



# Diversity in Test Suite Optimization

Annibale Panichella – PhD Candidate

apanichella@unisa.it

Software Engineering Lab , University of Salerno, Italy



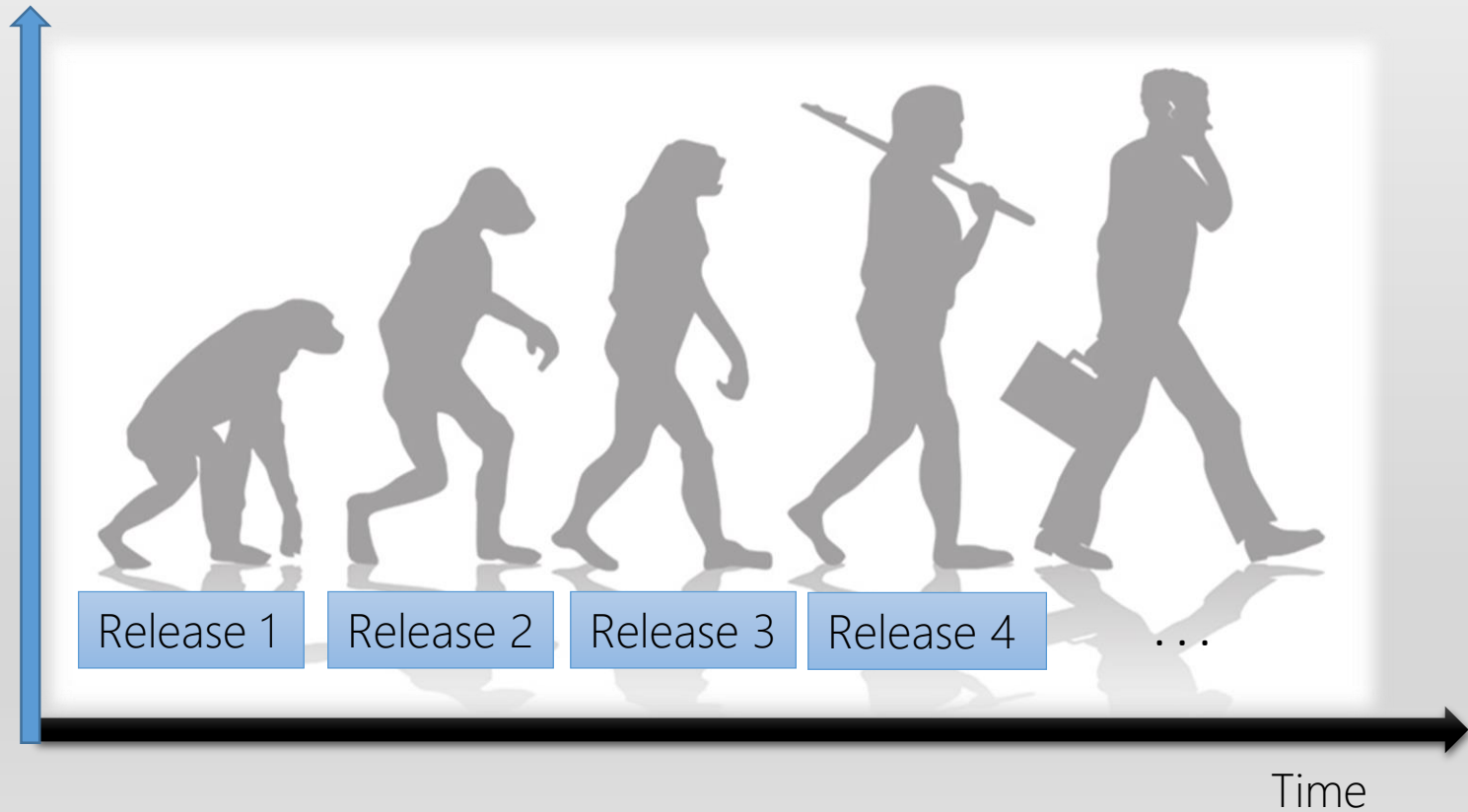
Andrea De Lucia  
University of Salerno



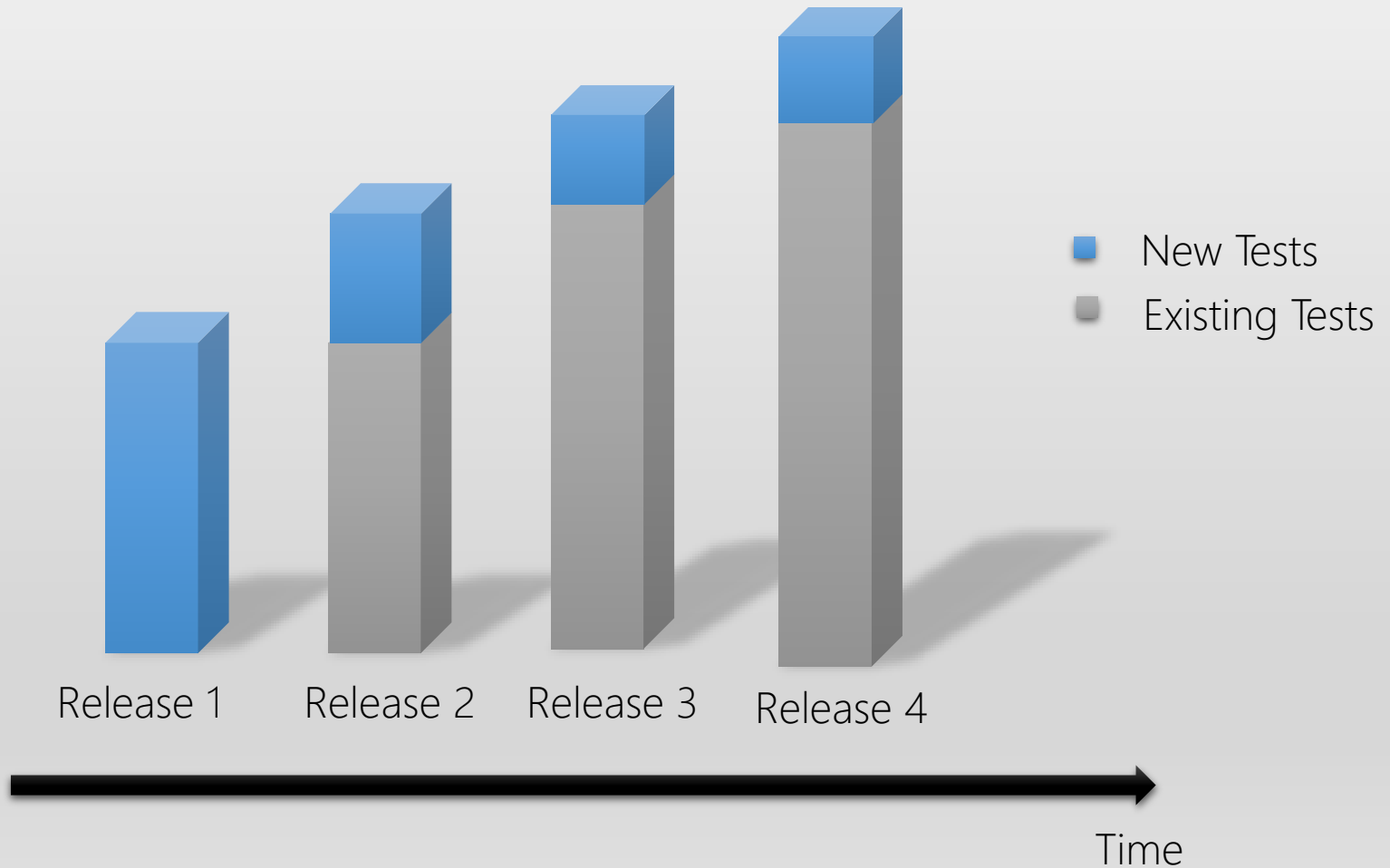
Rocco Oliveto  
University of Molise

# Software Evolution

N. Test Cases



# Amount of regression testing



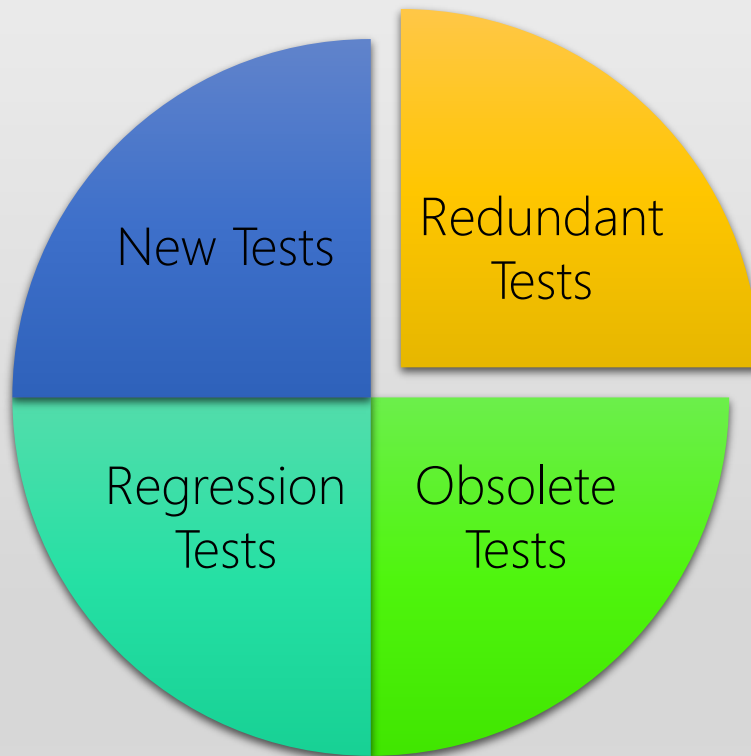
# Regression Testing is time consuming



1000 machine-hours to execute 30,000 functional test cases for a software product...

Mirarab, et al. *The effects of time constraints on test case prioritization: A series of controlled experiments.* **Transaction on Software Engineering 2010**

# Test Cases Redundancy



Reduce the Test Suite Size  
Reduce the Test Suite Cost  
Preserve the effectiveness



# Coverage Criteria

## Code coverage

- Statement coverage
- Block coverage

### A Methodology for Controlling the Size of a Test Suite

MARY JEAN HARROLD  
Clemson University

IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 22, NO. 8, AUGUST 1996

### Analyzing Regression Test Selection Techniques

Gregg Rothermel, Member, IEEE  
and Mary Jean Harrold, Member, IEEE

**Abstract**—Regression testing is a necessary but expensive maintenance activity aimed at showing that code has not been adversely affected by changes. Regression test selection techniques reuse tests from an existing test suite to test a modified program. Many regression test selection techniques have been proposed; however, it is difficult to compare and evaluate these techniques because they have different goals. This paper defines the issues relevant to regression test selection techniques, and uses these issues as the basis for a framework with which to evaluate the techniques. We illustrate the application of our framework by using it to evaluate existing regression test selection techniques. The evaluation reveals the strengths and weaknesses of existing techniques, and highlights some problems that have not been discussed in the literature.

**Index Terms**—Software maintenance, regression testing, selective testing, regression test selection.

#### 1 INTRODUCTION

ESTIMATES indicate that software maintenance activities account for as much as two-thirds of the cost of software production [36]. One necessary but expensive maintenance task is regression testing, performed on a modified program to instill confidence that changes are correct and have not adversely affected unchanged portions of the program. An important difference between regression testing and development testing is that during regression testing an established suite of tests may be available for reuse. One regression testing strategy, the retail-off approach, reuses all such tests, but this strategy may consume excessive time and resources. Regression test selection techniques, in contrast, attempt to reduce the time required to retest a modified program by selecting some subset of the existing test suite.<sup>1</sup> Although some regression test selection techniques select tests based on information collected from program specifications [28], [40] most techniques select tests based on information about the code of the program and the modified version [11, 12, 13, 15, 17, 110, 111, 131, 133, 135, 156, 172, 181, 190, 191, 194, 195, 197, 198, 199, 131, 133, 135, 136, 137, 138, 139, 141, 142, 143]. These code-based techniques pursue three distinct goals. Coverage techniques locate program components that have been modified or affected by modifications, and select tests in  $T$  that exercise those components. Minimization techniques seek like coverage techniques, but select minimal sets of tests through modified or affected program components. Safe techniques select every test in  $T$  that can expose one or more faults in  $P$ . Given this abundance of regression test selection techniques, if we wish to choose a technique for practical application, we need a way to compare and evaluate the techniques.

Differences in underlying goals lead regression test selection techniques to distinctly different results in test selection. Despite these philosophical differences, we have identified categories in which regression test selection techniques can be compared and evaluated. These categories are inclusiveness, precision, efficiency, and generality. Inclusiveness measures the extent to which a technique chooses tests that will cause the modified program to produce different output than the original program, and thereby expose faults caused by modifications. Precision measures the ability of a technique to avoid choosing tests that will not cause the modified program to produce different output than the original program. Efficiency measures the computational cost, and thus, practicality, of a technique. Generality measures the ability of a technique to handle realistic and diverse language constructs, arbitrarily complex code modifications, and realistic testing applications. These categories form a framework for evaluation and comparison of regression test selection techniques. In this paper, we present this framework, and demonstrate its usefulness by applying it to the code-based regression test selection techniques that we cited above.

The main benefit of our framework is that it provides a way to evaluate and compare existing regression test selection techniques. Evaluation and comparison of existing techniques helps us choose appropriate techniques for particular applications. For example, if we require very reliable code, we may insist on a safe selective retest technique regardless of cost. On the other hand, if we must reduce

1. A second important task for regression testing is to find ways in which the existing test suite is not adequate for testing a modified program, and indicate where new tests might be needed. In this work, however, we are concerned only with the process of reusing existing tests. We discuss this further in Section 2.

