optimization” ,International Symposium on Electronic System Design (ISED), 2010.

[40] P. R. Srivastava, K. Baby, and G. Raghurama, “An approach of optimal path generation using ant colony optimization,” in Proceedings of TENCON 2009, IEEE Press, pp. 1-6, 2009.

[41] A. S. Ghiduk, “A new software data-flow testing approach via ant colony algorithms,” *Universal Journal of Computer Science and Engineering Technology*, pp. 64-72, 2010.

[42] H. Li and C. Peng LAM , “An Ant Colony Optimization Approach to Test Sequence Generation for State based Software Testing”, Proceedings of the Fifth International Conference on Quality Software (QSIC‘05), pp 255 – 264, 2005.

[43] P. Rabanal, I. Rodríguez, and F. Rubio, "Using river formation dynamics to design heuristic algorithms", *Unconventional Computation*, LNCS 4618, pages 163–177. Springer, 2007.

[44] P. Rabanal, I. Rodríguez, and F. Rubio," Solving Dynamic TSP by using River Formation Dynamics", Fourth International Conference on Natural Computation, pp.246-250.

[45] P. Rabanal, I. Rodríguez, and F. Rubio, " Finding Minimum Spanning /Distance Trees by using River Formation Dynamics", *Ant colony Optimization and Swarm Intelligence* , ANTS'08, LNCS 5217, Springer, pp. 60-71, 2009.

[46] P. Rabanal, I.Rodríguez, and F. Rubio, "Applying RFD to Construct Optimal Quality Investment Trees", *Journal of Universal Computer Science,* Vol. 16, No. 14, 2010.

[47] K.D. Cooper, P.J. Schielke, and D. Subramanian, "Optimizing for reduced code space using genetic algorithms", in Proceedings of LCTES, Vol. 34, No. 7. 1999.

[48] E. Díaz, J. Tuya, and R. Blanco, "Automated software testing using a metaheuristic technique based on tabu search", in Proceedings of the 18th IEEE International Conference on Automated Software Engineering (ASE '03), pp.310–313, IEEE, Montreal, Canada, 2003.

[49] C. Doungsa-ard, K. Dahal, M.A. Hossain, and T. Suwannasart, "GA-based for automatic test data generation for UML state diagrams with parallel paths", Proceedings of the International Conference on Advanced Design and Manufacture (ICADAM), Springer Verlag, China, 2008.

[50] P.R. Srivastava, C. Sravya, Ashima, S. Kamisetti and M. lakshmi, "Test sequence optimization: an intelligent approach via cuckoo search", *IJBIC,* Vol. 4, No. 3, 2012.

[51] P.R Srivastava, B. Mallikarjun, Xin-She-Yang, "Optimal test sequence generation using firefly algorithm", *Swarm and Evolutionary Computation*, Vol. 8, pp 44-53, 2013.

[52] G. Redlarski, A. Palkowski, M. Dabkowski, "Using River Formation Dynamics Algorithm in Mobile Robot Navigation", *Solid State Phenomena,* 2013.

[53] K. Venkateswarlu, T.V. Rao, "River Formation Dynamics Based Data Aggregation in Wireless Sensor Network", *IJETER,* Vol.3, No. 6, pp. 100-105, 2015.

[54] D. Dholakiya, T. Doshi, S. Ghiya and P. Patel, “Advanced River Formation Dynamics for Location Management in GSM”, *IJERT*, Vol. 4, No. 9, 2015.

APPENDIX A

List of Publication(s)

Published

1. A. Sharma & Y. Khatri, “SWOT Analysis of Search Based Software Engineering”, Proceedings of International Conference on Computing for Sustainable Global Development, INDIACom-2016.

Communicated

1. Y. Khatri & A. Sharma, “Optimal Test Sequence Generation using River Formation Dynamics*”, International Journal of Engineering and Technology (IJET),* (2016).