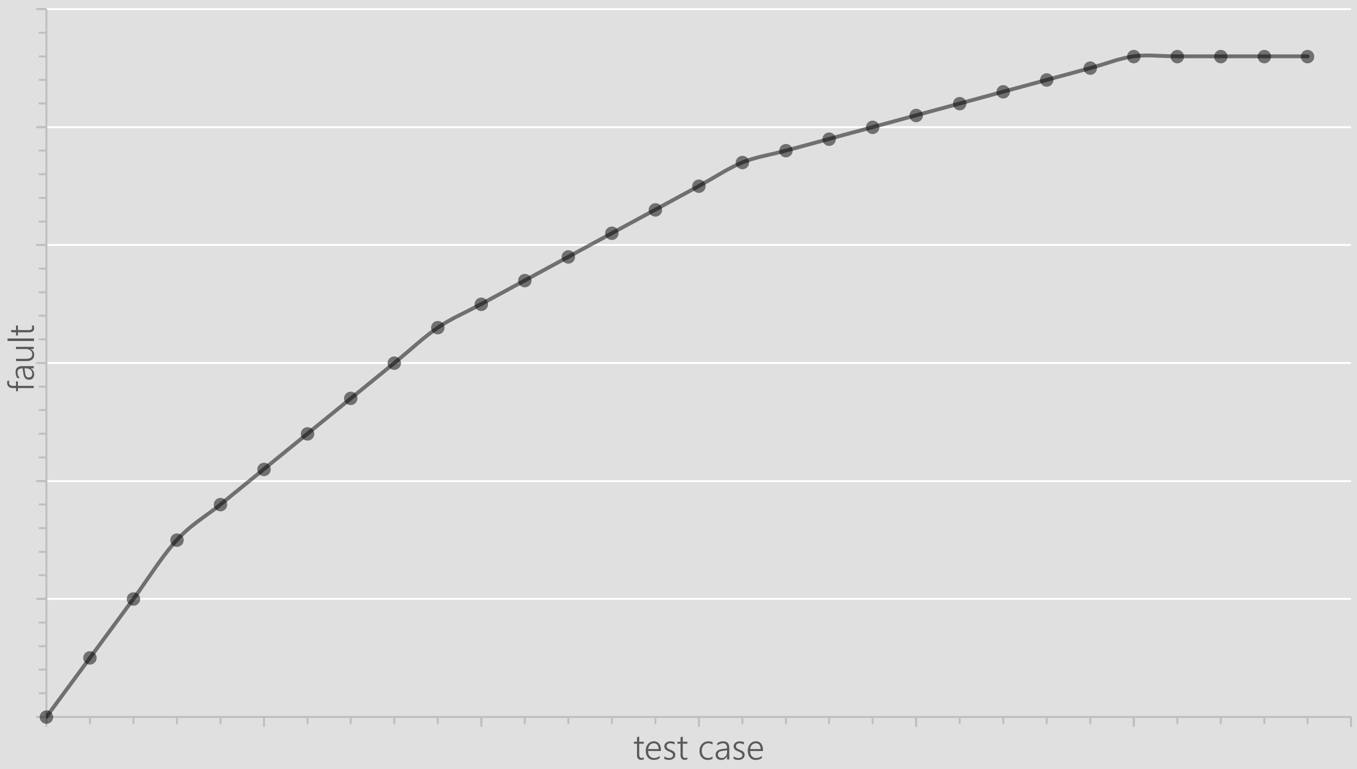
Hypervolume-based Search for Test Case Prioritization



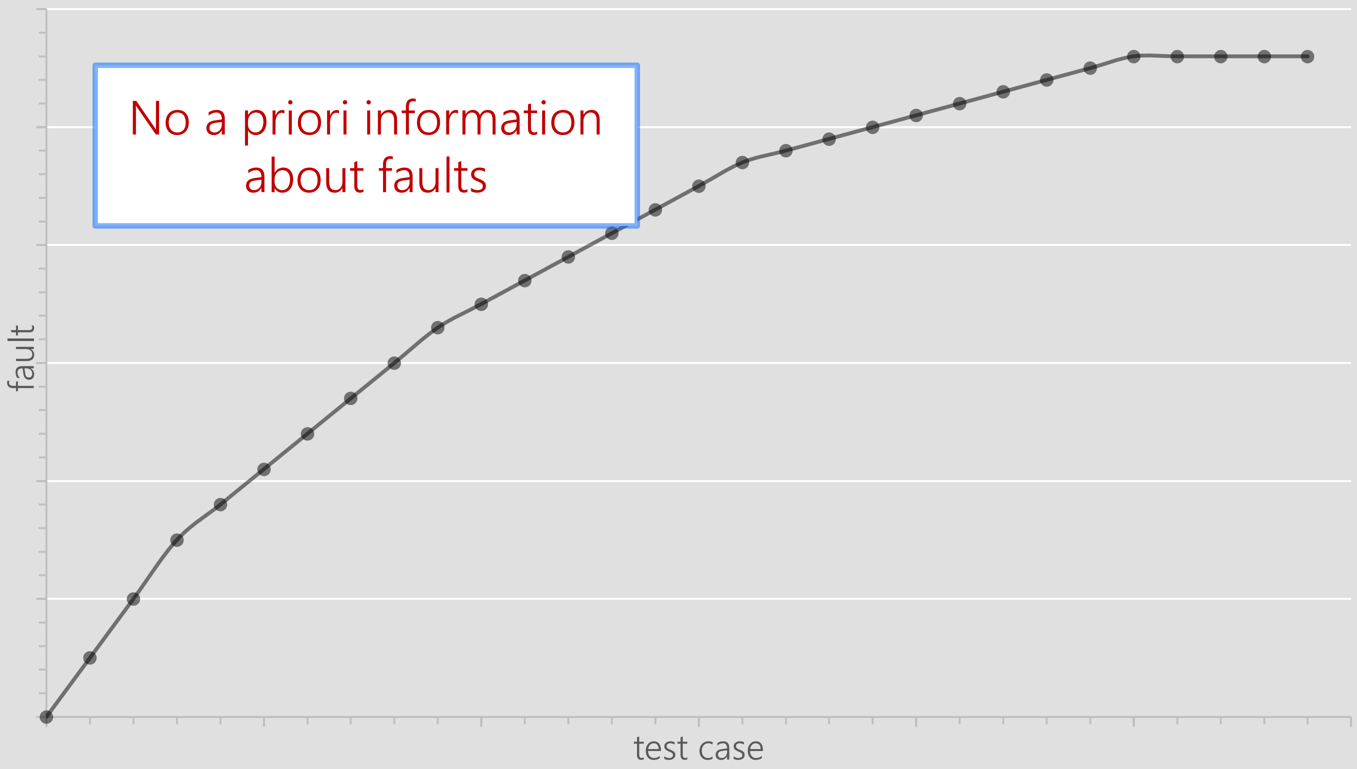

Dario Di Nucci Annibale Panichella Andy Zaidman Andrea De Lucia

Test Case Prioritization



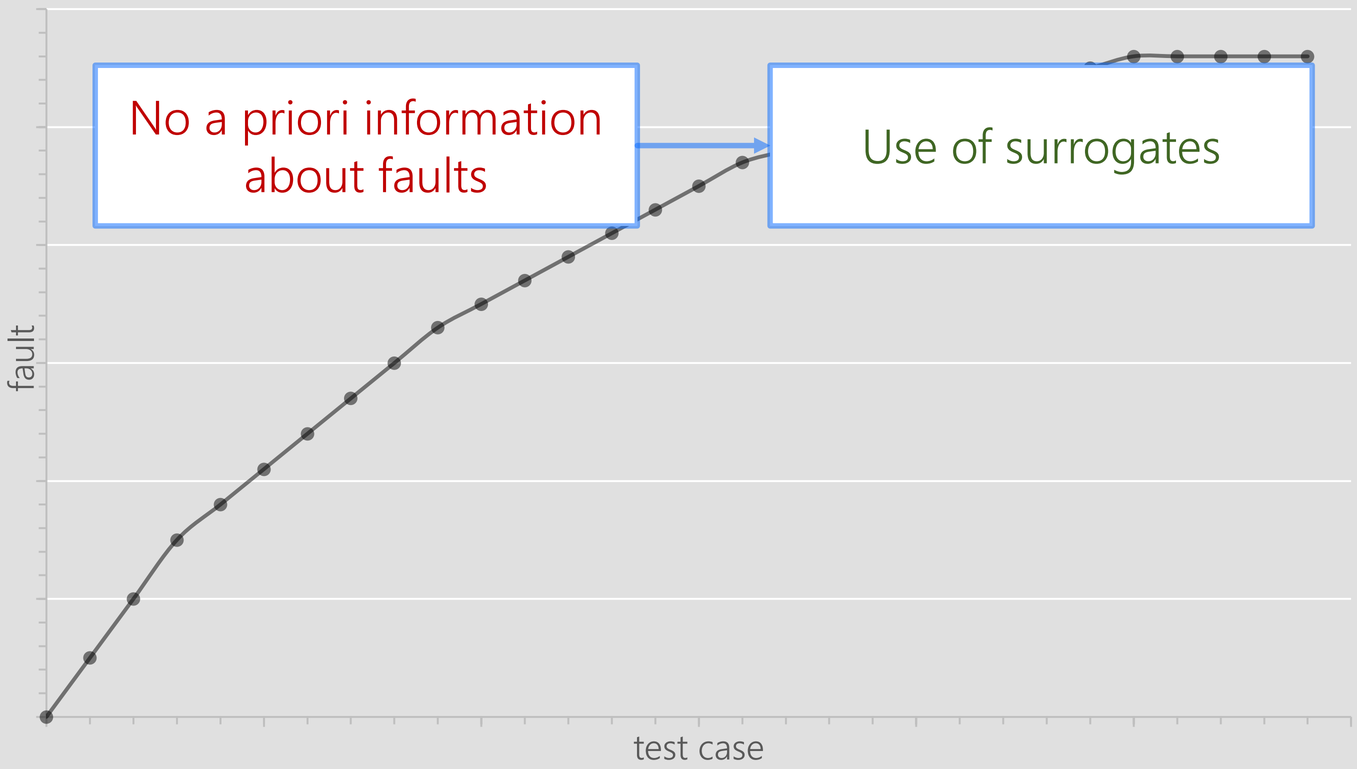
Find an ideal sorting for executing test cases in order to reveal faults earlier

Test Case Prioritization



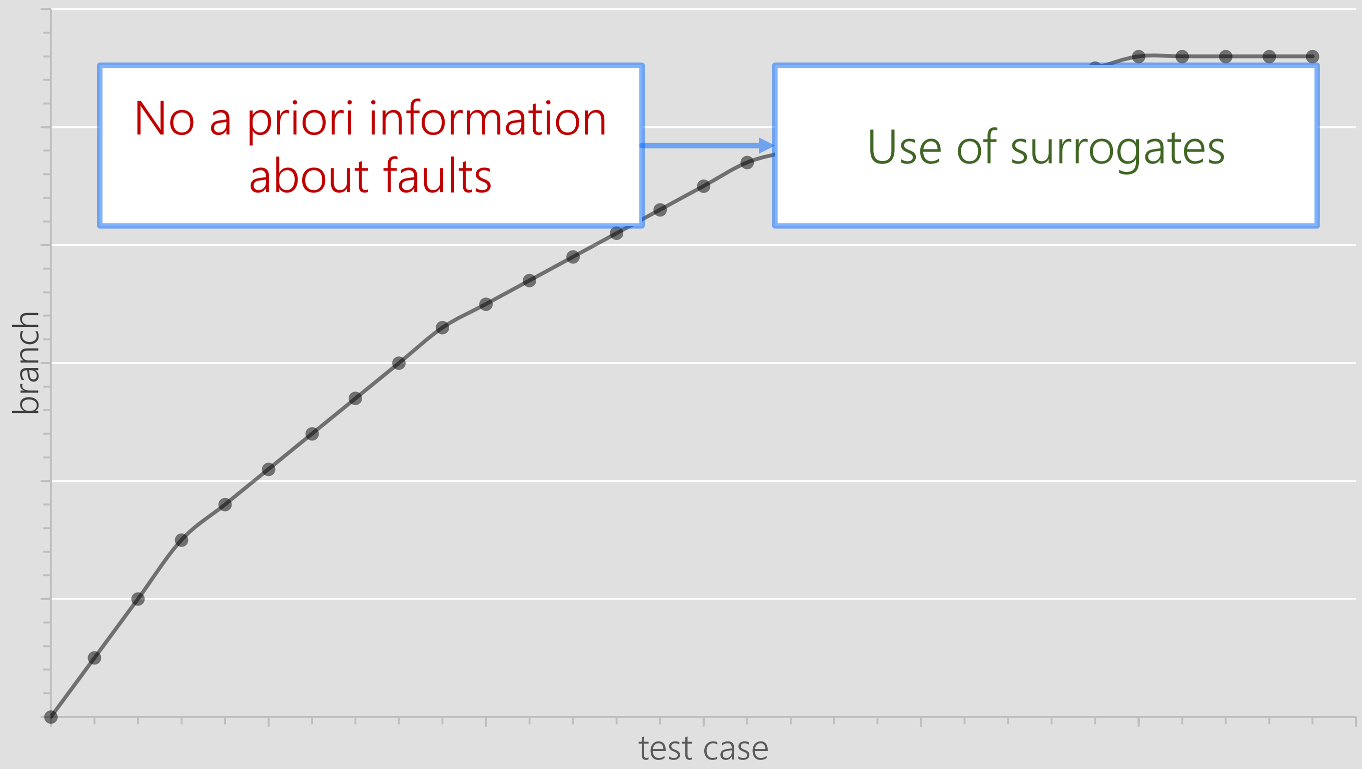
Find an ideal sorting for executing test cases in order to reveal faults earlier

Test Case Prioritization



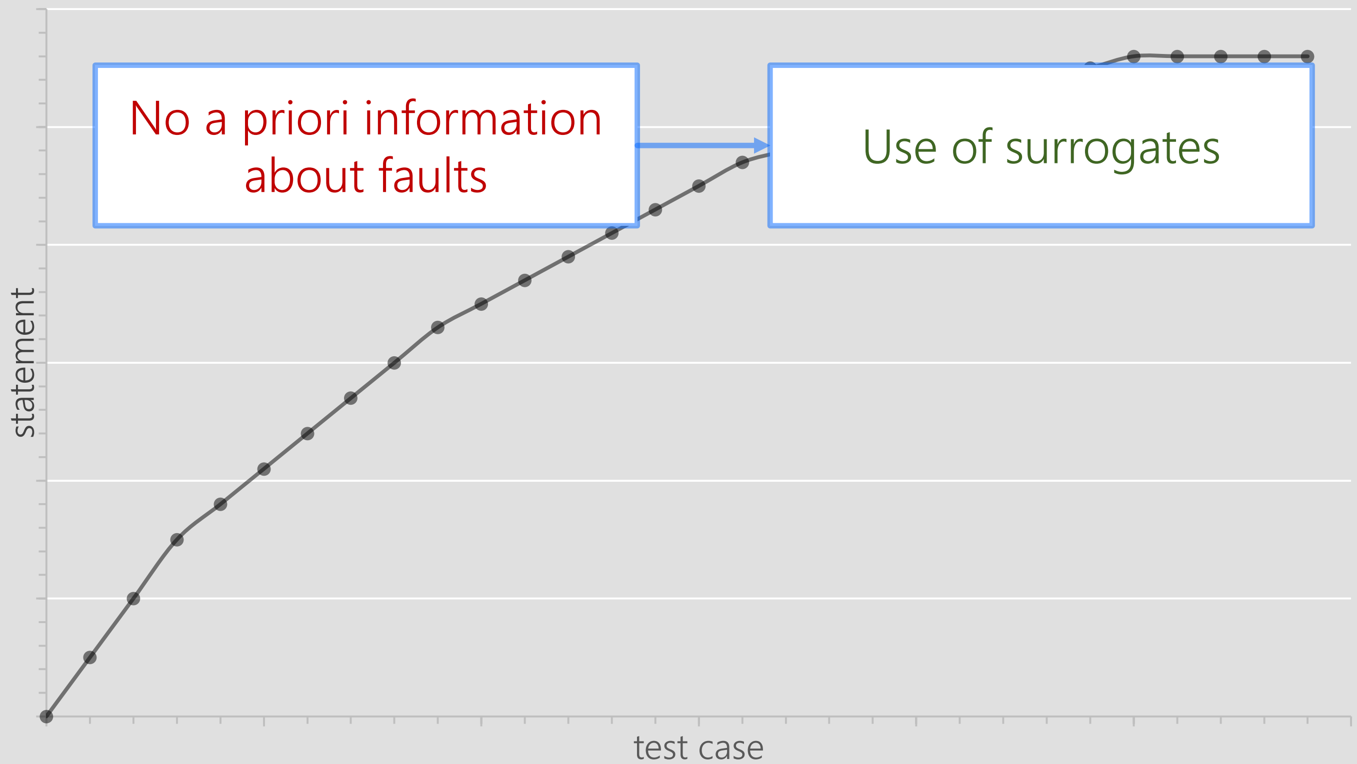
Find an ideal sorting for executing test cases in order to reveal faults earlier

