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

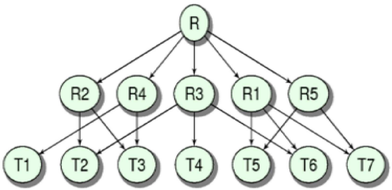
Important Contributions

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

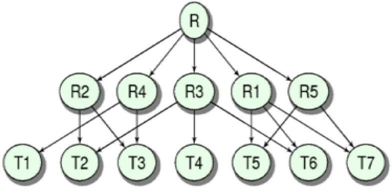


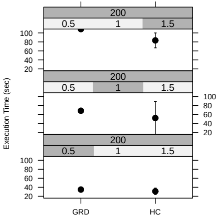
Synthetic Test Suites

Detailed Empirical Study

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

Important Contributions



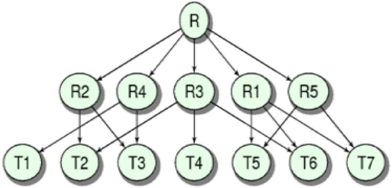


Synthetic Test Suites

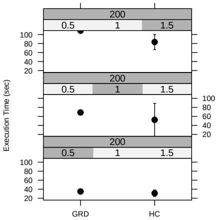
Detailed Empirical Study

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

Important Contributions



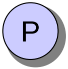
Synthetic Test Suites



Detailed Empirical Study

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

Overview of Regression Testing

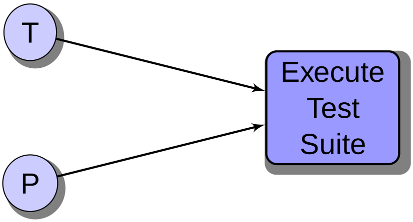


Correct programing defect

Overview of Regression Testing

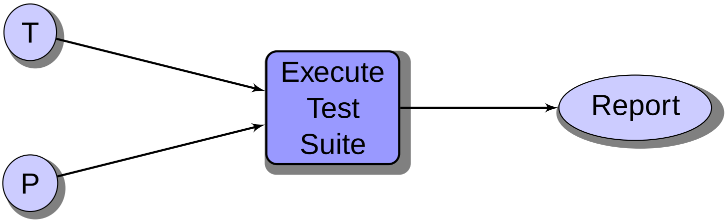
Correct programing defect

Overview of Regression Testing



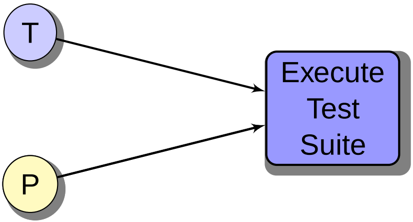
Correct programing defect

Overview of Regression Testing



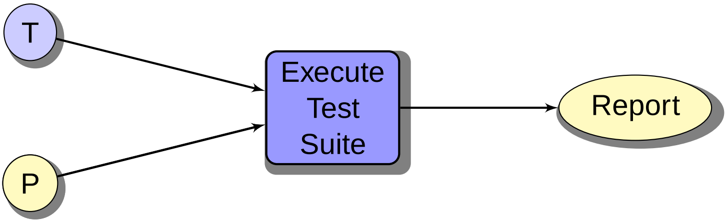