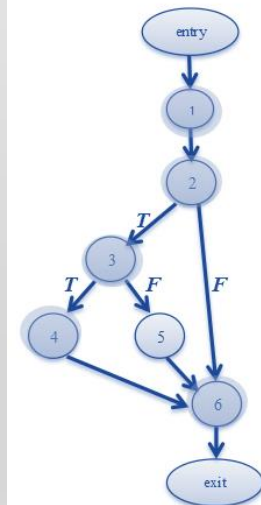
© G. Rothermel is with the Department of Computer Science, Oregon State University, Donkey Hall 547A, Corvallis, OR 97331-1332. E-mail: gprother@ccs.orst.edu.  
M.J. Harrold is with the Department of Computer and Information Science, The Ohio State University, 385 Drewes Lab, 3515 Neil Ave., Columbus, OH 43210-1277. E-mail: harrold@ccs.ohio-state.edu.

Manuscript received Oct. 27, 1994; revised Feb. 26, 1996.

Revised manuscript for acceptance by T. Chabre.  
For information on obtaining reprints of this article, please send e-mail to: ieee@computer.org, and reference IEEECS Log Number 990644.

```

Program
integer i, j, k
1. read i, j, k
2. if(i < j)
3.   if(j < k)
4.     i = k;
   else
5.     k = i;
   endif
6. print i, j, k
end
end
    
```



# Coverage Criteria

## Decision-To-Decision Graph

- All paths coverage
- All branches coverage
- All uses
- ...

### A Methodology for Controlling the Size of a Test Suite

MARY JEAN HARROLD  
Clemson University

IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 22, NO. 8, AUGUST 1996

### Analyzing Regression Test Selection Techniques

Gregg Rothermel, Member, IEEE

### Using Spanning Sets for Coverage Testing

Martina Marré and Antonia Bertolino

**Abstract.** A test coverage criterion defines a set  $E_c$  of entities of the program flowgraph and requires that every entity in this set is covered under some test case. Coverage criteria are also used to measure the adequacy of the executed test cases. In this paper, we introduce the notion of spanning sets of entities for coverage testing. A spanning set is a minimum subset of  $E_c$  such that a test suite covering the entities in this subset is guaranteed to cover every entity in  $E_c$ . When the coverage of an entity always guarantees the coverage of another entity, the former is said to subsume the latter. Based on the subsumption relation between entities, we provide a generic algorithm to find spanning sets for control flow and data flow based test coverage criteria. We suggest several useful applications of spanning sets. They help reduce and estimate the number of test cases needed to satisfy coverage criteria. We also empirically investigate how the use of spanning sets affects the fault detection effectiveness.

**Index Terms.** Control flow, coverage criteria, data flow, digraph, spanning sets, subsumption.

#### 1 INTRODUCTION

VARIOUS approaches to testing exist. The "classical" way to facilitate testing (as opposed to the more recent test-driven design [1]) is to establish a collection of requirements to be fulfilled, which defines a *test criterion*. In particular, structural coverage criteria map these requirements onto a set of entities in the program flowgraph that must be covered when the test cases are executed. These entities may be derived from the program's control flow or from the program's data flow.

We introduce the new concept of *spanning sets of entities* for a coverage criterion. A spanning set is a minimum subset of entities with the property that any test case covering the entities in this subset is guaranteed to cover every entity in the set.

improve the way existing test-coverage criteria are applied. We have been studying the use of spanning sets of entities in coverage testing for some time. In [6], [7], [9], we identify spanning sets of entities for the all-branches criterion and present some useful applications. In [19], we identify spanning sets of entities for the all-uses criterion. In this paper, we provide a general method for identifying a spanning set of entities for an entire family of test coverage criteria and discuss its applications.<sup>1</sup> The focus is on unit testing; the application of the approach at the inter-procedural level is an important extension, but is out of the scope of this paper.

<sup>1</sup> A concise, preliminary version of this paper appeared in [20].

• M. Marré is with the Dipartimento di Computazione, FC3/9, Università di Buenos Aires, Argentina. E-mail: mmarr@comp.uba.ar  
• A. Bertolino is with the Istituto di Scienza e Tecnologia dell'Informazione "A. Faedo", Area della Ricerca CNR di Pisa, 56100 Pisa, Italy. E-mail: antonia.bertolino@isti.cnr.it

To the best of our knowledge, there has been no previous work on identifying a minimum set of entities that guarantees full coverage for an entire family of criteria. Some authors [11], [11] have independently recognized this idea for the simple strategy of all-branches coverage, but failed to generalize it to other criteria.

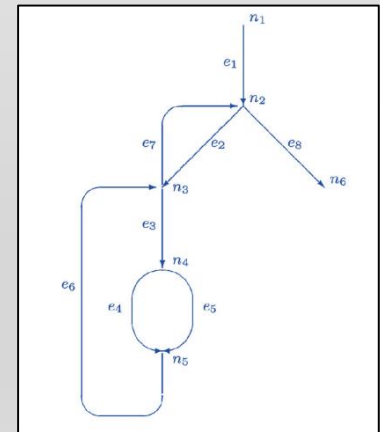
Some work has been done to reduce the size of a test suite. In particular, Gupta and Sofia [13] have investigated ways to guide test case generation, so that a single test case satisfies multiple coverage requirements. They gather coverage requirements into test groups and generate test cases for each group. However, these techniques are applied to a redundant set of test cases only after the test suite has been generated. Thus, such approaches do not actually reduce the effort of generating the test cases.

A word of caution is appropriate. By targeting test-case selection and avoiding redundant test cases, spanning sets can make coverage testing more efficient, but not necessarily more effective [22]. On the contrary, the testers should be aware that every single test case that they discard could have been the one that found the bug. Indeed, it is not our recommendation to use spanning sets to reduce the number of test cases at any rate; more pragmatically, we say that in those cases in which test resources are scarce and only few more test cases can be executed, then spanning sets can help in selecting those test cases that maximize coverage.

In some empirical studies [24], [22], the effect on fault detection of reducing the size of a test set, while holding coverage constant, was analyzed. The results in [24] showed that minimizing the test set produces little or no reduction

```
void sort(a, n) /* SORT Program */
1  int a[];
2  int n;
3  {
4  int sortupto;
5  int maxpos;
6  int mymax;
7  int index;

8  sortupto = 1;
9  maxpos = 1;
10 while (sortupto < n) {
11   mymax = a[sortupto];
12   index = sortupto + 1;
13   while (index <= n) {
14     if (a[index] > mymax) {
15       mymax = a[index];
16       maxpos = index;
17     }
18     index = index + 1;
19   }
20   index = a[sortupto];
21   a[sortupto] = mymax;
22   a[maxpos] = index;
23   sortupto = sortupto + 1;
24 }
25 }
```





# Coverage Criteria

## Model-Based coverage

- State Machine transitions
- Transition Tree

### A Methodology for Controlling the Size of a Test Suite

MARY JEAN HARROLD  
Clemson University

IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 22, NO. 8, AUGUST 1996

### Analyzing Regression Test Selection Techniques

### Achieving Scalable Model-Based Testing Through Test Case Diversity

HADI HEMMATI, Simula Research Laboratory, Norway and University of Oslo, Norway  
ANDREA ARCURI, Simula Research Laboratory, Norway  
LIONEL BRIAND, Simula Research Laboratory, Norway and University of Oslo, Norway

The increase in size and complexity of modern software systems requires scalable, systematic, and automated testing approaches. Model-based testing (MBT), as a systematic and automated test case generation technique, is being successfully applied to verify industrial-scale systems and is supported by commercial tools. However, scalability is still an open issue for large systems, as in practice there are limits to the amount of testing that can be performed in industrial contexts. Even with standard coverage criteria, the resulting test suites generated by MBT techniques can be very large and expensive to execute, especially for system level testing on real deployment platforms and network facilities. Therefore, a scalable MBT technique should be feasible regarding the size of the generated test suites and should be easily accommodated to fit resource and time constraints. Our approach is to select a subset of the generated test suite in such a way that it can be realistically executed and analyzed within the time and resource constraints, while preserving the fault revealing power of the original test suite to a maximum extent. In this article, to address this problem, we introduce a family of similarity-based test case selection techniques for test suites generated from state machines. We evaluate 320 different similarity-based selection techniques and then compare the effectiveness of the best similarity-based selection technique with other common selection techniques in the literature. The results based on two industrial case studies, in the domain of embedded systems, show significant benefits and a large improvement in performance when using a similarity-based approach. We complement these analyses with further studies on the scalability of the technique and the effects of failure rate on its effectiveness. We also propose a method to identify optimal tradeoffs between the number of test cases to run and fault detection.

Categories and Subject Descriptors: D.2.4 [Software Engineering]: Software/Program Verification; D.2.5 [Software Engineering]: Testing and Debugging

General Terms: Verification

Additional Key Words and Phrases: Test case selection, test case diversity, test case reduction, test case minimization, test case optimization

ACM Reference Format:

Hemmati, H., Arcuri, A., and Briand, L.

Achieving Scalable Model-Based Testing Through Test Case Diversity. ACM Trans. Softw. Eng. Meth.

DOI = 10.1145/2430536.2430540 <http://doi.acm.org/10.1145/2430536.2430540>

## Hemmati et al. TOSEM 2013

H. Hemmati is currently affiliated with the Software Architecture Group, David R. Chertson School of Computer Science, University of Waterloo.  
L. Briand is currently affiliated with the SnT Centre, University of Luxembourg.  
This work is a revised and extended version of papers presented at FSR 2010, ISSRE 2010, and ICTSS 2010.  
Authors' addresses: H. Hemmati, David R. Chertson School of Computer Science, University of Waterloo, Ontario, N2L 3G1, Canada; email: hemmati@gmail.com; A. Arcuri Simula Research Laboratory, PO Box 134, 1325 Lysaker Norway; L. Briand, SnT Centre, University of Luxembourg, 6 rue Richard Coudenhove-Kalergi, L-1359, Luxembourg.  
Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 2 Penn Plaza, Suite 701, New York, NY 10121-0701 USA, fax +1 (212) 869-0481, or [permissions@acm.org](mailto:permissions@acm.org).  
© 2013 ACM 1049-331X/2013/02-ART1 \$15.00  
DOI 10.1145/2430536.2430540 <http://doi.acm.org/10.1145/2430536.2430540>

### Coverage Testing

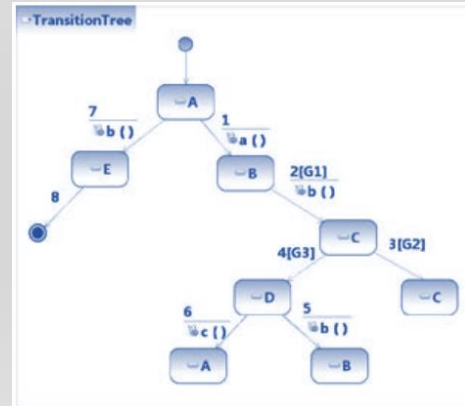
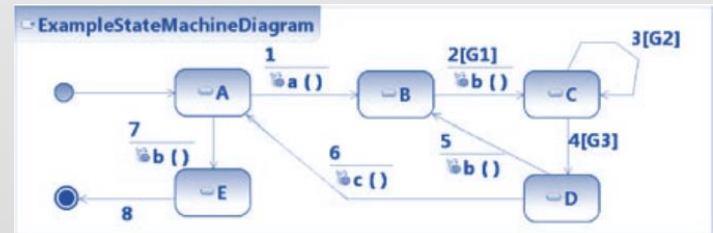
rolino

graph and requires that every entity in this set is a member of  $E_c$ . In this paper, we propose a technique for generating a minimum subset of  $E_c$  such that a test suite for coverage of an entity always guarantees the transition relation between entities, we provide a heuristic criterion. We suggest several useful heuristics to satisfy coverage criteria. We also discuss the impact of these heuristics on the size of the test suite.

Assumptions:

Our knowledge, there has been no previous work on this topic. In this paper, we propose a technique for generating a minimum subset of  $E_c$  such that a test suite for coverage of an entity always guarantees the transition relation between entities, we provide a heuristic criterion. We suggest several useful heuristics to satisfy coverage criteria. We also discuss the impact of these heuristics on the size of the test suite.

