Volume 5, Issue 10, October-2015 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

A New Method for Generating Minimal Test Cases for Regression Testing using Dynamic Approach

Shrankhla Saxena *
Department of Computer Science & Engineering
Uttarakhand Technical University, India

Dr. B. M Singh
Department of IT
COER, India

Abstract*— Testing the software is very important level in the development of the software life cycle. So to the test software automatically is the best way to test the software because it consume less time where testing software manually is consuming process. To software automatically test, test case creation is the best method. One method to produce test cases is to help of the UML figures. One of the basic regression testing objectives is to test that improved system works specification according, at the same time of duration optimizing various test cases through creating it effective and efficient. This paper shows the black-box method that uses Steiner Tree algorithm to create a minimal test set to check functionality. Test cases are produced from requirement represented applying UML. A terminal set are provide as input to the Dynamic Steiner Tree algorithm with graph G. The modification dnodes are defined as terminal nodes to guarantee presence in the test set. A minimal test cases set is created as an indicator to the efficiency of the change. Initial outcomes present that the technique is applicable for rapid testing to guarantee that basic functionality works properly.

Keywords*— Regression Testing, Test case generation, Unified Modeling language, Steiner Algorithm, Dynamic Programming, Backtracking.

I. INTRODUCTION

Testing of the Regression is the testing that program has not regressed, that is, functionalities that were functioning in earlier version are still working in novel version. At time of the software system maintenance or as the software changes, regression testing is the classy but certainly required task.

Since the software maintenance account cost for about whole software part of two-thirds, both managers of project and also researchers have to pay regression testing large attention. There have been regression testing numerous researches [1, 3, 4, 5]. Since regression testing is the fairly classy, researches mostly attention on how to decrease such cost. The topics conclude prioritization, test selection, minimization, etc.

The UML is a widely accepted standard for modeling software systems. It consists of a set of modeling concepts (primitives) to support and object oriented approach to software development. UML consists of a set of diagrams that model both static and dynamic behaviour of a system. Various aspects of the system are elaborated at different levels of abstraction using diagrams like use diagram of case, diagram of class, diagram of activity, diagram of sequence and diagram of state. As the current work involves functionality level, activity diagram is used. Activity diagrams elaborate scenarios related to each use case. Each functionality (use case) is elaborated using an activity diagram. The diagram models the main, alternate and exception scenarios with reference to the functionality[UML]. The number of scenarios generated from activity diagrams using automated generation is exhaustive.

A lot of work is available for generating regression test cases both, strategies of white-box and black-box. This work focuses on black-box testing where changes related to a functionality of a system are regression tested. Further, this work looks at selecting a minimum regression test set that acts as a precursor for elaborate regression testing. This minimal regression test set indicates those functionality that have defects and hence need rework before elaborate testing and those that may be elaborately tested as they have passed the minimal regression test. Given the high cost of software development and testing effort there is scope to introduce new ways of enhancing the testing process. In this direction, the paper proposes method that utilize an algorithm of Steiner tree to create the minimum regression test set.

II. BACKGROUND

2.1 Regression Testing Technologies

Scientists had addressed testing of the regression for various years and also research on regression testing conclude a extensive topics variety which cover the above level.

2.1.1 Test environments and Automation

Hoffman [8], Brown and Hoffman[9], Ziegler, Grasso, and Burgermeister [10] define the test atmospheres and automation challenges of the regression testing procedure[1]. Their aims are to increase system of quality and also costs maintenance by systematic regression testing. They tried to describe a common regression test procedure and also tried to the use scripts automate test cases creation and performance. Such as, [8] defines their own test script language which can be used to the define test cases. Then they utilized test program creator PGMGEN to generate test drivers in the C language.

2.1.2 Test Suite Management

Lewis[16], Hartmann and Robson[17], Taha, et al. [18], and Harrold et al. [19] focus on the issue of test suite management[1]. At whatever point changes happen, a test's piece cases will be chosen from the first experiments, some piece of the experiments are out of date and should be erased, and for the new usefulness, new experiments ought to be included.

Every one of these progressions ought to be overseen amid the process of regression testing. [17] proposed a particular retesting tool. The device has a test library which isstores the experiments and test information, and it would naturally get criticism as to the effect of the changesof programming, including a complete reanalysis of the objective framework and the extraction of reusable experiments from the current test library, and the test subset determination information to revalidate the given changes.

Besides, tool would offer, if required, proposals as to the any extra tests that may be need to exercise improvements or novel information. Lastly, tool will be restore each improved or novel test cases and information into the test library. [18] conserves test suites through applying knowledge flow analysis tool to test cases partition into relevant, non relevant and invalid programs.