Correct programing defect

Overview of Regression Testing



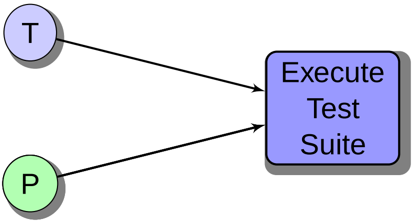
Correct programing defect

Overview of Regression Testing



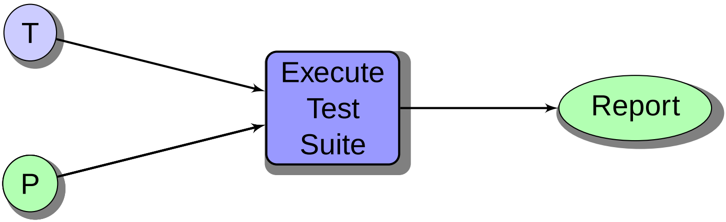
Correct programing defect

Overview of Regression Testing



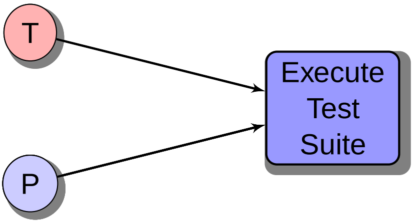
Add new functionallity

Overview of Regression Testing



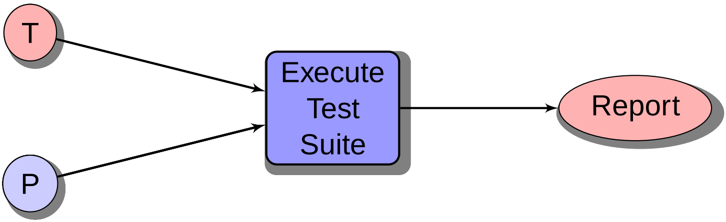
Add new functionallity

Overview of Regression Testing



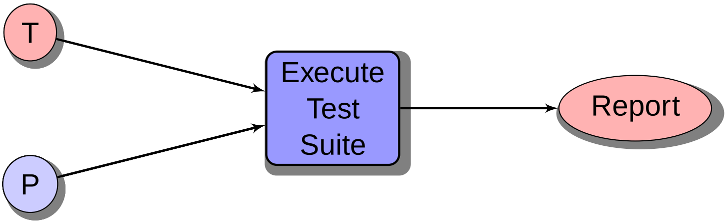
Modify test suite

Overview of Regression Testing



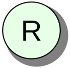
Modify test suite

Overview of Regression Testing



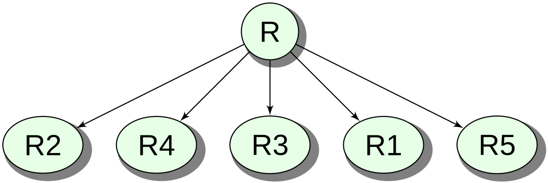
Complete retesting is often prohibitively expensive

Regression Test Suite Prioritization



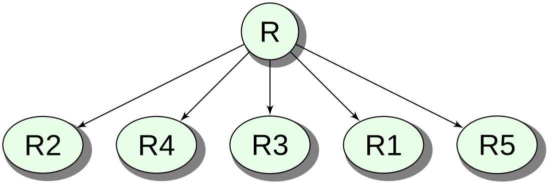
Requirements necessitate the coverage of the **state** and/or **structure** of a program under test

Regression Test Suite Prioritization



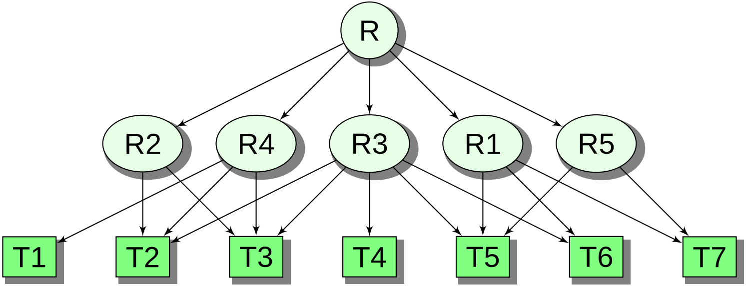
Requirements necessitate the coverage of the **state** and/or **structure** of a program under test

