Rafi: Parallel Dynamic Test-Suite Reduction for Software

Najneen
Rajasthan Technical University Kota,
Rajasthan

***Abstract*: A trend in software testing is reducing the size of a test suite while preserving its overall quality. For software, requirements and a set of test cases are given. Each test case is covering some requirements. In this paper, our goal is to find the method for test-suite reduction (TSR) to calculate the minimal subset of test cases that cover all the requirements across versions. While this problem has gained significant attention, it is still difficult to find the smallest subset of test cases and widely used methods to solve this problem with only approximate solutions. In this paper, our goal is to find the greedy method for test-suite reduction (TSR) to calculate the minimal subset of test cases that cover all the requirements across versions. There are already existing exponential-time algorithms and greedy algorithms to find the TRS in a version-specific and across versions. We proposed a new parallel greedy heuristic method RAFI to find minimal test sets in across versions. Our approach shows that: (i) RAFI is much faster than the exponential time algorithms and approximately 1000x time faster than the traditional greedy method. (ii) RAFI method achieves roughly the same reduction rate compared to the traditional greedy method**

Keywords: Test-Suite Reduction, Traditional Greedy, Traditional Greedy parallel, Test-Coverage graph, Dynamic Reduction, Flower, RAFI, Reduced test set.

1 INTRODUCTION

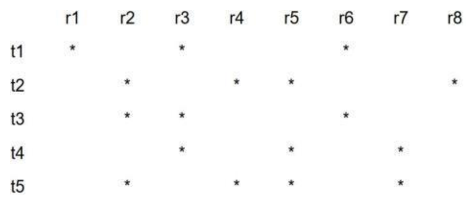
We can testify the quality of software products is a massive problem for the software industry, and software failure is a significant loss for a software company. Software testing is an essential technique for software quality solace, and it is a challenging but necessary process of growth information technology. If we increase software size, then we increase the number of test cases, and automatically cost will be increased. If we design software test cases, then we have to fulfill the demand of the software quality. So a cardinal set of the test cases is redundancy. We are selecting a minimum cardinal subset for selecting a test case of requirements. Test Suite Reduction(TSR) [3 - 5] can solve the problem of the condition and decrease the activity load, so automatically reduce the cost of software testing. Test Suite Reduction[6 - 8] comes under regression testing [1, 2]. In this paper, we can satisfy how we select a minimum number of test cases and fulfill all the requirements. It is the key to this topic test casereduction**.**

Next, every software has some defect, less feature and some BUGs, So we are releasing the new version of software with BUG fix or additional feature. Before releasing the feature again we have to test our software for quality and correctness purposes. For this, the normal way is again to find the minimal test set to execute our software. Then it will become more costly to again find the reduced test set. In this paper we propose a RAFI method to find the reduced test set across the versions by using the information of added features and previous version's reduced test set.

Now we will understand what the test suite reduction problem is? We will define the test suite reduction problem [10, 11] as:

1. If T = {r1, r2, r3, r4, r5,...... rn} are the requirements of the software.
2. If R = {t1, t2, t3, t4, t5,...... tn} are the test cases of software.
3. If S = {(t,r)} R is the relationship between test cases and the requirements.

Each requirement is covered by some test cases and each test case will cover some requirements. In the example (see Fig. 1), * indicates that there is a relationship between the corresponding requirement and the test case [9]. The above notation we can say the array representation of the coverage graph. Coverage Graph is the graph which will show the relationship between the test cases and the requirements. We can also represent this relation by bipartite graphs [11 - 13].



	r1	r2	r3	r4	r5	r6	r7	r8
t1	*		*			*		
t2		*		*	*			*
t3		*	*			*		
t4			*		*		*	
t5		*		*	*		*	

Fig. 1. Test Cases Cover the Requirements

In the above, there are a lot of test cases which are covering all the requirements. For example {t1, t2, t4, t5}, {t1, t2, t4}, etc. But we observe that the second subset is minimal than the first one. So we can say that subset {t1, t2, t4} is the minimal subset which is covering all the requirements. In this t1 is covering {r1,r3}, t2 is covering {r2, r4, r5, r8} and t3 is covering {r3, r5, r7}. So, we can say that {t1, t2, t4} is covering all the requirements and hence the reduced test set. There are many approaches for TSR, some are exponential and some are polynomial-timeapproaches.

