

Using Synthetic Test Suites to Empirically Compare Search-Based and Greedy Prioritizers

Zachary Williams
Gregory M. Kapfhammer



Department of Computer Science
Allegheny College
<http://www.cs.allegheny.edu/>

Genetic and Evolutionary Computation Conference
Late Breaking Abstract Workshop
July 2010

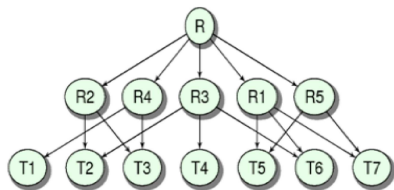
Important Contributions

Synthetic Test Suites

Detailed Empirical Study

Use **synthetic test suites** to empirically evaluate the **efficiency** and **effectiveness** of search-based and greedy prioritizers

Important Contributions

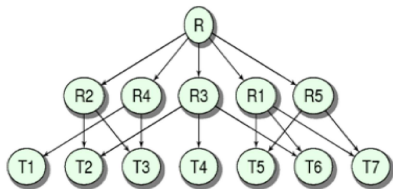


Synthetic Test Suites

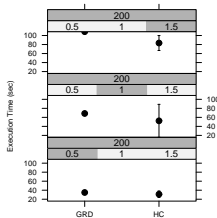
Detailed Empirical Study

Use **synthetic test suites** to empirically evaluate the **efficiency** and **effectiveness** of search-based and greedy prioritizers

Important Contributions



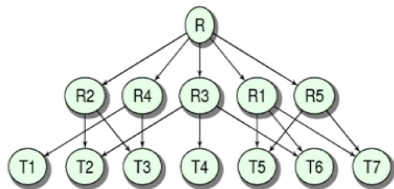
Synthetic Test Suites



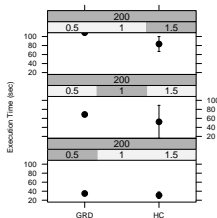
Detailed Empirical Study

Use **synthetic test suites** to empirically evaluate the **efficiency** and **effectiveness** of search-based and greedy prioritizers

Important Contributions



Synthetic Test Suites



Detailed Empirical Study

Use **synthetic test suites** to empirically evaluate the **efficiency** and **effectiveness** of search-based and greedy prioritizers

Overview of Regression Testing



Correct programing defect

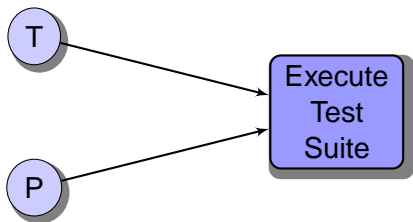
Overview of Regression Testing

T

P

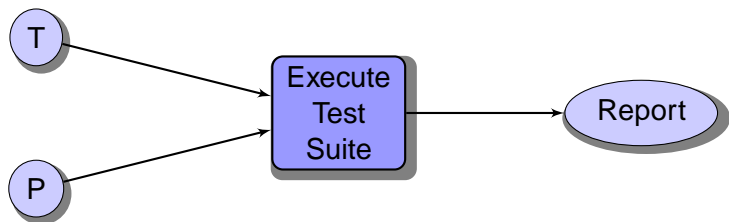
Correct programing defect

Overview of Regression Testing



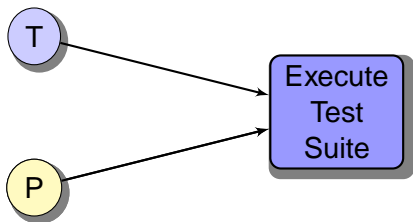
Correct programming defect

Overview of Regression Testing



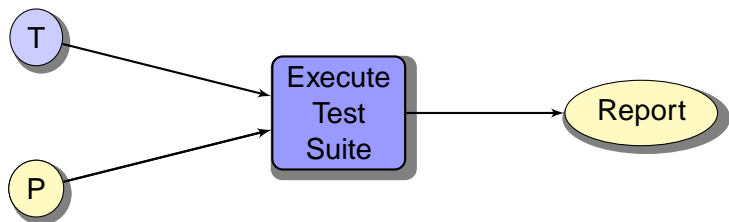
Correct programming defect

Overview of Regression Testing



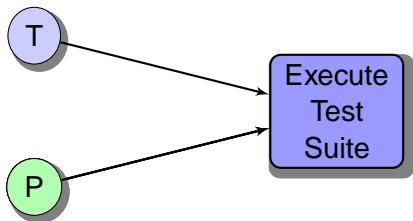
Correct programming defect

Overview of Regression Testing



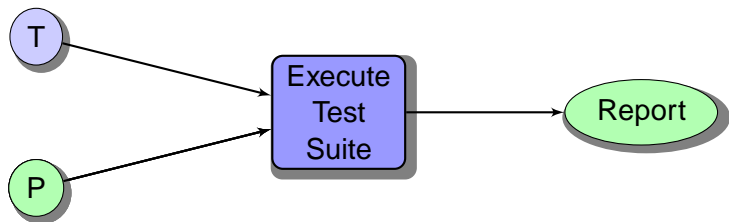
Correct programming defect

Overview of Regression Testing



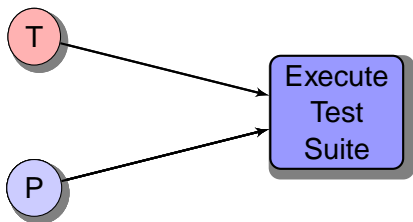
Add new functionality

Overview of Regression Testing



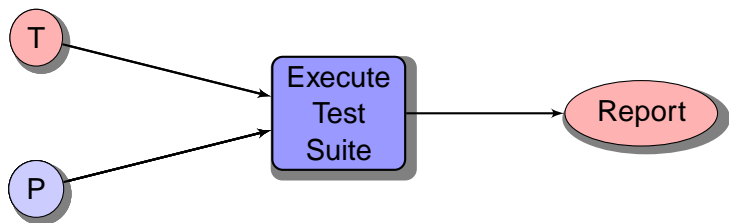
Add new functionality

Overview of Regression Testing



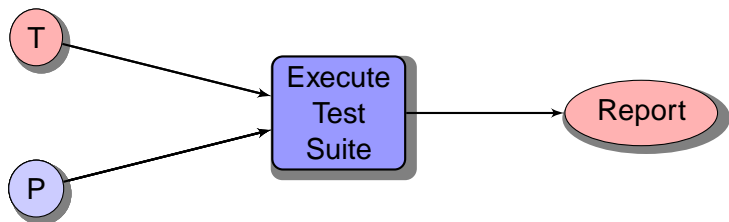
Modify test suite

Overview of Regression Testing



Modify test suite

Overview of Regression Testing



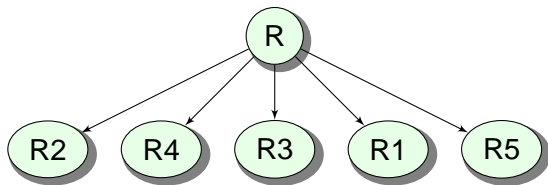
Complete retesting is often prohibitively expensive

Regression Test Suite Prioritization



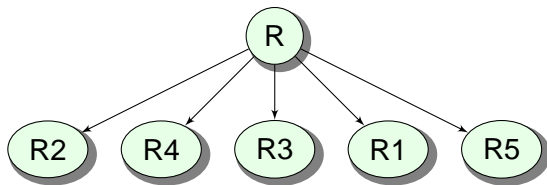
Requirements necessitate the coverage of the **state** and/or **structure** of a program under test

Regression Test Suite Prioritization



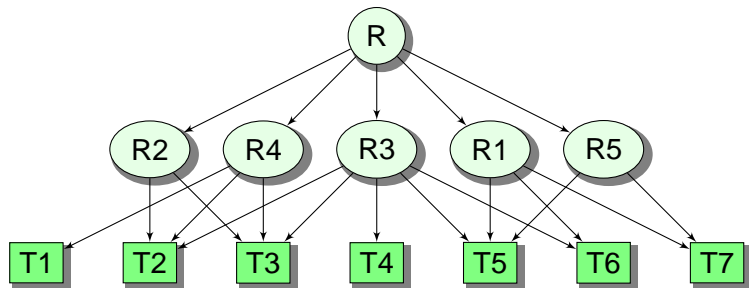
Requirements necessitate the coverage of the **state** and/or **structure** of a program under test