Regression Test Suite Prioritization



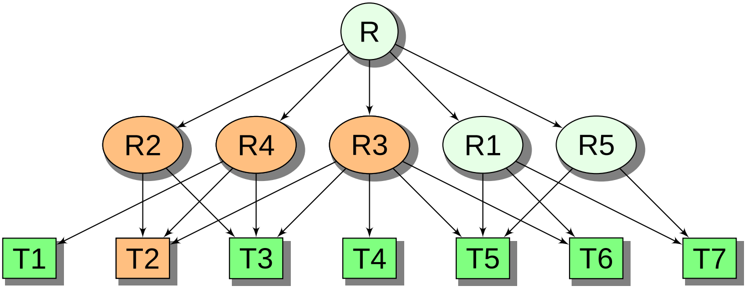
Requirements necessitate the coverage of the **state** and/or **structure** of a program under test

Regression Test Suite Prioritization



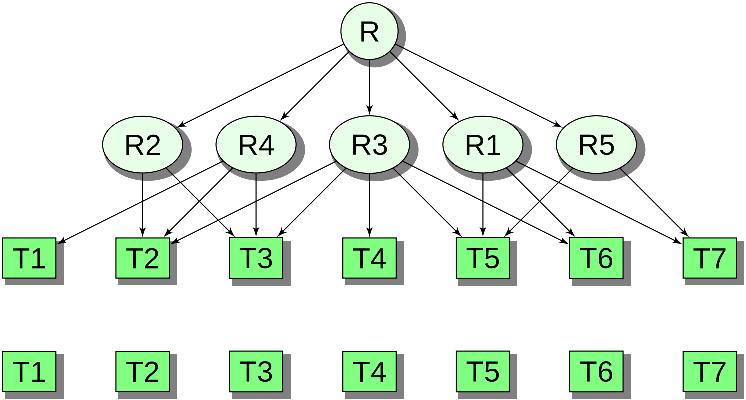
Each test covers specific **requirements** in a certain amount of **time** and thus the **ordering** is critical

Regression Test Suite Prioritization



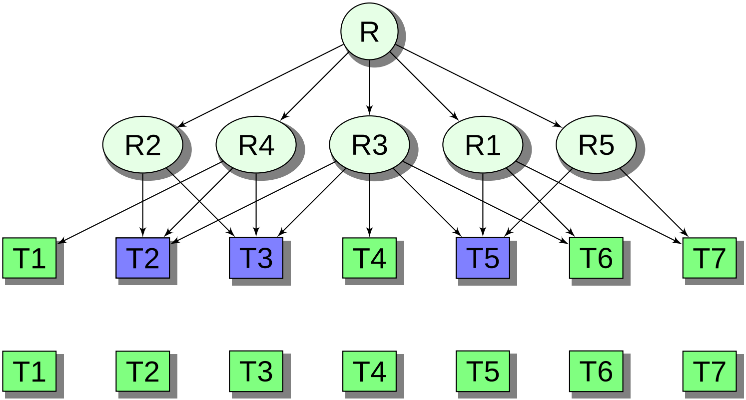
Each test covers specific **requirements** in a certain amount of **time** and thus the **ordering** is critical

Regression Test Suite Prioritization



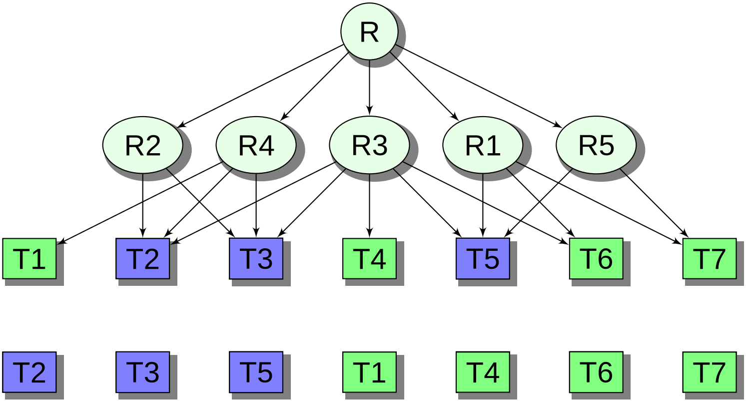
Prioritized test suites cover requirements **faster** thus enabling the **rapid** detection of defects