2 BACKGROUND AND RELATED WORK

In this section, we talked about some existing methods of implementing Test Suite Reduction techniques. For some of them are, Flower method and the Traditional greedy method.

2.1 Exponential Time Algorithms

There are so many algorithms for Test Suite reduction whose time complexity is exponential and giving the exact result, Flower method is one of the methods.

FLOWER Method

Flower method [14] comes under the exact approach. We can get the optimal solution by using this method. In this method, we solve a problem as a flow network [15] and use the Ford Fulkerson algorithm [15]. The flower method represents a flow network where every edge has a capacity. Also, given two vertices source S and sink D, in the graph, find out the maximum possible flow from S to D. Flow on edge doesn't exceed the given capacity of the edge. Inflow is equal to outflow for every vertex except S to D. time complexity of this algorithm is exponential which is equal to O((2^|test cases|)*edges * |requirements|) which seems an exponential time approach. This Flower method is an integer linear programming (ILP) [16 - 18]

2.2 Polynomial Time Algorithms

There are so many algorithms for Test Suite reduction whose time complexity is polynomial and giving the approximate result, traditional greedy method is one of the methods.

Tradition greedy method

Traditional greedy [19] is a heuristic approach to find the reduced test set that covers all requirements. In this, we will not get the exact result that we are expecting but we can get the approximate solution for this. In this approach, we iteratively select a test case that will cover all the maximum number of remaining requirements. We performed this iteratively until all requirements got not covered. A traditional Greedy method is a heuristic approach, so time complexity is polynomial. Its time complexity depends on the number of requirements, the number of test cases, and the relationship between the test case and requirements. Time complexity of Basic Greedy method is O(r * t * minimum(r, t)) where t is the number of test cases and r is the number of requirements.

3 OUR APPROACH: RAFI SEQUENTIAL METHOD

In every software we are releasing the new version of that software. So before releasing the new version of that software, that software must pass all the test cases for the quality purpose. Due to the new version of software our old coverage got changed. Some edges can be added or deleted. If the edges are added then we can say that some test cases are now covering the more requirements and if edges got deleted then we can say that some tests are not covering the requirement which they previously covered. Now we have to find the reduced test set in the new coverage graph. In this section, we Implemented the RAFI method to find the Reduced test set in the new Coverage graph. In the RAFI method we are using some information and those informations are 1) a reduced test set of the old coverage graph 2) edges got modified (added and deleted).

There are two types of edge modification in the old coverage graph.

1. Edge deletion
2. Edge addition

In this we will see how we are handling the edge modification to find the reduced test set and how we are taking the advantage of the old reduced test set.

3.1 Handel edge deletion:

There are two type of edge deletion are possible

1. Deleted edge belongs to old reduced test set
2. Deleted edge not belongs to old reduced test set

Deleted edge belongs to old reduced test set

Below graph shows an example of edge deletion which belongs to the old reduced test set. When we ran the old coverage graph in a traditional greedy method then we found that the reduced test was {t1, t2}. After deletion of the edge {t1, r1}. We found that t1 belongs to the old reduced test set and r1 is now uncovered by the old reduced test set so we have to cover the r1. For this first we will check whether r1 is covered by the current old reduced test set or not. In the below example r1 is not covered by any selected test case {t1, t2} so we will add a test case which will cover this r1 requirement, because t3 is covering the r1 requirement so added this t3 test case in the old reduced test case now reduced test case will become {t1, t2, and t3}. Now we try to optimize the reduced test set. We had added the t3 test case because t1 is not covering the r1 so we try to cover all requirements without the presence of t1 test case in the new reduced test set. In this case we can cover all the requirements by {t2 , t3}. So new reduced test set will be {t2 and t3}

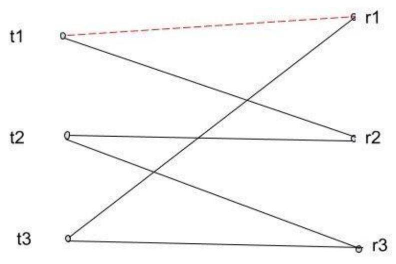


Fig. 2. new coverage graph after after deletion {t1, r1} edge

Deleted edge not belongs to old reduced test set

Below graph shows an example of edge deletion which does not belong to the old reduced test set. When we ran the old coverage graph in a traditional greedy method then we found that the reduced test was {t1, t2}. Now we deleted the edge {t3, r1}. We know that t1 does not belong to the old reduced test set. We found that r1 still got covered by the old reduced test set so nothing will be affected. The new reduced test set will be {t2 and t3} as the old reducedtest set.