Regression Test Suite Prioritization



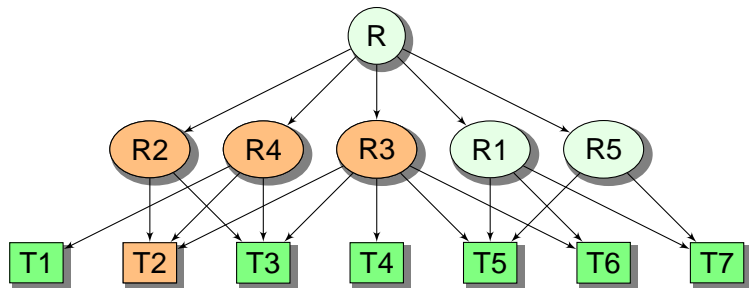
Requirements necessitate the coverage of the **state** and/or **structure** of a program under test

Regression Test Suite Prioritization



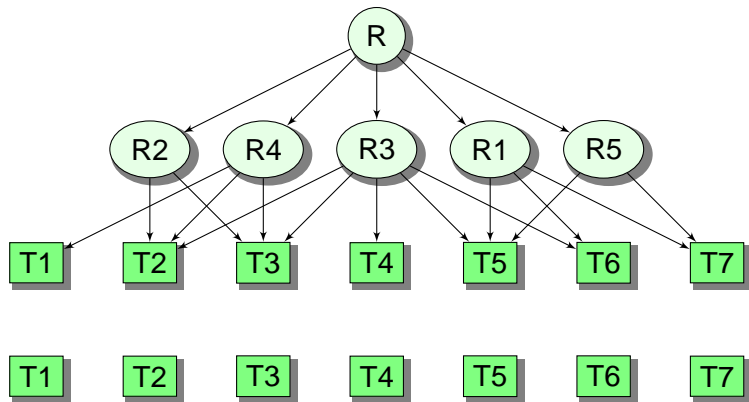
Each test covers specific **requirements** in a certain amount of **time** and thus the **ordering** is critical

Regression Test Suite Prioritization



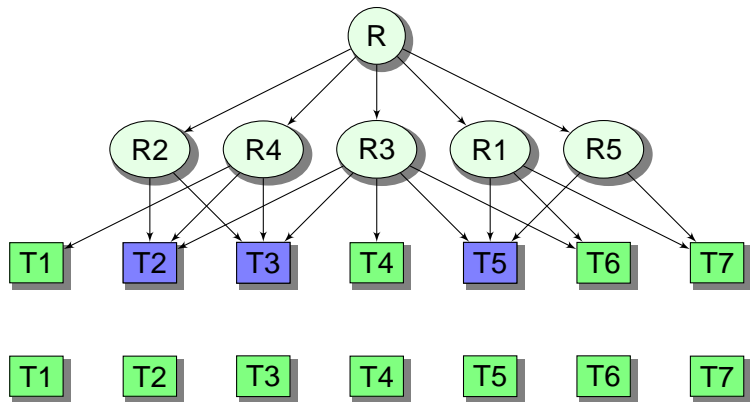
Each test covers specific **requirements** in a certain amount of **time** and thus the **ordering** is critical

Regression Test Suite Prioritization



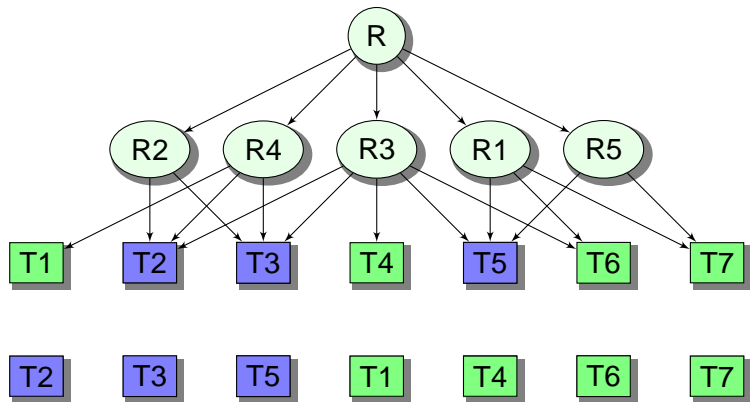
Prioritized test suites cover requirements **faster** thus enabling the **rapid** detection of defects

Regression Test Suite Prioritization



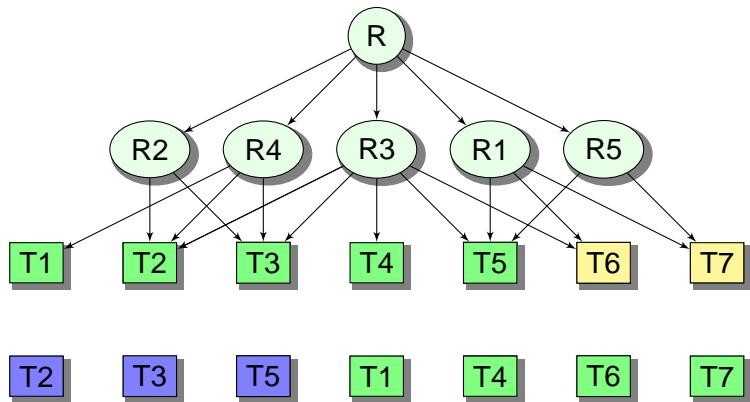
Prioritized test suites cover requirements **faster** thus enabling the **rapid** detection of defects

Regression Test Suite Prioritization



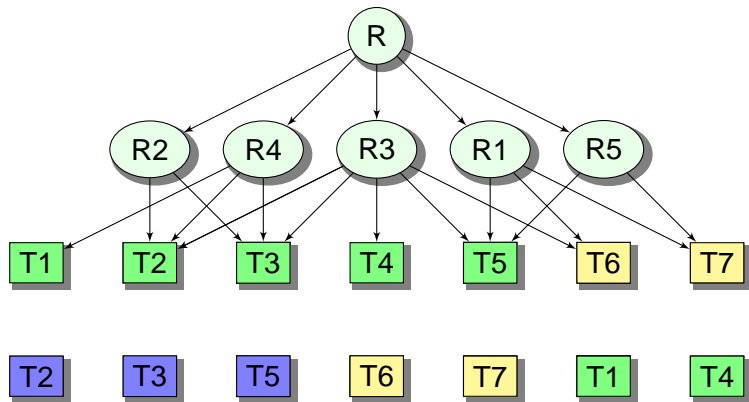
Prioritized test suites cover requirements **faster** thus enabling the **rapid** detection of defects

Regression Test Suite Prioritization



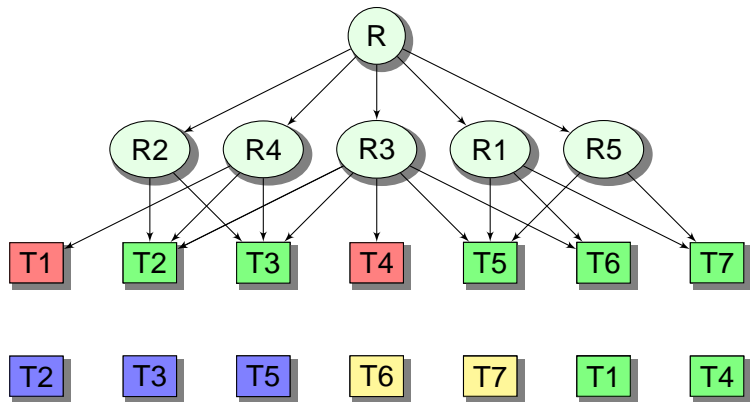
Prioritized test suites cover requirements **faster** thus enabling the **rapid** detection of defects