These techniques are applied to a redundancy after the test suite has been generated, they do not actually reduce the effort of cases. It is appropriate. By targeting test-case sets to reduce the number of test cases, we are more efficient, but not necessarily. On the contrary, the testers should use single test case that they discarded could have found the bug. Indeed, it is not our use spanning sets to reduce the number of test cases, we say that in test resources are scarce and only few are executed, then spanning sets can help cases that maximize coverage. In studies [24], [22], the effect on fault size of a test set, while holding, is analyzed. The results in [24] showed test set produces little or no reduction



# Test Suite Optimization

## Redundant Test Cases

### Input:

- Program  $P = \{e_1, e_2, \dots, e_n\}$
- Test Cases covering parts of P

$$T = \{t_1, t_2, \dots, t_n\}$$

### Problem:

Finding the minimal sub-set

$$T^* \subseteq T \quad \text{such that} \quad \bigcup_{t \in T^*} t(P) = P$$

# Test Suite Optimization

## Redundant Test Cases

### Input:

- Program  $P = \{e_1, e_2, \dots, e_n\}$
- Test Cases covering parts of P

$$T = \{t_1, t_2, \dots, t_n\}$$

### Problem:

Finding the minimal sub-set

$$T^* \subseteq T \text{ such that } \bigcup_{t \in T^*} t(P) = P$$

## Hitting set

### Input:

- Set  $P = \{e_1, e_2, \dots, e_n\}$
- A collection of sub-sets of P

$$T = \{t_1, t_2, \dots, t_n\}$$

### Problem:

Finding the minimal sub-set

$$T^* \subseteq T \text{ such that } \bigcup_{t \in T^*} t_i = P$$

# Test Suite Optimization

Redundant Test Cases

Input:

- Program
- Test Cases

Problem:

Finding the minimal sub-set

$$T^* \subseteq T \text{ such that } \bigcup_{t \in T^*} t(P) = P$$

Hitting set

Input:

$\{e_1, \dots, e_n\}$

Finding a minimal sub-set

$$T^* \subseteq T \text{ such that } \bigcup_{t \in T^*} t_i = P$$

*The problem is NP-hard*

# Search

100 Test Cases

N. of possible subsets  $2^{100} \approx 1,26 \cdot 10^{30} >$

$4.354 \pm 0.012 \cdot 10^{17}$  seconds

Age of the  
Universe



# Additional Greedy Algorithm

---

## Greedy Algorithm (P, T)

(1)  $C \leftarrow \emptyset$  covered statements of P

(2)  $S \leftarrow \emptyset$  set of selected test cases

(2) repeat

(3)  $j \leftarrow \max \{T_j - (C \cap T_j)\}$

(4)  $C = C \cup T_j$

(5) Remove  $T_j$  from T

$S = S \cup \{T_j\}$

(6) until  $C=P$

---

- 1) Start with the test case having the highest coverage
- 2) Iteratively add the most distant test case
- 3) Perform step 1-2 until max coverage is reached

# Additional Greedy Algorithm

---

## Greedy Algorithm (P, T)

(1)  $C \leftarrow \emptyset$  covered statements of P

(2)  $S \leftarrow \emptyset$  set of selected test cases

(2) repeat

(3)  $j \leftarrow \max \{T_j - (C \cap T_j)\}$

(4)  $C = C \cup T_j$

(5) Remove  $T_j$  from T

$S = S \cup \{T_j\}$

(6) until  $C=P$

---

1) Start with the test case having the highest coverage

2) Iteratively add the most distant test case

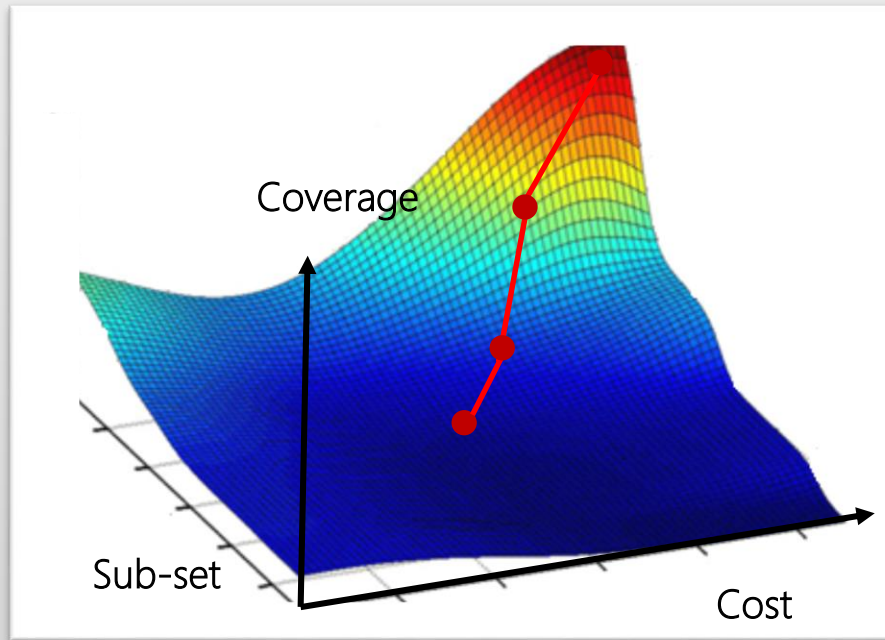
3) Perform step 1-2 until max coverage is reached

**Greedy step** =  $T_j - (C \cap T_j) \propto (C \cup T_j) - (C \cap T_j) \propto$  Jaccard Distance

Greedy step is proportional to the Jaccard Distance

# Additional Greedy Algorithm

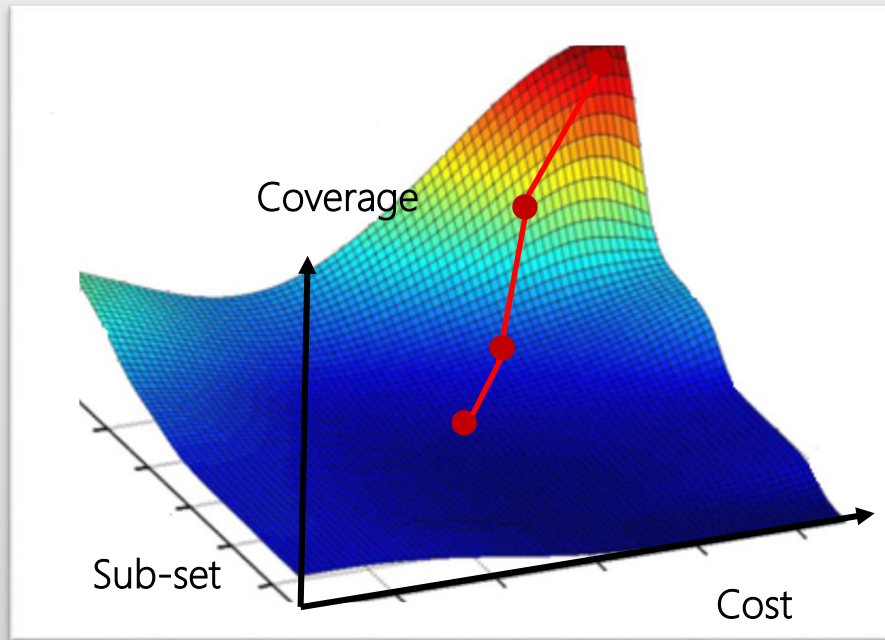
Ideal Problem



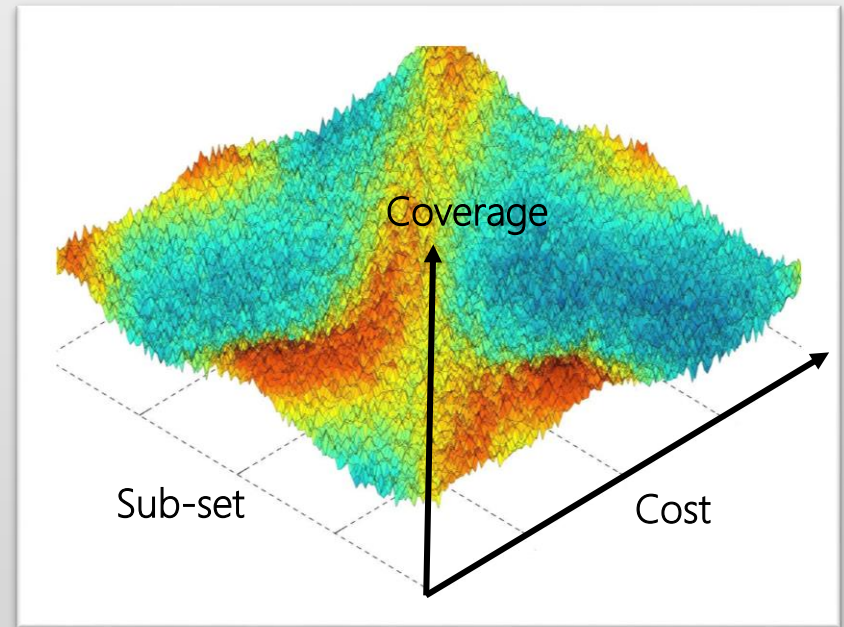


# Additional Greedy Algorithm

Ideal Problem



Real Problem



# What is the best search algorithm?

## Achieving Scalable Model-Based Testing Through Test Case Diversity

HADI HEMMATI, Simula Research Laboratory, Norway and University of Oslo, Norway  
ANDREA ARCURI, Simula Research Laboratory, Norway  
LIONEL BRIAND, Simula Research Laboratory, Norway and University of Oslo, Norway

The increase in size and complexity of modern software systems requires scalable, systematic, and automated testing approaches. Model-based testing (MBT), as a systematic and automated test case generation technique, is being successfully applied to verify industrial-scale systems and is supported by commercial tools. However, scalability is still an open issue for large systems, as in practice there are limits to the amount of testing that can be performed in industrial contexts. Even with standard coverage criteria, the resulting test suites generated by MBT techniques can be very large and expensive to execute, especially for system level testing on real deployment platforms and network facilities. Therefore, a scalable MBT technique should be flexible regarding the size of the generated test suites and should be easily accommodated to fit resource and time constraints. Our approach is to select a subset of the generated test suite in such a way that it can be realistically executed and analyzed within the time and resource constraints, while preserving the fault revealing power of the original test suite to a maximum extent. In this article, to address this problem, we introduce a family of similarity-based test case selection techniques for test suites generated from state machines. We evaluate 320 different similarity-based selection techniques and then compare the effectiveness of the best similarity-based selection technique with other common selection techniques in the literature. The results based on two industrial case studies, in the domain of embedded systems, show significant benefits and a large improvement in performance when using a similarity-based approach. We complement these analyses with further studies on the scalability of the technique and the effects of failure rate on its effectiveness. We also propose a method to identify optimal tradeoffs between the number of test cases to run and fault detection.

Categories and Subject Descriptors: D.2.4 [Software Engineering]: Software/Program Verification; D.2.5 [Software Engineering]: Testing and Debugging

General Terms: Verification

Additional Key Words and Phrases: Test case selection, test case minimization, model-based testing, similarity functions, search-based software engineering

ACM Reference Format:

Hemmati, H., Arcuri, A., and Briand, L. 2013. Achieving scalable model-based testing through test case diversity. ACM Trans. Softw. Eng. Methodol. 22, 1, Article 6 (February 2013), 42 pages.  
DOI = 10.1145/2430536.2430540 <http://doi.acm.org/10.1145/2430536.2430540>

H. Hemmati is currently affiliated with the Software Architecture Group, David R. Cheriton School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada.

Hemmati et al. **TOSEM 2013**

L-1359, Luxembourg

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 2 Penn Plaza, Suite 701, New York, NY 10121-0701 USA, fax +1 (212) 869-0481, or [permissions@acm.org](mailto:permissions@acm.org).

© 2013 ACM 1049-331X/2013/02. ARTS \$15.00  
DOI 10.1145/2430536.2430540 <http://doi.acm.org/10.1145/2430536.2430540>

ACM Transactions on Software Engineering and Methodology, Vol. 22, No. 1, Article 6, Feb. date: February 2013.

- 1) Model Driven Testing
  - Model-based coverage
  - Abstract Test Cases
- 2) Search Algorithms
  - Greedy Algorithm
  - Random Search Algorithm
  - Hill climbing
  - Evolutionary Algorithms
- 3) Similarity functions
  - Set distance
  - Hamming distance
  - Jaccard distance
  - Clustering distance

# What is the best search algorithm?

## Achieving Scalable Model-Based Testing Through Test Case Diversity

HADI HEMMATI, Simula Research Laboratory, Norway and University of Oslo, Norway  
ANDREA ARCURI, Simula Research Laboratory, Norway  
LIONEL BRIAND, Simula Research Laboratory, Norway and University of Oslo, Norway

The increase in size and complexity of modern software systems requires scalable, systematic, and automated testing approaches. Model-based testing (MBT), as a systematic and automated test case generation technique, is being successfully applied to verify industrial-scale systems and is supported by commercial tools. However, scalability is still an open issue for large systems, as the amount of testing that can be performed in industrial contexts. Even resulting test suites generated by MBT techniques can be very large as system level testing on real deployment platforms and network facilities. A technique should be flexible regarding the size of the generated test suites to fit resource and time constraints. Our approach is to select a subset of that it can be realistically executed and analyzed within the time and the fault revealing power of the original test suite to a maximum problem, we introduce a family of similarity-based test case selection from state machines. We evaluate 520 different similarity-based selection effectiveness of the test similarity-based selection technique with the literature. The results based on two industrial case studies, in significant benefits and a large improvement in performance when complement these analyses with further studies on the scalability ratio on its effectiveness. We also propose a method to identify optimal cases to run and fault detection.

Categories and Subject Descriptors: D.2.4 [Software Engineering]: Software Testing  
[Software Engineering]: Testing and Debugging

General Terms: Verification

Additional Key Words and Phrases: Test case selection, test case minimization, model-based testing, similarity functions, search-based software engineering

ACM Reference Format:

Hemmati, H., Arcuri, A., and Briand, L. 2013. Achieving scalable model-based testing through test case diversity. ACM Trans. Softw. Eng. Methodol. 22, 1, Article 6 (February 2013), 42 pages.  
DOI = 10.1145/2430536.2430540 <http://doi.acm.org/10.1145/2430536.2430540>

H. Hemmati is currently affiliated with the Software Architecture Group, David R. Cheriton School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada.

Hemmati et al. **TOSEM 2013**

L-1350, Luxembourg

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted.

To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 2 Penn Plaza, Suite 701, New York, NY 10121-0701 USA, fax +1 (212) 869-0481, or [permissions@acm.org](mailto:permissions@acm.org).

© 2013 ACM 1049-331X/2013/02-ART6 \$15.00  
DOI 10.1145/2430536.2430540 <http://doi.acm.org/10.1145/2430536.2430540>

ACM Transactions on Software Engineering and Methodology, Vol. 22, No. 1, Article 6, Feb. date: February 2013.

- 1) Model Driven Testing
  - Model-based coverage
  - Abstract Test Cases

“Evolutionary Algorithm to be the most cost-effective technique on average on two industrial case studies”. Hemmati et al.