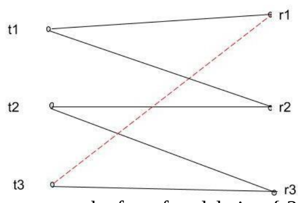


Fig. 3. new coverage graph after after deletion {t3, r1} edge

Algorithm

```
 Reduced test set as ReducedSet ;Delete
 the edge as DeletedEdge;
   delete DeletedEdge from CoverageGraph ;
   if DeletedEdge belongs in TestCase(t1) which is in ReducedSet thennothing to do
       for DeletedEdge ;
         else if DeletedEdge belongs in Requirement(r1) which iscovered
         by TestCase(ti) which is in ReducedSet then nothing to do for
         DeletedEdge ;
       else
          Add one TestCases which satisfy that Requirement(r1);end
                      if all requirements that covered by TestCase(t1)canbe
                      covered by ReducedSet - { TestCase(t1) } then
            ReducedSet = ReducedSet - { TestCase(t1) } ;end
     end
   return Reduced test set as ReducedSet ;
```

Line1 In the ReducedSet set we will store the final reduced test set. Line2 deleted edge we store the test case and the requirement relationship. Line4-5 If the deleted edge does not belong to the reduced test set then we will do nothing. Line6-7

If the deleted edge belongs to the reduced test set and r1 is still covered by another test case which belongs to the reduced test set then we will do nothing. Line9 If not then we will add the test case which covered r1. Line 11- 12 if all requirements which are covered by t1, get covered by the current reduced test set except t1 then we can remove t1 form the reduced test set. Line15 return reduced test set.

3.2 Handel edge addition:

There are two type of edge addition are possible

1. Added edge belongs to old reduced test set
2. Added edge not belongs to old reduced test set

Added edge belongs to old reduced test set

Below graph shows an example of edge addition which belongs to the old reduced test set. When we ran the old coverage graph in a traditional greedy method then we found that the reduced test was {t1, t2}. After addition of the edge {t2, r1}. We found that t2 belongs to the old reduced test set and r1 is now covered by the other test case which belongs to the old reduced test set so try to optimize the reduced test set. For this first we will check the cardinality of the test case which was covering the r1. If that cardinality is one then we can remove that test case form the old reduced test set. In the below example the cardinality of t1 is one so we will remove the t1 form the old reduced test set. In this case we can cover all the requirements by {t3} only. So new reduced test set will be {t3}

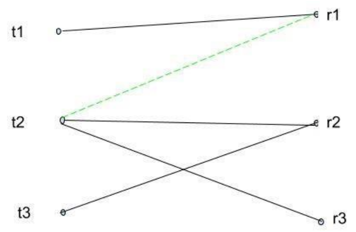


Fig. 4. new coverage graph after after adding {t2, r1} edge

Added edge not belongs to old reduced test set

Below graph shows an example of edge addition which does not belong to the old reduced test set. When we ran the old coverage graph in a traditional greedy method then we found that the reduced test was {t1, t2}. Now we added the edge {t3, r2}. We know that t3 does not belong to the old reduced test set. We found that r1 is not covered by another test case which belongs to the old reduced test set so nothing will be affected. The new reduced test set will be {t2 and t3} as the old reduced test set.

Algorithm

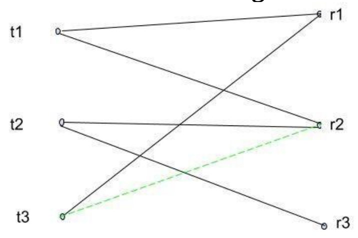


Fig. 5. new coverage graph after after adding {t3, r2} edge

```
1. Reduced testSet as ReducedSet ;
2. Add edge as AddEdge(e1) ;
3.Add AddEdge(e1) ( connected by Requirement(r1) and TestCase(t1) ) in
  CoverageGraph ;
4.    if AddEdge(e1) belongs to TestCase(t1) which is not in Reduced_Setthen
5.      do nothing for AddEdge(e1) ;
6.    else
7.      if AddEdge(e1) belongs to Requirement(r1) that is coveredby
          ReducedSet - { TestCase(ti) } and degree of TestCase(ti) is
                    one then
8.        ReducedSet = ReducedSet - { TestCase(ti) }
9.      else
10.         do nothing for AddEdge(e1) ;
11.     end
12.   end
13. return Reduced testSet as ReducedSet ;
```