Thus, the revalidating cost a program following alteration will decrease. [19] suggested a procedure to select a representative test cases set from a test suite that gives the similar coverage as the complete test suite. The selection was achieved through classifying and then removing redundant and also test cases of obsolete. The representative set changes original test suite and thus, potentially creates a smaller test suite.

2.1.3 Reuse of existing test cases

2.1.3.1 Retest-all

At the point when the project is adjusted, by and large the analyzers have two principle methodologies to test the altered system. One is that select piece of the experiments from the first test suites so as to diminish the expense. The other is to rerun all the first experiments which is known as retest-all procedure. [2, 21] present the retest-all system: re-utilize all beforehand created test suite T, executing on the adjusted project P'.

2.1.3.2 Regression test selection

As depicted in 2.2.3.1, rather than rerunning all the experiments, the regression test determination advancements attempt to decrease the expense by picking a subset T' of T, and just run T' on the altered system P'. The presumption here is that the choice's expense procedure won't exceed the execution of the extra testcases (the experiments which are not chose by the technology).[1] did a review of relapse test choice advances. Totally technologies of the 12 test selection are studied, from data-flow technology to the technology of symbolic execution, and so on.

2.1.3.3 Test case prioritization

Regression testing is classy and in the order to decrease such cost, scientists have to complete numerous works other than test selection technologies. [22, 23, 24] address technology of the test case prioritization issue. They order (prioritize) the test cases through various measures. Then in cycling testing of the regression, test cases will be utilized to test the altered program P' according to order, so that "better" test cases can run first. The basic aim of the increase prioritization the rate of fault detection (how rapidly test suite can be detect faults at the time test procedure), or, growth the code coverage rate (how rapidly test suite can growth the program coverage). For such as, let t1, t2, t3 be the three different test cases.

2.1.3.4 Test Suite Reduction

[25, 26, 27] attention on test suite reduction methodology. They try to permanently eliminate test cases from test suite so that future regression testing cost will reduced and the span of test suite can be controlled. For instance, [27] presents a strategy to choose an agent set of experiments from a test suite which gives the same scope as the entire unique test suite. They utilize the information stream system to break down the scope. They first distinguish experiments into three classes helpful, repetitive and outdated, and after that kill the excess and out of date experiments in the test suite. The rest agent experiments supplant the first test suite. What's more, therefore, a possibly littler test suite is delivered.

The regression test selection techniquehas been used in [6][7][14] to select a subset of test cases. The objective isto check if the modified program has the same behaviour as the previous acceptable version of the program runningon T, a set of test cases [5]. UML activity diagrams have been used for specification. In [22], the authors present aframework for analyzing regression test selection technique. Four categories are included in the framework :inclusiveness, precision, efficiency and generality. The authors analyze the several method on the four factors. In[7], a CFG is constructed for a program or procedure and it altered version. CFG is used to the select tests thatperform changed code from the original test suite. Risk based test selection for regression testing is the focus ofwork done by Chen et al [15]. Risk information pertaining to test cases is provided by test engineers usingexperience gained. Risk metrics are used to measure quality of the test suite. Risk is based on the cost of each test case(valued by both customer and test engineer) as well as severity. Risk exposure (RE), a product of cost andseverity is taken as the basis for test selection. Scenarios that cover most critical cases are considered first.

2.2 UML Diagrams

UML is a widely accepted standard for modeling object oriented software systems. Being a semi-formal language,it is widely used to specify requirements and depict design of the software. UML provides diagrams to represent thestatic as well as the dynamic behaviour of a system. Class, component and deployment diagrams are used torepresent the static behaviour of the system whereas activity, sequence and state diagrams are used to represent thedynamic behaviour. Scenarios repesent the sequence of events in a software system and defines a system'sbehaviour. Use Case, activity,

sequence and collaboration diagrams can be used to represent scenarios. Eachscenario can be said to represent a requirement goal of the system. In practice, generation of scenarios is mostlydone manually making it labor-intensive and error-prone. Hence, there is a need to generate test scenarios to achievetest adequacy and to ensure software quality [28]. Each scenario is considered as a test case. In this regard,automation of test case generation gains importance.

An activity diagram consists of activities and transitions, showing the flow of control from activity to activity asshown in Figure 1 for the functionality 'Customer does Online Shopping'. Online customers can srowse or search items, show particular item, add a thing to the shopping cart, present and also update shopping cart and do checkout. Likewise, clients can be see the shopping cart whenever. Checkout is accepted to incorporate client enlistment and login.. Eachaction is represented using an activity construct and the flow of control from one activity to another, representedusing arrows (edges). UML activity diagrams are developed using two types of nodes, namely, action nodes andcontrol nodes as shown in Figure 1. Action nodes include Activity, CallBehaviourAction, SendSignal andAcceptEvent. Control nodes include InitialNode, FinalNode, Flow-Final, Decision, Merge, Fork and Join [29].