Test Case Prioritization



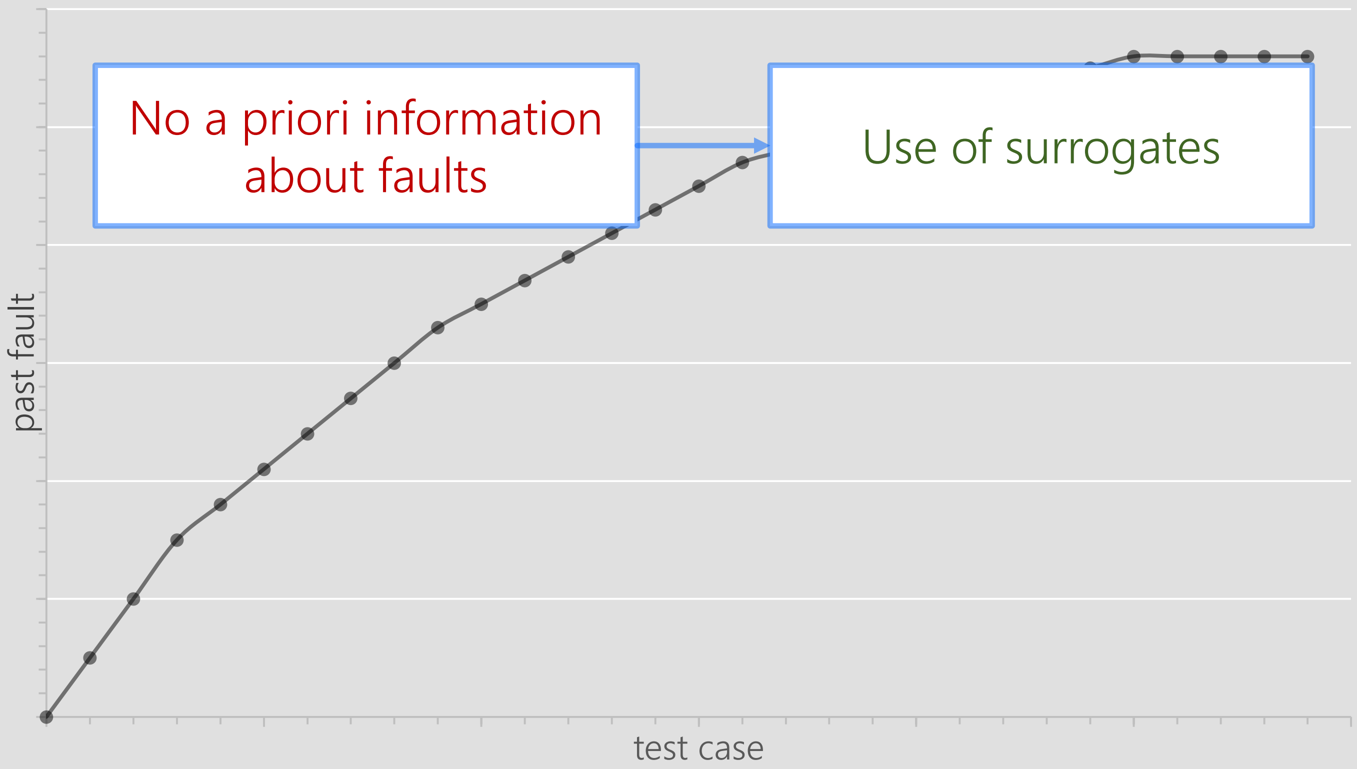
Find an ideal sorting for executing test cases in order to reveal faults earlier

Test Case Prioritization



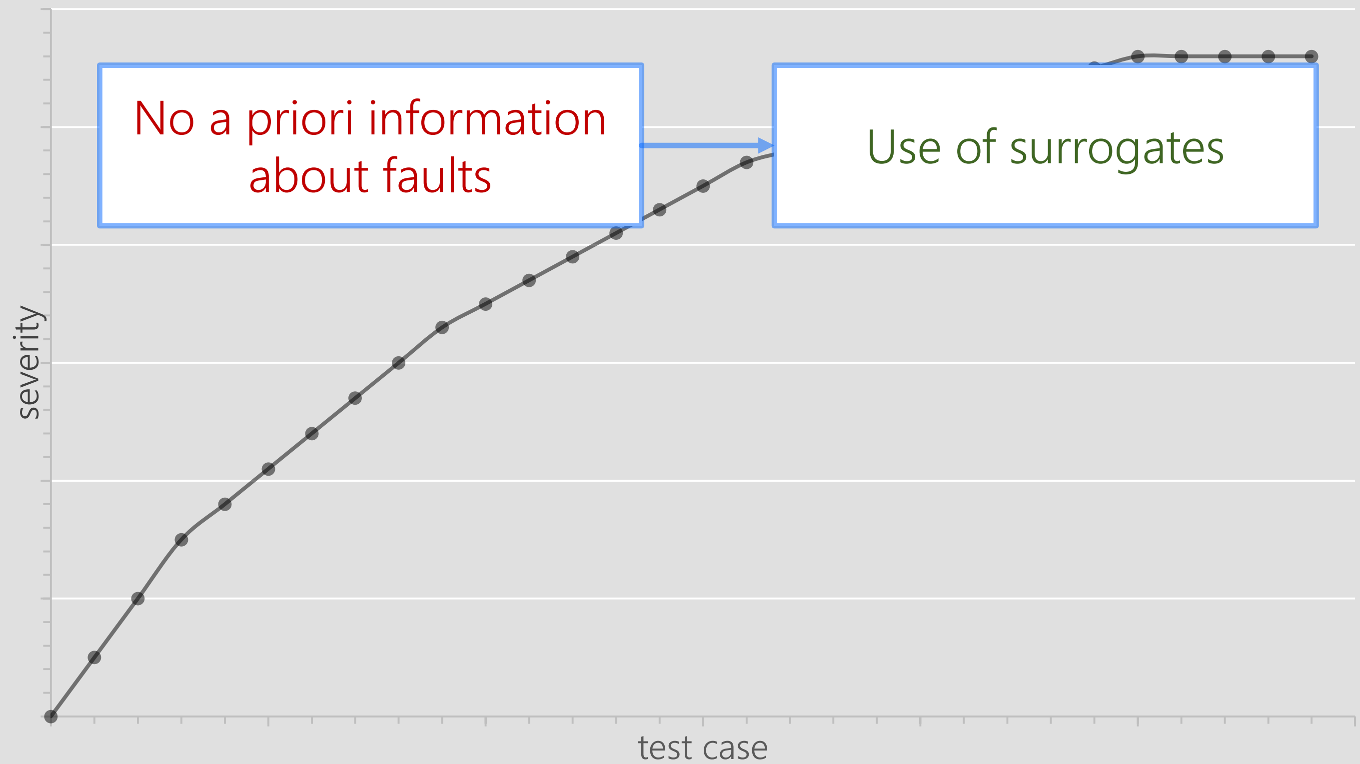
Find an ideal sorting for executing test cases in order to reveal faults earlier

Test Case Prioritization



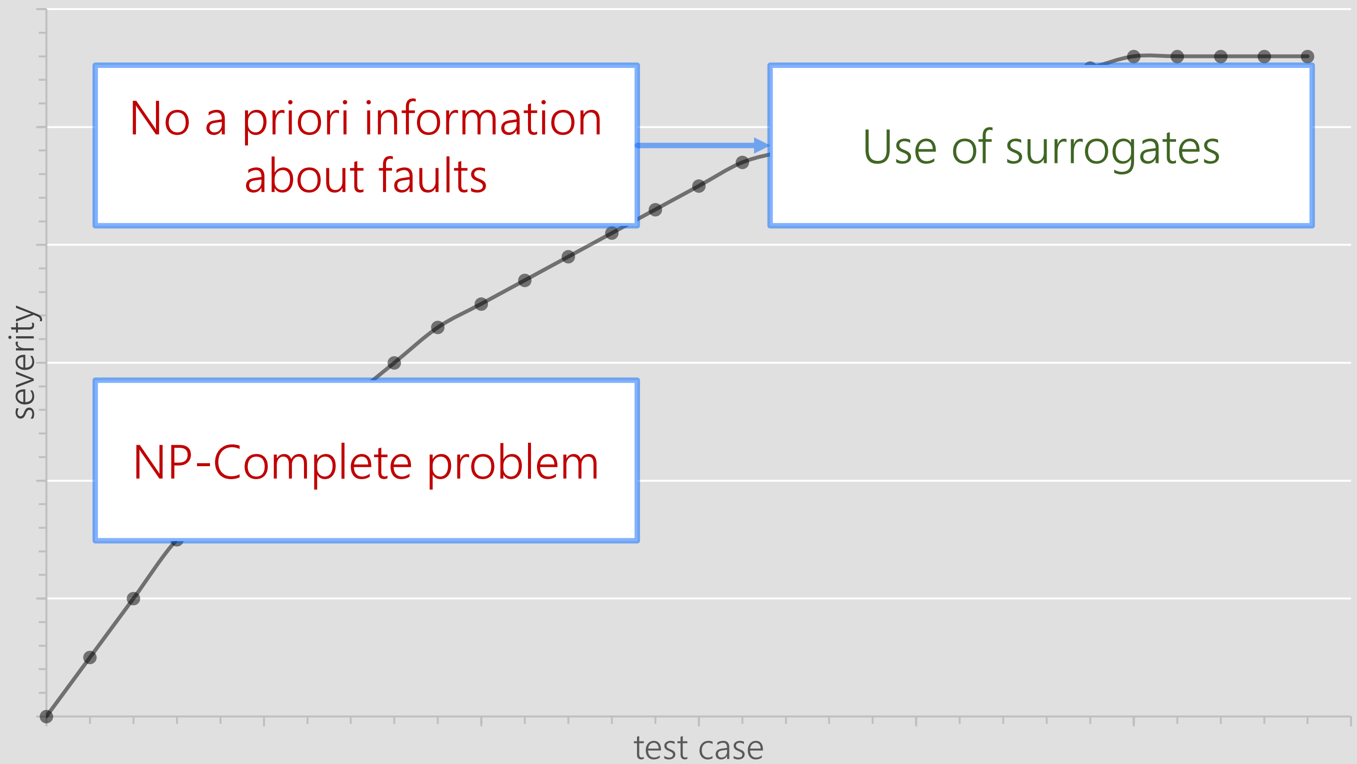
Find an ideal sorting for executing test cases in order to reveal faults earlier

Test Case Prioritization



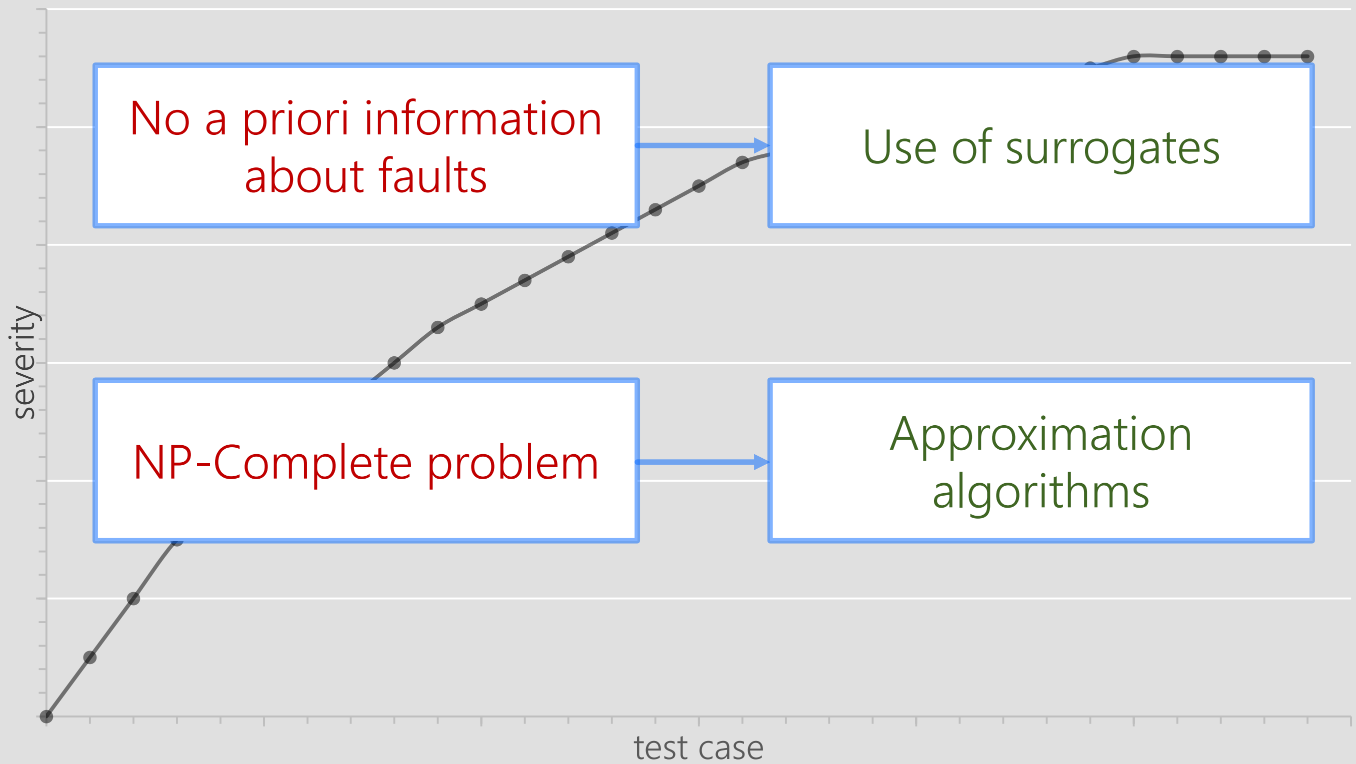
Find an ideal sorting for executing test cases in order to reveal faults earlier

Test Case Prioritization



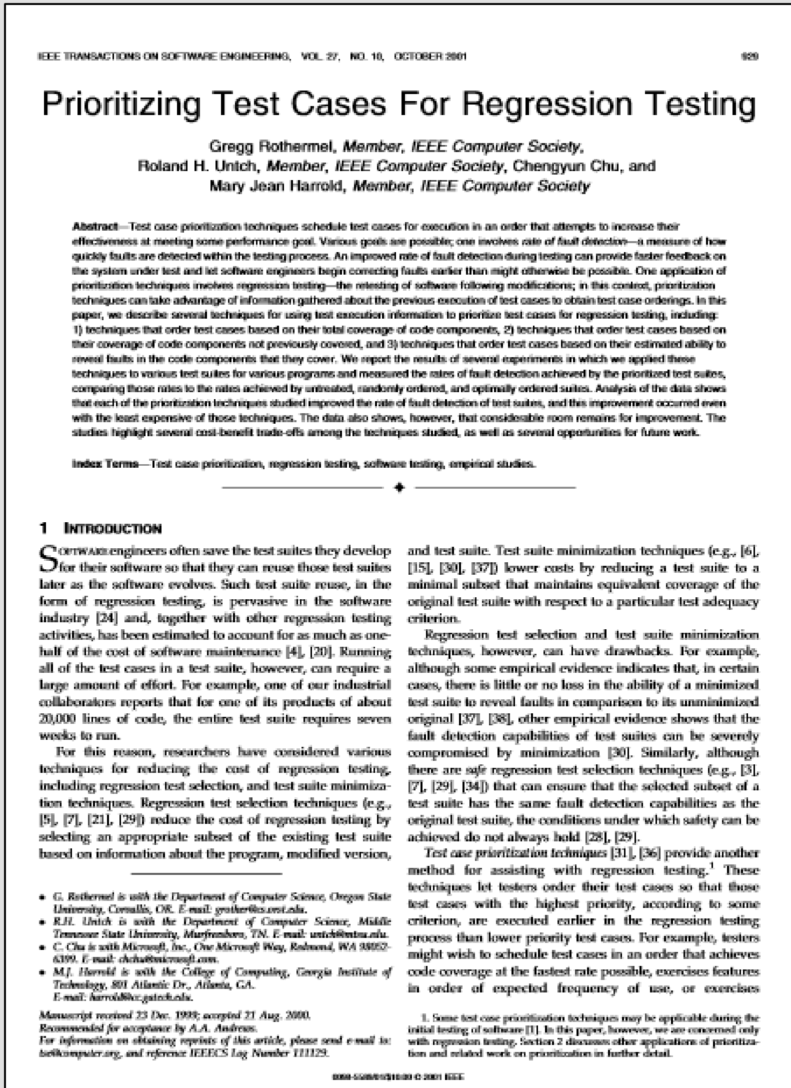
Find an ideal sorting for executing test cases in order to reveal faults earlier

Test Case Prioritization



Find an ideal sorting for executing test cases in order to reveal faults earlier

Greedy Algorithms



Prioritizing Test Cases For Regression Testing

Gregg Rothermel, *Member, IEEE Computer Society*,
Roland H. Untch, *Member, IEEE Computer Society*, Chengyun Chu, and
Mary Jean Harrold, *Member, IEEE Computer Society*

Abstract—Test case prioritization techniques schedule test cases for execution in an order that attempts to increase their effectiveness at meeting some performance goal. Various goals are possible; one involves *rate of fault detection*—a measure of how quickly faults are detected within the testing process. An improved rate of fault detection during testing can provide faster feedback on the system under test and let software engineers begin correcting faults earlier than might otherwise be possible. One application of prioritization techniques involves regression testing—the retesting of software following modifications; in this context, prioritization techniques can take advantage of information gathered about the previous execution of test cases to obtain test case orderings. In this paper, we describe several techniques for using test execution information to prioritize test cases for regression testing, including: 1) techniques that order test cases based on their total coverage of code components, 2) techniques that order test cases based on their coverage of code components not previously covered, and 3) techniques that order test cases based on their estimated ability to reveal faults in the code components that they cover. We report the results of several experiments in which we applied these techniques to various test suites for various programs and measured the rates of fault detection achieved by the prioritized test suites, comparing those rates to the rates achieved by untreated, randomly ordered, and optimally ordered suites. Analysis of the data shows that each of the prioritization techniques studied improved the rate of fault detection of test suites, and this improvement occurred even with the least expensive of those techniques. The data also shows, however, that considerable room remains for improvement. The studies highlight several cost-benefit trade-offs among the techniques studied, as well as several opportunities for future work.

Index Terms—Test case prioritization, regression testing, software testing, empirical studies.

1 Introduction

Software engineers often save the test suites they develop for their software so that they can reuse those test suites later as the software evolves. Such test suite reuse, in the form of regression testing, is pervasive in the software industry [24] and, together with other regression testing activities, has been estimated to account for as much as one-half of the cost of software maintenance [4], [20]. Running all of the test cases in a test suite, however, can require a large amount of effort. For example, one of our industrial collaborators reports that for one of its products of about 20,000 lines of code, the entire test suite requires seven weeks to run.

For this reason, researchers have considered various techniques for reducing the cost of regression testing, including regression test selection, and test suite minimization techniques. Regression test selection techniques (e.g., [5], [7], [21], [29]) reduce the cost of regression testing by selecting an appropriate subset of the existing test suite based on information about the program, modified version, and test suite. Test suite minimization techniques (e.g., [6], [15], [30], [37]) lower costs by reducing a test suite to a minimal subset that maintains equivalent coverage of the original test suite with respect to a particular test adequacy criterion.

Regression test selection and test suite minimization techniques, however, can have drawbacks. For example, although some empirical evidence indicates that, in certain cases, there is little or no loss in the ability of a minimized test suite to reveal faults in comparison to its unminimized original [37], [38], other empirical evidence shows that the fault detection capabilities of test suites can be severely compromised by minimization [30]. Similarly, although there are *safe* regression test selection techniques (e.g., [3], [7], [29], [34]) that can ensure that the selected subset of a test suite has the same fault detection capabilities as the original test suite, the conditions under which safety can be achieved do not always hold [28], [29].