Line1 In the ReducedSet set we will store the final reduced test set. Line2 deleted edge we store the test case and the requirement relationship. Line4-5 If the added edge does not belong to the reduced test set then we will do nothing. Line7-8 If the added edge belongs to the reduced test set and cardinality of the test case which was covering the requirement in the old reduced test set is one then we will remove that test case. Line13 return reduced test set.

4 EXPERIMENT EVALUATION

In this, we evaluated Flower Method, Traditional greedy sequential method, and Traditional greedy parallel method, parallel RAFI method. We evaluated different factors in different benchmarks.

- **RQ1:** Which method is the time-efficient Traditional greedy method or the Flower method.
- **RQ2:** How much can we reduce the size of the test suite?
- **RQ3:** Is the RAFI method is time efficient in the term to find the reduced test set compared with traditionalgreedy across versions?
- **RQ4:** Is the RAFI method efficient in the term of reduction compared with traditional greedy across versions?

4.1 Experiment Subject

In this, we will evaluate our experiment in the set of standard benchmarks (SIR). In this there are a total of eight benchmarks. tcas, totinfo, schedule, printtokens, printtokens2, replace, schedule2, space. From the below benchmarks, we can say that the requirements are between 58 to 1515. Total test cases are between 1052 to 13585. The total numbers of edges are between 12825 to 5713638. We will run our Flower method, traditional greedy method sequential and parallel method in these standard benchmarks. We can see the property of each benchmark from the below table.

Table 1. Standard Benchmarks

Sr.No.	Benchmark	Test Cases	Requirements	Edges
1	printtokens	4130	157	344020
2	printtokens2	4115	200	12825
3	replace	5542	229	628464
4	schedule	2650	100	203000
5	schedule2	2710	125	258929
6	space	13585	1514	5713638
7	tcas	1608	58	39413
8	totinfo	1052	132	88682

5 EXPERIMENT SETUP

Our aim is 1) Find the minimal test suite such that it can cover all the requirements. 2) Find the time taken to compute the minimal test suite for Flower, Traditional greedy sequential. 3) Speedup for RAFI when we vary the number of threads 5) Speedup for RAFI when we vary the percentage of adding edges 4) Reduction rate for RAFI when we vary the percentage of adding edges 5) Speedup for RAFI when we vary dynamic changes in edges 6) Reduction rate for RAFI when we vary dynamic changes in edges. We will run our all implemented methods in the standard benchmarks(SIR). For evaluating the RAFI, there are three parameters on which the performance can be varied. The parameters are

- ➢ The number of threads on which we run the RAFI.
- ➢ Structural change in the old coverage graph, mena percentage of edges got modified.
- ➢ In edge modification how much percentage edges are added and remaining percentage edges deleted We will evaluate our RAFI for these parameters, we will fix two parameters and will vary the remaining oneparameter.

We are using the machine as Operating System as Linux version 4.4.0-159-generic, Operating SystemType: 64-bit processor, RAM: 4 GB and Processor: Intel 2.60GHz*4

5.1 Result and analysis

In this, we will explore the result of the Flower method, traditional greedy sequential for finding the Reduced test setand RAFI parallel method for finding the reduced test set in the modified coverage graph.

Time Efficient

We run Flower and traditional greedy methods to find the minimal test set which can cover all the requirements. We run our experiment in the benchmarks given in the table1. We found the time complexity of the Flower method is exponential so it is taking the exponential time and failing to find the Reduced test set in a reasonable time. On the other hand, the traditional greedy method has a polynomial-time algorithm. So we were able to find the reduced test set in a few milliseconds. Hence we can say that the traditional greedy algorithm is capable of finding the reduced test set in a reasonable time while the Flower method fails to do this. We can refer to the below figure to find how much the traditional greedy method is taking to find the reduced test set for different benchmarks.