2.2.1 Usecase Diagram:-

In the software engineering and systems, a use case is a level list, typically describing interactions between a role (known in UML as an "actor") and a system,

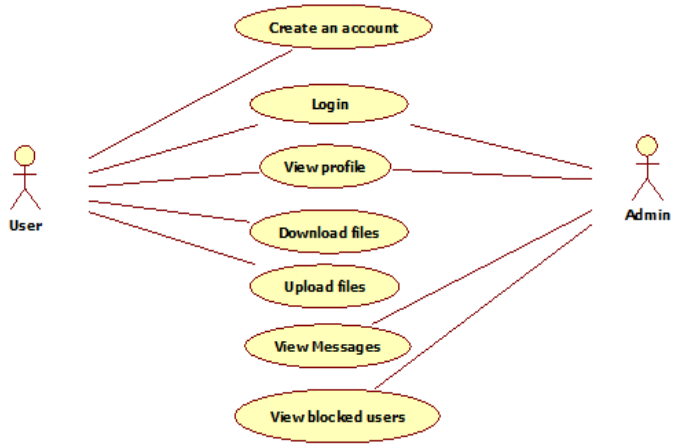

Fig 1. Use Case Diagram

2.2.2. Activity Diagram:-

Activity diagrams are graphical workflows stepwise activities representations and also actions with support, concurrency and iteration. Activity diagrams have been used to generate test cases. A path is defined as a set of nodes and edges startingfrom the initial state to the final state. Both true and false branches of all conditions are executed.

Definition: An Activity diagram D is a tuple,D = (A, T, F, J, R,a_1,a_F) where

A = a_0, a_1, a_2, ..., a_m is a finite set of activities (nodes),T = t_0, t_1, t_2, ..., t_n is a finite set of transitions (edges),

F = f_1, f_2, ... f_k is a finite set of forks,J = j_1, j_2, ..., j_k is a finite set of joins,

R ⊂(A X T) ⊂(T X A) is the flow relation,a0 is the initial state, anda_m is the final state.

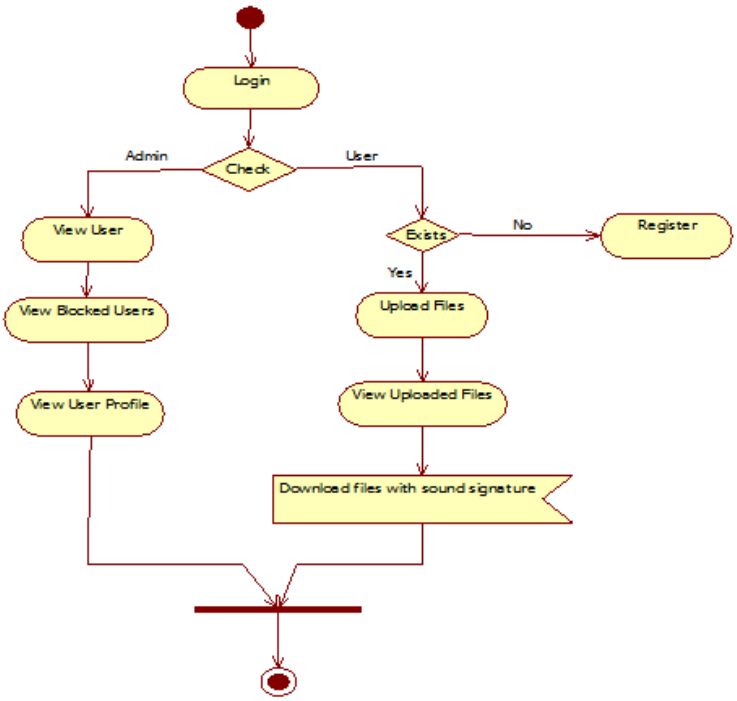

Fig 2. Activity Diagram

2.2.3. Sequence Diagram:-

A sequence figure in UML is an interaction diagram type that show how procedures operate with one another and in what order. It is a construct of a data Sequence Chart.