Regression Test Suite Prioritization



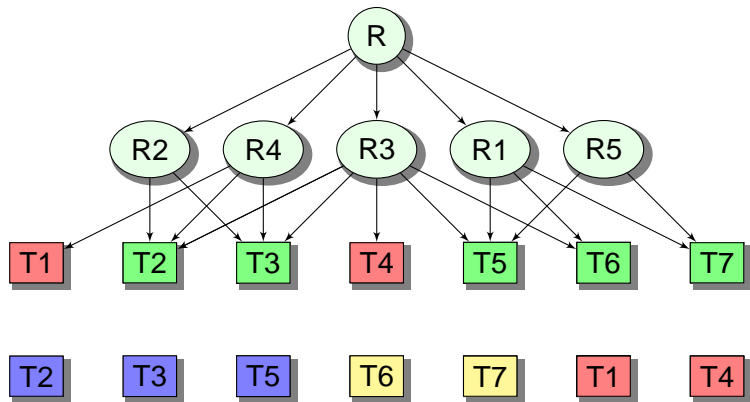
Prioritized test suites cover requirements **faster** thus enabling the **rapid** detection of defects

Regression Test Suite Prioritization



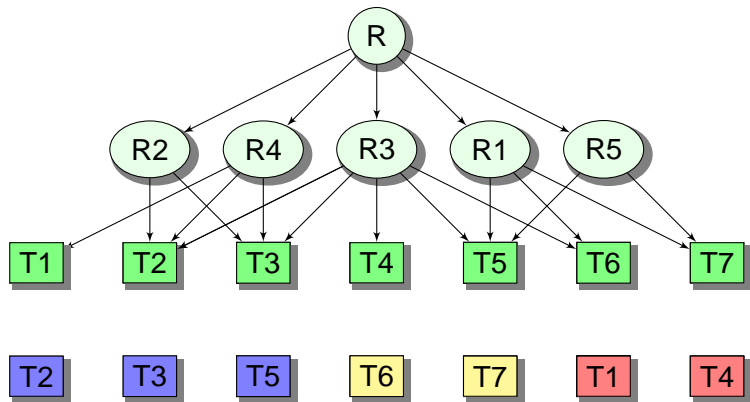
Prioritized test suites cover requirements **faster** thus enabling the **rapid** detection of defects

Regression Test Suite Prioritization



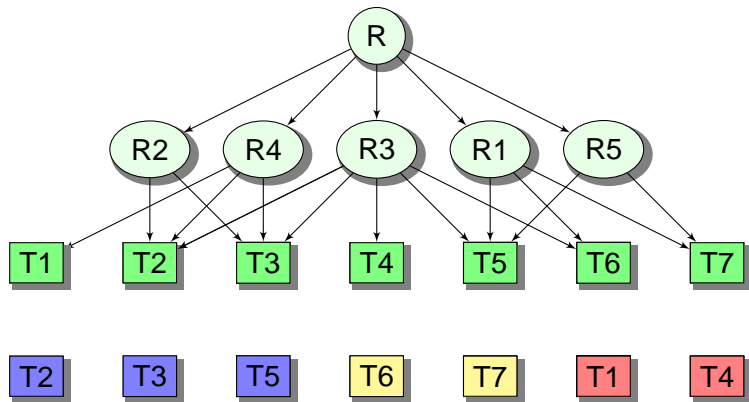
Prioritized test suites cover requirements **faster** thus enabling the **rapid** detection of defects

Regression Test Suite Prioritization



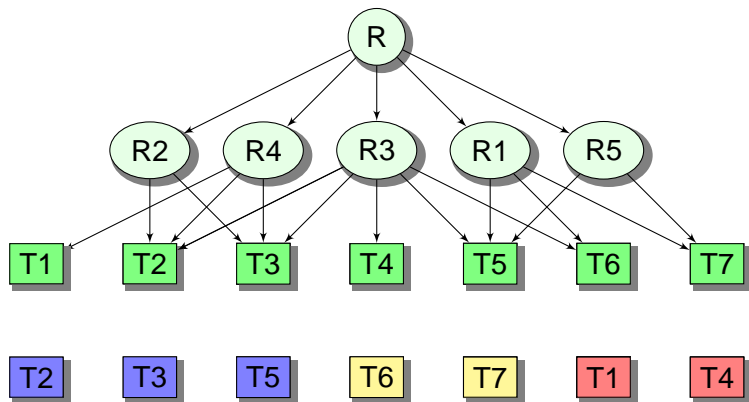
Prioritized test suites cover requirements **faster** thus enabling the **rapid** detection of defects

Regression Test Suite Prioritization



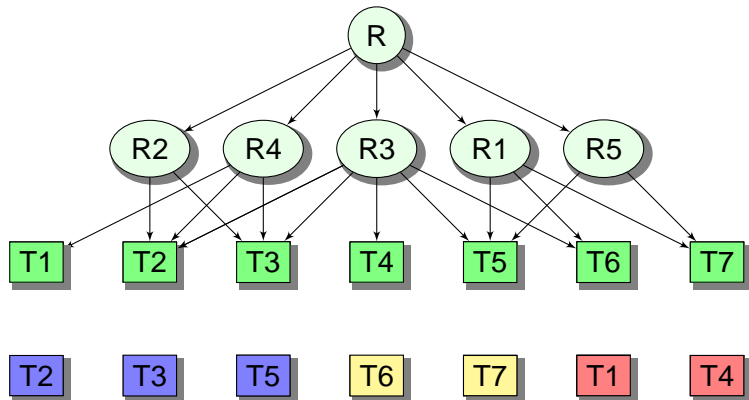
Prioritized test suites cover requirements **faster** thus enabling the **rapid** detection of defects

Regression Test Suite Prioritization



Testers can use **greedy** (Rothermel et al. TSE 2001) and **search-based** (Li et al. TSE 2007) methods to reorder suites

Regression Test Suite Prioritization



QUESTION: Which prioritization technique is the best?

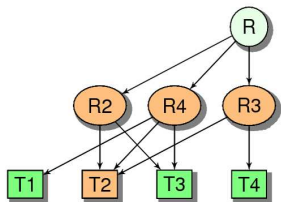
Existing Prioritization Techniques

Greedy approaches select the next best test case

Hill climbers search the state space for improved orderings

Conventional wisdom dictates that greedy generally out performs hill climbing in terms of both efficiency and effectiveness

Existing Prioritization Techniques

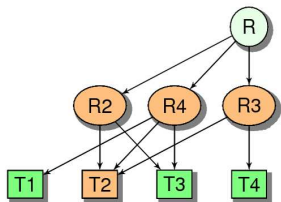


Greedy approaches select the next best test case

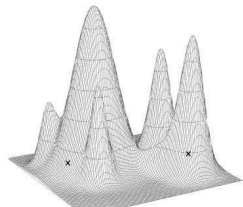
Hill climbers search the state space for improved orderings

Conventional wisdom dictates that greedy generally outperforms hill climbing in terms of both efficiency and effectiveness

Existing Prioritization Techniques



Greedy approaches select the next best test case

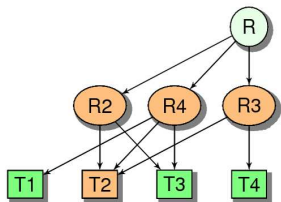


http://artedi.ebc.uu.se/course/Embo01/Phylogeny/phylogeny_readme.html

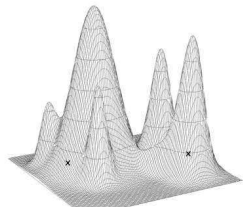
Hill climbers search the state space for improved orderings

Conventional wisdom dictates that greedy generally outperforms hill climbing in terms of both efficiency and effectiveness

Existing Prioritization Techniques



Greedy approaches select the next best test case



http://artedi.ebc.uu.se/course/Embo01/Phylogeny/phylogeny_readme.html

Hill climbers search the state space for improved orderings

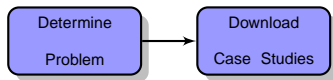
Conventional wisdom dictates that greedy generally outperforms hill climbing in terms of both efficiency and effectiveness

Conducting an Empirical Evaluation

Determine
Problem

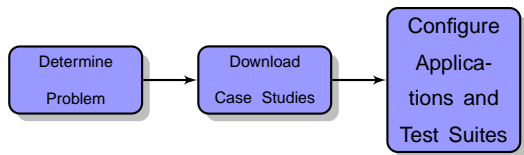
These highlighted tasks are **manual**, **expensive**, and
prone to error