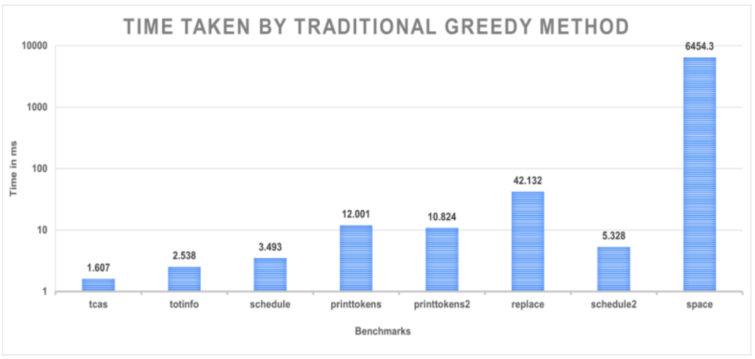


Fig. 6. Time taken by the traditional greedy method to find the reduced set size of the test cases for different benchmarks.

Reduced size of the test set

We ran our traditional greedy method for different benchmarks and found how much they are reducing the test set tofind the reduced test set to cover all the requirements. From the below image we can find the reduced set size of all benchmarks which we are using for evaluation.

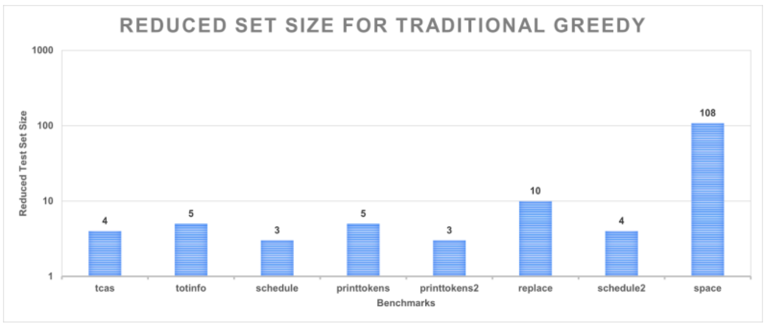


Fig. 7. Reduced set size of the test cases for different benchmarks.

Rate Of Speedup when we vary the number of threads

In this we will vary the number of threads from 1 to 8, percentage of dynamic changes in edges will be fixed which is 10% and fixed the 25% of edges added and 75% deleted. Now we calculated the speedup which is the ratio of time taken by Greedy and time taken by RAFI. We found that the maximum speedup we got when the number of threads is 4.

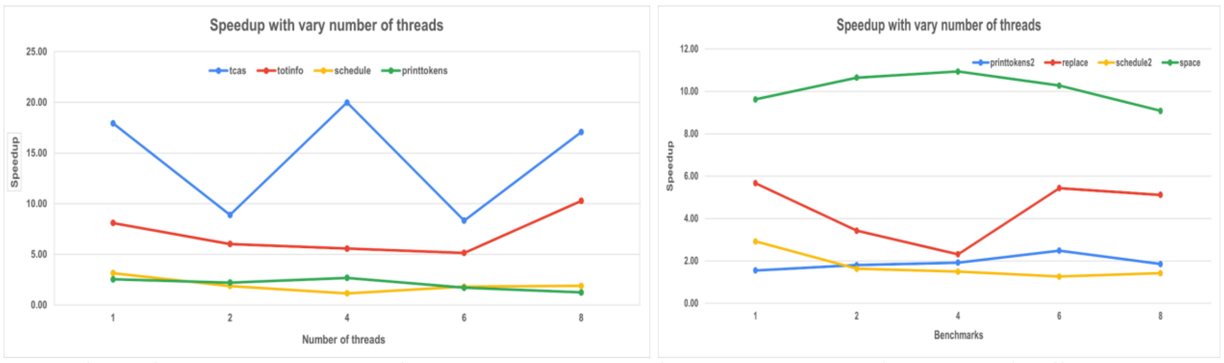


Fig. 8. Speedup of the Rafi method with comparison of traditional greedy parallel to find the reduced set size of the test cases fordifferent benchmarks.

From these two graphs, we can see that as we are increasing the number of threads we are getting speedup but after a certain time speedup got reduced. Because when the number of threads got increased then the most time we were wasting scheduling of the thread instead of doing required work.