Regression Test Suite Prioritization



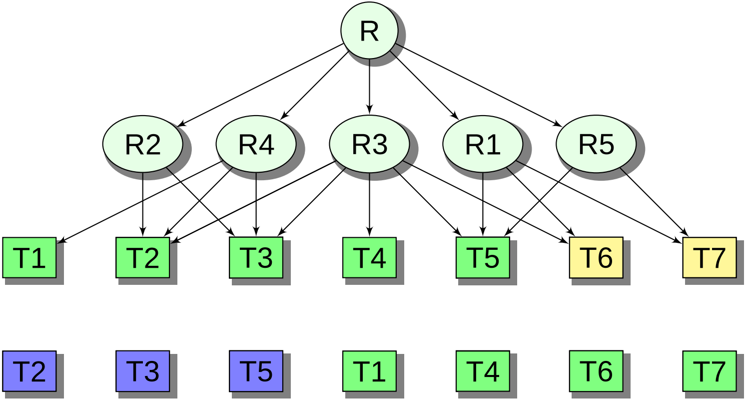
Prioritized test suites cover requirements **faster** thus enabling the **rapid** detection of defects

Regression Test Suite Prioritization



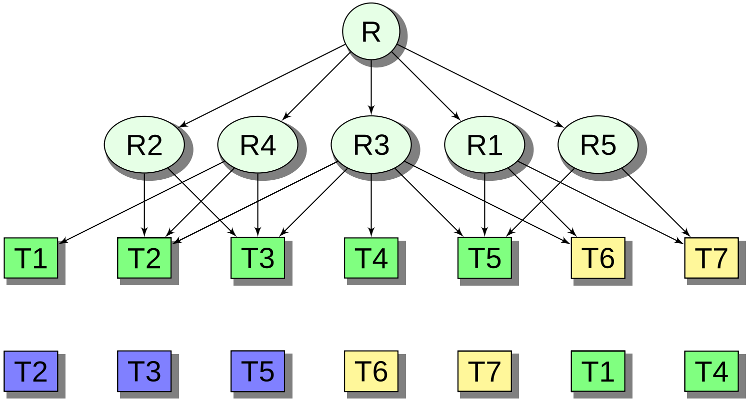
Prioritized test suites cover requirements **faster** thus enabling the **rapid** detection of defects

Regression Test Suite Prioritization



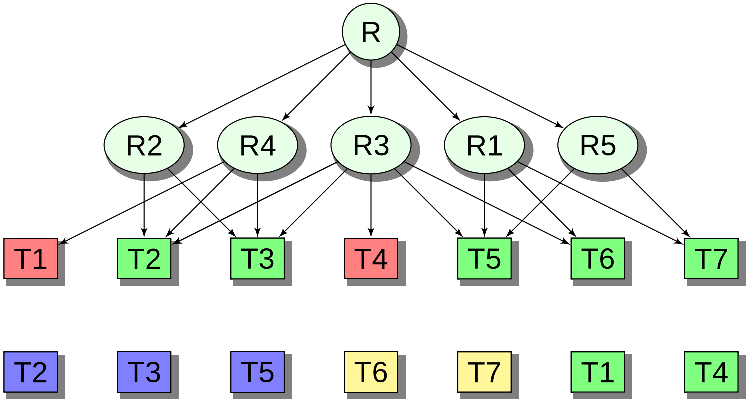
Prioritized test suites cover requirements **faster** thus enabling the **rapid** detection of defects

Regression Test Suite Prioritization



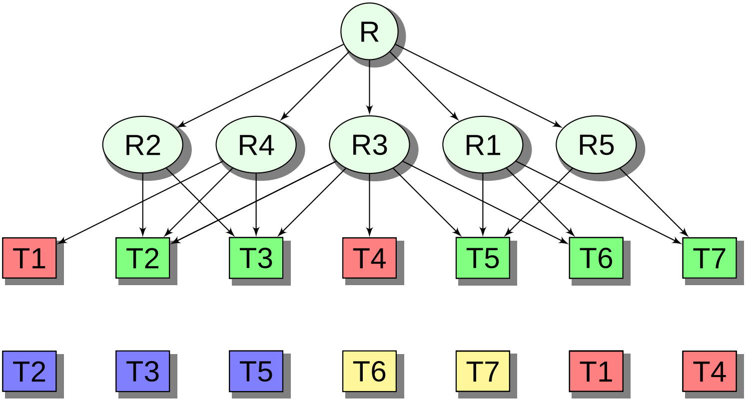
Prioritized test suites cover requirements **faster** thus enabling the **rapid** detection of defects