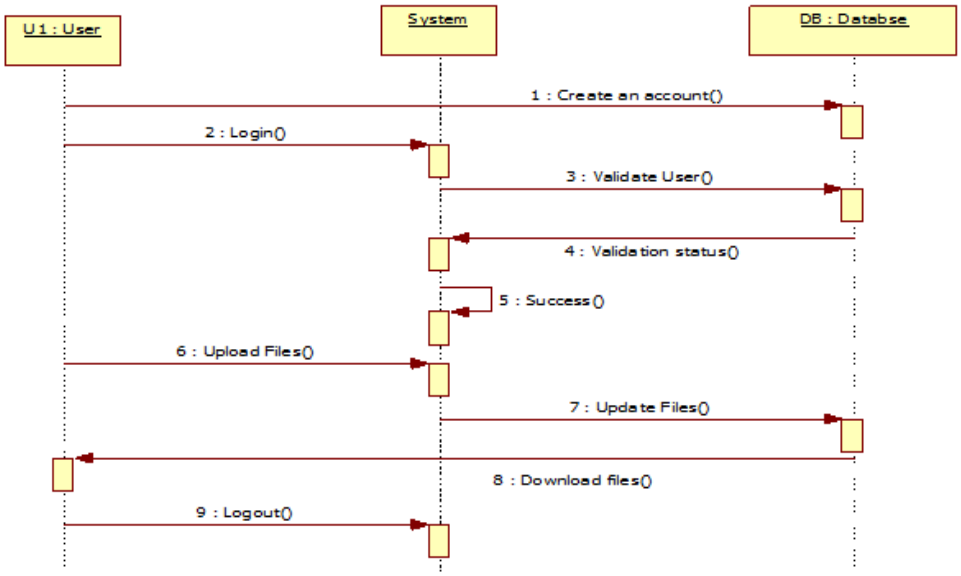

Fig 3. Sequence Diagram

2.3 Dynamic Programming:-

Dynamic programming is the optimal method complex problem transforms into the number of simpler issues; its vital characteristic is the optimization procedure nature of multistage. Further so than optimization method define earlier, dynamic programming gives a common framework for analyzing numerous issue kinds. Within this framework an optimization method variation can be acted to solve the specific features of a more common formulation. Generally creativity is need before we can identify that a specific issue can be cast efficiently as a dynamic program; and often subtle insights are needed to reorganize the formulation so that it can be solved efficiently.

With the help of common insight, giving into the dynamic programming method through treating a simple example in few detail. We then provide a dynamic programming, formal characterization under certain, followed through an in-depth such as dealing with the expansion of the optimal capacity. Other different topics covered in the chapter conclude the future returns discounting, the relationship between the dynamic-programming issue and in networks shortest paths, an such as a continuous-state-space problem, and dynamic programming introduction under uncertainty [30].In the linear programming contrast, there does not exist the standard mathematical formulation of "the" dynamic programming problem. Rather, dynamic programming is a common type of the problem solving method, and the specific equations used must be developed to fit all conditions. Therefore, a certain ingenuity degree and insight into the dynamic programming problems common structure is need to recognize when and how a problem can be solved through dynamic programming techniques.

2.3.1 A Prototype Example for Dynamic Programming

The STAGECOACH PROBLEM is a problem, particularly constructed1 to the show features and to dynamic programming terminology introduce. It concerns an imaginary fortune seeker in the Missouri who west to join decided the California gold rush at the time of the mid-19th century. The journey would need traveling through the stagecoach from an unsettled country where there was serious danger of attack through marauders.

Though his initial point and endpoint were fixed, he had a significant choice as to which level to travel with the route. The probable routes are present then in the Fig. 4, where all state are presented through a circled letter and the travel direction is always from left to right in figure. Thus, four different level (stagecoach runs) were need to travel from his embarkation point in the level *A* (Missouri) to his endpoint in level *J* (California). This fortune seeker was the prudent man who was fairly concerned about his safety. After some idea, he thought of a somewhat smart method for deciding the most secure course. Extra security strategies were offered to stagecoach travelers. Since the approach's expense for taking any given stage mentor run depended on a watchful assessment of the security of that run, the most secure course ought to be the one with the least expensive aggregate life coverage arrangement.

The cost for the standard policy on the stagecoach run from state *i* to state *j*, which will be denoted by *cij*, is

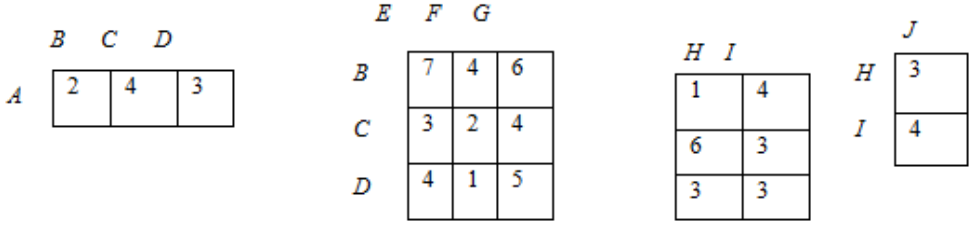

Fig 4.Costs for Various States

We shall now focus on the question of which route minimizes the complete cost of the policy.