Rate of speedup when we vary the percentage of adding edges

In this we will vary the percentage of edge addition from 0% to 100% and rest will be edge deletion, we fixed the number of threads to 4 and fixed percentage dynamic change to 1%. Now we calculated the speedup which is the ratio of time taken by Greedy and time taken by RAFI. We found the maximum speed 489 for schedule2 benchmarks. The maximum speedup we got when the percentage of adding edges was 100%.

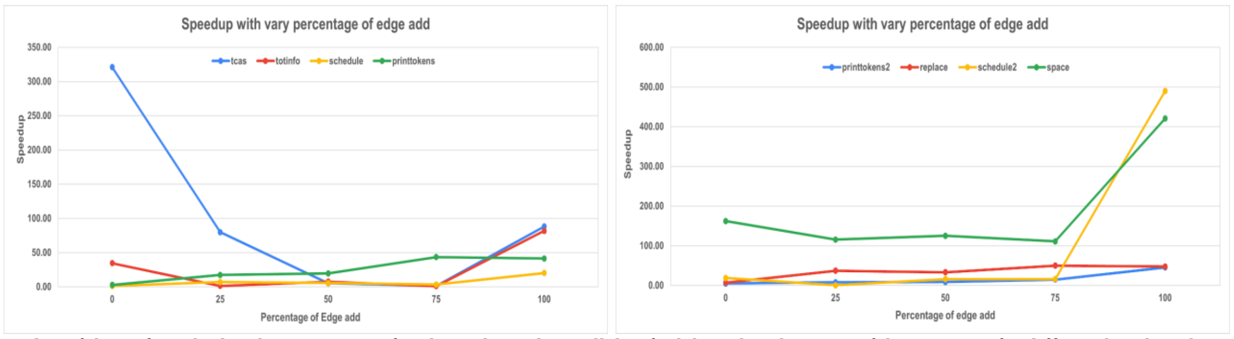


Fig. 9 Speedup of the Rafi method with comparison of traditional greedy parallel to find the reduced set size of the test cases for different benchmarks.

From these two graphs, we can see that as we are increasing the percentage of edge edition then speedgetting increased except the tcas benchmarks.

Reduction rate when we vary the percentage of adding edges

In this we evaluated the normalized reduced set size for the above three parameter parameters. Now we calculated the reduction rate which is the ratio of reduced test set size of traditional Greedy and reduced set size of RAFI method.