Test case prioritization techniques [31], [36] provide another method for assisting with regression testing.[1] These techniques let testers order their test cases so that those test cases with the highest priority, according to some criterion, are executed earlier in the regression testing process than lower priority test cases. For example, testers might wish to schedule test cases in an order that achieves code coverage at the fastest rate possible, exercises features in order of expected frequency of use, or exercises

- *G. Rothermel is with the Department of Computer Science, Oregon State University, Corvallis, OR. E-mail: grother@cs.orst.edu.*
- *R.H. Untch is with the Department of Computer Science, Middle Tennessee State University, Murfreesboro, TN. E-mail: untch@mtsu.edu.*
- *C. Chu is with Microsoft, Inc., One Microsoft Way, Redmond, WA 98052-6399. E-mail: chchu@microsoft.com.*
- *M.J. Harrold is with the College of Computing, Georgia Institute of Technology, 801 Atlantic Dr., Atlanta, GA. E-mail: harrold@cc.gatech.edu.*

Manuscript received 23 Dec. 1999; accepted 21 Aug. 2000.
Recommended for acceptance by A.A. Andrews.
For information on obtaining reprints of this article, please send e-mail to: tse@computer.org, and reference IEEECS Log Number 111129.

1. Some test case prioritization techniques may be applicable during the initial testing of software [1]. In this paper, however, we are concerned only with regression testing. Section 2 discusses other applications of prioritization and related work on prioritization in further detail.

Greedy Algorithms

IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 27, NO. 10, OCTOBER 2001 929

Prioritizing Test Cases For Regression Testing

Gregg Rothermel, *Member, IEEE Computer Society*,
Roland H. Untch, *Member, IEEE Computer Society*, Chengyun Chu, and
Mary Jean Harrold, *Member, IEEE Computer Society*

Abstract—Test case prioritization techn
effectiveness at meeting some performa
quickly faults are detected within the test
the system under test and let software e
prioritization techniques involves regres
techniques can take advantage of inform
paper, we describe several techniques f
1) techniques that order test cases base
their coverage of code components not p
reveal faults in the code components th
techniques to various test suites for vario
comparing those rates to the rates achie
that each of the prioritization techniques
with the least expensive of those techni
studies highlight several cost-benefit tra

Index Terms—Test case prioritization,

1 Introduction

Software engineers often save the tes
for their software so that they can r
later as the software evolves. Such tes
form of regression testing, is pervas
industry [24] and, together with othe
activities, has been estimated to accoun
half of the cost of software maintenanc
all of the test cases in a test suite, how
large amount of effort. For example, o
collaborators reports that for one of it
20,000 lines of code, the entire test s
weeks to run.

For this reason, researchers have
techniques for reducing the cost of
including regression test selection, and
tion techniques. Regression test selecti
[5], [7], [21], [29]) reduce the cost of r
selecting an appropriate subset of th
based on information about the progra

• G. Rothermel is with the Department of Comp
University, Corvallis, OR. E-mail: grother@cs.
• R.H. Untch is with the Department of C
Tennessee State University, Murfreesboro, TN.
• C. Chu is with Microsoft, Inc., One Microsoft W
6399. E-mail: chchu@microsoft.com.
• M.J. Harrold is with the College of Compu
Technology, 801 Atlantic Dr., Atlanta, GA.
E-mail: harrold@cc.gatech.edu.

*Manuscript received 23 Dec. 1999; accepted 21 Au
Recommended for acceptance by A.A. Andrews.
For information on obtaining reprints of this arti
tse@computer.org, and reference IEEECS Log Num*

Technical Report TR-UNL-CSE-2006-0004, Department of Computer Science and Engineering, University of Nebraska–Lincoln, Lincoln, Nebraska, U.S.A., 12 March 2006

Cost-cognizant Test Case Prioritization

Alexey G. Malishevsky* Joseph R. Ruthruff† Gregg Rothermel† Sebastian Elbaum†

Abstract

Test case prioritization techniques schedule test cases for regression testing in an order that increases their ability to meet some performance goal. One performance goal, *rate of fault detection*, measures how quickly faults are detected within the testing process. Previous work has provided a metric, *APFD*, for measuring rate of fault detection, and techniques for prioritizing test cases in order to improve *APFD*. This metric and these techniques, however, assume that all test case and fault costs are uniform. In practice, test case and fault costs can vary, and in such cases the previous *APFD* metric and techniques designed to improve *APFD* can be unsatisfactory. This paper presents a new metric for assessing the rate of fault detection of prioritized test cases, $APFD_C$, that incorporates varying test case and fault costs. The paper also describes adjustments to previous prioritization techniques that allow them, too, to be "cognizant" of these varying costs. These techniques enable practitioners to perform a new type of prioritization: *cost-cognizant test case prioritization*. Finally, the results of a formative case study are presented. This study was designed to investigate the cost-cognizant metric and techniques and how they compare to their non-cost-cognizant counterparts. The study's results provide insights regarding the use of cost-cognizant test case prioritization in a variety of real-world settings.

Keywords: test case prioritization, empirical studies, regression testing

1 Introduction

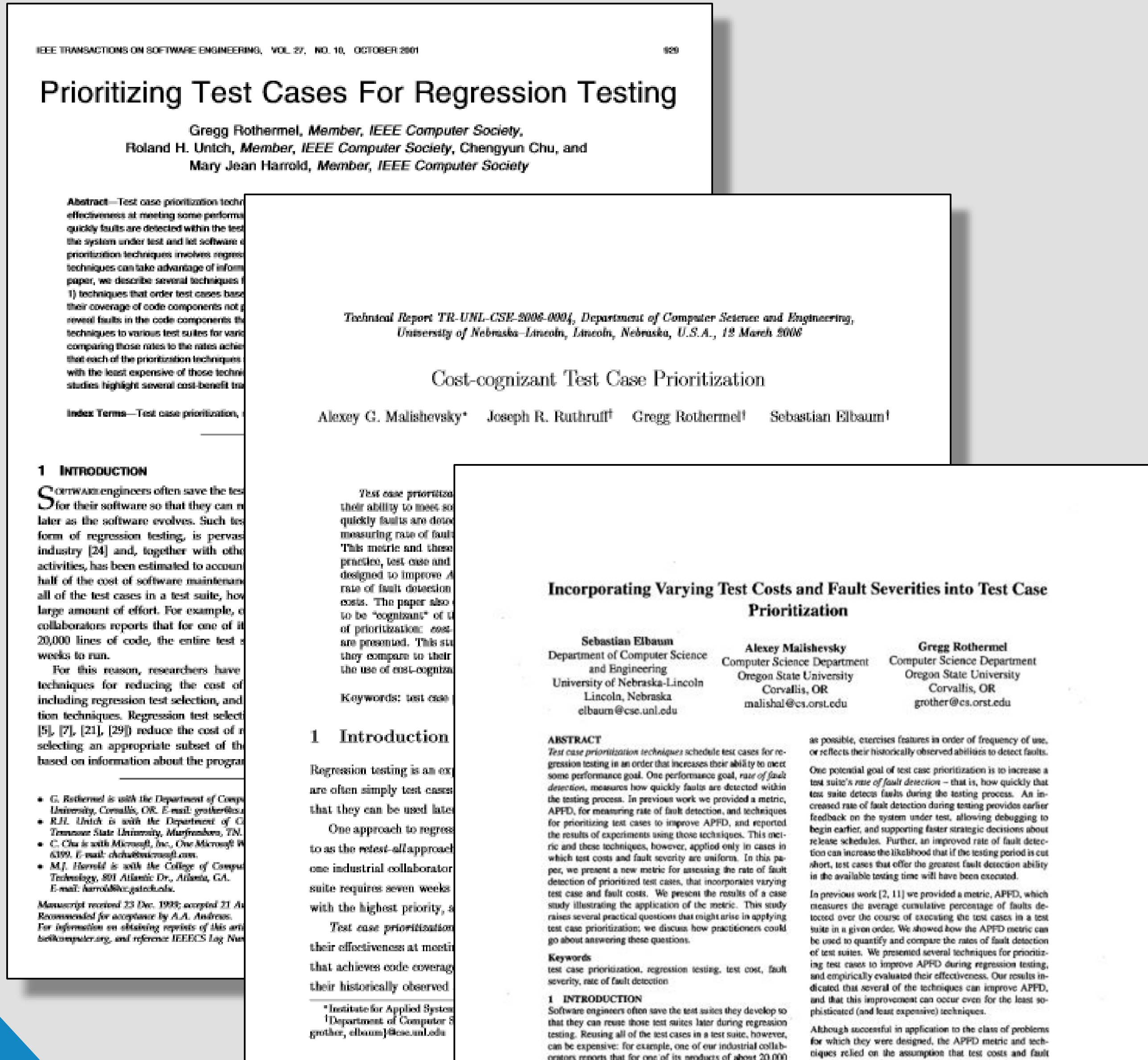
Regression testing is an expensive testing process used to detect regression faults [30]. Regression test suites are often simply test cases that software engineers have previously developed, and that have been saved so that they can be used later to perform regression testing.

One approach to regression testing is to simply rerun entire regression test suites; this approach is referred to as the *retest-all* approach [25]. The retest-all approach, however, can be expensive in practice: for example, one industrial collaborator reports that for one of its products of about 20,000 lines of code, the entire test suite requires seven weeks to run. In such cases, testers may want to order their test cases such that those with the highest priority, according to some criterion, are run earlier than those with lower priority.

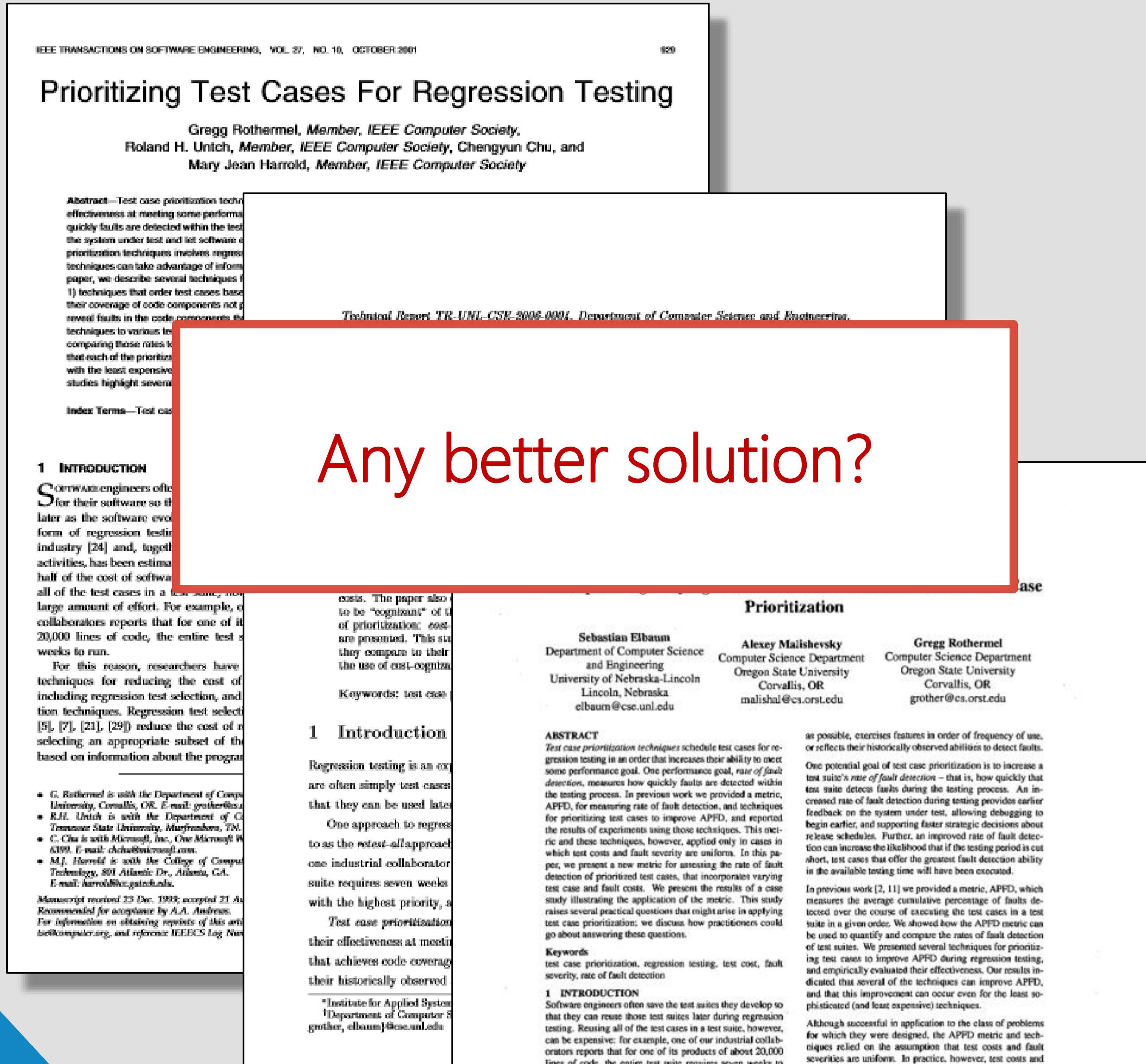
Test case prioritization techniques [9, 10, 11, 39, 40, 44] schedule test cases in an order that increases their effectiveness at meeting some performance goal. For example, test cases might be scheduled in an order that achieves code coverage as quickly as possible, exercises features in order of frequency of use, or reflects their historically observed abilities to detect faults.

*Institute for Applied System Analysis, National Technical University of Ukraine "KPI", Kiev, Ukraine, malishal@cse.unl.edu
†Department of Computer Science and Engineering, University of Nebraska–Lincoln, Lincoln, Nebraska, U.S.A., {ruthruff, grother, elbaum}@cse.unl.edu

Greedy Algorithms

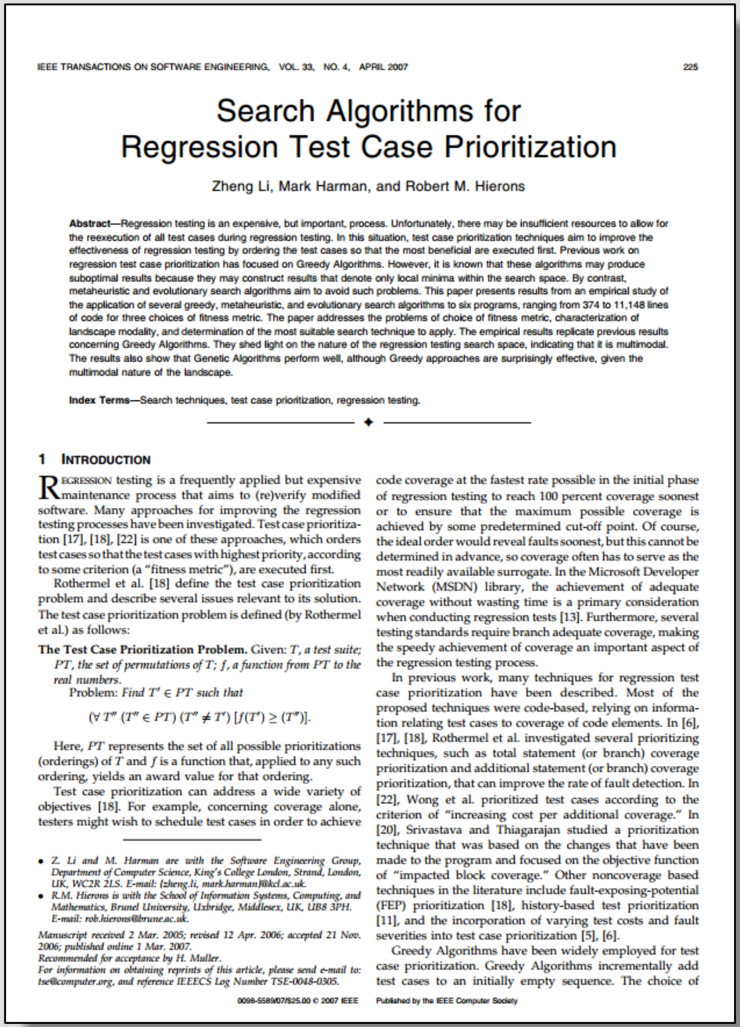


Greedy Algorithms



Any better solution?

Single Objective Metaheuristics



IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 33, NO. 4, APRIL 2007

Search Algorithms for Regression Test Case Prioritization

Zheng Li, Mark Harman, and Robert M. Hierons

Abstract—Regression testing is an expensive, but important, process. Unfortunately, there may be insufficient resources to allow for the reexecution of all test cases during regression testing. In this situation, test case prioritization techniques aim to improve the effectiveness of regression testing by ordering the test cases so that the most beneficial are executed first. Previous work on regression test case prioritization has focused on Greedy Algorithms. However, it is known that these algorithms may produce suboptimal results because they may construct results that denote only local minima within the search space. By contrast, metaheuristic and evolutionary search algorithms aim to avoid such problems. This paper presents results from an empirical study of the application of several greedy, metaheuristic, and evolutionary search algorithms to six programs, ranging from 374 to 11,148 lines of code for three choices of fitness metric. The paper addresses the problems of choice of fitness metric, characterization of landscape modality, and determination of the most suitable search technique to apply. The empirical results replicate previous results concerning Greedy Algorithms. They shed light on the nature of the regression testing search space, indicating that it is multimodal. The results also show that Genetic Algorithms perform well, although Greedy approaches are surprisingly effective, given the multimodal nature of the landscape.

Index Terms—Search techniques, test case prioritization, regression testing.

1 Introduction

Regression testing is a frequently applied but expensive maintenance process that aims to (re)verify modified software. Many approaches for improving the regression testing processes have been investigated. Test case prioritization [17], [18], [22] is one of these approaches, which orders test cases so that the test cases with highest priority, according to some criterion (a "fitness metric"), are executed first.

Rothermel et al. [18] define the test case prioritization problem and describe several issues relevant to its solution. The test case prioritization problem is defined (by Rothermel et al.) as follows:

The Test Case Prioritization Problem. Given: T, *a test suite;* PT, *the set of permutations of* T*;* f, *a function from* PT *to the real numbers.*

Problem: *Find* $T' \in PT$ *such that*

$$(\forall T'' \; (T'' \in PT) \; (T'' \neq T') \; [f(T') \geq (T'')].$$

Here, PT represents the set of all possible prioritizations (orderings) of T and f is a function that, applied to any such ordering, yields an award value for that ordering.

Test case prioritization can address a wide variety of objectives [18]. For example, concerning coverage alone, testers might wish to schedule test cases in order to achieve code coverage at the fastest rate possible in the initial phase of regression testing to reach 100 percent coverage soonest or to ensure that the maximum possible coverage is achieved by some predetermined cut-off point. Of course, the ideal order would reveal faults soonest, but this cannot be determined in advance, so coverage often has to serve as the most readily available surrogate. In the Microsoft Developer Network (MSDN) library, the achievement of adequate coverage without wasting time is a primary consideration when conducting regression tests [13]. Furthermore, several testing standards require branch adequate coverage, making the speedy achievement of coverage an important aspect of the regression testing process.