Solving the Problem

First, note that the selecting short sighted method the inexpensive run provide through all successive level need not yield an complete optimal decision. Following this approach would provide route *A _ B _ F _ I _ J*, at a complete cost of 13. However, sacrificing a little on one level may be permit greater savings thereafter. Such as, *A _ D _ F* is cheaper complete than *A _ B _ F*.

One possible method to the solving this problem is to use trial and error.1 However, various possible way is large (18), and consuming to calculate the whole cost for all route is not an appealing task.

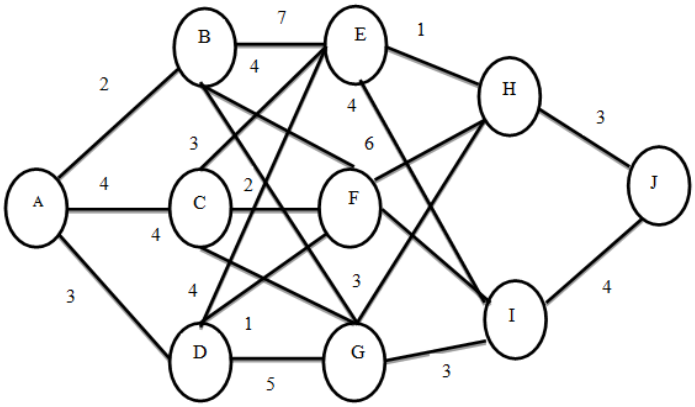

Fig 5. Graph Representation of Nodes

Fortunately, dynamic programming gives a solution with the much less effort than thorough enumeration. (The computational savings are huge for bigger forms of this problem.) Dynamic programming starts with the original problem small portion and also discovers the optimal solution for this slighter problem. It then slowly increases the problem, finding the present optimal solution from preceding one, until main problem is solved in its total.For the stagecoach issue, we start with the smaller problem where the fortune seeker has nearly completed his journey and has only one more stage (stagecoach run) to go. The obvious optimal solution for this smaller problem is to go from his current state (whatever it is) to his ultimate destination (state *J*). At the each subsequent iteration, the problem is enlarged through increasing with the 1 the number of level left to go to complete the journey.For this issue of enlarged, optimal solution for where go to another from all possible level can be found relatively simply from outcomes found at the preceding iteration. The specifics conclude in implementing this method follow.

2.4 Backtracking

Backtracking is the common algorithm for finding each (or some) solutions to the few computational problems, particularly constraint satisfaction problems, that incrementally builds solutions candidates, and abandons all partial candidate *c* ("backtracks") as soon as it concludes that *c* cannot probably be complete to the valid solution.

The typical textbook such as backtracking use is the eight queens puzzle, that asks for each arrangements of eight chess queens on a standard chessboard so that no queen attacks any other different. In the general backtracking method, the partial candidates are in *k* queens arrangements in the first *k* rows of the board, each in various rows and columns. Any partial solution that conclude two different mutually attacking queens can be abandoned [31].Backtracking can only for problems apply purpose which admits basic idea of a "partial candidate solution" and a comparatively rapid test of whether it can probably be finished to the valid solution. It is unusable, such as, for locating a provide value in an unordered table. When it is applicable, however, backtracking is frequently much faster than the brute force enumeration of very complete candidates, since it can remove a big amount of candidates with a single test.

In order to apply backtracking to the particular class of problems, one must give the Information *P* for specific instance of the problem that is to be solved, and also six dfferent procedural parameters, *root*, *next*, *reject*, *accept*, *first*, and *output*. These techniques should take the instance information *P* as a parameter and should do the following:

1. *root*(*P*): return the partial candidate at the root of the search tree.
2. *reject*(*P*,*c*): return *true* only if the partial candidate *c* is not worth completing.
3. *accept*(*P*,*c*): return *true* if *c* is a solution of *P*, and *false* otherwise.
4. *first*(*P*,*c*): generate the first extension of candidate *c*.
5. *next*(*P*,*s*): generate the next alternative extension of a candidate, after the extension *s*.
6. *output*(*P*,*c*): use the solution *c* of *P*, as appropriate to the application.

Backtracking is a refinement of brute force method, which methodically solution searches to the problem among each presented options. It does so through assuming that solutions are represented through vectors (v_1, ..., v_m) of values and through traversing, in a depth first manner, vectors domains until the solutions are found.

When invoked, algorithm starts with an empty vector. At all level it extends the partial vector with a novel value. Upon reaching a partial vector (v_1, ..., v_i) which can't represent a partial solution, the algorithm backtracks through eliminating the trailing value from the vector, and then proceeds through trying to extend the vector with alternative values.


```
Algorithm: Backtrack (v1,...,vi)
Input: input vertices v1,..., vi;
Output: Solution to the problem.
    1.    IF (v1,...,vi) is a solution THEN RETURN (v1,...,vi)
    2.    FOR each v DO
    3.        IF (v1,...,vi,v) is acceptable vector  THEN
                sol = backtrack(v1,...,vi,v)
    4.        IF sol != () THEN RETURN sol
            END
  END
```

2.5 Dynamic Steiner Tree Algorithm

Definition: Given a directed graph, G, a set of vertices, V, and edges, E, the goal of the directed Steiner tree problem is to find a minimum cost tree in a directed graph G = (V,E) that connects all terminals X to a given root r [32][33].