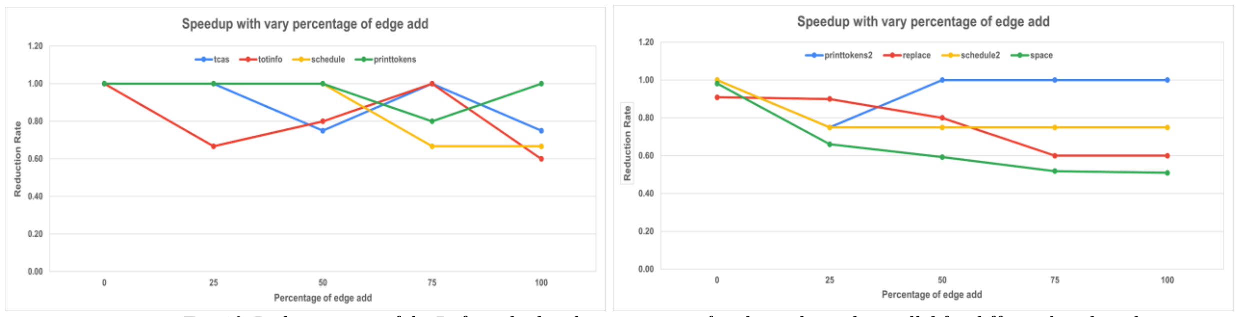


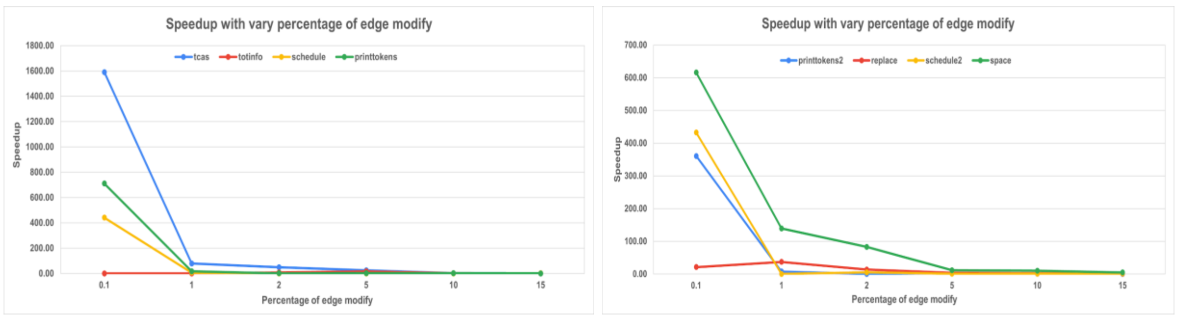
Fig. 10. Reduction rate of the Rafi method with comparison of traditional greedy parallel for different benchmarks.

From these two graphs, we can see that the reduction rate varies from 0.52 to 1. Still there is a possibility that we can get a reduction rate greater than one. The maximum speedup of reduction rate we are getting when the percentage of adding edges is 0% and deleting edges is 100%

Rate of speedup when we vary dynamic changes in edges

In this we will vary the percentage of dynamic changes in edges from 0.1 to 15%, we fixed the number of threads to 4 and fixed the 25% of edges added and 75% deleted. Now we calculated the speedup which is the ratio of time taken by Greedy and time taken by RAFI. We found that the maximum speedup we got when dynamic changes in edges is 0.1%.

Fig. 11. Speedup of the Rafi method with comparison of traditional greedy parallel to find the reduced set size of the test



cases for different benchmarks.

From these two graphs, we can see that maximum speedup we get for benchmark tcas, which is equal to 1590 and maximum speedup when the percentage of dynamic changes is 0.1%.

Reduction rate when we vary dynamic changes in edges

In this we evaluated the normalized reduced set size for the above three parameter parameters. Now we calculated the reduction rate which is the ratio of reduced test set size of traditional Greedy and reduced set size of RAFI method.

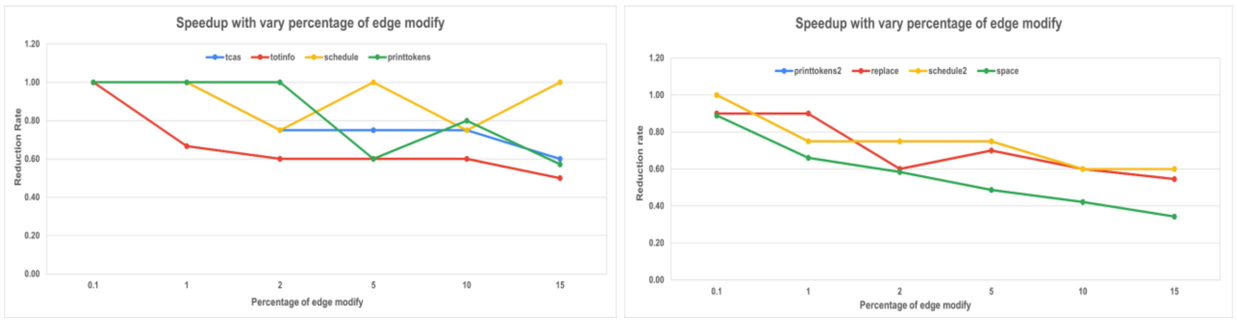


Fig. 12. Reduction rate of the Rafi method with comparison of traditional greedy parallel for different benchmarks.

From these two graphs, we can see that the reduction rate varies from 0.34 to 1. Still there is a possibility that we can get a reduction rate greater than one. The maximum speedup of reduction rate we are getting when the percentage of dynamic changes is 0.1%.

6 CONCLUSION AND FUTURE WORK

Given a set of test cases and the requirement of the software. Our aim was to find the reduced test such that it covers all the requirements. For this, we implemented some existing exponential methods such as Flower and some polynomial methods such as the Traditional greedy method. We run these methods in some standard benchmarks(SIR) and found that the traditional greedy method is able to find the reduced test set in a reasonable time but the Flower method failed to do this in a reasonable time. Further, we found that the traditional greedy method is able to reduce the test set up to 0.07%. Further we implemented the RAFI method to find the reduced test set in the modified coverage graph. Our aim was to find the reduced test set by using the old reduced test set and edges got modified such that it covers all the requirements. We run RAFI method and for comparison we run Traditional greedy method. We found that RAFI gives a reduced test set in very less time compared to Traditional Greedy, we got speedup up to 1590X. In the term of reduction rate, it varies from 0.38 to 1 for the RAFI method.