In previous work, many techniques for regression test case prioritization have been described. Most of the proposed techniques were code-based, relying on information relating test cases to coverage of code elements. In [6], [17], [18], Rothermel et al. investigated several prioritizing techniques, such as total statement (or branch) coverage prioritization and additional statement (or branch) coverage prioritization, that can improve the rate of fault detection. In [22], Wong et al. prioritized test cases according to the criterion of "increasing cost per additional coverage." In [20], Srivastava and Thiagarajan studied a prioritization technique that was based on the changes that have been made to the program and focused on the objective function of "impacted block coverage." Other noncoverage based techniques in the literature include fault-exposing-potential (FEP) prioritization [18], history-based test prioritization [11], and the incorporation of varying test costs and fault severities into test case prioritization [5], [6].

Greedy Algorithms have been widely employed for test case prioritization. Greedy Algorithms incrementally add test cases to an initially empty sequence. The choice of

- *Z. Li and M. Harman are with the Software Engineering Group, Department of Computer Science, King's College London, Strand, London, UK, WC2R 2LS. E-mail: {zheng.li, mark.harman}@kcl.ac.uk.*
- *R.M. Hierons is with the School of Information Systems, Computing, and Mathematics, Brunel University, Uxbridge, Middlesex, UK, UB8 3PH. E-mail: rob.hierons@brune.ac.uk.*

Manuscript received 2 Mar. 2005; revised 12 Apr. 2006; accepted 21 Nov. 2006; published online 1 Mar. 2007.
Recommended for acceptance by H. Muller.
For information on obtaining reprints of this article, please send e-mail to: tse@computer.org, and reference IEEECS Log Number TSE-0048-0305.

0098-5589/07/$25.00 © 2007 IEEE Published by the IEEE Computer Society

Single Objective Metaheuristics

Search Algorithms for Regression Test Case Prioritization

Zheng Li, Mark Harman, and Robert M. Hierons

Abstract—Regression testing is an expensive, but important, process. Unfortunately, there may be insufficient resources to allow for the reexecution of all test cases during regression testing. In this situation, test case prioritization techniques aim to improve the effectiveness of regression testing by ordering the test cases so that the most beneficial are executed first. Previous work on regression test case prioritization has focused on Greedy Algorithms. However, it is known that these algorithms may produce suboptimal results because they may construct results that denote only local minima within the search space. By contrast, metaheuristic and evolutionary search algorithms aim to avoid such problems. This paper presents results from an empirical study of the application of several greedy, metaheuristic, and evolutionary search algorithms to six programs, ranging from 374 to 11,148 lines of code for three choices of fitness metric. The paper addresses the problems of choice of fitness metric, characterization of landscape modality, and determination of the most suitable search technique to apply. The empirical results replicate previous results concerning Greedy Algorithms. They shed light on the nature of the regression testing search space, indicating that it is multimodal. The results also show that Genetic Algorithms perform well, although Greedy approaches are surprisingly effective, given the multimodal nature of the landscape.

Index Terms—Search techniques, test case prioritization, regression testing.

1 Introduction

Regression testing is a frequently applied but expensive maintenance process that aims to (re)verify modified software. Many approaches for improving the regression testing processes have been investigated. Test case prioritization [17], [18], [22] is one of these approaches, which orders test cases so that the test cases with highest priority, according to some criterion (a "fitness metric"), are executed first.

Rothermel et al. [18] define the test case prioritization problem and describe several issues relevant to its solution. The test case prioritization problem is defined (by Rothermel et al.) as follows:

The Test Case Prioritization Problem. Given: *T, a test suite; PT, the set of permutations of T; f, a function from PT to the real numbers.*

Problem: *Find $T' \in PT$ such that*

$$(\forall T'' \ (T'' \in PT) \ (T'' \neq T') \ [f(T') \geq (T'')].$$

Here, PT represents the set of all possible prioritizations (orderings) of T and f is a function that, applied to any such ordering, yields an award value for that ordering.

Test case prioritization can address a wide variety of objectives [18]. For example, concerning coverage alone, testers might wish to schedule test cases in order to achieve code coverage at the fastest rate possible in the initial phase of regression testing to reach 100 percent coverage soonest or to ensure that the maximum possible coverage is achieved by some predetermined cut-off point. Of course, the ideal order would reveal faults soonest, but this cannot be determined in advance, so coverage often has to serve as the most readily available surrogate. In the Microsoft Developer Network (MSDN) library, the achievement of adequate coverage without wasting time is a primary consideration when conducting regression tests [13]. Furthermore, several testing standards require branch adequate coverage, making the speedy achievement of coverage an important aspect of the regression testing process.

In previous work, many techniques for regression test case prioritization have been described. Most of the proposed techniques were code-based, relying on information relating test cases to coverage of code elements. In [6], [17], [18], Rothermel et al. investigated several prioritizing techniques, such as total statement (or branch) coverage prioritization and additional statement (or branch) coverage prioritization, that can improve the rate of fault detection. In [22], Wong et al. prioritized test cases according to the criterion of "increasing cost per additional coverage." In [20], Srivastava and Thiagarajan studied a prioritization technique that was based on the changes that have been made to the program and focused on the objective function of "impacted block coverage." Other noncoverage based techniques in the literature include fault-exposing-potential (FEP) prioritization [18], history-based test prioritization [11], and the incorporation of varying test costs and fault severities into test case prioritization [5], [6].

Greedy Algorithms have been widely employed for test case prioritization. Greedy Algorithms incrementally add test cases to an initially empty sequence. The choice of

- *Z. Li and M. Harman are with the Software Engineering Group, Department of Computer Science, King's College London, Strand, London, UK, WC2R 2LS. E-mail: {zheng.li, mark.harman}@kcl.ac.uk.*
- *R.M. Hierons is with the School of Information Systems, Computing, and Mathematics, Brunel University, Uxbridge, Middlesex, UK, UB8 3PH. E-mail: rob.hierons@brune.ac.uk.*

Manuscript received 2 Mar. 2005; revised 12 Apr. 2006; accepted 21 Nov. 2006; published online 1 Mar. 2007.
Recommended for acceptance by H. Muller.
For information on obtaining reprints of this article, please send e-mail to: tse@computer.org, and reference IEEECS Log Number TSE-0048-0305.

0098-5589/07/$25.00 © 2007 IEEE Published by the IEEE Computer Society

Integer permutation for representing a test suite

3	5	1	4	7	2	8	6	9
8	5	1	9	6	2	3	7	4

Single Objective Metaheuristics

Search Algorithms for Regression Test Case Prioritization

Zheng Li, Mark Harman, and Robert M. Hierons

Abstract—Regression testing is an expensive, but important, process. Unfortunately, there may be insufficient resources to allow for the reexecution of all test cases during regression testing. In this situation, test case prioritization techniques aim to improve the effectiveness of regression testing by ordering the test cases so that the most beneficial are executed first. Previous work on regression test case prioritization has focused on Greedy Algorithms. However, it is known that these algorithms may produce suboptimal results because they may construct results that denote only local minima within the search space. By contrast, metaheuristic and evolutionary search algorithms aim to avoid such problems. This paper presents results from an empirical study of the application of several greedy, metaheuristic, and evolutionary search algorithms to six programs, ranging from 374 to 11,148 lines of code for three choices of fitness metric. The paper addresses the problems of choice of fitness metric, characterization of landscape modality, and determination of the most suitable search technique to apply. The empirical results replicate previous results concerning Greedy Algorithms. They shed light on the nature of the regression testing search space, indicating that it is multimodal. The results also show that Genetic Algorithms perform well, although Greedy approaches are surprisingly effective, given the multimodal nature of the landscape.

Index Terms—Search techniques, test case prioritization, regression testing.

1 Introduction

Regression testing is a frequently applied but expensive maintenance process that aims to (re)verify modified software. Many approaches for improving the regression testing processes have been investigated. Test case prioritization [17], [18], [22] is one of these approaches, which orders test cases so that the test cases with highest priority, according to some criterion (a "fitness metric"), are executed first.

Rothermel et al. [18] define the test case prioritization problem and describe several issues relevant to its solution. The test case prioritization problem is defined (by Rothermel et al.) as follows:

The Test Case Prioritization Problem. Given: *T, a test suite; PT, the set of permutations of T; f, a function from PT to the real numbers.*

Problem: *Find $T' \in PT$ such that*

$$(\forall T'' \, (T'' \in PT) \, (T'' \neq T') \, [f(T') \geq (T'')].$$

Here, PT represents the set of all possible prioritizations (orderings) of T and f is a function that, applied to any such ordering, yields an award value for that ordering.

Test case prioritization can address a wide variety of objectives [18]. For example, concerning coverage alone, testers might wish to schedule test cases in order to achieve code coverage at the fastest rate possible in the initial phase of regression testing to reach 100 percent coverage soonest or to ensure that the maximum possible coverage is achieved by some predetermined cut-off point. Of course, the ideal order would reveal faults soonest, but this cannot be determined in advance, so coverage often has to serve as the most readily available surrogate. In the Microsoft Developer Network (MSDN) library, the achievement of adequate coverage without wasting time is a primary consideration when conducting regression tests [13]. Furthermore, several testing standards require branch adequate coverage, making the speedy achievement of coverage an important aspect of the regression testing process.

In previous work, many techniques for regression test case prioritization have been described. Most of the proposed techniques were code-based, relying on information relating test cases to coverage of code elements. In [6], [17], [18], Rothermel et al. investigated several prioritizing techniques, such as total statement (or branch) coverage prioritization and additional statement (or branch) coverage prioritization, that can improve the rate of fault detection. In [22], Wong et al. prioritized test cases according to the criterion of "increasing cost per additional coverage." In [20], Srivastava and Thiagarajan studied a prioritization technique that was based on the changes that have been made to the program and focused on the objective function of "impacted block coverage." Other noncoverage based techniques in the literature include fault-exposing-potential (FEP) prioritization [18], history-based test prioritization [11], and the incorporation of varying test costs and fault severities into test case prioritization [5], [6].

Greedy Algorithms have been widely employed for test case prioritization. Greedy Algorithms incrementally add test cases to an initially empty sequence. The choice of

- *Z. Li and M. Harman are with the Software Engineering Group, Department of Computer Science, King's College London, Strand, London, UK, WC2R 2LS. E-mail: {zheng.li, mark.harman}@kcl.ac.uk.*
- *R.M. Hierons is with the School of Information Systems, Computing, and Mathematics, Brunel University, Uxbridge, Middlesex, UK, UB8 3PH. E-mail: rob.hierons@brune.ac.uk.*

Manuscript received 2 Mar. 2005; revised 12 Apr. 2006; accepted 21 Nov. 2006; published online 1 Mar. 2007.
Recommended for acceptance by H. Muller.
For information on obtaining reprints of this article, please send e-mail to: tse@computer.org, and reference IEEECS Log Number TSE-0048-0305.

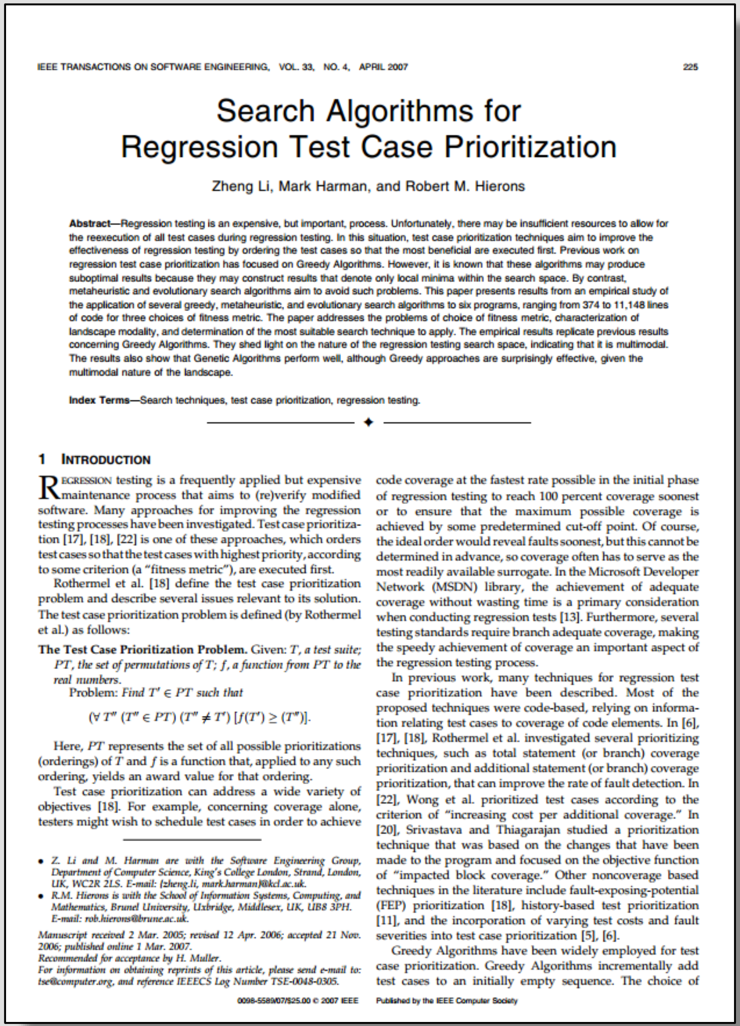
 Published by the IEEE Computer Society

Integer permutation for representing a test suite

3	5	1	4	7	2	8	6	9
8	5	1	9	6	2	3	7	4

Maximize AUC
APBC – APDC - APSC

Single Objective Metaheuristics



Search Algorithms for Regression Test Case Prioritization

Zheng Li, Mark Harman, and Robert M. Hierons

Abstract—Regression testing is an expensive, but important, process. Unfortunately, there may be insufficient resources to allow for the reexecution of all test cases during regression testing. In this situation, test case prioritization techniques aim to improve the effectiveness of regression testing by ordering the test cases so that the most beneficial are executed first. Previous work on regression test case prioritization has focused on Greedy Algorithms. However, it is known that these algorithms may produce suboptimal results because they may construct results that denote only local minima within the search space. By contrast, metaheuristic and evolutionary search algorithms aim to avoid such problems. This paper presents results from an empirical study of the application of several greedy, metaheuristic, and evolutionary search algorithms to six programs, ranging from 374 to 11,148 lines of code for three choices of fitness metric. The paper addresses the problems of choice of fitness metric, characterization of landscape modality, and determination of the most suitable search technique to apply. The empirical results replicate previous results concerning Greedy Algorithms. They shed light on the nature of the regression testing search space, indicating that it is multimodal. The results also show that Genetic Algorithms perform well, although Greedy approaches are surprisingly effective, given the multimodal nature of the landscape.

Index Terms—Search techniques, test case prioritization, regression testing.

1 Introduction

Regression testing is a frequently applied but expensive maintenance process that aims to (re)verify modified software. Many approaches for improving the regression testing processes have been investigated. Test case prioritization [17], [18], [22] is one of these approaches, which orders test cases so that the test cases with highest priority, according to some criterion (a "fitness metric"), are executed first.

Rothermel et al. [18] define the test case prioritization problem and describe several issues relevant to its solution. The test case prioritization problem is defined (by Rothermel et al.) as follows:

The Test Case Prioritization Problem. Given: *T, a test suite; PT, the set of permutations of T; f, a function from PT to the real numbers.*

Problem: *Find $T' \in PT$ such that*

$$(\forall T'' \; (T'' \in PT) \; (T'' \neq T') \; [f(T') \geq (T'')].$$

Here, PT represents the set of all possible prioritizations (orderings) of T and f is a function that, applied to any such ordering, yields an award value for that ordering.

Test case prioritization can address a wide variety of objectives [18]. For example, concerning coverage alone, testers might wish to schedule test cases in order to achieve code coverage at the fastest rate possible in the initial phase of regression testing to reach 100 percent coverage soonest or to ensure that the maximum possible coverage is achieved by some predetermined cut-off point. Of course, the ideal order would reveal faults soonest, but this cannot be determined in advance, so coverage often has to serve as the most readily available surrogate. In the Microsoft Developer Network (MSDN) library, the achievement of adequate coverage without wasting time is a primary consideration when conducting regression tests [13]. Furthermore, several testing standards require branch adequate coverage, making the speedy achievement of coverage an important aspect of the regression testing process.

In previous work, many techniques for regression test case prioritization have been described. Most of the proposed techniques were code-based, relying on information relating test cases to coverage of code elements. In [6], [17], [18], Rothermel et al. investigated several prioritizing techniques, such as total statement (or branch) coverage prioritization and additional statement (or branch) coverage prioritization, that can improve the rate of fault detection. In [22], Wong et al. prioritized test cases according to the criterion of "increasing cost per additional coverage." In [20], Srivastava and Thiagarajan studied a prioritization technique that was based on the changes that have been made to the program and focused on the objective function of "impacted block coverage." Other noncoverage based techniques in the literature include fault-exposing-potential (FEP) prioritization [18], history-based test prioritization [11], and the incorporation of varying test costs and fault severities into test case prioritization [5], [6].

Greedy Algorithms have been widely employed for test case prioritization. Greedy Algorithms incrementally add test cases to an initially empty sequence. The choice of

• *Z. Li and M. Harman are with the Software Engineering Group, Department of Computer Science, King's College London, Strand, London, UK, WC2R 2LS. E-mail: {zheng.li, mark.harman}@kcl.ac.uk.*
• *R.M. Hierons is with the School of Information Systems, Computing, and Mathematics, Brunel University, Uxbridge, Middlesex, UK, UB8 3PH. E-mail: rob.hierons@brune.ac.uk.*

Manuscript received 2 Mar. 2005; revised 12 Apr. 2006; accepted 21 Nov. 2006; published online 1 Mar. 2007.
Recommended for acceptance by H. Muller.
For information on obtaining reprints of this article, please send e-mail to: tse@computer.org, and reference IEEECS Log Number TSE-0048-0305.