- 2) Search Algorithms
  - Evolutionary Algorithm
  - Genetic Algorithm
  - Hill Climbing
  - Simulated Annealing
  - Tabu Search
  - Genetic Algorithms

- 3) Similarity functions
  - Set distance
  - Hamming distance
  - Jaccard distance
  - Clustering distance

# Multi-Criteria Regression Testing

## Mono-Objective Paradigm

## Multi-Objective Paradigm

### A Methodology for Con Test Suite

MARY JEAN HARROLD  
Clomson University  
and  
DAVID CHERTON

IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 22, NO. 8, AUGUST 1996

539

### Analyzing Regression Test Selection Techniques

Member, IEEE  
Member, IEEE

Recently arrived at showing that code has not been tested from an existing test suite to test a modified program. It is difficult to compare and evaluate these in relation to regression test selection techniques, and techniques. We examine the application of our test. The evaluation reveals the strengths and weaknesses of this area should address.

Regression test selection.

and select tests in  $T$  that exercise those components. Selection techniques work like coverage techniques, but minimal sets of tests through modified or affected components. Site techniques select every test in  $T$  that types one or more faults in  $P$ . Given this abundance of test selection techniques, if we wish to choose a type for practical application, we need a way to compare and evaluate the techniques. Differences in underlying goal lead regression test selection techniques to distinctly different results in test sets. Despite these philosophical differences, we have listed categories in which regression test selection techniques can be compared and evaluated. These categories include: effectiveness, precision, efficiency, and generality. Various measures the extent to which a technique selects tests that will cause the modified program to produce different output than the original program, and by expense limits caused by modifications. Precision measures the ability of a technique to avoid choosing tests will not cause the modified program to produce different than the original program. Efficiency measures computational cost, and thus, practicality, of a technique. Generality measures the ability of a technique to be realistic and diverse language constructs, arbitrary test code modifications, and realistic testing applications. These categories form a framework for evaluation comparison of regression test selection techniques. In this paper, we present this framework, and demonstrate its fitness by applying it to the code-based regression test selection techniques that we cited above.

The main benefit of our framework is that it provides a way to evaluate and compare existing regression test selection techniques. Evaluation and comparison of existing techniques helps us choose appropriate techniques for particular applications. For example, if we require very reliable test results, we may choose the selective reset technique. On the other hand, if we must reduce

### Achieving Scalable Model-Based Testing Through Test Case Diversity

HADI HEMMATI, Simula Research Laboratory, Norway and University of Oslo, Norway  
ANDREA ARCURI, Simula Research Laboratory, Norway  
LIONEL BRIAND, Simula Research Laboratory, Norway and University of Oslo, Norway

The increase in size and complexity of modern software systems requires scalable, systematic, and automated testing approaches. Model-based testing (MBT), as a systematic and automated test case generation technique, is being successfully applied to verify industrial-scale systems and is supported by commercial tools. However, scalability is still an open issue for large systems, as in practice there are limits to the amount of testing that can be performed in industrial contexts. Even with standard coverage criteria, the resulting test suites generated by MBT techniques can be very large and expensive to execute, especially for system level testing on real deployment platforms and network facilities. Therefore, a scalable MBT technique should be flexible regarding the size of the generated test suites and should be easily accommodated to fit resource and time constraints. Our approach is to select a subset of the generated test suite in such a way that it can be realistically executed and analyzed within the time and resource constraints, while preserving the fault revealing power of the original test suite to a maximum extent. In this article, to address this problem, we introduce a family of similarity-based test case selection techniques for test suites generated from state machines. We evaluate 520 different similarity-based selection techniques and then compare the effectiveness of the best similarity-based selection technique with other common selection techniques in the literature. The results based on two industrial case studies, in the domain of embedded systems, show significant benefits and a large improvement in performance when using a similarity-based approach. We complement these analyses with further studies on the scalability of the technique and the effects of failure rate on its effectiveness. We also propose a method to identify optimal tradeoffs between the number of test cases to run and fault detection.

Categories and Subject Descriptors: D.2.4 (Software Engineering): Software/Program Verification; D.2.5 (Software Engineering): Testing and Debugging

General Terms: Verification

Additional Key Words and Phrases: Test case selection, test case minimization, model-based testing, similarity function, search-based software engineering

#### ACM Reference Format:

Hemmati, H., Arcuri, A., and Briand, L. 2013. Achieving scalable model-based testing through test case diversity. *ACM Trans. Softw. Eng. Methodol.* 22, 1, Article 6 (February 2013), 42 pages.  
DOI = 10.1145/2430536.2430540 <http://doi.acm.org/10.1145/2430536.2430540>

H. Hemmati is currently affiliated with the Software Architecture Group, David R. Cherton School of Computer Science, University of Waterloo.  
L. Briand is currently affiliated with the SuT Centre, University of Luxembourg.  
This work is a revised and extended version of papers presented at PSE 2010, ISSRE 2010, and ICTSS 2010. Authors' addresses: H. Hemmati, David R. Cherton School of Computer Science, University of Waterloo, Ontario, N2L 2G1, Canada; email: hemmati@gmail.com; A. Arcuri Simula Research Laboratory, P.O. Box 154, 1325 Lysaker Norway; L. Briand, SuT Centre, University of Luxembourg, 6 rue Richard Coudenhove-Kalergi, L-1359, Luxembourg.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies show this notice on the first page or initial screen of a displaying along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 2 Penn Plaza, Suite 701, New York, NY 10121-0701 USA, fax +1 (212) 869-0481, or [permissions@acm.org](mailto:permissions@acm.org).  
© 2013 ACM 1049-3312/2013/02-ART6 \$15.00  
DOI 10.1145/2430536.2430540 <http://doi.acm.org/10.1145/2430536.2430540>

### Pareto Efficient Multi-Objective Test Case Selection

Shin Yoo and Mark Harman  
Kings College London  
Strand, London  
WC2R 2LS, UK  
(Shin.Yoo,Mark.Harman)@kcl.ac.uk

#### ABSTRACT

Previous work has treated test case selection as a single objective optimization problem. This paper introduces the concept of Pareto efficiency to test case selection. The Pareto efficient approach takes multiple objectives such as code coverage, pass/fault-detection history and execution cost, and constraints a group of non-dominating, equivalently optimal test case subsets. The paper describes the potential benefits of Pareto efficient multi-objective test case selection, illustrating with empirical studies of two & three objective formulations.

#### 1. INTRODUCTION

Regression testing is the test performed in order to guarantee that newly introduced changes in a software do not affect the unchanged parts of the software. One possible approach to regression testing is the *run-all* method, in which the tester simply executes all of the existing test cases to ensure that the new changes are harmless. Unfortunately, this is a very expensive process, time limitations force a consideration of test case selection and prioritization techniques [2, 8, 13, 17, 19, 20, 25].

Test case selection techniques try to reduce the number of test cases to be executed, while satisfying the testing requirements denoted by a test criterion. Test case prioritization techniques try to order the test cases in such a way that increase the rate of early fault-detection.

In the real-world testing, there are often multiple test criteria. For example, different types of testing, such as functional testing and structural testing, require different testing criteria [9]. There also can be cases where it is beneficial for the tester to consider multiple test criteria because the strongest test case criterion is simply unobtainable. For example, testers face the problem that the real fault detection information cannot be known until the regression testing is actually finished. Code coverage is one possible surrogate test adequacy criterion that is used in place of fault detection, but it is not the only one. However one cannot be

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies show this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.  
July 2013, Vol. 22, No. 1, Article 6, Pages 6-15  
Copyright 2013 ACM 978-1-4558-3056-2/13/02-ART6 \$15.00.

contains a link between code coverage and fault detection it would be natural to supplement coverage with other test criteria, for example, pass/fault-detection history.

Of course, the quality of the test data is not the only concern. Cost is also one of the essential criteria, because the whole purpose of test case selection and prioritization is to achieve more efficient testing in terms of the cost. One important cost driver, considered by other researchers [13, 20] is the execution time of the test suite.

In order to provide automated support to the selection of regression test data it therefore seems inevitable that a multi-objective approach is required that is capable of taking into account the subtasks inherent in balancing many, possibly competing and conflicting objectives. Existing approaches to regression test case selection (and prioritization) have been single objective approaches that have sought to optimize a single objective function.

For the prioritization problem, there has been recent work on a two objective formulation [15], that takes account of coverage and cost, using a single objective of coverage per unit cost. However, this approach conflates the two objectives into a single objective. Where there are multiple competing and conflicting objectives the optimization heuristic recommends the consideration of a Pareto optimal optimization approach [4, 16]. Such a Pareto optimal approach is able to take account of the need to balance the conflicting objectives, all of which the software engineer seeks to optimize.

This paper presents the first multi-objective formulation of the test case selection problem, showing how multiple objectives can be optimized using a Pareto efficient approach. We believe that such an approach is well suited to the regression test case selection problem, because it is likely that a tester will want to optimize several possible conflicting constraints.

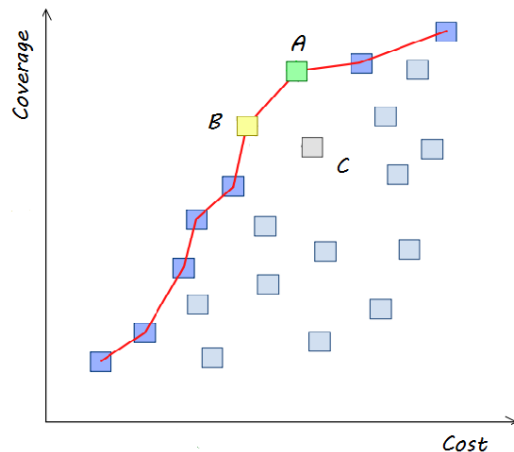
The primary contributions of this paper are as follows:

1. The paper introduces a multi-objective formulation of the regression test case selection problem and instantiates this with two variants: A two objective formulation that combines coverage and cost, and a three objective formulation that combines coverage, cost and fault history. The formulation facilitates a theoretical treatment of the optimality of the greedy algorithms and allows us to establish a relationship between the multi-objective problems of test case prioritization and test case selection.
2. The paper presents three algorithms for solving the two and three objective instances of the test case selection problem.

# Multi-Criteria Regression Testing

Multiple objectives are optimized using Pareto efficient approaches

Multiple optimal solutions can be found



Pareto Optimality: all solutions that are not dominated by any other solutions form the Pareto optimal set.

## Multi-Objective Paradigm

### Pareto Efficient Multi-Objective Test Case Selection

Shin Yoo and Mark Harman  
Kings College London  
Strand, London  
WC2R 2LS, UK  
(Shin.Yoo,Mark.Harman)@kol.ac.uk

#### ABSTRACT

Previous work has treated test case selection as a single objective optimization problem. This paper introduces the concept of Pareto efficiency to test case selection. The Pareto efficient approach takes multiple objectives such as code coverage, past fault-detection history and execution cost, and constructs a group of non-dominating, equivalently optimal test case subsets. The paper describes the potential benefits of Pareto efficient multi-objective test case selection, illustrating with empirical studies of two & three objective formulations.

#### 1. INTRODUCTION

Regression testing is the test performed in order to guarantee that newly introduced changes in a software do not affect the unchanged parts of the software. One possible approach to regression testing is the ruses-all method, in which the tester simply executes all of the existing test cases to ensure that the new changes are harmless. Unfortunately, this is a very expensive process; time limitations force a consideration of test case selection and prioritization techniques [2, 8, 13, 17, 19, 20, 25].

Test case selection techniques try to reduce the number of test cases to be executed, while satisfying the testing requirements denoted by a test criterion. Test case prioritization techniques try to order the test cases in such a way that increase the rate of early fault-detection.

In the real-world testing, there are often multiple test criteria. For example, different types of testing, such as functional testing and structural testing, require different testing criteria [9]. There also can be cases where it is beneficial for the tester to consider multiple test criteria because the single most ideal test criterion is simply unobtainable. For example, testers face the problem that the real fault detection information cannot be known until the regression testing is actually finished. Code coverage is one possible surrogate test adequacy criterion that is used in place of fault detection, but it is not the only one. However one cannot be

certain of a link between code coverage and fault detection it would be natural to supplement coverage with other test criteria, for example, past fault detection history.

Of course, the quality of the test data is not the only concern. Cost is also one of the essential criteria, because the whole purpose of test case selection and prioritization is to achieve more efficient testing in terms of the cost. One important cost driver, considered by other researchers [13, 20] is the execution time of the test suite.

In order to provide automated support to the selection of regression test data it therefore seems inevitable that a multi-objective approach is required that is capable of taking into account the subtleties inherent in balancing many, possibly competing and conflicting objectives. Existing approaches to regression test case selection (and prioritization) have been single objective approaches that have sought to optimize a single objective function.

For the prioritization problem, there has been recent work on a two objective formulation [15], that takes account of coverage and cost, using a single objective of coverage per unit cost. However, this approach conflates the two objectives into a single objective. Where there are multiple competing and conflicting objectives the optimization literature recommends the consideration of a Pareto optimal optimization approach [4, 18]. Such a Pareto optimal approach is able to take account of the need to balance the conflicting objectives, all of which the software engineer seeks to optimize.