In future work, we can test different data structures like trees and adjacency lists to store the graph to reduce the time for memory access and space. Further, Adding the notion of test case execution time. If edges are weighted, weighted means that some test cases may take longer to execute. Then in this situation how we will find the reduced test set. Because if test cases are weighted then there is a possibility that the reduced test set whose cardinality is more can give better results.

REFERENCES

1. Lin, Chu-Ti, Kai-Wei Tang, Jiun-Shiang Wang, and Gregory M. Kapfhammer. "Empirically evaluating Greedy-based test suite reduction methods at different levels of test suite complexity." *Science of Computer Programming* 150 (2017): 1-25.
2. Kapfhammer, G. M., Regression testing. In Encyclopedia of Software Engineering Three-Volume Set (Print). Auerbach Publications, 2010, 893–915.
3. Rothermel, G., & Harrold, M. J. (1997). A safe, efficient regression test selection technique. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, *6*(2), 173-210.
4. Rothermel, Gregg, et al. "An empirical study of the effects of minimization on the fault detection capabilities of test suites." *Proceedings. International Conference on Software Maintenance (Cat. No. 98CB36272)*. IEEE, 1998.
5. Wong, W. Eric, et al. "Effect of test set minimization on fault detection effectiveness." *Software: Practice andExperience* 28.4 (1998): 347-369.
6. Gligoric, Milos, Lamyaa Eloussi, and Darko Marinov. "Practical regression test selection with dynamic file dependencies." *Proceedings of the 2015 International Symposium on Software Testing and Analysis*. 2015.
7. Marchetto, Alessandro, et al. "A multi-objective technique to prioritize test cases." *IEEE Transactions on Software Engineering* 42.10 (2015): 918-940.
8. Wong, W. Eric, et al. "Effect of test set minimization on fault detection effectiveness." *Software: Practice andExperience* 28.4 (1998): 347-369.
9. Zhang, Lingming, et al. "An empirical study of junit test-suite reduction." *2011 IEEE 22nd International Symposium on Software Reliability Engineering*. IEEE, 2011.
10. Lin, Jun-Wei, and Chin-Yu Huang. "Analysis of test suite reduction with enhanced tie-breaking techniques." *Information and Software Technology* 51.4 (2009): 679-690.
11. Jeffrey, Dennis, and Neelam Gupta. "Improving fault detection capability by selectively retaining test cases during test suite reduction." *IEEE Transactions on software Engineering* 33.2 (2007): 108-123.
12. Qu, Xiao, Myra B. Cohen, and Gregg Rothermel. "Configuration-aware regression testing: an empirical study of sampling and prioritization." *Proceedings of the 2008 international symposium on Software testing and analysis*. 2008.
13. Jeffrey, Dennis, and Neelam Gupta. "Test suite reduction with selective redundancy." *21st IEEE International Conference on Software Maintenance (ICSM'05)*. IEEE, 2005.
14. Gotlieb, Arnaud, and Dusica Marijan. "FLOWER: optimal test suite reduction as a network maximum flow."

Proceedings of the 2014 International Symposium on Software Testing and Analysis. 2014.

15. Cormen, Thomas H., Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to algorithms*.MIT press, 2009.
16. Black, Jennifer, Emanuel Melachrinoudis, and David Kaeli. "Bi-criteria models for all-uses test suite reduction." *Proceedings. 26th International Conference on Software Engineering*. IEEE, 2004.
17. Chen, Z., Zhang, X., & Xu, B. (2008, July). A Degraded ILP Approach for Test Suite Reduction. In *SEKE* (pp. 494-499).
18. Hao, D., Zhang, L., Wu, X., Mei, H., & Rothermel, G. (2012, June). On-demand test suite reduction. In *2012 34th International Conference on Software Engineering (ICSE)* (pp. 738-748). IEEE.
19. Li, Zheng, Mark Harman, and Robert M. Hierons. "Search algorithms for regression test caseprioritization." *IEEE Transactions on software engineering* 33.4 (2007): 225-237.