Integer permutation for representing a test suite

3	5	1	4	7	2	8	6	9
8	5	1	9	6	2	3	7	4

Maximize AUC
APBC – APDC - APSC

TCP problem has
a multi-objective nature

Multi Objective Metaheuristics

2012 16th European Conference on Software Maintenance and Reengineering

A Multi-Objective Technique to Prioritize Test Cases based on Latent Semantic Indexing

Md. Mahfuzul Islam, Alessandro Marchetto, Angelo Susi
Fondazione Bruno Kessler
Trento, Italy
{mahfuzul,marchetto,susi}@fbk.eu

Giuseppe Scanniello
Università della Basilicata
Potenza, Italy
giuseppe.scanniello@unibas.it

***Abstract*—To early discover faults in source code, test case ordering has to be properly chosen. To this aim test prioritization techniques can be used. Several of these techniques leave out the execution cost of test cases and exploit a single objective function (e.g., code or requirements coverage).**

In this paper, we present a multi-objective test prioritization technique that determines sequences of test cases that maximize the number of discovered faults that are both technical and business critical. The technique uses the information related to the code and requirements coverage, as well as the execution cost of each test case. The approach also uses recovered traceability links among source code and system requirements via the Latent Semantic Indexing technique. We evaluated our proposal against both a random prioritization technique and two single-objective prioritization techniques on two Java applications. The results indicate that our proposal outperforms the baseline techniques and that additional improvements are still possible.

***Keywords*-Regression Testing; Requirements; Testing; Test Case Prioritization; Traceability.**

I. Introduction

Regression testing aims at guaranteeing that the integration of software components and modifications to the source code does not compromise the expected behavior of the software application. Relevant activities often conducted during regression testing are [31]: *(i)* test selection; *(ii)* test minimization; and *(iii)* test prioritization. Test selection chooses the test cases that are relevant for a specific part of the application or for the performed maintenance operation. Test minimization reduces the number of test cases to be executed by removing redundant test cases, thus preserving the capability of the suite in discovering faults. Test prioritization determines the execution order of test cases that maximizes the probability of early discovering faults. The objective of this activity is to identify test case orderings that are effective (in terms of capability of early revealing faults) and efficient (in terms of test cases execution cost). These factors are relevant because they represent technical and business criteria for the success of a software project [11].

In this paper, we propose a novel prioritization technique. It is multi-objective and determines test case orderings to maximize the number of faults to be discovered. The technique is based on a three-dimension analysis of test cases. The *structural* dimension concerns information regarding test cases under analysis (i.e., how they exercise the application under test), while *functional* dimension regards the coverage of users' and system requirements. The latter dimension is *cost* and concerns the time to execute test cases.

A test case ordering is achieved as a multi-objective optimization problem to balance the considered dimensions with respect to the relationships (also named traceability links in the following) among software artifacts. Since very often traceability links are not available in the project documentation, we use an Information Retrieval (IR) approach [16], namely Latent Semantic Indexing (LSI). Our approach exploits this technique to recover relationships among software artifacts (i.e., application code, test cases, and requirements specifications) and to measure their strength.

Test prioritization techniques usually consider several algorithms to prioritize test cases and are mostly based on a single dimension (e.g., code or requirements coverage). These techniques also assume that faults have the same relevance. Conversely, we identify test case orderings that early reveal both technical (e.g., coding faults) and business critical faults (e.g., due to the misunderstanding of requirements) by explicitly considering structural and functional information and the time to execute test cases.

We have evaluated the proposed technique against more traditional techniques on two applications implemented in Java. The results indicate that the new technique on average outperforms the baseline techniques in revealing both technical and business critical faults, and also show that there is room for further improvement.

The contributions of this paper are: (1) a test prioritization technique using a multi-objective optimization problem to consider three dimensions based on high and low -level information; (2) the definition of an IR-based approach to recover traceability links among: requirements specifications, source code, and test cases. The proposed multi-objective technique is built on that approach; (3) a preliminary case study on two small Java applications to assess the validity of the technique.

1534-5351/12 $26.00 © 2012 IEEE
DOI 10.1109/CSMR.2012.13

IEEE computer society

Pareto-ranking

Algorithms (NSGA-II)

Multi Objective Metaheuristics


2012 16th European Conference on Software Maintenance and Reengineering

A Multi-Objective Technique to Prioritize Test Cases based on Latent Semantic Indexing

Md. Mahfuzul Islam, Alessandro Marchetto, Angelo Susi
Fondazione Bruno Kessler
Trento, Italy
{mahfuzul,marchetto,susi}@fbk.eu

Giuseppe Scanniello
Università della Basilicata
Potenza, Italy
giuseppe.scanniello@unibas.it

1534-5351/12 $26.00 © 2012 IEEE
DOI 10.1109/CSMR.2012.13

A Fine-Grained Parallel Multi-objective Test Case Prioritization on GPU

Zheng Li[1], Yi Bian[1], Ruilian Zhao[1], and Jun Cheng[2]

[1] Department of Computer Science
Beijing University of Chemical Technology
Beijing 100029, P.R. China
[2] Chongqing Institute of Green and Intelligent Technology
Chinese Academy of Sciences
Chongqing 401122, P.R. China

Abstract. Multi-Objective Evolutionary Algorithms (MOEAs) have been widely used to address regression test optimization problems, including test case selection and test suite minimization. GPU-based parallel MOEAs are proposed to increase execution efficiency to fulfill the industrial demands. When using binary representation in MOEAs, the fitness evaluation can be transformed a parallel matrix multiplication that is implemented on GPU easily and more efficiently. Such GPU-based parallel MOEAs may achieve higher level of speed-up for test case prioritization because the computation load of fitness evaluation in test case prioritization is more than that in test case selection or test suite minimization. However, the non-applicability of binary representation in the test case prioritization results in the challenge of parallel fitness evaluation on GPU. In this paper, we present a GPU-based parallel fitness evaluation and three novel parallel crossover computation schemes based on ordinal and sequential representations, which form a fine-grained parallel framework for multi-objective test case prioritization. The empirical studies based on eight benchmarks and one open source program show a maximum of 120x speed-up achieved.

Keywords: Test Case Prioritization, Mulit-Objective Optimization, NSGA-II, GPU, CUDA.

1 Introduction

Pareto-ranking

Algorithms (NSGA-II)

Multi Objective Metaheuristics

Pareto-ranking

Algorithms (NSGA-II)

2012 16th European Conference on Software Maintenance and Reengineering

A Multi-Objective Technique to Prioritize Test Cases based on Latent Semantic Indexing

Md. Mahfuzul Islam, Alessandro Marchetto, Angelo Susi
Fondazione Bruno Kessler
Trento, Italy
{mahfuzul,marchetto,susi}@fbk.eu

Giuseppe Scanniello
Università della Basilicata
Potenza, Italy
giuseppe.scanniello@unibas.it

1534-5351/12 $26.00 © 2012 IEEE
DOI 10.1109/CSMR.2012.13

A Fine-Grained Parallel Multi-objective Test Case Prioritization on GPU

Zheng Li[1], Yi Bian[1], Ruilian Zhao[1], and Jun Cheng[2]

[1] Department of Computer Science
Beijing University of Chemical Technology
Beijing 100029, P.R. China
[2] Chongqing Institute of Green and Intelligent Technology
Chinese Academy of Sciences
Chongqing 401122, P.R. China

Empirical Evaluation of Pareto Efficient Multi-objective Regression Test Case Prioritisation

Michael G. Epitropakis
Computing Science and Mathematics
University of Stirling
Stirling, UK
mge@cs.stir.ac.uk

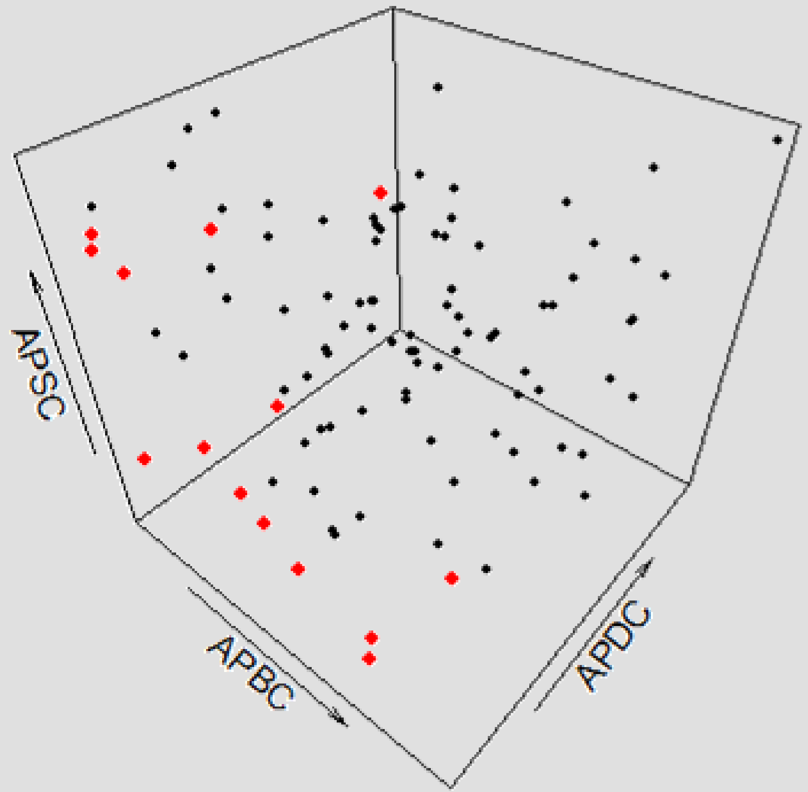
Shin Yoo
Department of Computer Science,
University College London,
London, UK
shin.yoo@ucl.ac.uk

Mark Harman
Department of Computer Science,
University College London,
London, UK
mark.harman@ucl.ac.uk

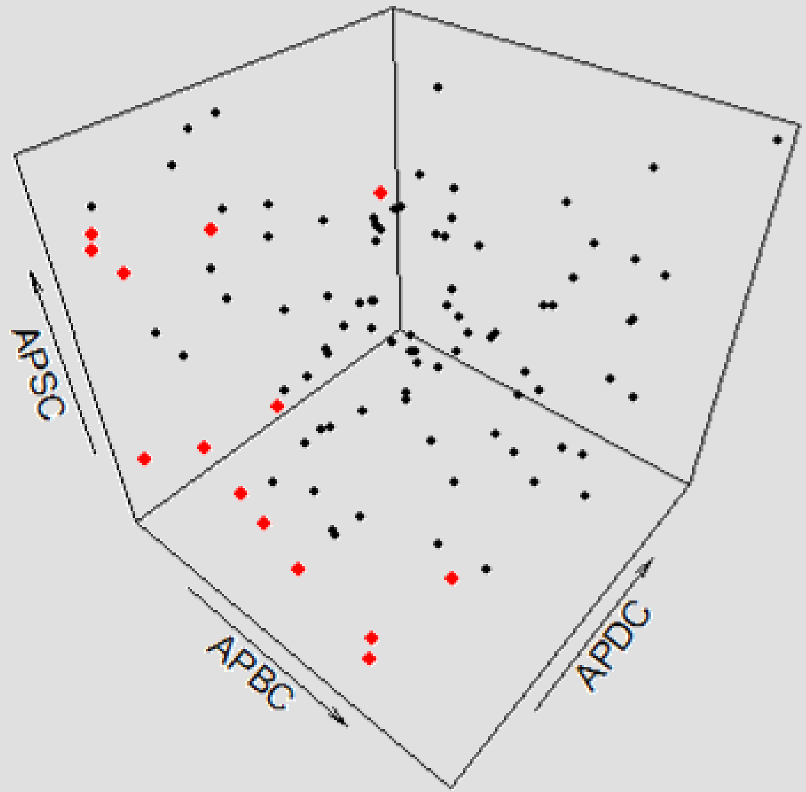
Edmund K. Burke
Computing Science and Mathematics
University of Stirling
Stirling, UK
e.k.burke@stir.ac.uk

ABSTRACT

Multi Objective Metaheuristics

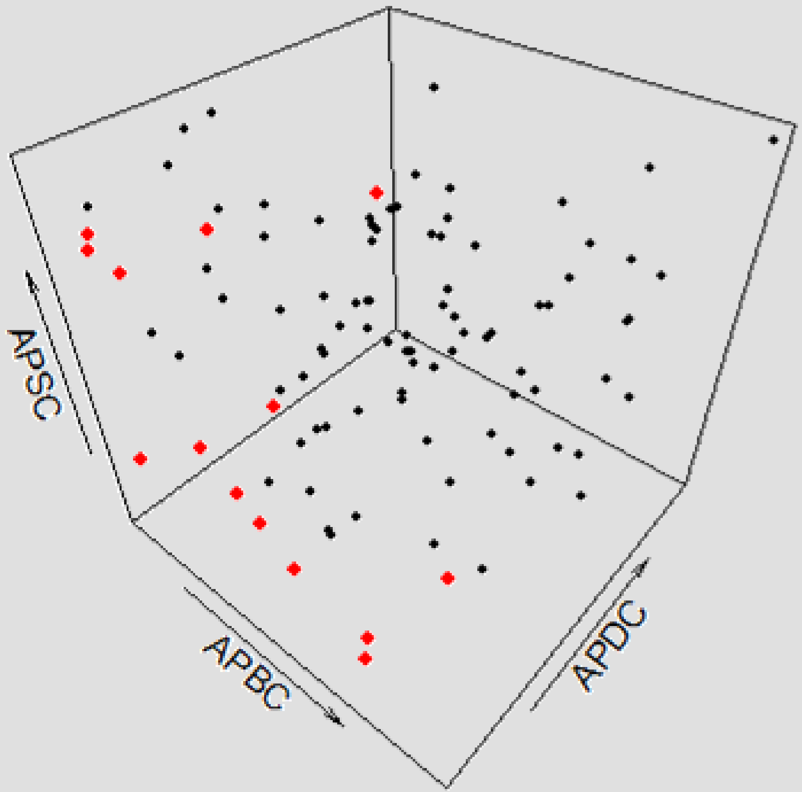


Multi Objective Metaheuristics



Bad effectiveness as the problem dimensionality increases

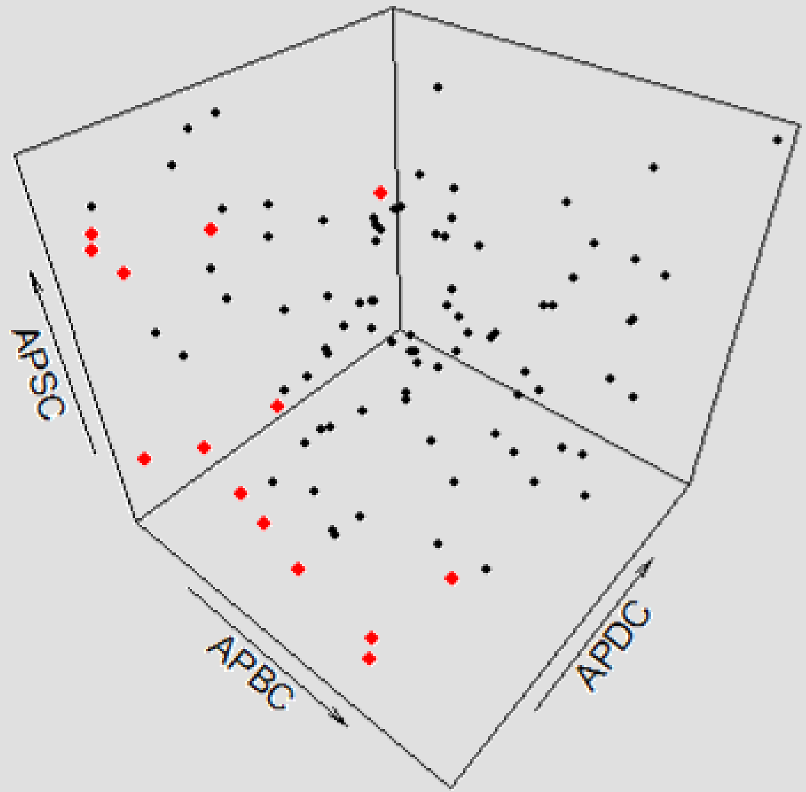
Multi Objective Metaheuristics



Bad effectiveness as the problem dimensionality increases

For more than 3-objectives, all individuals are non-dominated → Poor Selective Pressure

Multi Objective Metaheuristics



Bad effectiveness as the problem dimensionality increases

For more than 3-objectives, all individuals are non-dominated → Poor Selective Pressure

No strong empirical evidence of the cost-effectiveness with respect to simpler heuristics

Hypervolume-based **G**enetic **A**lgorithm

Hypervolume

In many-objective optimization there is a growing trend to solve many-objective problems using ***quality scalar indicators*** *to condense multiple objectives into a single objective.*

Auger et al. - Theory of the hypervolume indicator: optimal distributions and the choice of the reference point - FOGA 2009

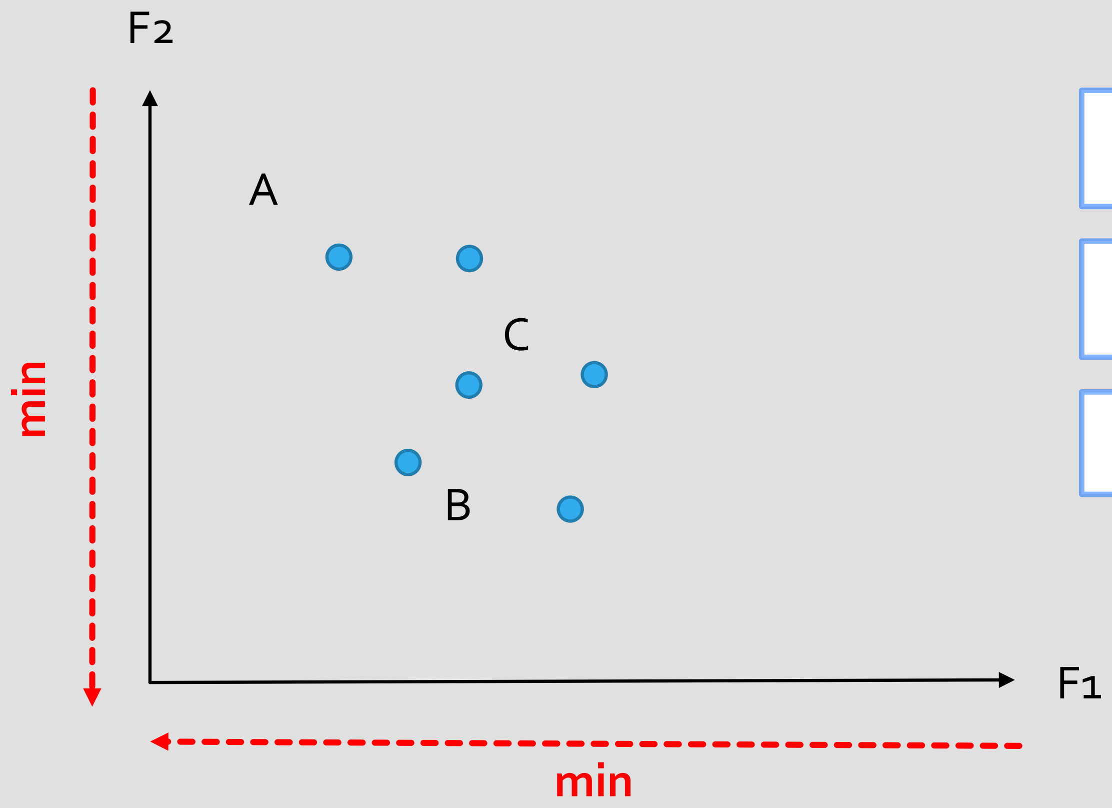
Hypervolume

In many-objective optimization there is a growing trend to solve many-objective problems using ***quality scalar indicators*** *to condense multiple objectives into a single objective.*