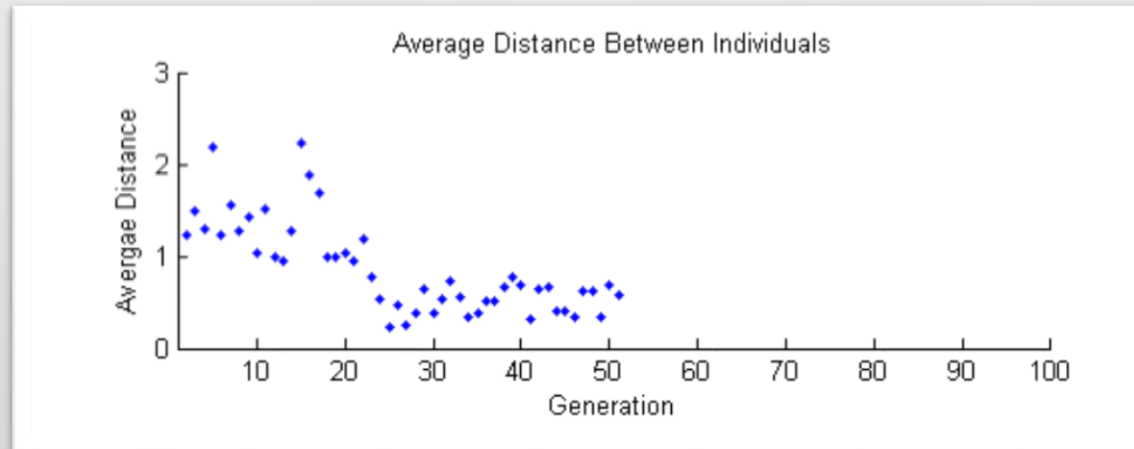
This paper presents the first multi-objective formulation of the test case selection problem, showing how multiple objectives can be optimized using a Pareto efficient approach. We believe that such an approach is well suited to the regression test case selection problem, because it is likely that a tester will want to optimize several possible conflicting constraints.

The primary contributions of this paper are as follows:

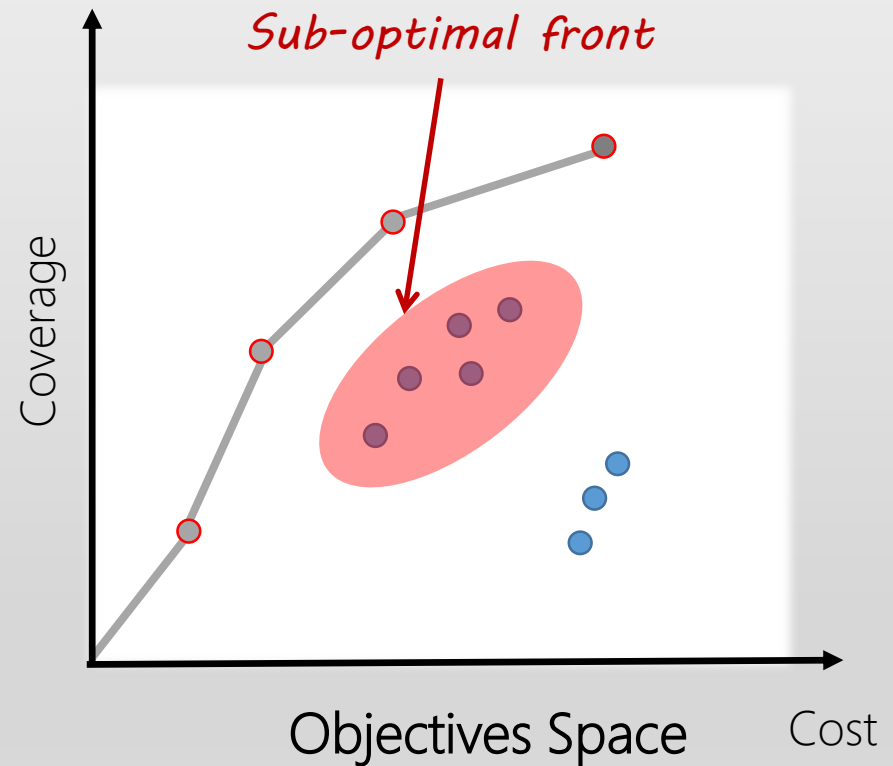
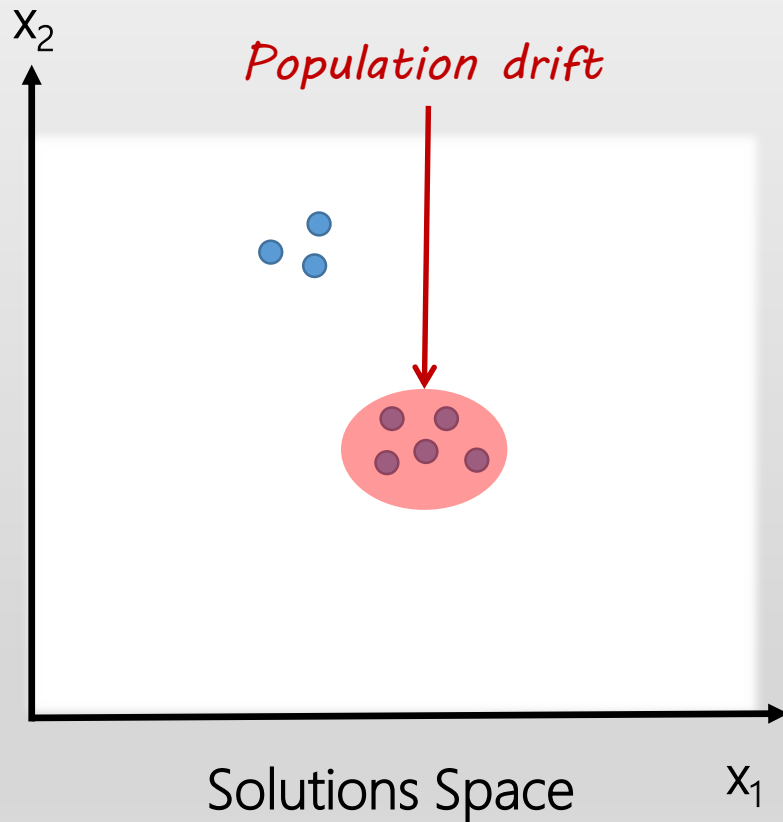
1. The paper introduces a multi-objective formulation of the regression test case selection problem and illustrates this with two variants: A two objective formulation that combines coverage and cost and a three objective formulation that combines coverage, cost and fault history. The formulation facilitates a theoretical treatment of the optimality of the greedy algorithms and allows us to establish a relationship between the multi-objective problems of test case prioritization and test case selection.
2. The paper presents three algorithms for solving the two and three objective instances of the test case selection

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear the notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.  
Submitted to ICSE '07, July 9-12, London, UK.  
Copyright 2007 ACM 978-0-905660-30-0/07...\$5.00.

# Diversity in Evolutionary Algorithms



# Diversity in Evolutionary Algorithms



# Injecting Diversity during the Evolution





# Orthogonal exploration

## Estimating the Evolution Direction of Populations to Improve Genetic Algorithms

Andrea De Lucia<sup>1</sup>, Massimiliano Di Penta<sup>2</sup>, Rocco Oliveto<sup>3</sup>, Annibale Panichella<sup>1</sup>

## Orthogonal Exploration of the Search Space in Evolutionary Test Case Generation

Fitsum M. Kifetew  
Fondazione Bruno Kessler  
Trento, Italy  
kifetew@fbk.eu

Annibale Panichella  
University of Salerno  
Fisciano (SA), Italy  
apanichella@unisa.it

Andrea De Lucia  
University of Salerno  
Fisciano (SA), Italy  
adelucia@unisa.it

Rocco Oliveto  
University of Molise  
Pesche (IS), Italy  
rocco.oliveto@unimol.it

Paolo Tonella  
Fondazione Bruno Kessler  
Trento, Italy  
tonella@fbk.eu

### ABSTRACT

The effectiveness of evolutionary test case generation based on Genetic Algorithms (GAs) can be seriously impacted by genetic drift, a phenomenon that inhibits the ability of such algorithms to effectively diversify the search and look for alternative potential solutions. In such cases, the search becomes dominated by a small set of similar individuals that lead GAs to converge to a sub-optimal solution and to stagnate, without reaching the desired objective. This problem is particularly common for hard-to-cover program branches, associated with an extremely large solution space.

In this paper, we propose an approach to solve this problem by integrating a mechanism for orthogonal exploration of the search space into standard GA. The diversity in the population is enriched by adding individuals in orthogonal directions, hence providing a more effective exploration of the solution space. To the best of our knowledge, no prior work has addressed explicitly the issue of evolution direction based diversification in the context of evolutionary testing. Results achieved on 17 Java classes indicate that the proposed enhancements make GA much more effective and efficient in automating the testing process. In particular, effectiveness (coverage) was significantly improved in 47% of the subjects and efficiency (search budget consumed) was improved in 85% of the subjects on which effectiveness remains the same.

### Categories and Subject Descriptors

D.2.5 [Software Engineering]: Testing and Debugging

### General Terms

Reliability, Verification

### Keywords

Search based testing, test case generation, orthogonal exploration, genetic algorithms, genetic drift

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.  
ISSTA '13, July 15-20, 2013, Lugano, Switzerland.  
Copyright 2013 ACM 978-1-4503-2159-4/13/07...\$15.00

### 1. INTRODUCTION

Search based techniques have been successfully applied to several areas of Software Engineering in general, and Software Testing in particular [14]. Specifically, a class of algorithms known as Evolutionary Algorithms (EA) has been widely used in automating the generation of test data [21].

In evolutionary test case generation [27], candidate test cases are encoded into a population of individuals (solutions). These individuals are then evaluated by executing them against the System Under Test (SUT) and their fitness (goodness) is measured with respect to a given criterion, e.g., branch coverage. Evolutionary operators such as selection, crossover and mutation are then used to derive a new generation of individuals with better quality than their parents. The process is repeated for several generations either until the criterion is satisfied or the fixed search budget is finished. A particularly popular flavor of EA are Genetic Algorithms (GA) [11], quite often and successfully adopted in the testing community [2, 10, 26, 27].

The success of evolutionary test case generation techniques in general, and GA in particular, depends on several factors. One of these factors is the level of diversity among the individuals in the population, which directly affects the exploration ability of the search. Indeed, given the fact that the search space is usually extremely large, the GA has to maintain an adequate level of diversity in the population in order to effectively explore the search space and look for alternative, potential solutions. The basic genetic operators play an important role in maintaining this diversity [13]: crossover is generally considered as a background operator to maintain diversity for exploration, while selection and mutation are used to exploit the current solution to find nearby better ones [24]. However, these operators might not suffice by themselves in maintaining enough diversity. Depending on the difficulty of the current search target and the type of selection scheme in use, individuals in the population can become too similar to one another, eventually converging to a single, sub-optimal individual. Hence, the ability of the search to explore new areas of the search space is greatly reduced. This phenomenon is referred to as genetic drift [23].

Evolutionary test case generation techniques that employ GAs could be seriously affected by genetic drift. Especially for hard-to-cover program branches, it can happen that nearly all the candidate individuals in the search become similar. Consequently, genetic operators simply recombine genetic material among these similar individuals, resulting in convergence to a sub-optimal solution, which does not achieve the desired objective. Hence, test data generation

stagnates when the search space is trapped in

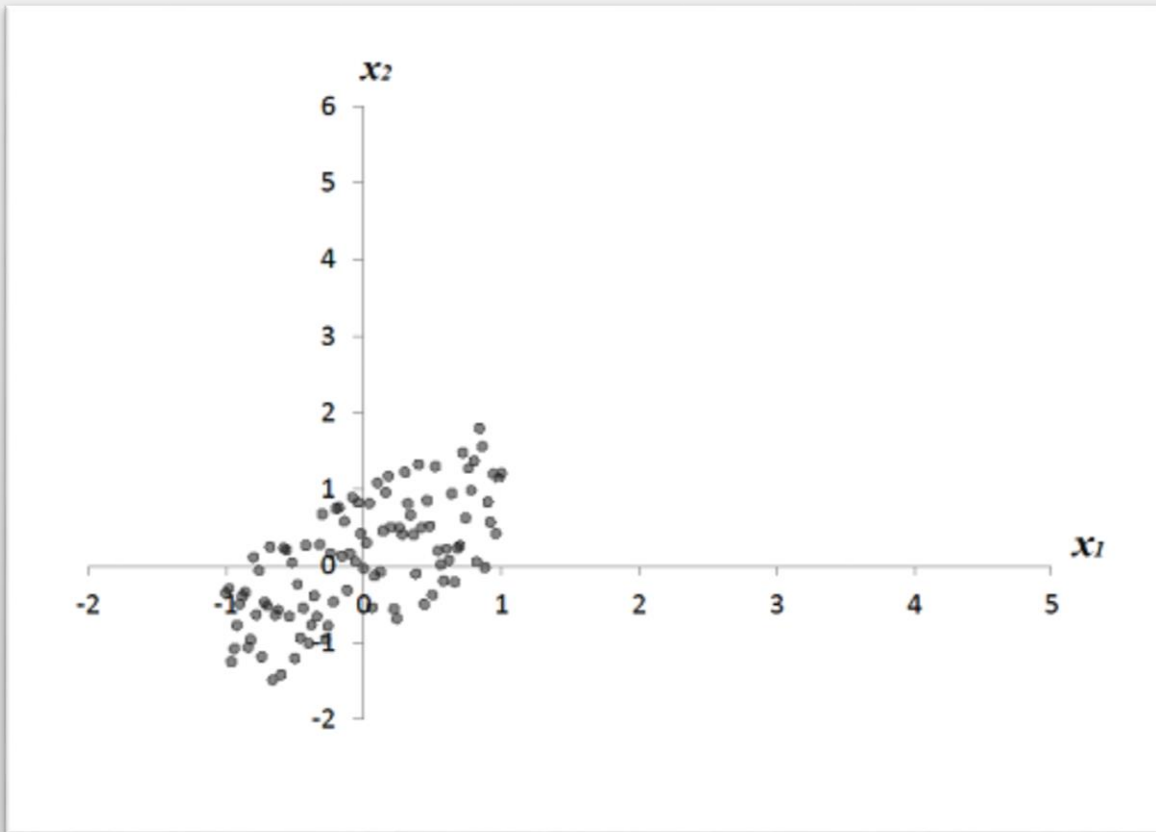
techniques used for evolutionary test case generation for Algorithms that classically applied evolutionary test case generation to dynamic (C/C++, due to its population) the risk of finding standard GA (evolutionary solving high case remain

individuals to generate new good (previously non-existing), fit) into niche found within time, a by-product of the search process between the individuals and the population (individuals) the performance of diversity convergence (23) to (6)