Conducting an Empirical Evaluation



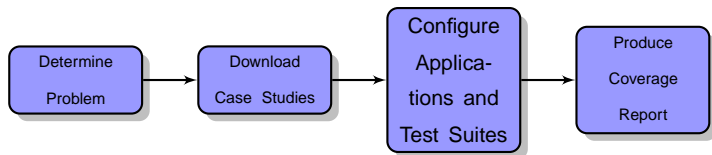
These highlighted tasks are **manual**, **expensive**, and **prone to error**

Conducting an Empirical Evaluation



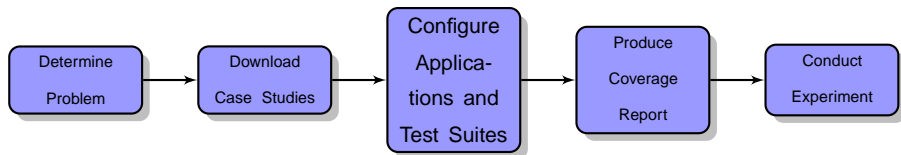
These highlighted tasks are **manual**, **expensive**, and **prone to error**

Conducting an Empirical Evaluation



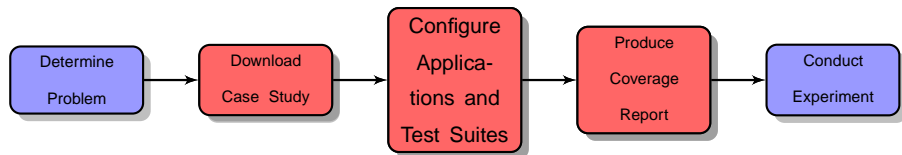
These highlighted tasks are **manual**, **expensive**, and **prone to error**

Conducting an Empirical Evaluation



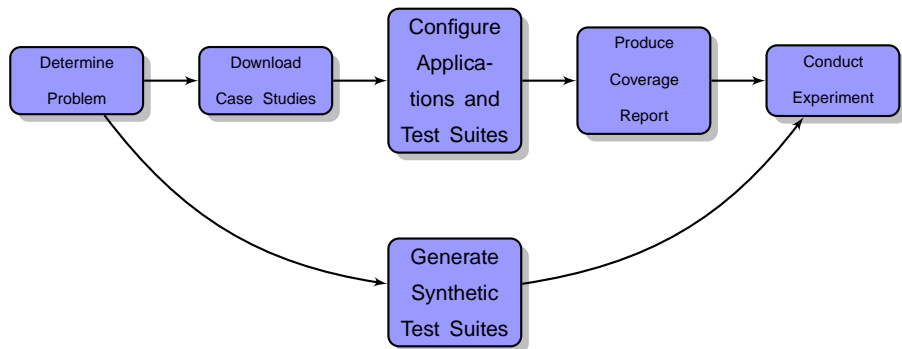
These highlighted tasks are **manual, expensive, and prone to error**

Conducting an Empirical Evaluation



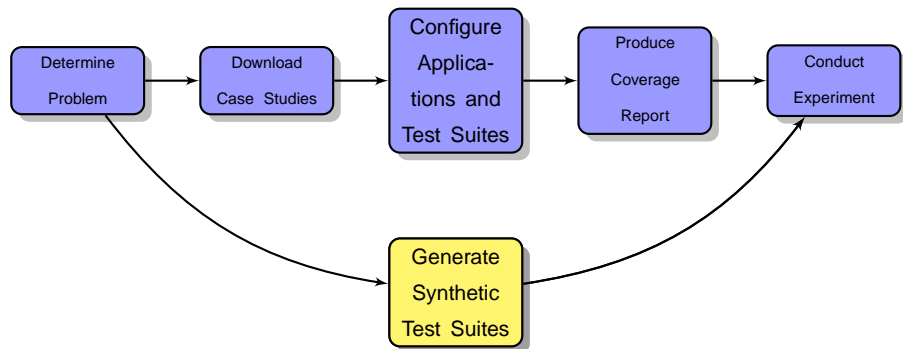
These highlighted tasks are **manual**, **expensive**, and **prone to error**

Conducting an Empirical Evaluation



Synthetically generating a test suite is **automated**, **effective**, and **efficient**

Conducting an Empirical Evaluation



Synthetically generating a test suite is **automated**, **effective**, and **efficient**

Generating Synthetic Test Suite



Tests

The **total number of tests** controls how many tests the suite will contain

Generating Synthetic Test Suite



Tests

Requirements

The **total number of requirements** governs how many requirements the test suite will cover

Generating Synthetic Test Suite

Tests

Requirements

Coverage Points

The **total number of coverage points** controls how many unique test-requirement pairs the test suite will contain

Generating Synthetic Test Suite

Tests

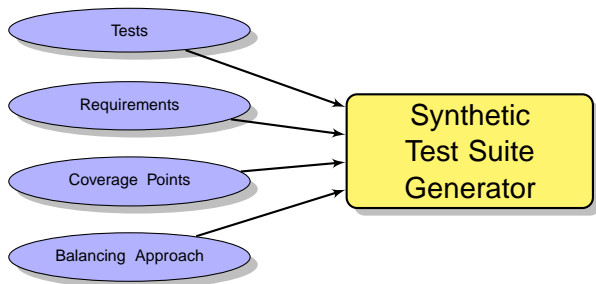
Requirements

Coverage Points

Balancing Approach

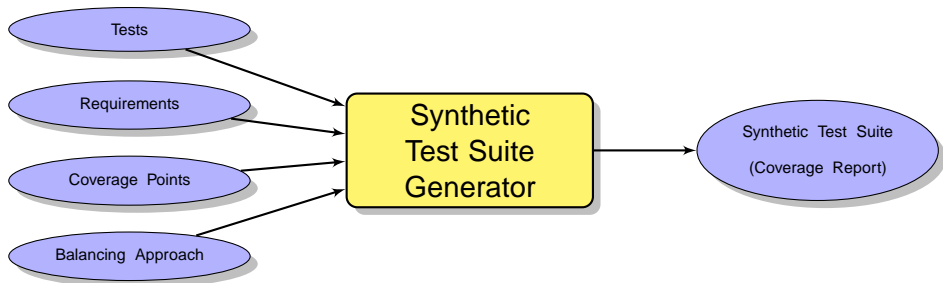
The **balancing configuration** dictates how the coverage points will be distributed in the synthetic test suite

Generating Synthetic Test Suite



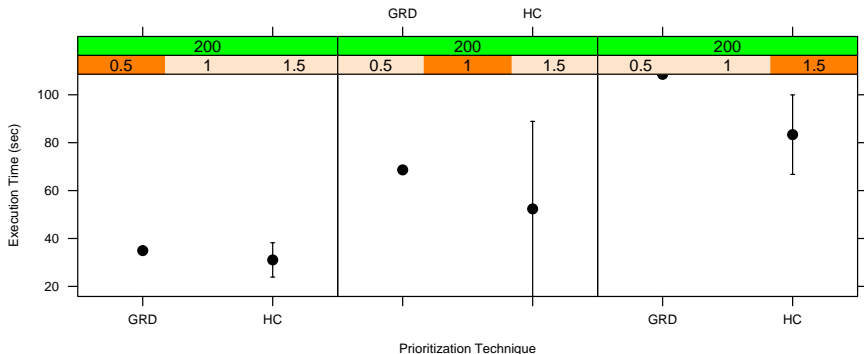
Our empirical results show that synthetic generation takes less than **0.2 seconds** for extremely large test suites

Generating Synthetic Test Suite



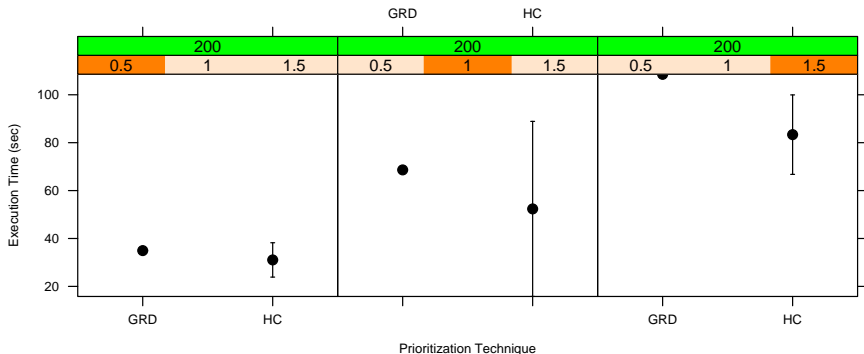
Contains information concerning the **requirements covered** and the **execution time** of each test