Auger et al. - Theory of the hypervolume indicator: optimal distributions and the choice of the reference point - FOGA 2009

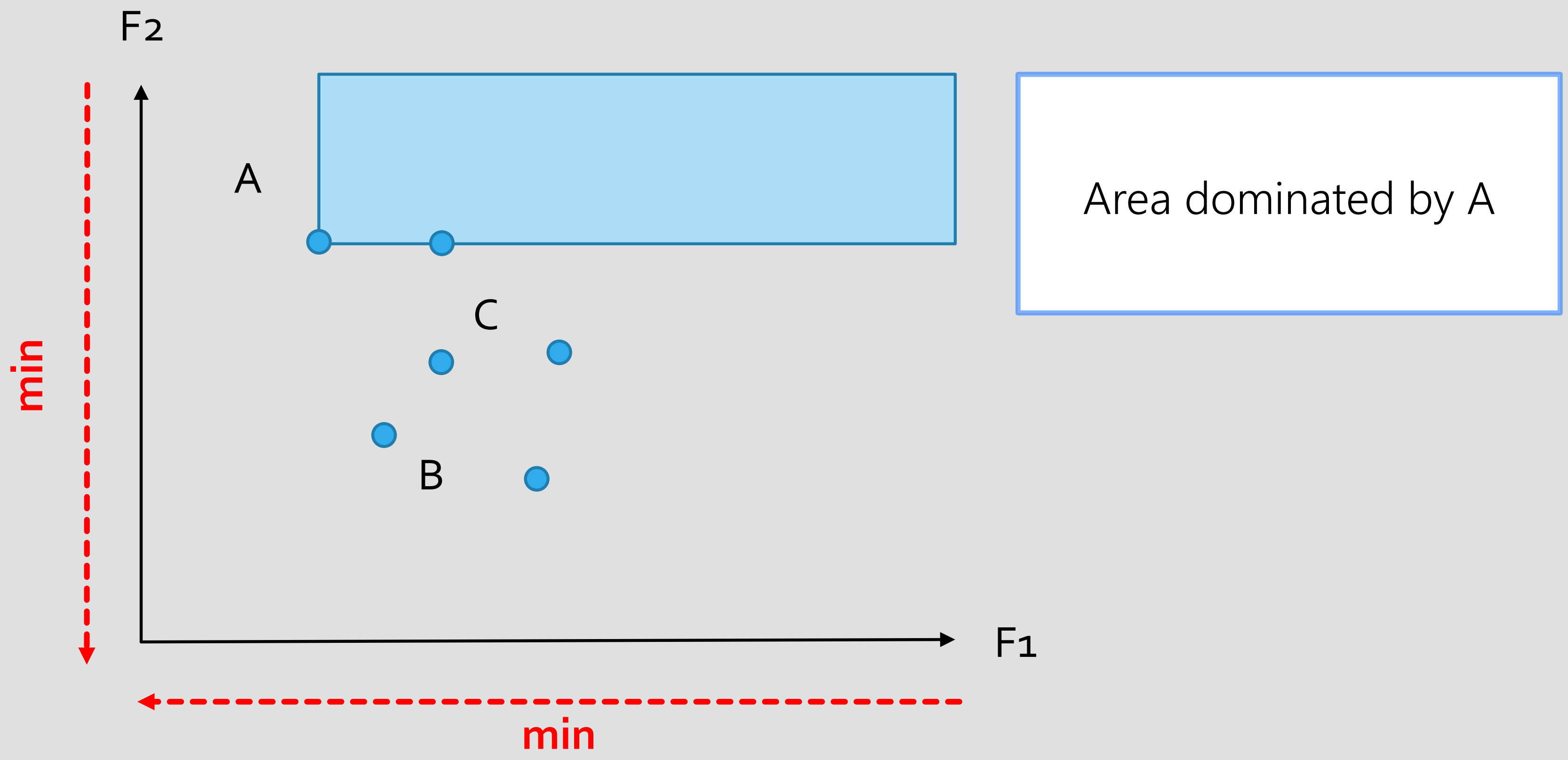
The **hypervolume** measures the quality of a set of solutions as the total size of the objective space that is dominated by one (or more) of such solutions.

Hypervolume

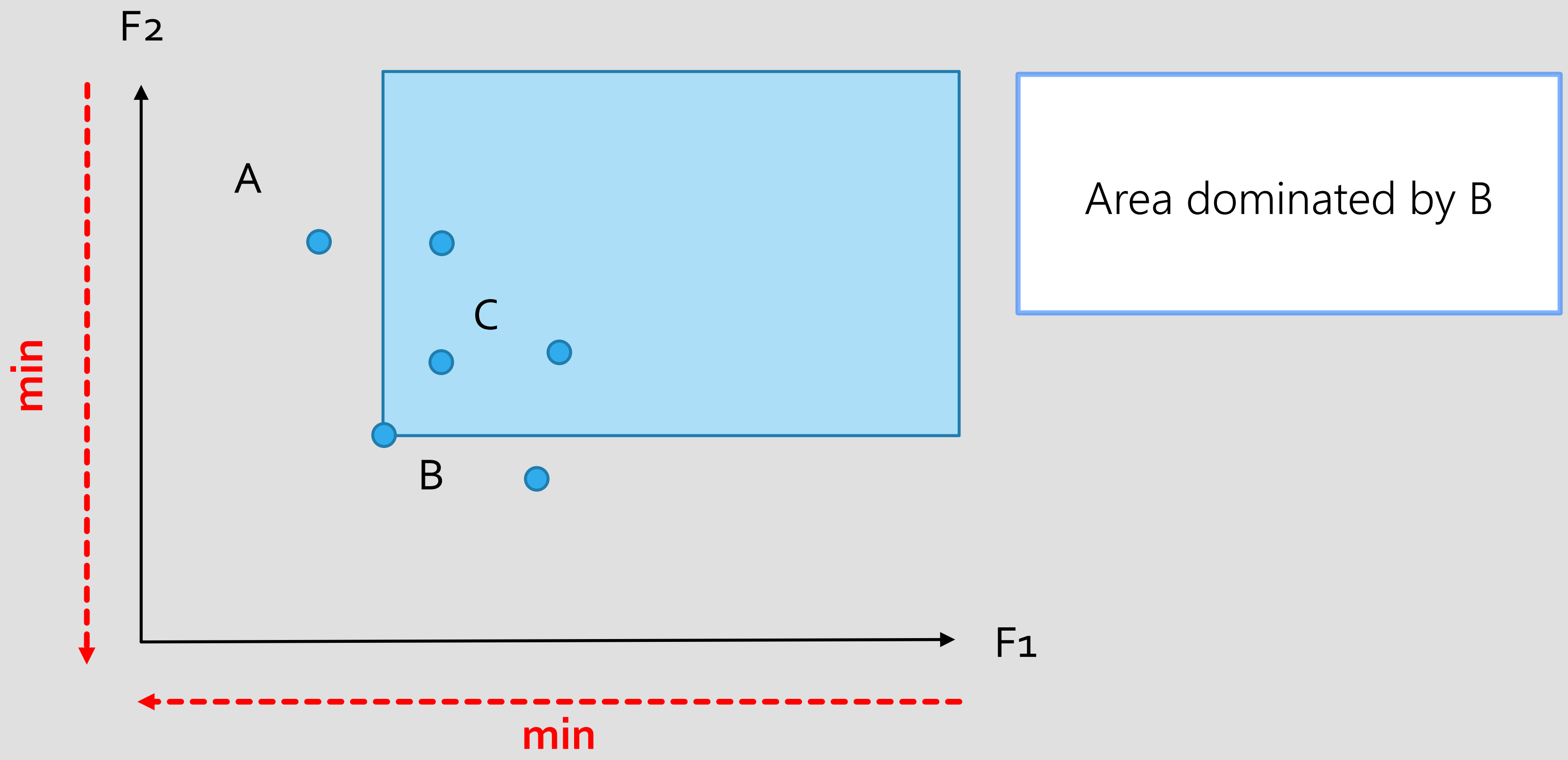


- A non-dominates B
- B non-dominates A
- B dominates C

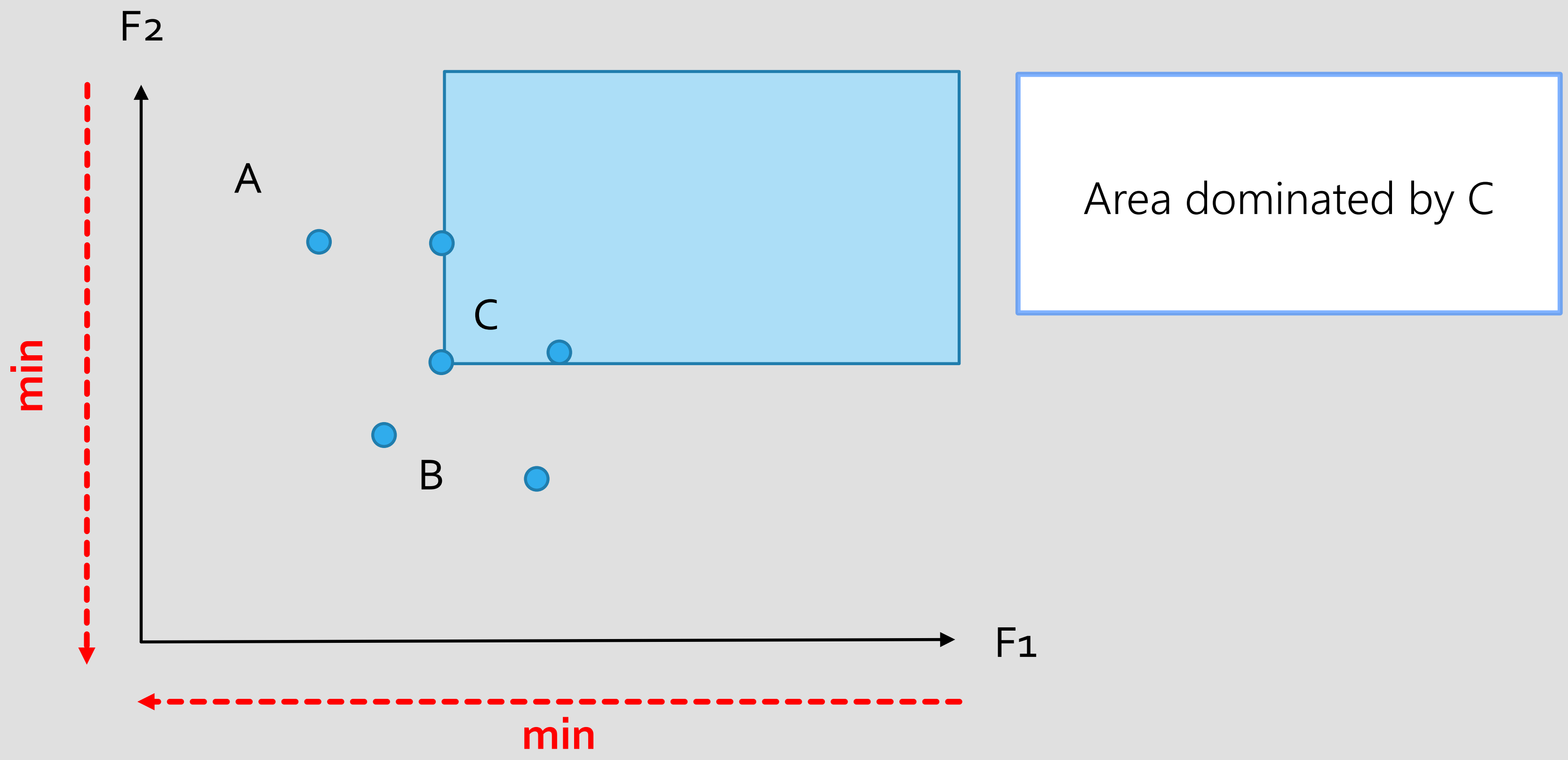
Hypervolume



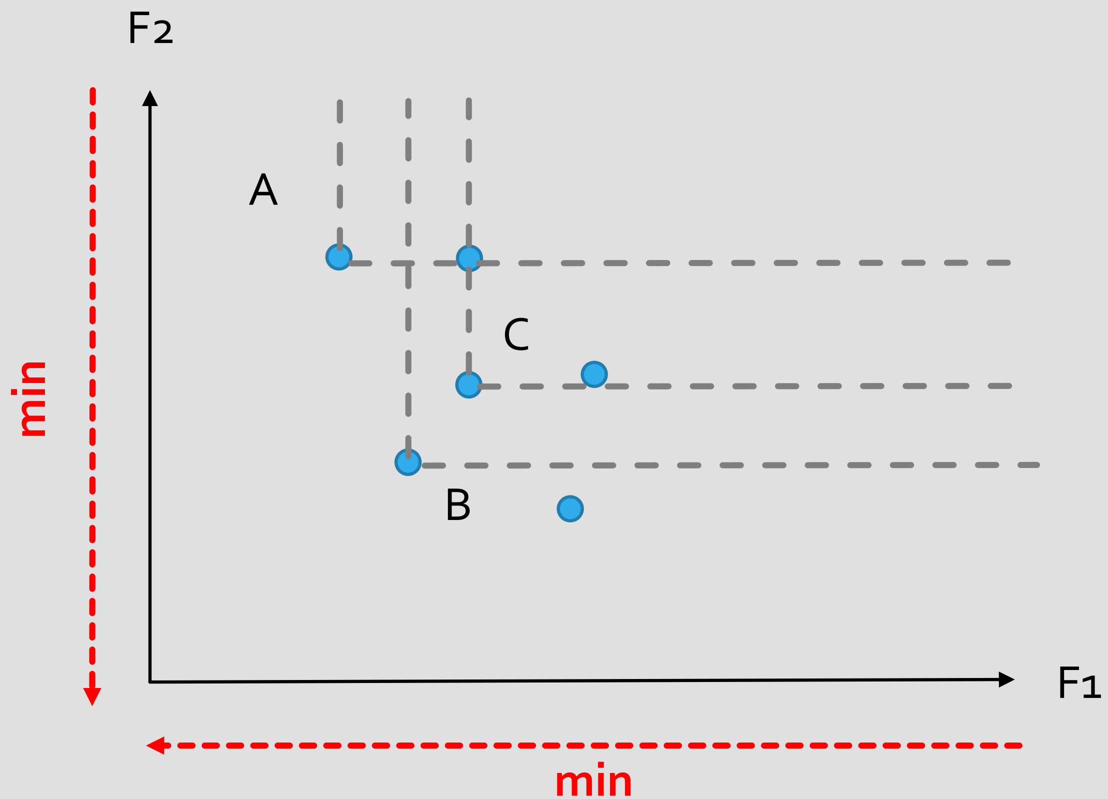
Hypervolume



Hypervolume



Hypervolume

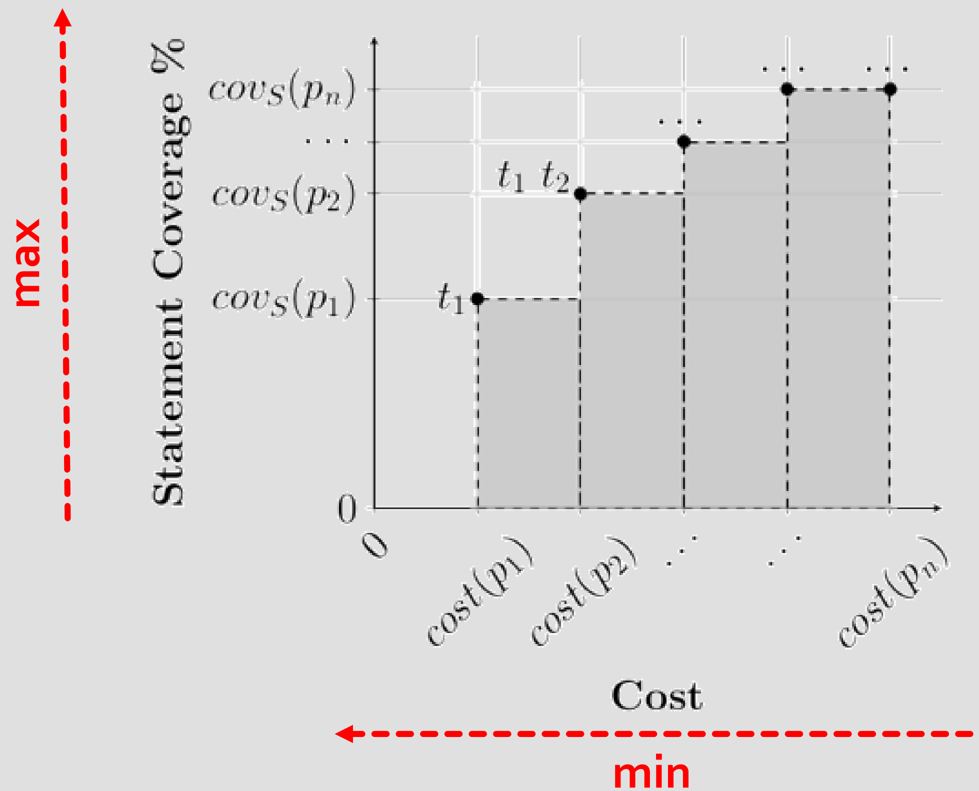


Area(B) > Area(A)

Area(B) > Area(C)

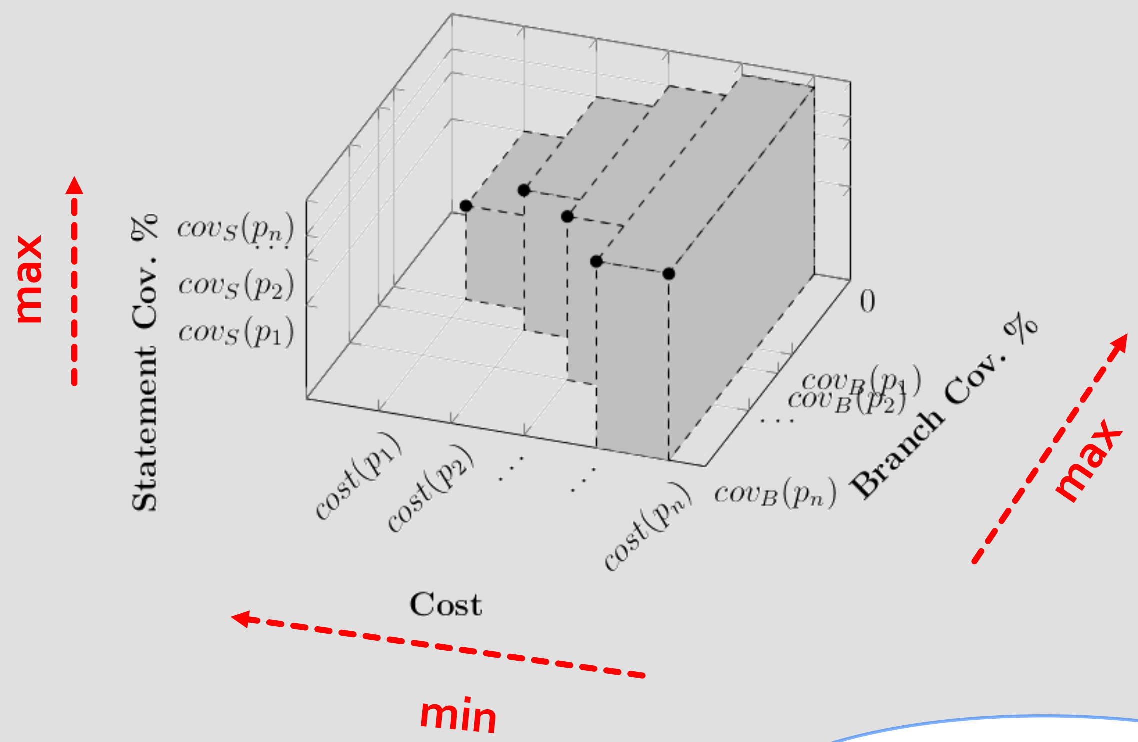
Stronger Selective Pressure

Hypervolume Indicator for TCP



2-objectives

Hypervolume Indicator for TCP



3-objectives

Hypervolume Indicator for TCP

It can be used as **fitness function** in a Single Objective Metaheuristic

Hypervolume Indicator for TCP

It can be used as **fitness function** in a Single Objective Metaheuristic

Computational Time $\boldsymbol{O(n \cdot m)}$

n test cases

m testing criteria

Empirical Evaluation

Research Questions

RQ1 : What is the **cost-effectiveness** of HGA, compared to cost-aware additional greedy algorithms?