The Steiner Tree is distinguished from the Spanning Tree in that while a spanning tree spans all vertices of a given graph, a Steiner tree spans a given subset of vertices. In the case of the Steiner minimal tree problem, vertices are divided into two parts : terminals and non terminals. Terminals are the given vertices that must be included in the solution. Non terminals may be included if necessary to connect terminals. The cost of a Steiner tree is the total edge weight. In order to reduce the cost, non terminal vertices may be included.

Let G = (V, E, w) be an undirected graph with non-negative weights of edge. Given a set L ⊂ V of the terminals, a Steiner minimal tree is the tree T ⊂ G of minimum complete edge weight such that T includes each vertices in the L. Problem: Graph Steiner Minimal Tree Instance: A graph G = (V, E, w) and a set L ⊂ V of terminals Aim: Find a tree T with L ⊂ V (T) so as to minimize w(T).

The decision version problem of the SMT was present to be NP-complete through the transformation from the Exact Cover through 3-Sets problem [34]. We shall focus on some approximation results. A well-known technique to approximate an SMT is to use a MST.

```
Algorithm: MST-Dynamic_Steiner
Input: A graph G = (V, E, w) and a terminal set L ⊂ V ; root node r;
Output: A Dynamic Steiner tree T.
1: Set the r and the terminal nodes in the graph.
2: Construct the metric closure for the terminal nodes.
3: Set the tree stack as empty i.e. T ← ∅.
4. Set T ← r.
4: for edge e = (u, v) ∈ E(TL) each i = 1 : TL in a depth-first-search order of TL do
        4.1: β(T, R, L1) = {min β(T {x}, R, i) + cost(e, x); x ∈ T, e ∈ (T {x} ∪ R)
                If(no path further) then
                        Backtrack(v)
                Else
                        Find a shortest path P from u to v on G.
5: Output T.
```

Mostly, we change all edge in the TL with corresponding shortest way at the level 4. But if there are two different vertices previously in the tree, adding the way will outcome in cycles. In this case we only insert the subpaths from the stations to the vertices previously in the tree. It avoids any cycle and guarantees that terminals are conclude. As a outcome, we can see that the algorithm returns a Steiner tree.

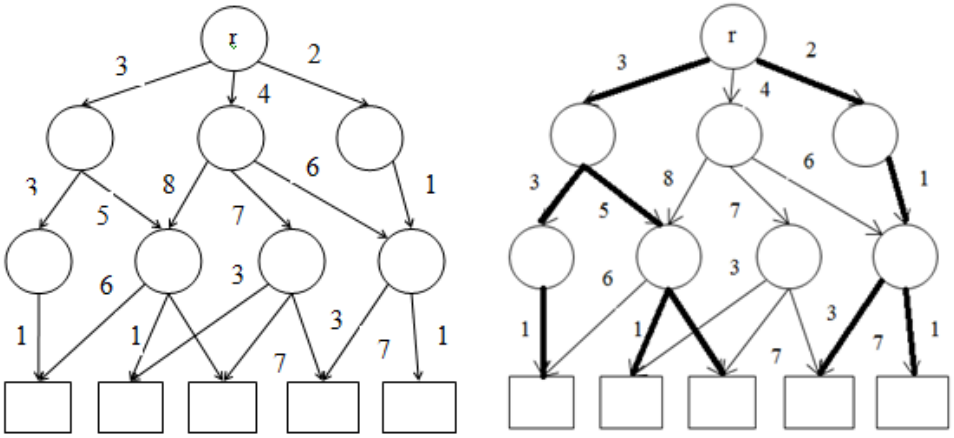

Fig 6. Dynamic Steiner Tree G

2.6 Generating Minimal Test Cases- The Approach

This work presents an approach to generate minimal regression test cases for a functionality which will give an indication of whether defects have been fixed. In this work, an activity diagram is used to elaborate functionality.

Preprocessing

The activity diagram is converted into a Control Flow Graph (CFG). Each activity in the activity diagram is a node in the CFG and a control flow in the activity diagram corresponds to an edge. In the case of loops, each loop is unfolded to a maximum of two iterations. For fork join constructs that represent concurrency, it is considered as one node in this work. The weight for each edge is calculated using a measure e.g. distance measure.