Empirical Results – Prioritizer Efficiency



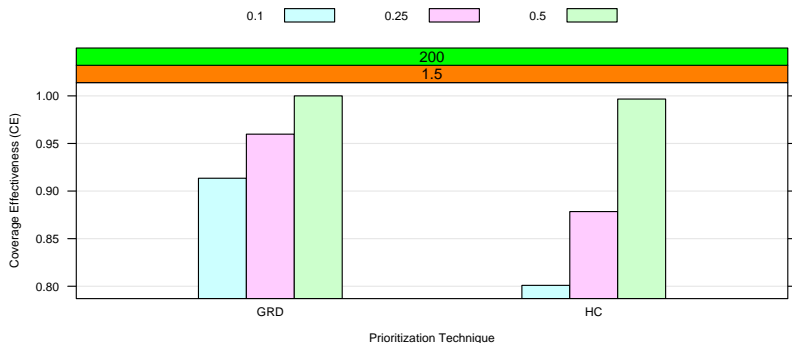
HC demonstrated to be more efficient than GRD for large test suites

Empirical Results – Prioritizer Efficiency



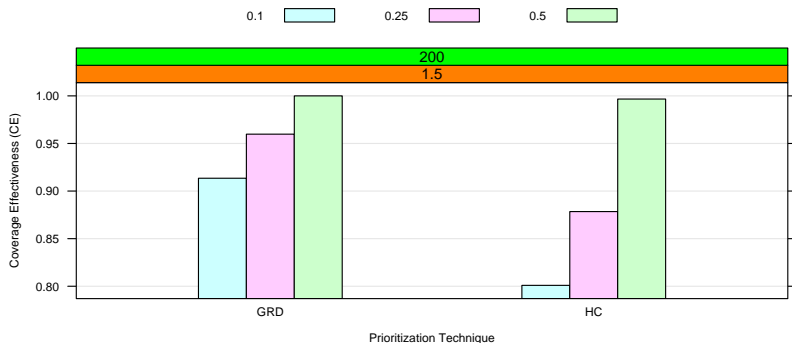
HC demonstrated to be more efficient than GRD for large test suites

Empirical Results – Prioritizer Effectiveness



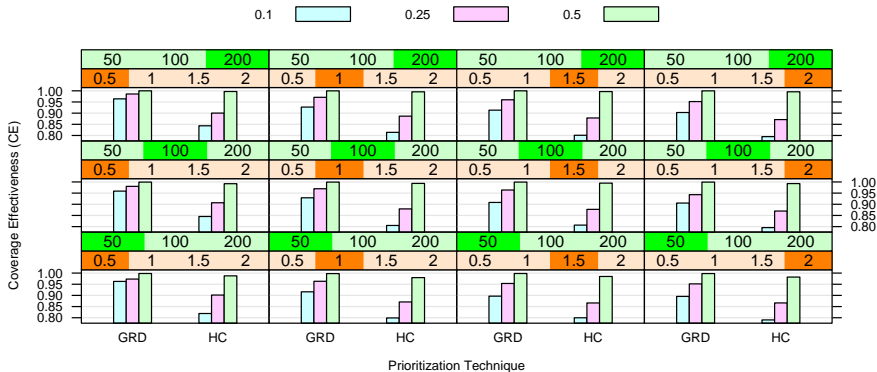
As the amount of coverage points in the test suite increases the performance of HC becomes comparable to that of GRD

Empirical Results – Prioritizer Effectiveness



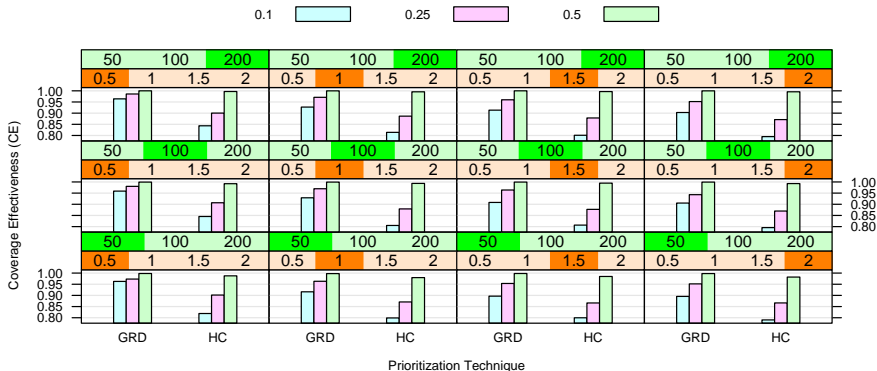
As the amount of coverage points in the test suite increases the performance of HC becomes comparable to that of GRD

Empirical Results – Prioritizer Effectiveness



The trend is evident over a wide range of experimental configurations

Empirical Results – Prioritizer Effectiveness



The trend is evident over a wide range of experimental configurations

Conclusion and Future Work

Using Synthetic Test Suites to Empirically Compare Search-Based and Greedy Prioritizers

Zachary D. Williams
Department of Computer Science
Allegheny College
williaz@allegheny.edu

Gregory M. Kapfhammer
Department of Computer Science
Allegheny College
gkapfham@allegheny.edu

ABSTRACT

The increase in the complexity of modern software has led to the corresponding growth in the size and execution time of the test suites for these programs. In order to address this alarming trend, developers use test suite prioritization to render the test cases so that faults can be detected at an early stage of testing. Yet, the implementation and evaluation of greedy and search-based test prioritizers requires access to case study applications and their associated test suites, which are often difficult to find, configure, and use in an empirical study. This paper presents two types of synthetically generated test suites that support this process of experimentally evaluating prioritization methods. Using synthetic test suites affords greater control over test case characteristics and supports the identification of empirical trends that contradict the established wisdom about search-based and greedy prioritization. For instance, we find that the hill-climbing algorithm often exhibits a lower time overhead than the greedy test suite prioritization while producing test orderings with comparable effectiveness scores.

Categories and Subject Descriptors: D.2.2 [Software Engineering]: Testing and Debugging

General Terms: Experimentation, Performance

Keywords: search-based and greedy test prioritization

1. INTRODUCTION

Software developers often introduce defects during the implementation process. Integration testing methods mitigate a confidence in the correctness of and isolate defects within a program by running a collection of tests known as a test suite. Since regression testing can be very time-consuming, testers use search-based and greedy prioritization techniques to produce a test ordering that will reveal faults earlier in the suite's execution than would otherwise be possible.

Suppose that a test suite $T = \{t_1, t_2, t_3, \dots, t_n\}$ covers the set of requirements $R(T) = \{r_1, r_2, r_3, \dots, r_m\}$. Each test case $t_i \in T$ is associated with the non-empty set $R(t_i) \subseteq R(T)$ [3, 9]. During the empirical study of search-based

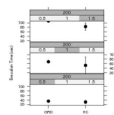


Figure 1: Execution Time - Fully Random.

and greedy test suite prioritizers, researchers often use the T and $R(T)$ associated with real-world case study applications. Yet, this practice can be difficult and time-consuming because of the need to tailor post-hoc test rendering tools for complex real-world programs. Furthermore, some small case-study applications may not be representative of all real-world programs, thus limiting empirical investigations of the efficiency and effectiveness of approaches to testing. The use of real-world programs also prohibits the experimenter from easily controlling the size of the test suite T and the coverage patterns within $R(T)$. Ultimately, the lack of a wide variety of test suites hinders the ability of researchers to quickly compare and contrast the plethora of newly developed techniques for test prioritization (e.g., [1, 3, 4, 9]).