Cost-cognizant Average Fault Detection Percentage (**$AFDP_c$**)

Research Questions

RQ1 : What is the **cost-effectiveness** of HGA, compared to cost-aware additional greedy algorithms?

Cost-cognizant Average Fault Detection Percentage (**$AFDP_c$**)

RQ2 : What is the **efficiency** of HGA, compared to

cost-aware additional greedy algorithms?

Execution time (in seconds)

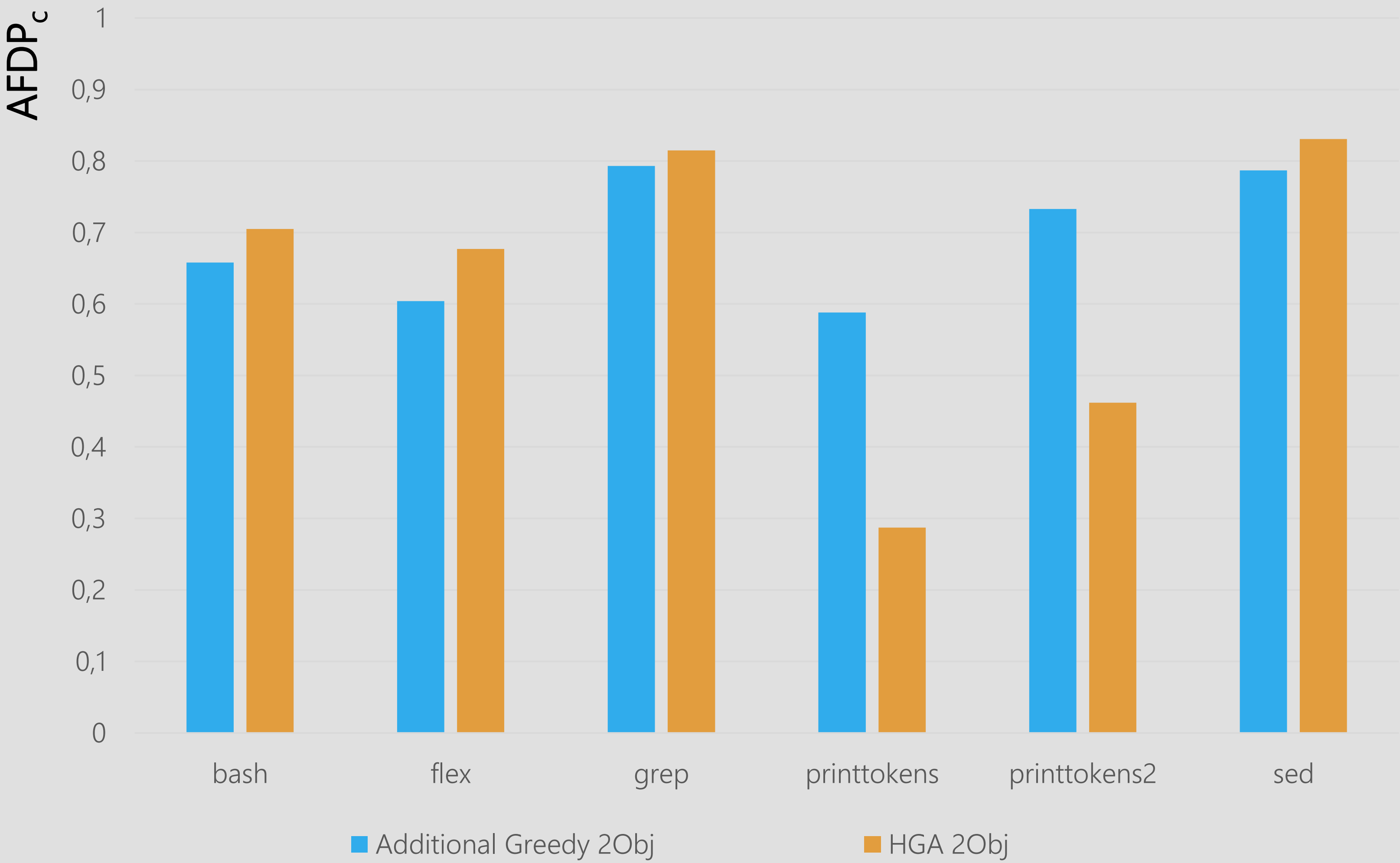
Design

Comparison with **Additional Greedy** algorithms
(2Obj and 3Obj)

6 programs from **SIR**

20 indipendent GA runs

Results RQ1



Average $AFDP_c$
2-Objective formulation

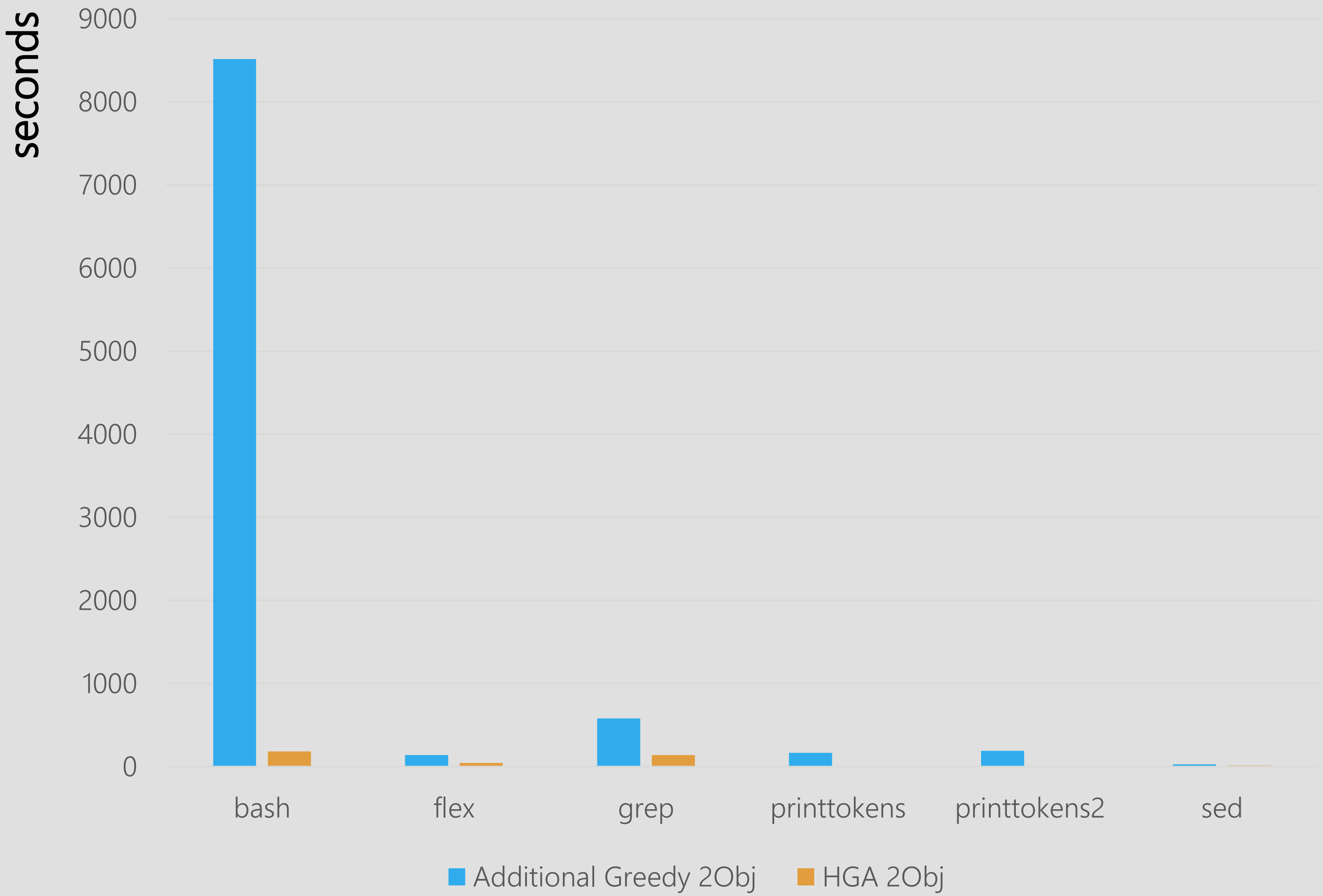
Results RQ1

Average $AFDP_c$
3-Objective formulation

Results RQ1

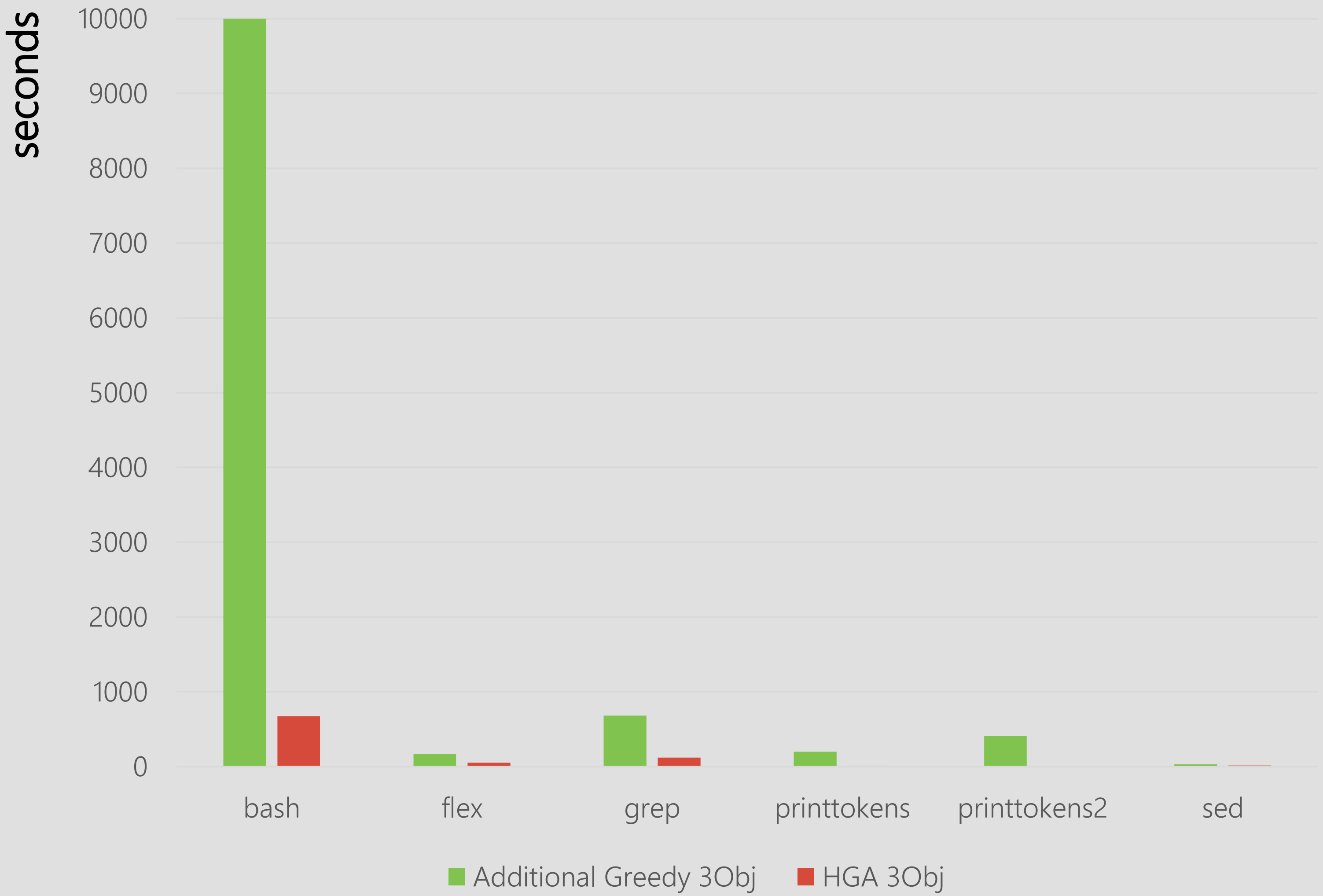
Program	2-Objective			3-Objective		
	p-value	$\hat{A}_{12}$	magnitude	p-value	$\hat{A}_{12}$	magnitude
bash	< 0.01	0.88	Large	< 0.01	0.95	Large
flex	< 0.01	0.70	Medium	< 0.01	0.75	Large
grep	< 0.01	0.85	Large	< 0.01	0.85	Large
printtokens	1	0.10	Large	1	0.10	Large
printtokens 2	1	0.30	Large	0.73	0.40	Small
sed	< 0.01	0.85	Large	0.01	0.80	Large

Results RQ2



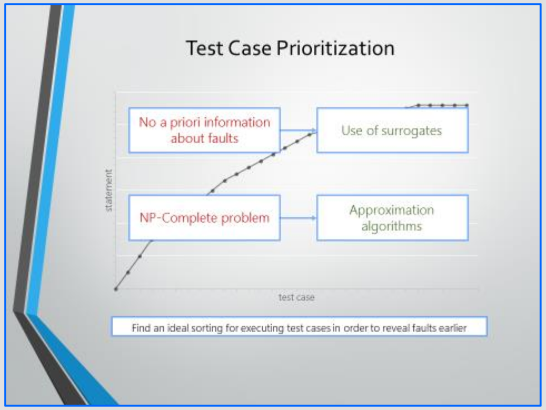
Average Execution Time
2-Objective formulation

Results RQ2

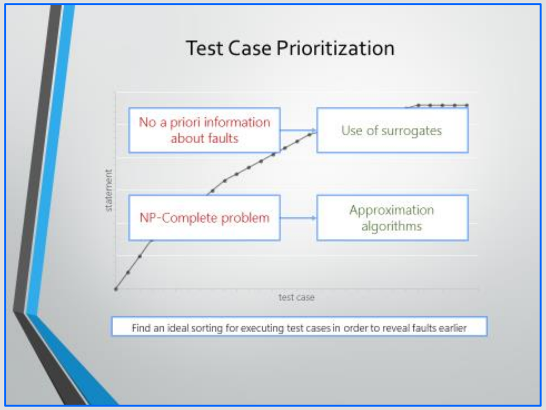


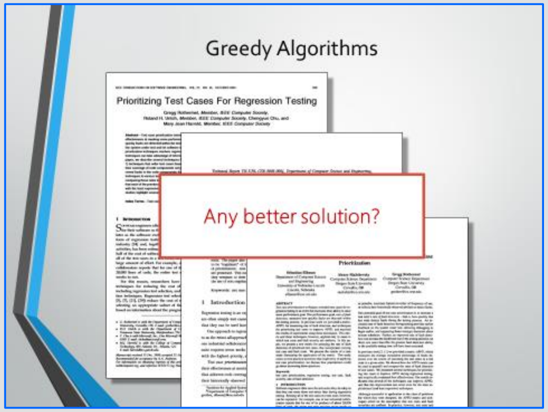
Average Execution Time
3-Objective formulation

Summary

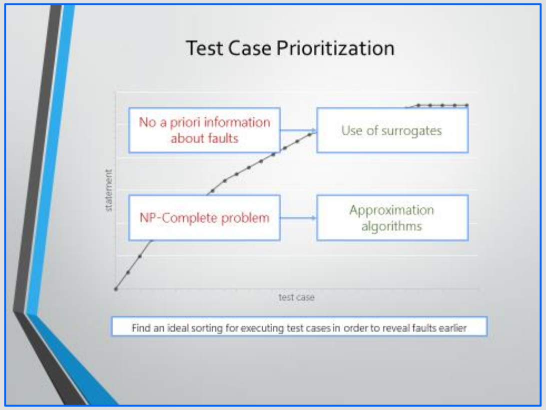

Test Case Prioritization
No a priori information about faults
Use of surrogates
NP-Complete problem
Approximation algorithms
statement
test case
Find an ideal sorting for executing test cases in order to reveal faults earlier

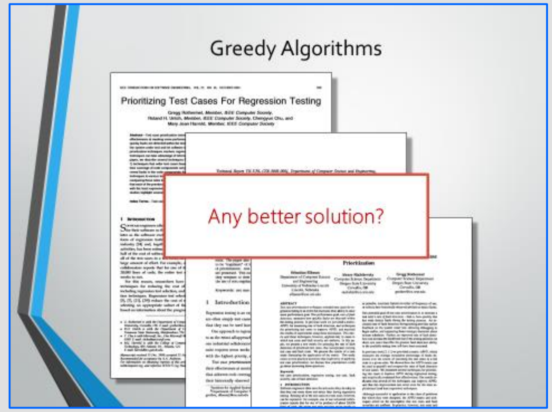
Summary

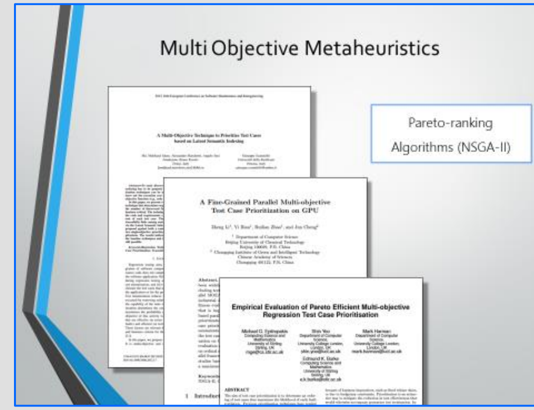

Test Case Prioritization
No a priori information about faults
Use of surrogates
NP-Complete problem
Approximation algorithms
statement
test case
Find an ideal sorting for executing test cases in order to reveal faults earlier


Greedy Algorithms
Prioritizing Test Cases For Regression Testing
Any better solution?