Given the Control Flow Graph (CFG) elaborating a functionality, the next step is to define terminal nodes. The following rules are followed:

a. Consider as terminal the root node of the CFG.
b. Consider as terminals all the nodes that provide output of value to the user.
c. Add as terminals all nodes where change has been made.

2.7 Algorithm

Given the Control Flow Graph (CFG) elaborating a functionality, the next step is to define terminal nodes. Thefollowing rules are followed:

a. Consider as terminal the root node of the CFG.
b. Consider as terminals all the nodes that provide output of value to the user.
c. Add as terminals all nodes where change has been made.

Given that the technique is black-box, the first two rules ensure that all I/O nodes are included. The third ruleincorporates change that has been done due to a bug fix or change in functionality. It is necessary that such nodes beadded to the minimal set of test cases as they should be part of the regression test cases.

Algorithm : GTC (Generate Test Cases)
Input: Activity diagram, A
A graph G = (V, E, w) and a terminal set L ⊂V
Output: TC, a set of Test Cases
1. For each node in activity diagram, A do
    1.1. If node is an InitialNode, FinalNode, Activity, Decision node, convert into node in CFG.
    1.2 If node is a Fork, find corresponding Join and group all nodes inside the Fork-Join into one node.
2. For each edge in activity diagram, A do
    2.1 If edge forms a loop, unfold the loop upto two iterations adding nodes and edges to CFG.
    2.2 Else, add edge in CFG.
3. Calculate edge value using a measure.
4. Define the terminal edges.
5. Use Dynamic_Steiner Tree algorithm, to generate Dynamic Steiner Tree, T.
6. Generate TC, the set of test cases by generating independent paths from the start node to end node of T.

2.8 Case Study

Consider an example an activity diagram that has been shown in fig 7 The equivalent graph is shown in Figure 3b.

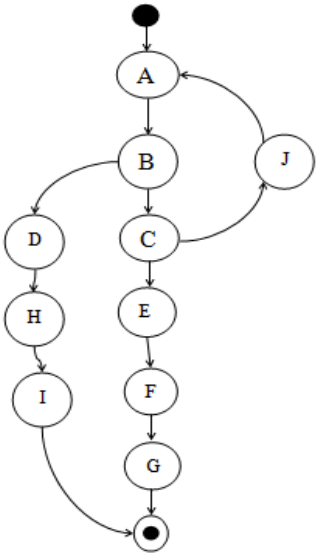

Fig 7. CFG for Activity Diagram

Given the CFG 'G', in Figure 3, the next step is to unfold cycles if any, and convert any fork-join constructs into a single node. The unfolded graph is shown in Fig. 4a.

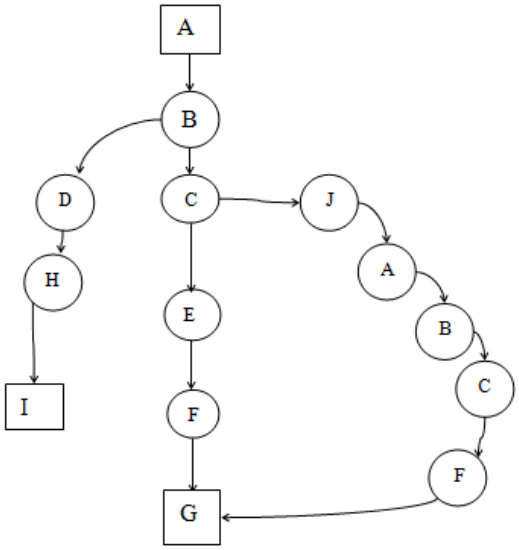

Fig 8.CFG in Unfolded with edges and weights

Figure 4 b shows the terminal edges defined. 'D' is considered as a terminal edge as a change was incorporated in the condition. 'D' is added to the set of terminals as it is an activity that must be exercised as part of testing.
Applying the Steiner minimum tree algorithm, it is found that minimal path to the terminal activities are the following :
Thus, two test cases form the minimal test case to test this functionality for regression testing.

1. A – B – C– E – F– G
2. A – B – D – H – I

III. CONCLUSIONS

In this paper we have discussed about the black box testing approach for regression testing. The main motive behind this approach is to create minimal test cases using regression testing in the best possible way. UML has been used to model requirements. Each functionality is elaborated using activity diagrams. Given a set of terminal nodes, the Steiner tree algorithm was used to generate the minimal regression test set. The approach is advantageous as the nodes that have a change can be added to the set of terminals and hence included in the test set.
The approach has been tested on individual functionality elaborated through activity diagrams. This work however imposed certain constraints. First, a loop was unfolded twice to contain the size of the graph. Secondly, a fork-join construct which represents concurrent activity is considered as a single node. The size of the case studies undertaken does not reflect the scale of actual software systems. Future work involves exploring further the following issues:
a. build case studies to understand the efficacy of the technique on large functionality; b. explore the use of other parameters like cost, risk, etc to calculate weight of edges; c. mechanism for testing of fork-join constructs and d.development of a tool to support the technique.