Estimating the Evolution Direction of Populations to Improve Genetic Algorithm. A. De Lucia , M. Di Penta, R. Oliveto, A. Panichella  
GECCO 2012

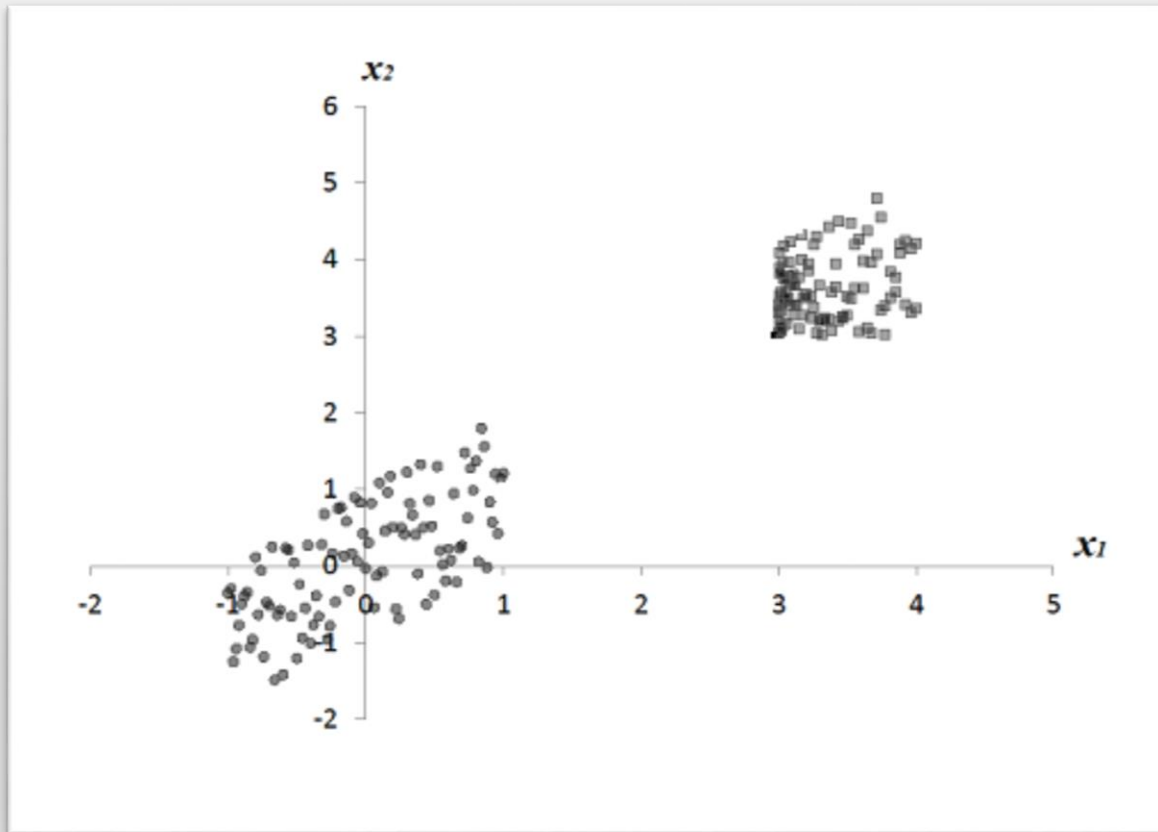
Orthogonal Exploration of the Search Space in Evolutionary Test Case Generation F. M. Kifetew, A. Panichella , A. De Lucia , R. Oliveto, P. Tonella  
ISSTA 2013

# What is the evolution direction?



$P(t)$  = Population at generation  $t$

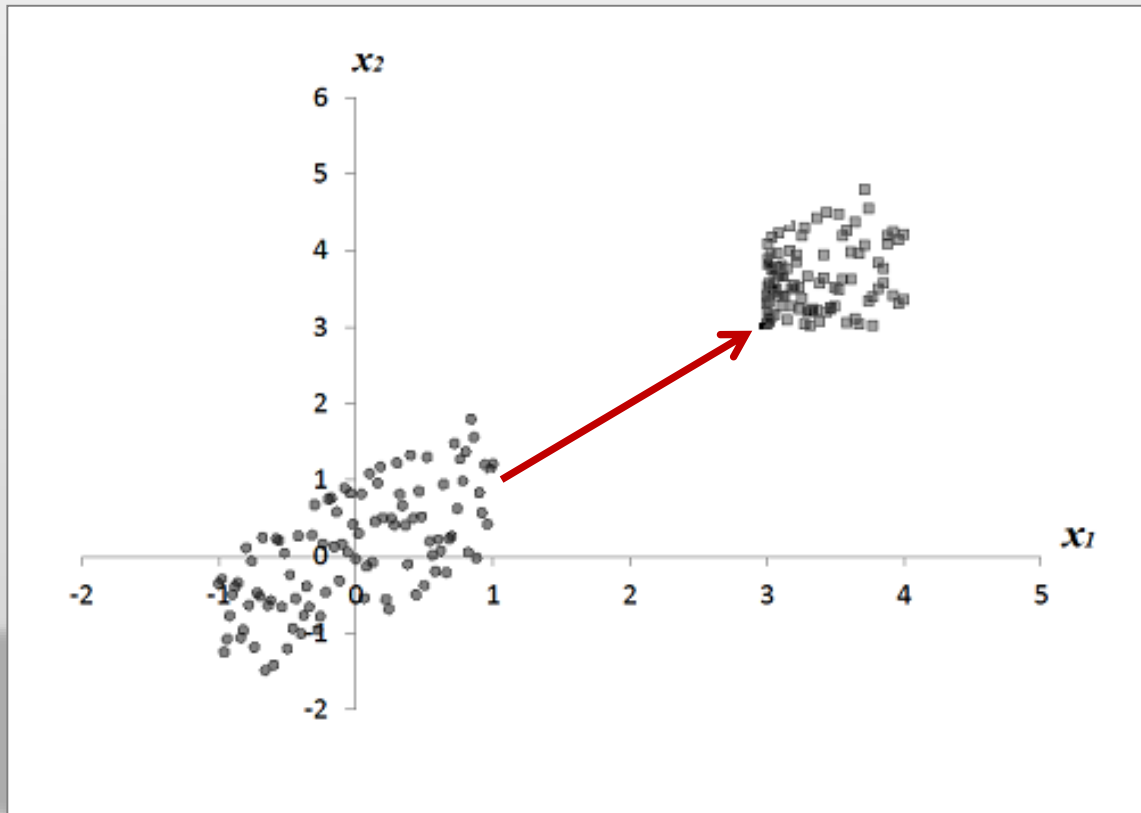
# What is the evolution direction?



$P(t)$  = Population at generation  $t$

$P(t+k)$  = Population after  $k$  generations

# What is the evolution direction?

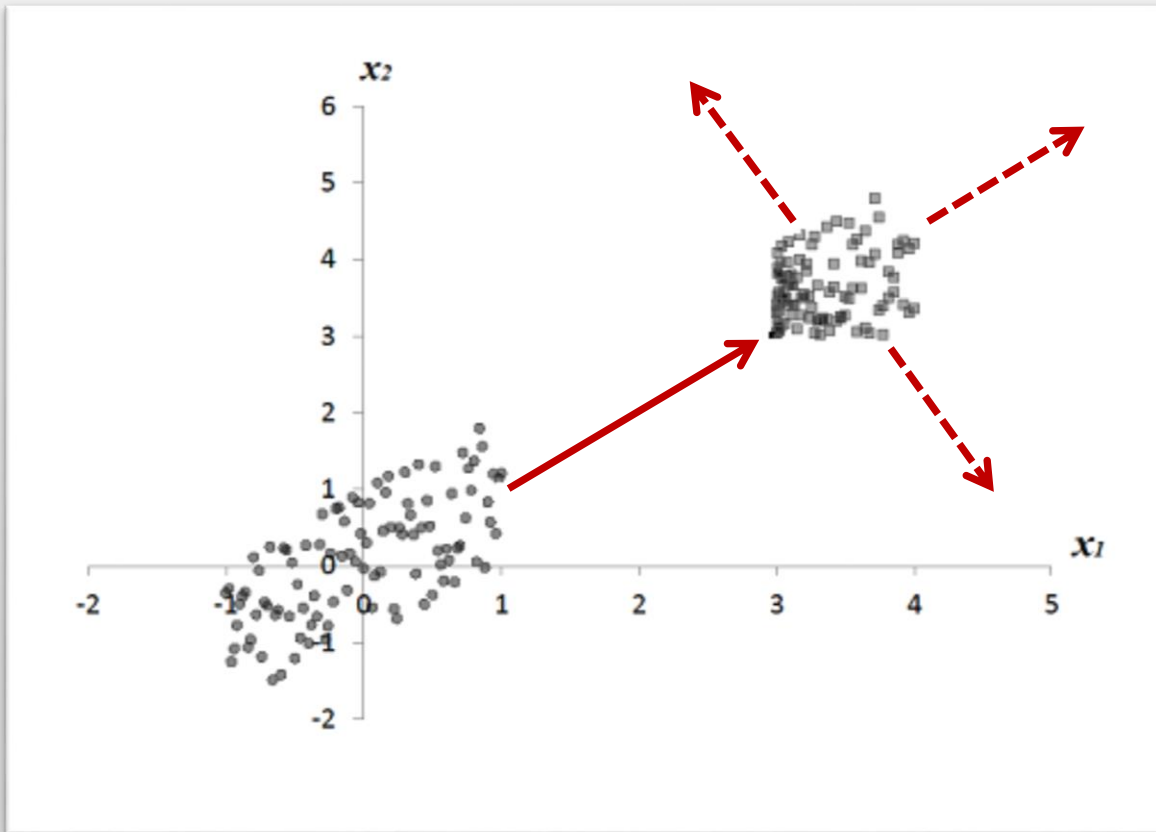


$P(t)$  = Population at generation  $t$

$P(t+k)$  = Population after  $k$  generations

Evolution Directions

# Why?



$P(t)$  = Population at generation  $t$

$P(t+k)$  = Population after  $k$  generations

Evolution Directions

Orthogonal Individuals

# How?

The basic idea is that a population of solutions  $P$  provided by GA at generation  $t$  can be viewed as a  $m \times n$  matrix

$$P = \begin{matrix} & \begin{matrix} \text{variable 1} \\ \text{variable 2} \\ \vdots \\ \text{variable } n \end{matrix} \\ \begin{bmatrix} p_{1,1} & p_{1,2} & \cdots & p_{1,n} \\ p_{2,1} & p_{2,2} & \cdots & p_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ p_{m,1} & p_{m,2} & \cdots & p_{m,n} \end{bmatrix} & \begin{matrix} \text{Individual 1} \\ \text{Individual 2} \\ \vdots \\ \text{Individual } m \end{matrix} \end{matrix}$$

# How? Singual Value Decomposition

Each matrix  $\mathbf{P}$  can be decomposed in the product of three different matrices:

$$\mathbf{P} = \mathbf{U} \cdot \mathbf{\Sigma} \cdot \mathbf{V}$$

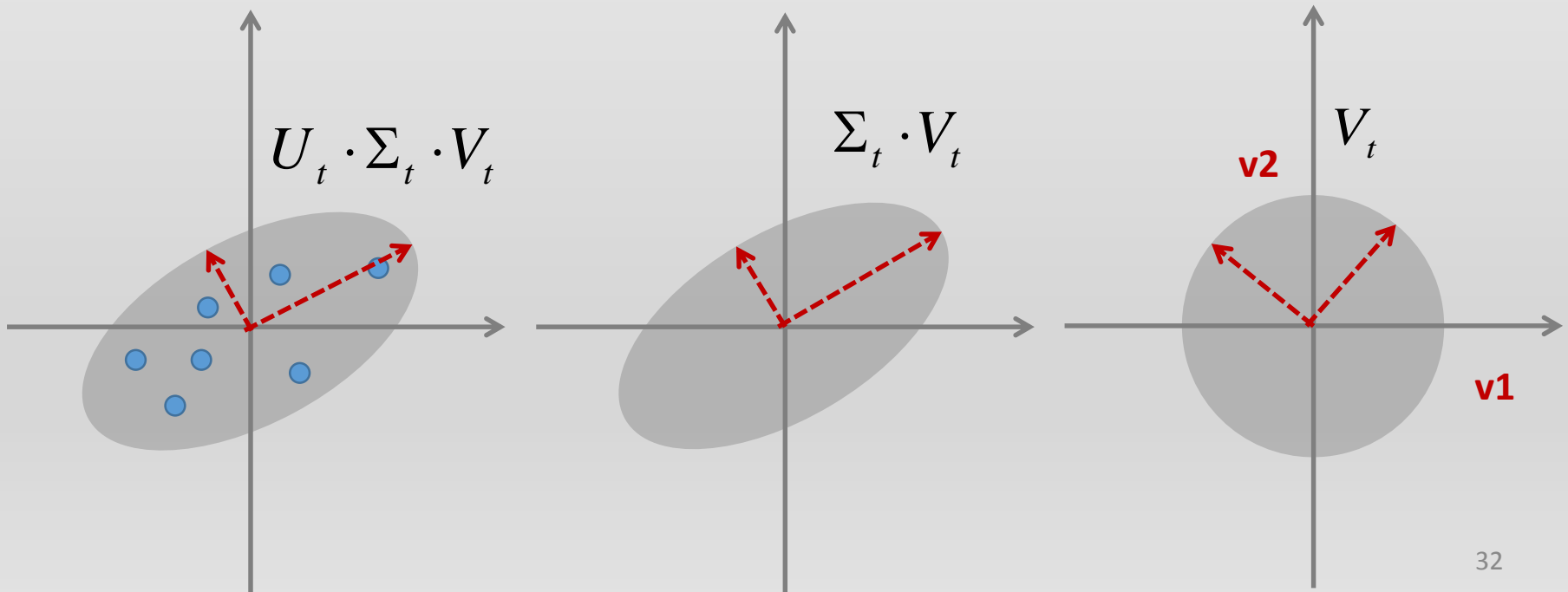
$m \times n$       $m \times k$       $k \times k$       $k \times n$