Synthetic Test Suites. Using efficiently generated synthetic test suites to study search-based and greedy prioritizers enables experiments to easily establish baseline results and control the key characteristics of the tests [2]. As such, this paper describes two simple methods for generating synthetic test suites and demonstrates how they reveal fundamental trade-offs in test prioritization techniques. We need two parameters to create the synthetic test suites: requirement factor F_r and coverage point factor F_c . For a chosen test suite size, F_c controls how many requirements the generated test suite will have, such that $|R(T)| = F_c \times |T|$. After setting the size of the test suite and requirement set, we use F_r to define the number of times the requirements are covered, denoted C_r , as a function of the total number of possible coverage points, so that $C_r = F_r \times (|R(T)| - |T|)$.

Conclusion

- Synthetic test suite generation is efficient
- Enable the identification of fundamental trade-offs

Future Work

- Apply our technique to genetic and other algorithms
- Implement and evaluate new and different synthetic generators

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted by copyright owner, provided that copies are not made or distributed for profit or commercial advantage and that copies bear the name and the full address on the top page. Any copying that exceeds this permission, in part or in whole, is prohibited. For more information on this, please refer to our copyright notice on page 2.

GRECO '20, July 7-11, 2020, Portland, Oregon, USA.
Copyright 2020 ACM 978-1-4503-7035-7/20/07...\$5.00.

<http://www.cs.allegheny.edu/~gkapfham/research/kanonizo/>

Conclusion and Future Work

Using Synthetic Test Suites to Empirically Compare Search-Based and Greedy Prioritizers

Zachary D. Williams
Department of Computer Science
Allegheny College
williaz@allegheny.edu

Gregory M. Kapfhammer
Department of Computer Science
Allegheny College
gkapfham@allegheny.edu

ABSTRACT

The increase in the complexity of modern software has led to the commensurate growth in the size and execution time of the test suites for these programs. In order to address this alarming trend, developers use test suite prioritization to render the test cases so that faults can be detected at an early stage of testing. Yet, the implementation and evaluation of greedy and search-based test prioritizers requires access to case study applications and their associated test suites, which are often difficult to find, configure, and use in an empirical study. This paper presents two types of synthetically generated test suites that support this process of experimentally evaluating prioritization methods. Using synthetic test suites affords greater control over test case characteristics and supports the identification of empirical trends that contradict the established wisdom about search-based and greedy prioritization. For instance, we find that the hill climbing algorithm often exhibits a lower time overhead than the greedy test suite prioritization while producing test orderings with comparable effectiveness scores.

Categories and Subject Descriptors: D.2.2 [Software Engineering]: Testing and Debugging

General Terms: Experimentation, Performance

Keywords: search-based and greedy test prioritization

1. INTRODUCTION

Software developers often introduce defects during the implementation process. Integration testing methods mitigate confidence in the correctness of and isolate defects within a program by running a collection of tests known as a test suite. Since regression testing can be very time consuming, testers use search-based and greedy prioritization techniques to produce a test ordering that will reveal faults earlier in the suite's execution than would otherwise be possible.

Suppose that a test suite $T = \{t_1, t_2, t_3, \dots, t_n\}$ covers the set of requirements $R(T) = \{r_1, r_2, r_3, \dots, r_m\}$. Each test case $t_i \in T$ is associated with the non-empty set $R(t_i) \subseteq R(T)$ [3, 9]. During the empirical study of search-based

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted by copyright owner, provided the copies are not made or distributed for profit or commercial advantage and the copies bear the notice and the full citation on the first page. Copyrights for components that may not be registered to IEEE Press are held by their respective owners. This paper is published in the proceedings of the 2010 ACM SIGSOFT International Symposium on Software Testing and Verification, June 11-13, 2010, Portland, Oregon, USA. Copyright 2010 ACM 978-9-781533-333-3/10/06...\$10.00.

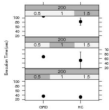


Figure 1: Execution Time - Fully Random.

and greedy suite prioritizers, researchers often use the F and $R(F)$ associated with real-world case study applications. Yet, this practice can be difficult and time-consuming because of the need to tailor post-hoc test reordering tools for complex real-world programs. Furthermore, some small case-study applications may not be representative of all real-world programs, thus limiting empirical investigations of the efficiency and effectiveness of approaches to testing. The use of real-world programs also prohibits the experimenter from easily controlling the size of the test suite T and the coverage patterns within $R(T)$. Ultimately, the lack of a wide variety of test suites hinders the ability of researchers to quickly compare and contrast the plethora of newly developed techniques for test prioritization (e.g., [1, 3, 4, 9]).

Synthetic Test Suites. Using efficiently generated synthetic test suites to study search-based and greedy prioritizers enables experimenters to easily establish baseline results and control the key characteristics of the tests [2]. As such, this paper describes two simple methods for generating synthetic test suites and demonstrates how they reveal fundamental trade-offs in test prioritization techniques. We need two parameters to create the synthetic test suites: requirement factor F_s and coverage point factor F_c . For a chosen test suite size, F_s controls how many requirements the generated test suite will have, such that $|R(T)| = F_s \times |T|$. After setting the size of the test suite and requirement set, we use F_c to define the number of times the specific requirements are covered, denoted C_i , as a function of the total number of possible coverage points, so that $C_i = F_c \times (|R(T)| - |T|)$.

Conclusion

- Synthetic test suite generation is efficient
- Enable the identification of fundamental trade-offs

Future Work

- Apply our technique to genetic and other algorithms
- Implement and evaluate new and different synthetic generators

<http://www.cs.allegheny.edu/~gkapfham/research/kanonizo/>

Conclusion and Future Work

Using Synthetic Test Suites to Empirically Compare Search-Based and Greedy Prioritizers

Zachary D. Williams
Department of Computer Science
Aleggheny College
williaz@alegheny.edu

Gregory M. Kapfhammer
Department of Computer Science
Aleggheny College
gkapfham@alegheny.edu

ABSTRACT

The increase in the complexity of modern software has led to the commensurate growth in the size and execution time of the test suites for these programs. In order to address this alarming trend, developers use test suite prioritization to render the test cases so that faults can be detected at an early stage of testing. Yet, the implementation and evaluation of greedy and search-based test prioritizers requires access to case study applications and their associated test suites, which are often difficult to find, configure, and use in an empirical study. This paper presents two types of synthetically generated test suites that support this process of experimentally evaluating prioritization methods. Using synthetic test suites affords greater control over test case characteristics and supports the identification of empirical trends that contradict the established wisdom about search-based and greedy prioritization. For instance, we find that the hill climbing algorithm often exhibits a lower time overhead than the greedy test suite prioritization while producing test orderings with comparable effectiveness scores.

Categories and Subject Descriptors: D.2.2 [Software Engineering]: Testing and Debugging

General Terms: Experimentation, Performance

Keywords: search-based and greedy test prioritization

1. INTRODUCTION

Software developers often introduce defects during the implementation process. Inefficient test methods mitigate confidence in the correctness of and isolate defects within a program by running a collection of tests known as a test suite. Since expensive testing can be very time consuming, testers use search-based and greedy prioritization techniques to produce a test ordering that will reveal faults earlier in the suite's execution than would otherwise be possible.

Suppose that a test suite $T = \{t_1, t_2, t_3, \dots, t_n\}$ covers the set of requirements $R(T) = \{r_1, r_2, r_3, \dots, r_m\}$. Each test case $t_i \in T$ is associated with the non-empty set $R(t_i) \subseteq R(T)$ [3, 5]. During the empirical study of search-based

techniques to make slight or large changes to all or part of this work are permitted whenever it is granted permission to publish the copies are not made or distributed for profit or commercial advantage and that copies bear the notice and the full notice on the lower page.

Copyright 2010 ACM 978-955-30-9120-9/10/0000...\$10.00

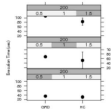


Figure 1: Execution Time - Fully Random.

and greedy test suite prioritizers, researchers often use the F and $R(T)$ associated with real-world case study applications. Yet, this practice can be difficult and time-consuming because of the need to tailor post-hoc test rendering tools for complex real-world programs. Furthermore, some small case-study applications may not be representative of all real-world programs, thus limiting empirical investigations of the efficiency and effectiveness of approaches to testing. The use of real-world programs also prohibits the experimenter from easily controlling the size of the test suite T and the coverage patterns within $R(T)$. Ultimately, the lack of a wide variety of test suites hinders the ability of researchers to quickly compare and contrast the plethora of newly developed techniques for test prioritization (e.g., [1, 3, 4, 5]).