REFERENCES

[1] G. Rothermel and M. J. Harrold, Analyzing Regression Test Selection Techniques, IEEE Transactions on Software Engineering, V.22, no. 8, August 1996, pages 529-551.
[2] K. Onoma, W-T. Tsai, M. Poonawala, and H. Suganuma. Regression testing in an industrial environment. Communications of the ACM, 41(5):81–86, May 1998.
[3] G. Rothermel, S. Elbaum, A. G. Malishevsky, P. Kallakuri, and X. Qiu, On Test Suite Composition and Cost-Effective Regression Testing; ACM Transactions on Software Engineering and Methodology, V. 13, no. 3, July, 2004, pages 277-331.
[4] G. Rothermel and M. J. Harrold, A Safe, Efficient Regression Test Set Selection Technique, ACM Transactions on Software Engineering and Methodology, V.6, no. 2, April 1997, pages 173-210.
[5] S. Elbaum, .A. Malishevsky and G. Rothermel, "Test case prioritization: A family of empirical studies", IEEE Trans. Software Engg. , vol. 28, no. 2, pp. 159-182, Feb. 2002.
[6] D.Hoffman and C.Brealey. Module test case generation. IN Proceedings of the Third Workshop on Software Testing, Analysis, and Verification, pages 97-102, December 1989.
[7] .P.A Brown and D. Hoffman. The application of module regression testing at TRIUMF. Nuclear Instruments and Methods in Pysics Research, Section A, . A293(1-2):377- 381, August 1990.
[8] .J.Ziegler, J.M. Grasso, and L.G. Burgermeister. An Ada based real-time closed-loop integration and regression test tool. In Proceedings of the Conference on Software Maintenance - 1989, pages 81-90, October 1989.
[9] R. Lewis, D.w. Beck, and J.Hartmann. Assay – a tool to support regression testing. In ESEC' 89.2nd European Software Engineering Conference Proceedings, pages 487-496, September 1989.

[10] J. Hartmann and D.J. Robson. Revalidation during the software maintenance phase. In Proceeding of the conference on Software Maintenance.

[11] A.B Taha, S.M. Thebaut, and S.S. Liu. An approach to software fault localization and revalidation based on incremental dat flow analysis. In proceeding ot the 13th Annual International Computer Software and Applications Conference.

[12] M.J. Harrold, R. Gupta, and M.L. Soffa. A methodology for controlling the size of a test suite. ACM Transactions on Software Engineering and Methodology.

[13] S. Elbaum, A. G. Malishevsky, and G. Rothermel. Test case prioritization: A family of empirical studies. IEEE Transactions on Software Engineering, 28(2):159–182, February 2002.

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

[15] W.E. Wong, J.R. Horgan, S. London, and H. Agrawal. A study of effective regression testing in practice.In Proceedings of the Eighth International Symposium on Software Reliability Engineering, pages 230–238, November 1997.

[16] T.Y. Chen and M.F. Lau. Dividing strategies for the optimization of a test suite. Information Processing Letters, 60(3):135–141, March 1996.

[17] M. J. Harrold, R. Gupta, and M. L. Soffa. A methodology for controlling the size of a test suite. ACM Transactions on Software Engineering and Methodology, 2(3):270–285, July 1993.

[18] J. Offutt, J. Pan, and J. M. Voas. Procedures for reducing the size of coverage-based test sets. In Proceedings of the Twelfth International Conference on Testing Computer Software, pages 111–123, June 1995.

[19] OMG. Unified modeling language (UML) Superstructure Specification, version 2.1. Technical report.

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

[21] D.D. Sleator and R.E. Tarjan. A data structure for dynamic trees. Journal of Computer and System Sciences, 26(3):362–391, 1983.

[22] Gurari, Eitan (1999). Backtracking algorithms "CIS 680: DATA STRUCTURES: Chapter 19: Backtracking Algorithms"

[23] Bang Ye Wu: A simple approximation algorithm for the internal Steiner minimum tree. CoRR abs/1307.3822, 2013.

[24] J. Buckner, J. Buchta, M. Petrenko, and V. Rajlich, JRipples: A tool for program comprehension during incremental change, in Proceedings of the 13th International Workshop on Program Comprehension (IWPC 05), pp. 149-151, May 2005.

[25] G. Rothermel and M. Harrold. A safe, efficient regression test selection technique. ACM Transactions on Software Engineering and Methodology, 6(2):173-210,April 1997.