# How? Singular Value Decomposition

Each matrix  $P$  can be decomposed in the product of three different matrices:

$$P = U \cdot \Sigma \cdot V$$

$m \times n$      $m \times k$      $k \times k$      $k \times n$





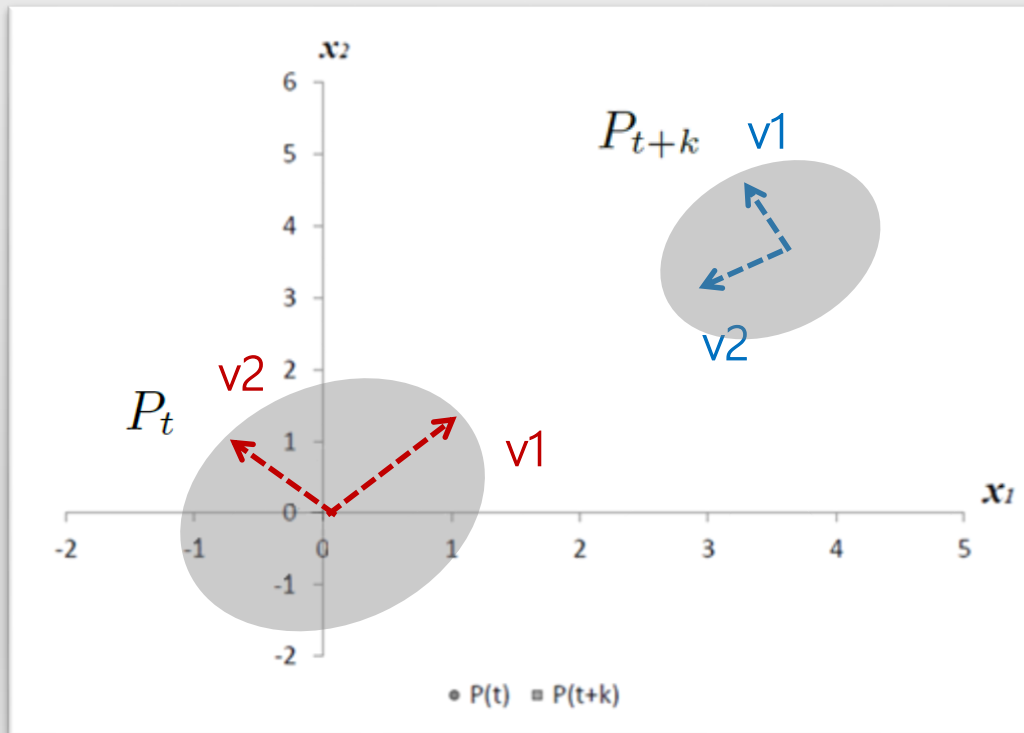
# Using SVD for Evolution Direction

Population at generation  $t$

$$P_t = U_t \cdot \Sigma_t \cdot V_t$$

Population at generation  $t + k$

$$P_{t+k} = U_{t+k} \cdot \Sigma_{t+k} \cdot V_{t+k}$$



The current evolution direction is related to

$$\bar{V} = V_{t+k} - V_t$$

$$\bar{\Sigma} = \Sigma_{t+k} - \Sigma_t$$

# Using SVD for Evolution Direction

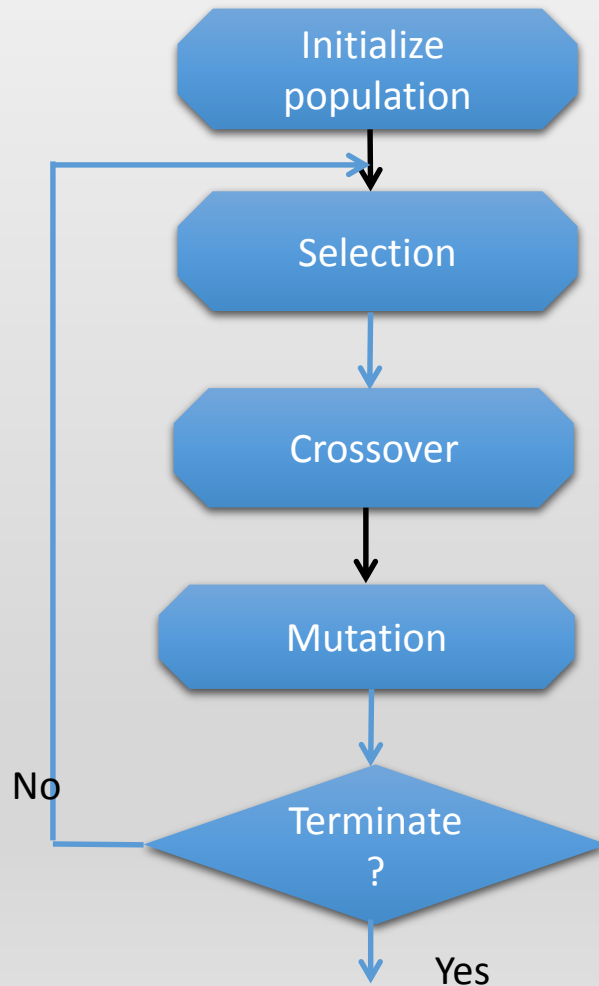
Generating new orthogonal individuals through the following equation:

$$P_{t+k}^* = U_{t+k} \cdot \left( \Sigma_{t+k} + \bar{\Sigma} \right) \cdot \left( V_{t+k} + \bar{V}_{\perp} \right)$$

Shifting Operator:  
 $P_{t+k}^*$  is  $\bar{\Sigma}$  shifted in  
the search space

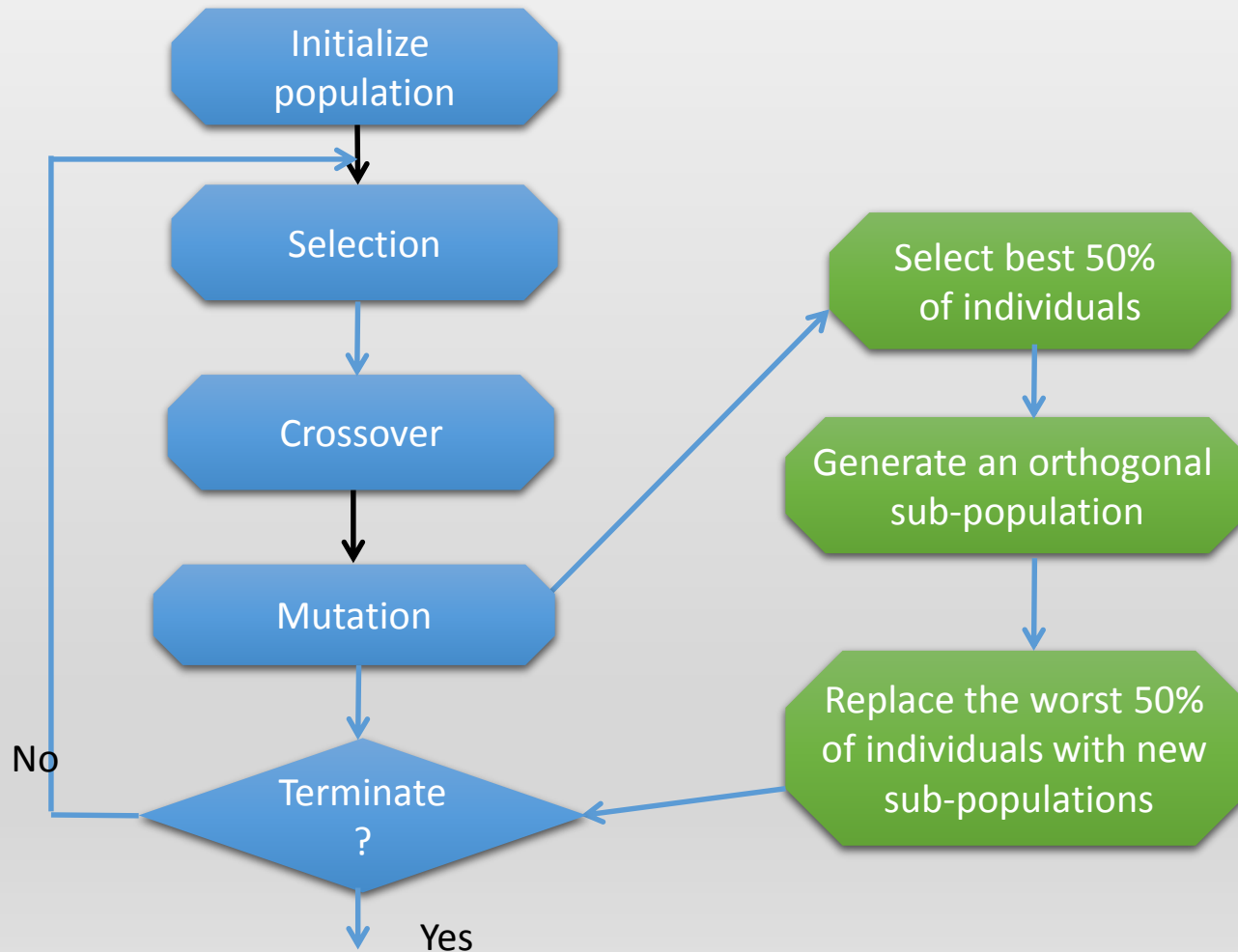
Rotating Operator:  
 $P_{t+k}^*$  is rotated with  
respect to  $P_{t+k}$

# Integration SVD with NSGA-II



- Non Dominated Sorting Algorithm
- Crowding Distance
- Tournament Selection
  
- Multi-points crossover
  
- Bit-flip mutation

# SVD + NSGA-II



# Empirical Evaluation

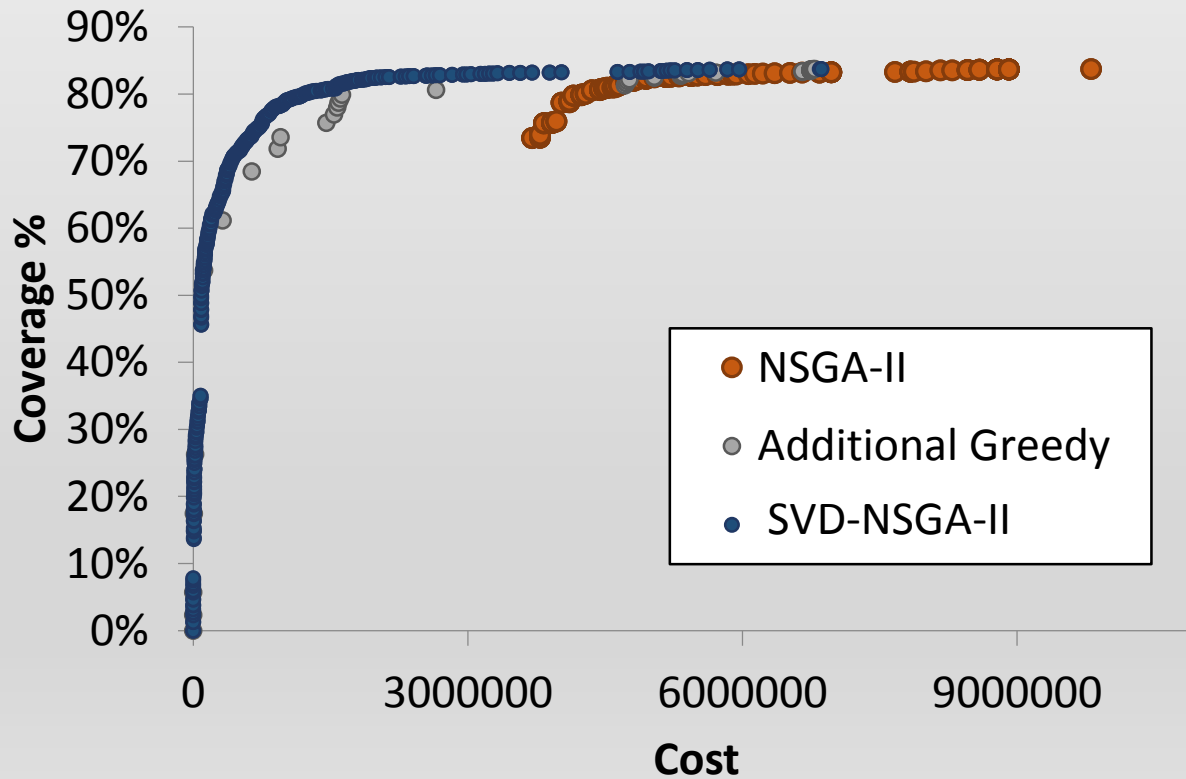


# Case Study Design

RQs	Systems	Algorithms	Metrics
<b>RQ1:</b> To what extent does SVD-NSGA-II produce near optimal solutions, compared to alternative techniques?	Bash, Flex, Grep, Gzip, Printtokens, Printtokens2, Schedule, Schedule2, Sed, Space, vim	Add.Greedy Algorithm NSGA-II SVD-NSGA-II	1. Pareto Front Size 2. N. non-dominated solutions
<b>RQ2:</b> What is the cost-effectiveness of SVD-NSGA-II compared to the alternative techniques?	Bash, Flex, Grep, Gzip, Printtokens, Printtokens2, Schedule, Schedule2, Sed, Space, vim	Add.Greedy Algorithm NSGA-II SVD-NSGA-II	1. Fault Detection Rate