Synthetic Test Suites. Using efficiently generated synthetic test suites to study search-based and greedy prioritizers enables experimenters to easily establish baseline results and control the key characteristics of the tests [2]. As such, this paper describes two simple methods for generating synthetic test suites and demonstrates how they reveal fundamental trade-offs in test prioritization techniques. We need two parameters to create the synthetic test suites: requirement factor F_c and coverage point factor F_r . For a chosen test suite size, F_c controls how many requirements the generated test suite will have, such that $|R(T)| = F_c \times |T|$. After setting the size of the test suite and requirement set, we use F_r to define the number of times the requirements are covered, denoted C , as a function of the total number of possible coverage points, so that $C = F_r \times (|R(T)| \times |T|)$.

Conclusion

- Synthetic test suite generation is efficient
- Enable the identification of fundamental trade-offs

Future Work

- Apply our technique to genetic and other algorithms
- Implement and evaluate new and different synthetic generators

<http://www.cs.alegheny.edu/~gkapfham/research/kanonizo/>

Conclusion and Future Work

Using Synthetic Test Suites to Empirically Compare Search-Based and Greedy Prioritizers

Zachary D. Williams
Department of Computer Science
Allegheny College
williaz@allegheny.edu

Gregory M. Kapfhammer
Department of Computer Science
Allegheny College
gkapfham@allegheny.edu

ABSTRACT

The increase in the complexity of modern software has led to the commensurate growth in the size and execution time of the test suites for these programs. In order to address this alarming trend, developers use test suite prioritization to render the test cases so that faults can be detected at an early stage of testing. Yet, the implementation and evaluation of greedy and search-based test prioritizers requires access to case study applications and their associated test suites, which are often difficult to find, configure, and use in an empirical study. This paper presents two types of synthetically generated test suites that support this process of experimentally evaluating prioritization methods. Using synthetic test suites affords greater control over test case characteristics and supports the identification of empirical trends that contradict the established wisdom about search-based and greedy prioritization. For instance, we find that the hill climbing algorithm often exhibits a lower time overhead than the greedy test suite prioritization while producing test orderings with comparable effectiveness scores.

Categories and Subject Descriptors: D.2.5 [Software Engineering]: Testing and Debugging

General Terms: Experimentation, Performance

Keywords: search-based and greedy test prioritization

1. INTRODUCTION

Software developers often introduce defects during the implementation process. Inefficient test methods mitigate confidence in the correctness of and isolate defects within a program by creating a collection of tests known as a test suite. Since regression testing can be very time-consuming, testers use search-based and greedy prioritization techniques to produce a test ordering that will reveal faults earlier in the suite's execution than would otherwise be possible.

Suppose that a test suite $T = \{t_1, t_2, t_3, \dots, t_n\}$ covers the set of requirements $R(T) = \{r_1, r_2, r_3, \dots, r_m\}$. Each test case $t_i \in T$ is associated with the non-empty set $R(t_i) \subseteq R(T)$ [3, 9]. During the empirical study of search-based

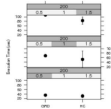


Figure 1: Execution Time - Fully Random.

and greedy test suite prioritizers, researchers often use the T and $R(T)$ associated with real-world case study applications. Yet, this practice can be difficult and time-consuming because of the need to tailor post-hoc test rendering tools for complex real-world programs. Furthermore, some small case-study applications may not be representative of all real-world programs, thus limiting empirical investigations of the efficiency and effectiveness of approaches to testing. The use of real-world programs also prohibits the experimenter from easily controlling the size of the test suite T and the coverage patterns within $R(T)$. Ultimately, the lack of a wide variety of test suites hinders the ability of researchers to quickly compare and contrast the plethora of newly developed techniques for test prioritization (e.g., [1, 3, 4, 9]).

Synthetic Test Suites. Using efficiently generated synthetic test suites to study search-based and greedy prioritizers enables experimenters to easily establish baseline results and control the key characteristics of the tests [2]. As such, this paper describes two simple methods for generating synthetic test suites and demonstrates how they reveal fundamental trade-offs in test prioritization techniques. We need two parameters to create the synthetic test suites: requirement factor F_s and coverage point factor F_c . For a chosen test suite size, F_s controls how many requirements the generated test suite will have, such that $|R(T)| = F_s \times |T|$. After setting the size of the test suite and requirement set, we use F_c to define the number of times the requirements are covered, denoted C , as a function of the total number of possible coverage points, so that $C = F_c \times (|R(T)| \times |T|)$.

Conclusion

- Synthetic test suite generation is efficient
- Enable the identification of fundamental trade-offs

Future Work

- Apply our technique to genetic and other algorithms
- Implement and evaluate new and different synthetic generators

<http://www.cs.allegheny.edu/~gkapfham/research/kanonizo/>

Conclusion and Future Work

Using Synthetic Test Suites to Empirically Compare Search-Based and Greedy Prioritizers

Zachary D. Williams
Department of Computer Science
Allegheny College
williaz@allegheny.edu

Gregory M. Kapfhammer
Department of Computer Science
Allegheny College
gkapfham@allegheny.edu

ABSTRACT

The increase in the complexity of modern software has led to the commensurate growth in the size and execution time of the test suites for these programs. In order to address this alarming trend, developers use test suite prioritization to render the test cases so that faults can be detected at an early stage of testing. Yet, the implementation and evaluation of greedy and search-based test prioritizers requires access to case study applications and their associated test suites, which are often difficult to find, configure, and use in an empirical study. This paper presents two types of synthetically generated test suites that support this process of experimentally evaluating prioritization methods. Using synthetic test suites affords greater control over test case characteristics and supports the identification of empirical trends that contradict the established wisdom about search-based and greedy prioritization. For instance, we find that the hill climbing algorithm often exhibits a lower time overhead than the greedy test suite prioritizer while producing test orderings with comparable effectiveness scores.

Categories and Subject Descriptors: D.2.2 [Software Engineering]: Testing and Debugging

General Terms: Experimentation, Performance

Keywords: search-based and greedy test prioritization

1. INTRODUCTION

Software developers often introduce defects during the implementation process. Inefficient testing methods mitigate confidence in the correctness of and isolate defects within a program by creating a collection of tests known as a test suite. Since execution testing can be very time consuming, testers use search-based and greedy prioritization techniques to produce a test ordering that will reveal faults earlier in the suite's execution than would otherwise be possible.

Suppose that a test suite $T = \{t_1, t_2, t_3, \dots, t_n\}$ covers the set of requirements $R(T) = \{r_1, r_2, r_3, \dots, r_m\}$. Each test case $t_i \in T$ is associated with the non-empty set $R(t_i) \subseteq R(T)$ [1, 5]. During the empirical study of search-based

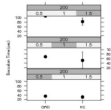


Figure 1: Execution Time - Fully Random.

and greedy test suite prioritizers, researchers often use the T and $R(T)$ associated with real-world case study applications. Yet, this practice can be difficult and time-consuming because of the need to tailor post-hoc test rendering tools for complex real-world programs. Furthermore, some small case-study applications may not be representative of all real-world programs, thus limiting empirical investigations of the efficiency and effectiveness of approaches to testing. The use of real-world programs also prohibits the experimenter from easily controlling the size of the test suite T and the coverage patterns within $R(T)$. Ultimately, the lack of a wide variety of test suites hinders the ability of researchers to quickly compare and contrast the plethora of newly developed techniques for test prioritization (e.g., [1, 3, 4, 5]).

Synthetic Test Suites. Using efficiently generated synthetic test suites to study search-based and greedy prioritizers enables experimenters to easily establish baseline results and control the key characteristics of the tests [2]. As such, this paper describes two simple methods for generating synthetic test suites and demonstrates how they reveal fundamental trade-offs in test prioritization techniques. We need two parameters to create the synthetic test suites: requirement factor F_r and coverage point factor F_c . For a chosen test suite size, F_c controls how many requirements the generated test suite will have, such that $|R(T)| = F_r \times |T|$. After setting the size of the test suite and requirement set, we use F_r to define the number of times the requirements are covered, denoted C , as a function of the total number of possible coverage points, so that $C = F_r \times |R(T)| = |T|$.