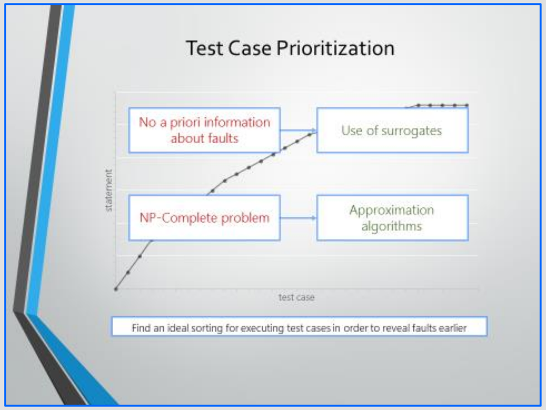
Summary

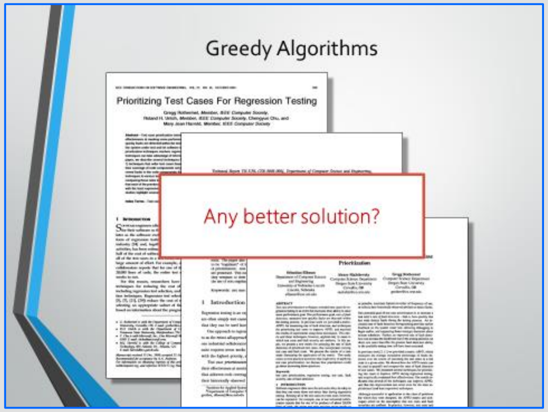

Test Case Prioritization
No a priori information about faults
Use of surrogates
NP-Complete problem
Approximation algorithms
statement
test case
Find an ideal sorting for executing test cases in order to reveal faults earlier

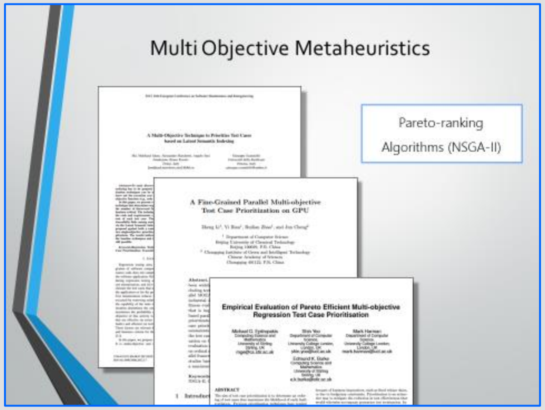

Greedy Algorithms
Prioritizing Test Cases For Regression Testing
Any better solution?

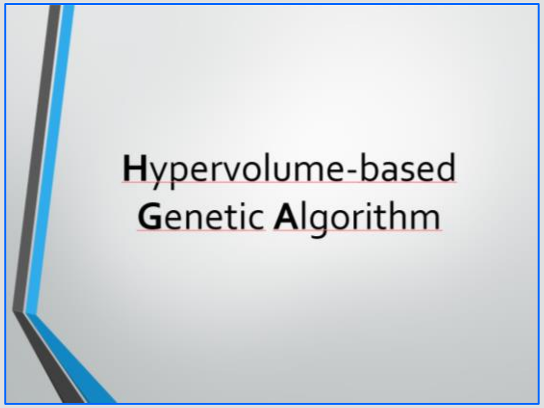

Multi Objective Metaheuristics
Pareto-ranking Algorithms (NSGA-II)
A Fine-Grained Parallel Multi-objective Test Case Prioritization on GPU
Empirical Evaluation of Pareto Efficient Multi-objective Regression Test Case Prioritisation

Summary

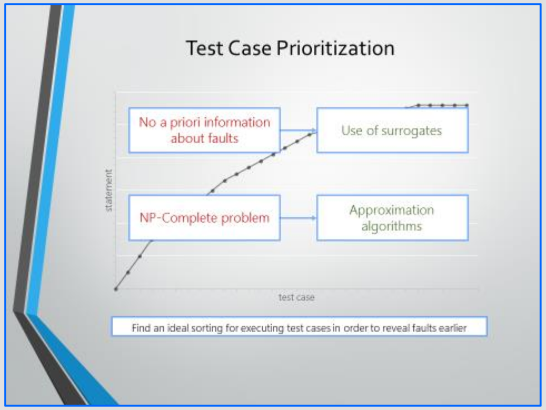

Test Case Prioritization
No a priori information about faults
Use of surrogates
NP-Complete problem
Approximation algorithms
statement
test case
Find an ideal sorting for executing test cases in order to reveal faults earlier

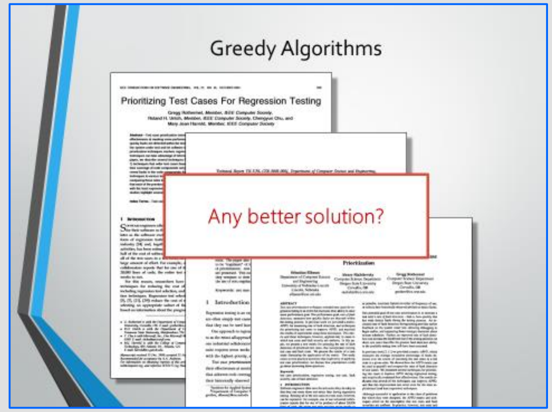

Greedy Algorithms
Prioritizing Test Cases For Regression Testing
Any better solution?

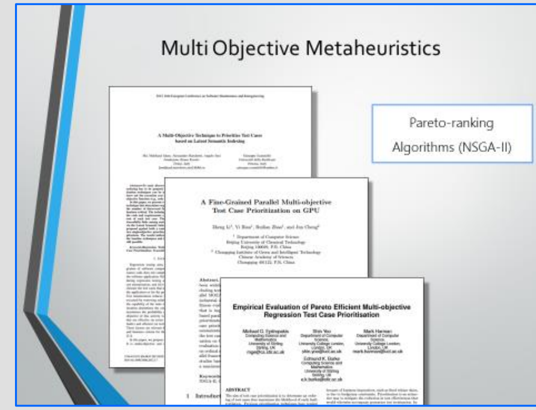

Multi Objective Metaheuristics
Pareto-ranking Algorithms (NSGA-II)
A Fine-Grained Parallel Multi-objective Test Case Prioritization on GPU
Empirical Evaluation of Pareto Efficient Multi-objective Regression Test Case Prioritisation

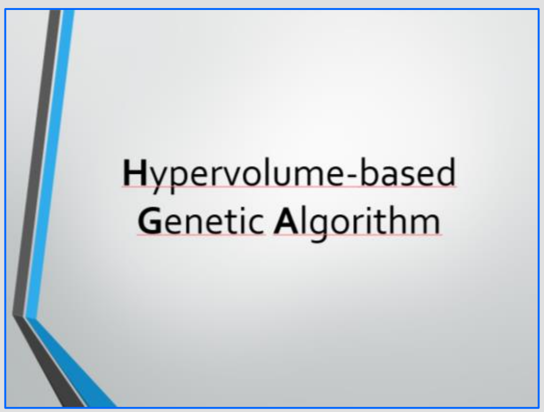

Hypervolume-based Genetic Algorithm

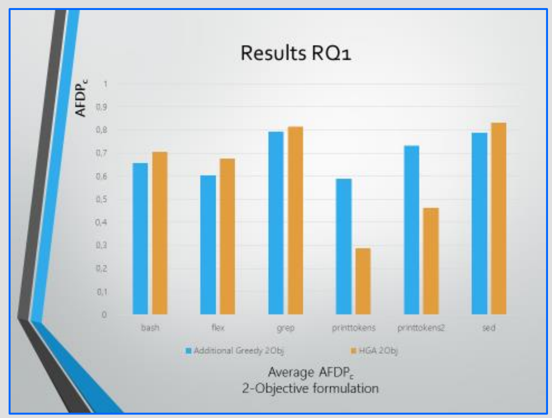
Summary


Test Case Prioritization
No a priori information about faults
Use of surrogates
NP-Complete problem
Approximation algorithms
statement
test case
Find an ideal sorting for executing test cases in order to reveal faults earlier

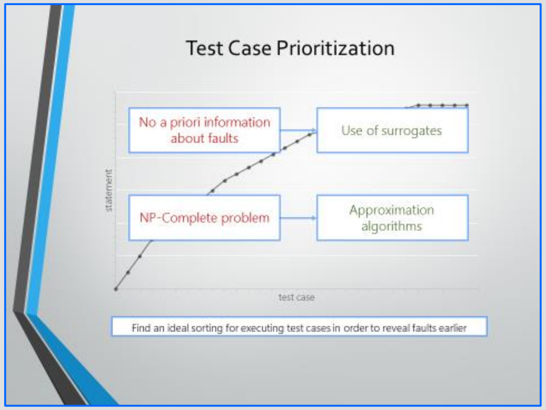

Greedy Algorithms
Prioritizing Test Cases For Regression Testing
Any better solution?

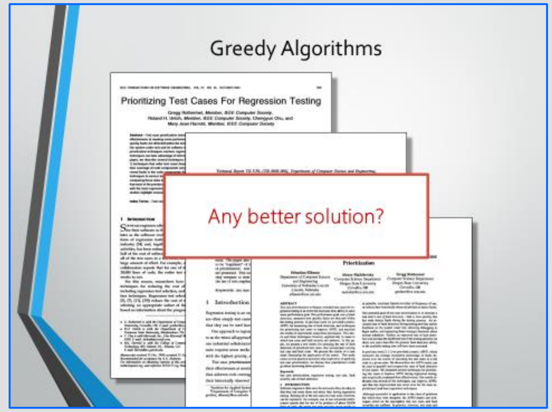

Multi Objective Metaheuristics
Pareto-ranking Algorithms (NSGA-II)
A Fine-Grained Parallel Multi-objective Test Case Prioritization on GPU
Empirical Evaluation of Pareto Efficient Multi-objective Regression Test Case Prioritisation

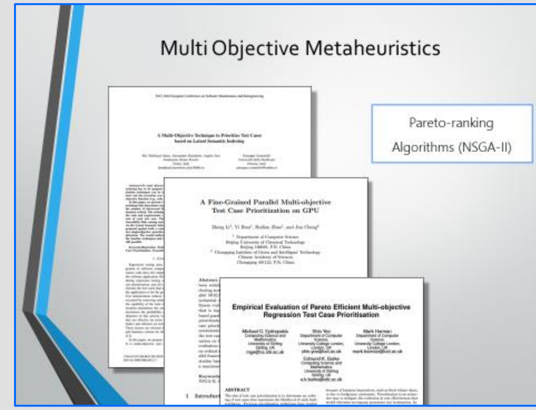

Hypervolume-based Genetic Algorithm

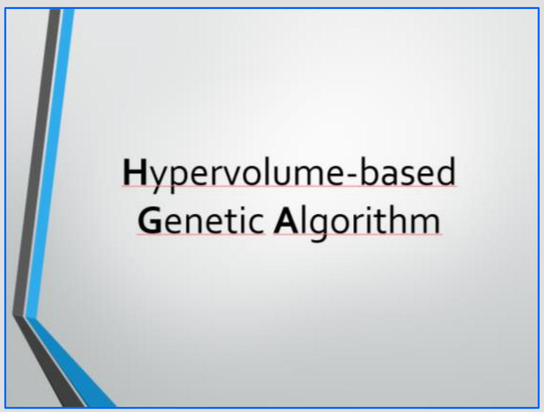

Results RQ1
AFDPc
1
0,9
0,8
0,7
0,6
0,5
0,4
0,3
0,2
0,1
0
bash
flex
grep
printtokens
printtokens2
sed
Additional Greedy 2Obj
HGA 2Obj
Average AFDPc
2-Objective formulation

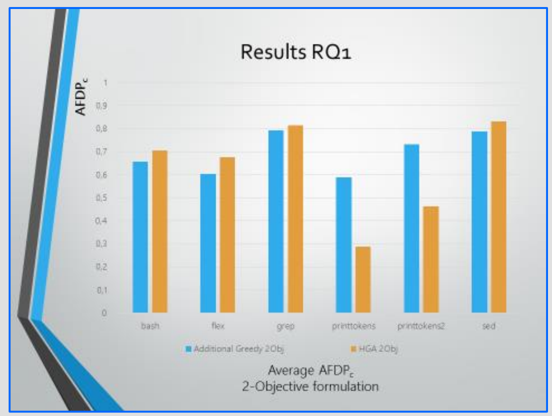
Summary

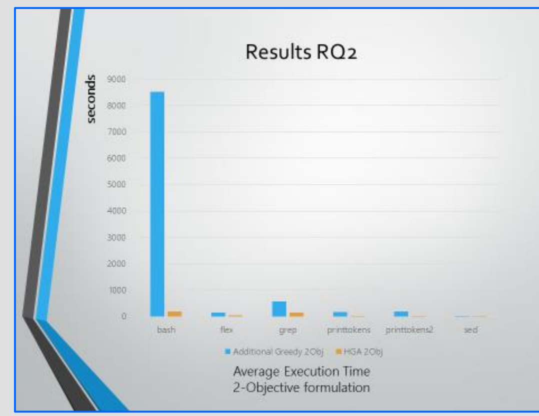

Test Case Prioritization
No a priori information about faults
Use of surrogates
NP-Complete problem
Approximation algorithms
statement
test case
Find an ideal sorting for executing test cases in order to reveal faults earlier


Greedy Algorithms
Prioritizing Test Cases For Regression Testing
Any better solution?


Multi Objective Metaheuristics
Pareto-ranking Algorithms (NSGA-II)
A Fine-Grained Parallel Multi-objective Test Case Prioritization on GPU
Empirical Evaluation of Pareto Efficient Multi-objective Regression Test Case Prioritisation


Hypervolume-based Genetic Algorithm


Results RQ1
AFDPc
bash
flex
grep
printtokens
printtokens2
sed
Additional Greedy 2Obj
HGA 2Obj
Average AFDPc
2-Objective formulation

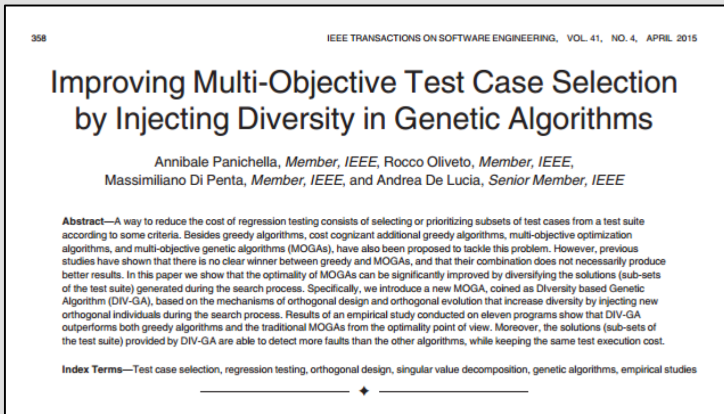

Results RQ2
seconds
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
bash
flex
grep
printtokens
printtokens2
sed
Additional Greedy 2Obj
HGA 2Obj
Average Execution Time
2-Objective formulation

Future Work

Investigate the scalability
for up than 3 testing criteria

3+

Future Work


358 IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 41, NO. 4, APRIL 2015

Improving Multi-Objective Test Case Selection by Injecting Diversity in Genetic Algorithms

Annibale Panichella, *Member, IEEE*, Rocco Oliveto, *Member, IEEE*,
Massimiliano Di Penta, *Member, IEEE*, and Andrea De Lucia, *Senior Member, IEEE*

Abstract—A way to reduce the cost of regression testing consists of selecting or prioritizing subsets of test cases from a test suite according to some criteria. Besides greedy algorithms, cost cognizant additional greedy algorithms, multi-objective optimization algorithms, and multi-objective genetic algorithms (MOGAs), have also been proposed to tackle this problem. However, previous studies have shown that there is no clear winner between greedy and MOGAs, and that their combination does not necessarily produce better results. In this paper we show that the optimality of MOGAs can be significantly improved by diversifying the solutions (sub-sets of the test suite) generated during the search process. Specifically, we introduce a new MOGA, coined as DIversity based Genetic Algorithm (DIV-GA), based on the mechanisms of orthogonal design and orthogonal evolution that increase diversity by injecting new orthogonal individuals during the search process. Results of an empirical study conducted on eleven programs show that DIV-GA outperforms both greedy algorithms and the traditional MOGAs from the optimality point of view. Moreover, the solutions (sub-sets of the test suite) provided by DIV-GA are able to detect more faults than the other algorithms, while keeping the same test execution cost.

Index Terms—Test case selection, regression testing, orthogonal design, singular value decomposition, genetic algorithms, empirical studies

Investigate the scalability
for up than 3 testing criteria

Incorporate diversity as a testing criteria

Future Work

Investigate the scalability for up than 3 testing criteria

Incorporate diversity as a testing criteria

Apply HGA for other test case optimization problems

Thank You
for
Your Attention!

Questions?

Dario Di Nucci
University of Salerno
ddinucci@unisa.it
http://www.sesa.unisa.it/people/ddinucci/