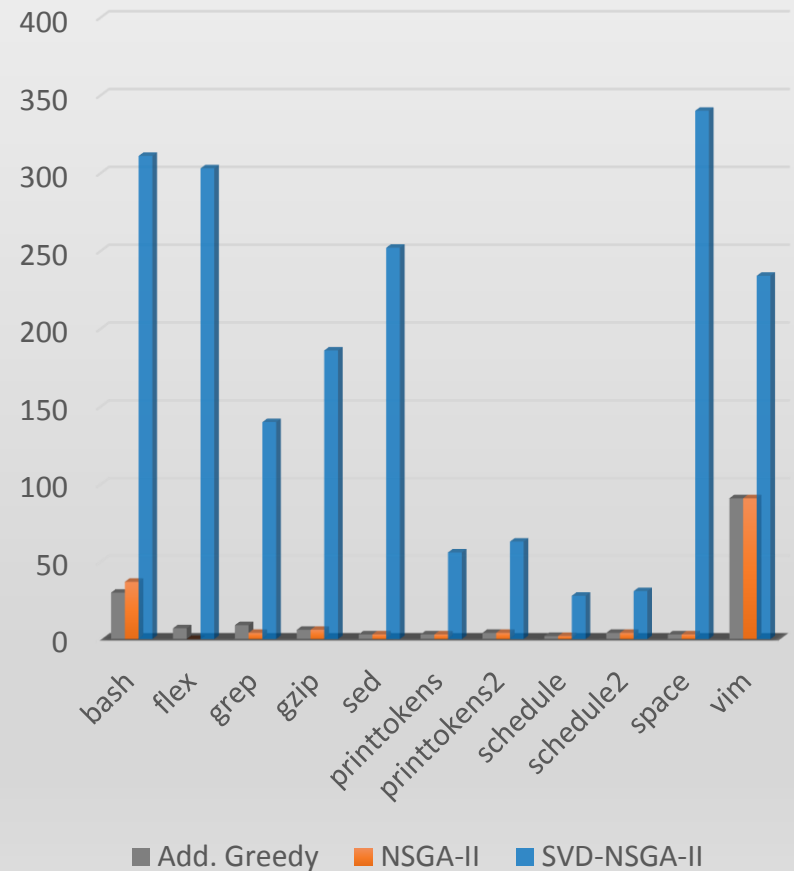
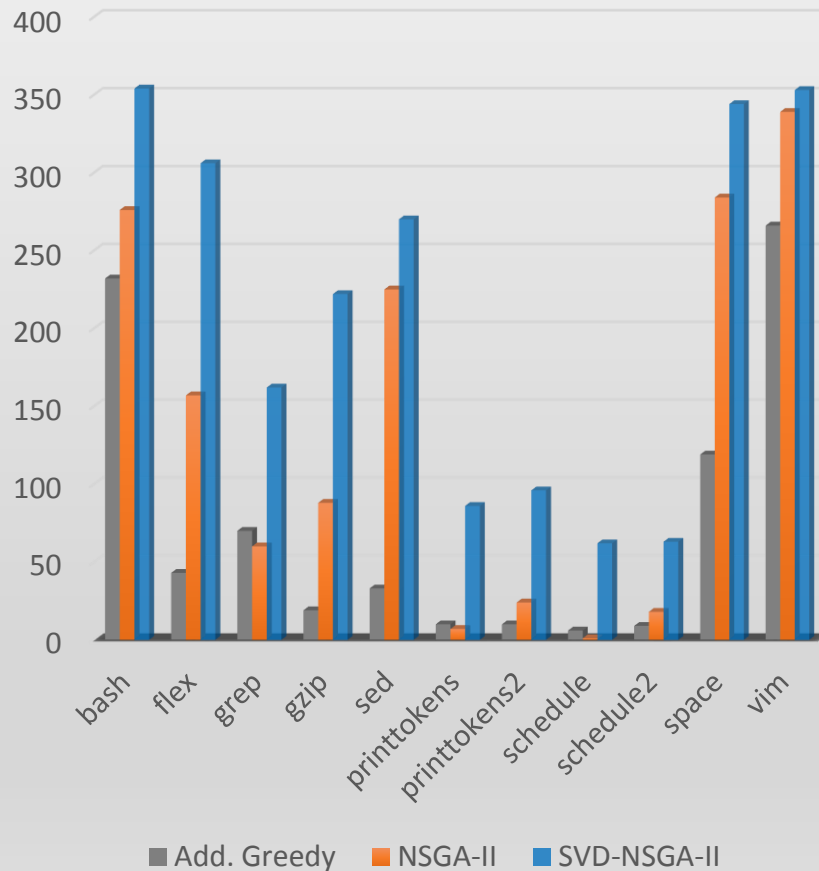
# Results

**RQ1:** To what extent does SVD-NSGA-II produce near optimal solutions, compared to alternative techniques?



# Results

**RQ1:** To what extent does SVD-NSGA-II produce near optimal solutions, compared to alternative techniques?

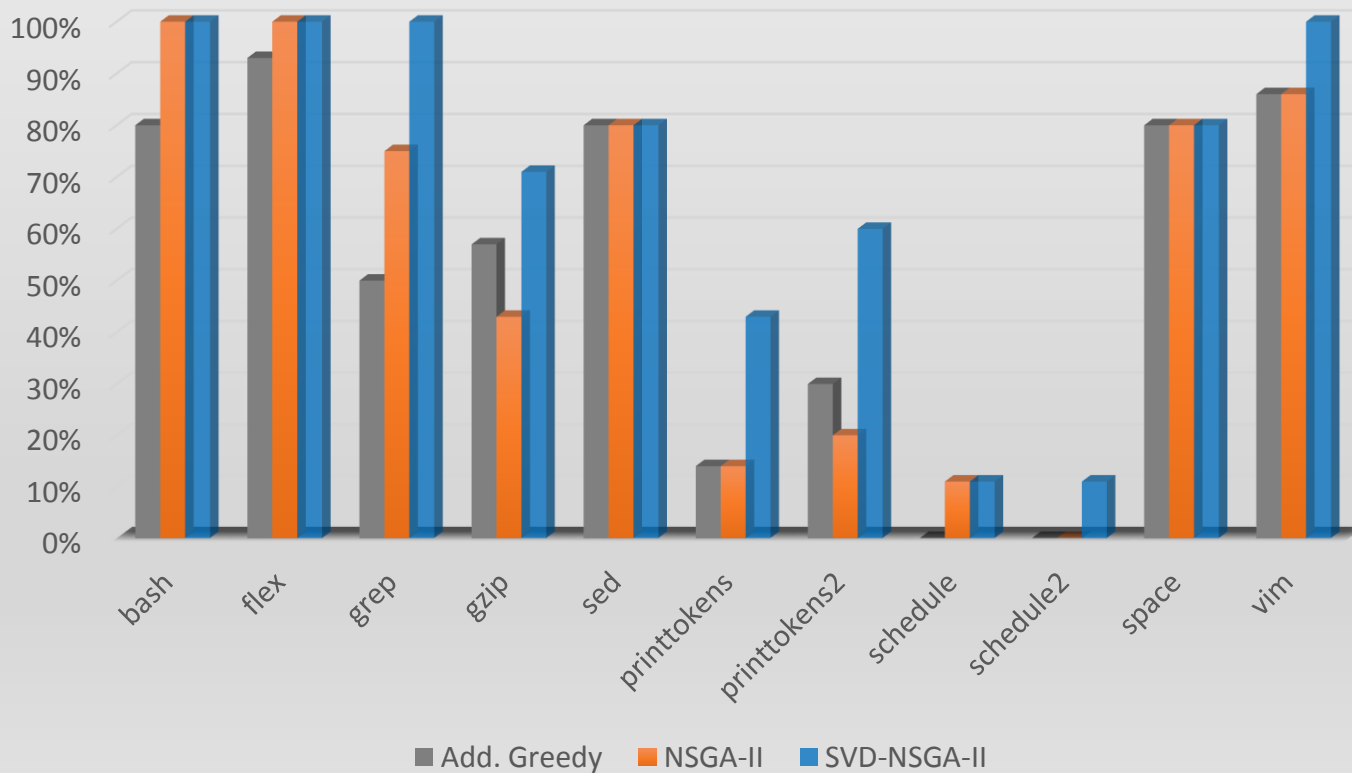




# Results

**RQ2:** What is the cost-effectiveness of SVD-NSGA-II compared to the alternative techniques?

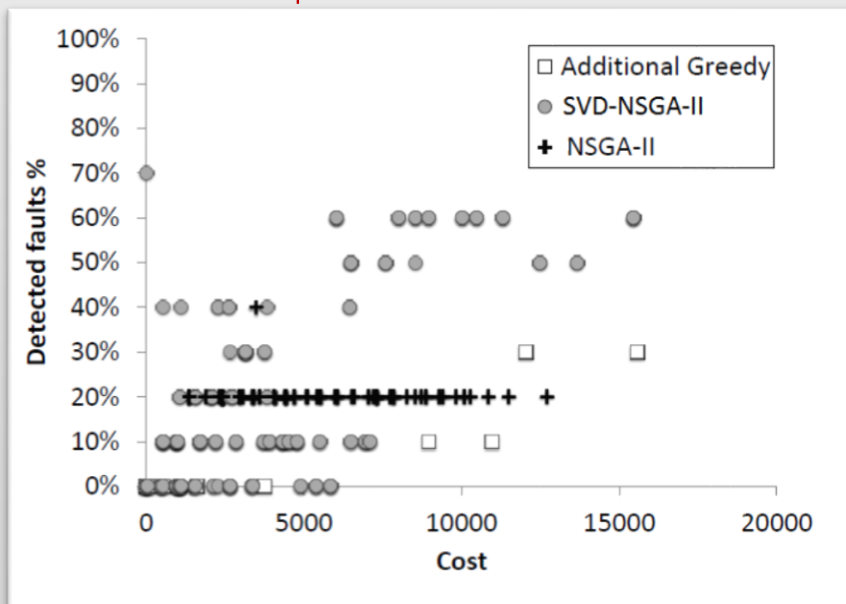
## Fault Detection Rate



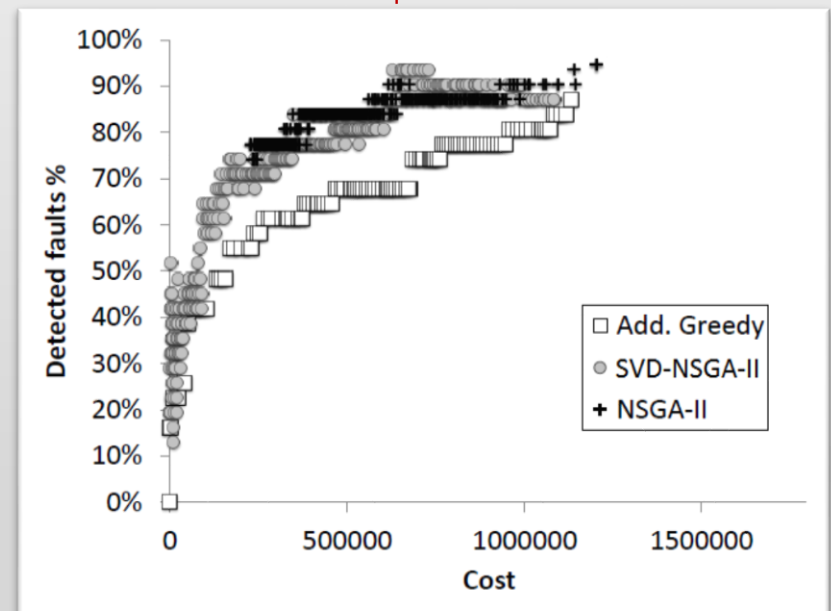
# Results

**RQ2:** What is the cost-effectiveness of SVD-NSGA-II compared to the alternative techniques?

printtokens2



space



# Summary

100%

The optimality was improved

The effectiveness was improved  
at same level of execution cost

64%

# Conclusion

## Test Suite Optimization

### Redundant Test Cases

#### Input

- Program  $P = \{e_1, e_2, \dots, e_n\}$
- Test Cases covering parts of P  
 $T = \{t_1, t_2, \dots, t_n\}$

#### Problem

Finding the minimal sub-set  
 $T^* \subseteq T$  such that  $\bigcup_{t \in T^*} t(P) = P$

### Hitting set

#### Input

- Set  $P = \{e_1, e_2, \dots, e_n\}$
- A collection of sub-sets of P  
 $T = \{t_1, t_2, \dots, t_n\}$

#### Problem

Finding the minimal sub-set  
 $T^* \subseteq T$  such that  $\bigcup_{t \in T^*} t_i = P$

19

# Conclusion



# Conclusion

Multi-C...

## Using SVD for Evolution Direction

Generating new orthogonal individuals through the following equation:

$$P_{t+k}^* = U_{t+k} \cdot (\underline{\Sigma_{t+k}} + \bar{\Sigma}) \cdot (\underline{V_{t+k}} + \bar{V}_{\perp})$$

**Shifting Operator:**  
 $P_{t+k}^*$  is  $\bar{\Sigma}$  shifted in the search space

**Rotating Operator:**  
 $P_{t+k}^*$  is rotated with respect to  $P_{t+k}$

63

# Conclusion

## Summary

100% The optimality was improved

The effectiveness was improved  
at same level of execution cost 64%

83

A green rectangular sign with rounded corners and a white border is mounted on two wooden posts. The sign features the words "Thank You" in a large, white, sans-serif font. The background is a bright blue sky filled with scattered white, fluffy clouds. The sign is tilted slightly to the right.

**Thank You**