Conclusion

- Synthetic test suite generation is efficient
- Enable the identification of fundamental trade-offs

Future Work

- Apply our technique to genetic and other algorithms
- Implement and evaluate new and different synthetic generators

<http://www.cs.allegheny.edu/~gkapfham/research/kanonizo/>

Conclusion and Future Work

Using Synthetic Test Suites to Empirically Compare Search-Based and Greedy Prioritizers

Zachary D. Williams
Department of Computer Science
Allegheny College
williaz@allegheny.edu

Gregory M. Kapfhammer
Department of Computer Science
Allegheny College
gkapfham@allegheny.edu

ABSTRACT

The increase in the complexity of modern software has led to the corresponding growth in the size and execution time of the test suites for these programs. In order to address this alarming trend, developers use test suite prioritization to render the test cases so that faults can be detected at an early stage of testing. Yet, the implementation and evaluation of greedy and search-based test prioritizers requires access to case study applications and their associated test suites, which are often difficult to find, configure, and use in an empirical study. This paper presents two types of synthetically generated test suites that support this process of experimentally evaluating prioritization methods. Using synthetic test suites affords greater control over test case characteristics and supports the identification of empirical trends that contradict the established wisdom about search-based and greedy prioritization. For instance, we find that the hill climbing algorithm often exhibits a lower time overhead than the greedy test suite prioritization while producing test orderings with comparable effectiveness scores.

Categories and Subject Descriptors: D.2.2 [Software Engineering]: Testing and Debugging

General Terms: Experimentation, Performance

Keywords: search-based and greedy test prioritization

1. INTRODUCTION

Software developers often introduce defects during the implementation process. Inexpensive methods mitigate a confidence in the correctness of and isolate defects within a program by running a collection of tests known as a test suite. Since expensive testing can be very time consuming, testers use search-based and greedy prioritization techniques to produce a test ordering that will reveal faults earlier in the suite's execution than would otherwise be possible.

Suppose that a test suite $T = \{t_1, t_2, \dots, t_n\}$ covers the set of requirements $R(T) = \{r_1, r_2, \dots, r_m\}$. Each test case $t_i \in T$ is associated with the non-empty set $R(t_i) \subseteq R(T)$ [3, 9]. During the empirical study of search-based

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted by copyright owner, provided the copies are not made or distributed for profit or commercial advantage and the origin, title, author and the full citation are given on the copy. For other permission, contact ACM, 375 Broadway, New York, NY 10013, USA.
Copyright 2010 ACM 978-955-31-0202-0/10/0001...\$10.00

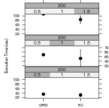


Figure 1: Execution Time - Fully Random.

and greedy test suite prioritizers, researchers often use the F and $R(F)$ associated with real-world case study applications. Yet, this practice can be difficult and time-consuming because of the need to tailor post-hoc test rendering tools for complex real-world programs. Furthermore, some small case-study applications may not be representative of all real-world programs, thus limiting empirical investigations of the efficiency and effectiveness of approaches to testing. The use of real-world programs also prohibits the experimenters from easily controlling the size of the test suite T and the coverage patterns within $R(T)$. Ultimately, the lack of a wide variety of test suites hinders the ability of researchers to quickly compare and contrast the plethora of newly developed techniques for test prioritization (e.g., [1, 3, 4, 9]).

Synthetic Test Suites. Using efficiently generated synthetic test suites to study search-based and greedy prioritizers enables experimenters to easily establish baseline results and control the key characteristics of the tests [2]. As such, this paper describes two simple methods for generating synthetic test suites and demonstrates how they reveal fundamental trade-offs in test prioritization techniques. We need two parameters to create the synthetic test suites: requirement factor F_s and coverage point factor C_s . For a chosen test suite size, F_s controls how many requirements the generated test suite will have, such that $|R(T)| = F_s \times |T|$. After setting the size of the test suite and requirement set, we use F_s to define the number of times the requirements are covered, denoted C_s , as a function of the total number of possible coverage points, so that $C_s = F_s \times |R(T)| = |T|$.

Conclusion

- Synthetic test suite generation is efficient
- Enable the identification of fundamental trade-offs

Future Work

- Apply our technique to genetic and other algorithms
- Implement and evaluate new and different synthetic generators

<http://www.cs.allegheny.edu/~gkapfham/research/kanonizo/>

Conclusion and Future Work

Using Synthetic Test Suites to Empirically Compare Search-Based and Greedy Prioritizers

Zachary D. Williams
Department of Computer Science
Allegheny College
williaz@allegheny.edu

Gregory M. Kapfhammer
Department of Computer Science
Allegheny College
gkapfham@allegheny.edu

ABSTRACT

The increase in the complexity of modern software has led to the corresponding growth in the size and execution time of the test suites for these programs. In order to address this alarming trend, developers use test suite prioritization to render the test cases so that faults can be detected at an early stage of testing. Yet, the implementation and evaluation of greedy and search-based test prioritizers requires access to one study application and their associated test suites, which are often difficult to find, configure, and use in an empirical study. This paper presents two types of synthetically generated test suites that support this process of experimentally evaluating prioritization methods. Using synthetic test suites affords greater control over test case characteristics and supports the identification of empirical trends that contradict the established wisdom about search-based and greedy prioritization. For instance, we find that the hill climbing algorithm often exhibits a lower time overhead than the greedy test suite prioritizer while producing test orderings with comparable effectiveness scores.

Categories and Subject Descriptors: D.2.2 [Software Engineering]: Testing and Debugging

General Terms: Experimentation, Performance

Keywords: search-based and greedy test prioritization

1. INTRODUCTION

Software developers often introduce defects during the implementation process. Inefficient test methods mitigate confidence in the correctness of and isolate defects within a program by creating a collection of tests known as a test suite. Since regression testing can be very time-consuming, testers use search-based and greedy prioritization techniques to produce a test ordering that will reveal faults earlier in the suite's execution than would otherwise be possible.

Suppose that a test suite $T = \{t_1, t_2, t_3, \dots, t_n\}$ covers the set of requirements $R(T) = \{R_1, R_2, R_3, \dots, R_n\}$. Each test case $t_i \in T$ is associated with the non-empty set $R(t_i) \subseteq R(T)$ [3, 5]. During the empirical study of search-based

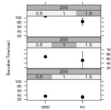


Figure 1: Execution Time - Fully Random.

and greedy test suite prioritizers, researchers often use the F and $R(F)$ associated with real-world case study applications. Yet, this practice can be difficult and time-consuming because of the need to tailor post-hoc test reordering tools for complex real-world programs. Furthermore, some small case study applications may not be representative of all real-world programs, thus limiting empirical investigations of the efficiency and effectiveness of approaches to testing. The use of real-world programs also prohibits the experimenters from easily controlling the size of the test suite T and the coverage patterns within $R(T)$. Ultimately, the lack of a wide variety of test suites hinders the ability of researchers to quickly compare and contrast the abilities of newly developed techniques for test prioritization (e.g., [3, 4, 5]).

Synthetic Test Suites. Using efficiently generated synthetic test suites to study search-based and greedy prioritizers enables experimenters to easily establish baseline results and control the key characteristics of the tests [2]. As such, this paper describes two simple methods for generating synthetic test suites and demonstrates how they reveal fundamental trade-offs in test prioritization techniques. We need two parameters to create the synthetic test suites: requirement factor F_r and coverage point factor F_c . For a chosen test suite size, F_c controls how many requirements the generated test suite will have, such that $|R(T)| = F_r \times |F|$. After setting the size of the test suite and requirement set, we use F_r to define the number of times the requirements are covered, denoted C_r , as a function of the total number of possible coverage points, so that $C_r = |R(T)| \times |F|$.

Conclusion

- Synthetic test suite generation is efficient
- Enable the identification of fundamental trade-offs

Future Work

- Apply our technique to genetic and other algorithms
- Implement and evaluate new and different synthetic generators

<http://www.cs.allegheny.edu/~gkapfham/research/kanonizo/>