Regression Test Suite Prioritization



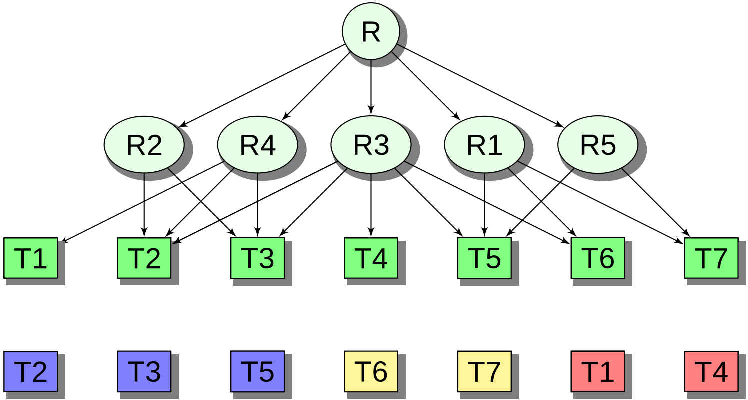
Prioritized test suites cover requirements **faster** thus enabling the **rapid** detection of defects

Regression Test Suite Prioritization



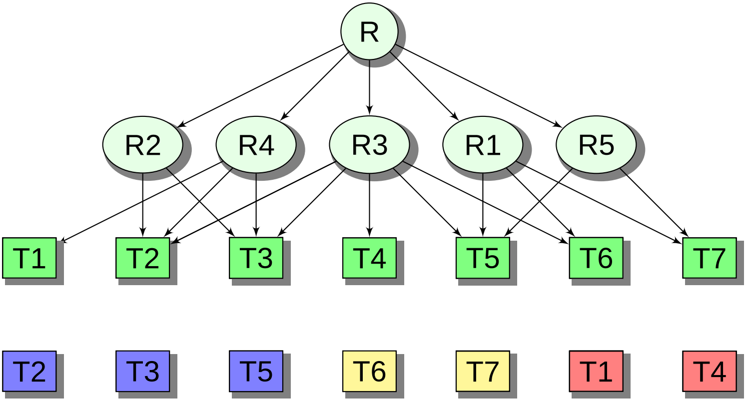
Prioritized test suites cover requirements **faster** thus enabling the **rapid** detection of defects

Regression Test Suite Prioritization



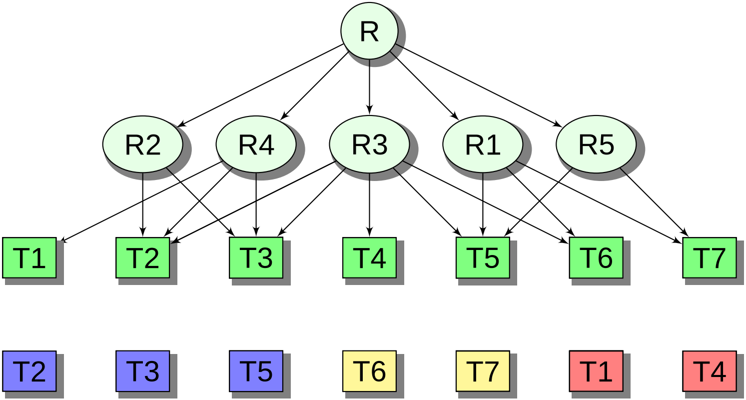
Prioritized test suites cover requirements **faster** thus enabling the **rapid** detection of defects

Regression Test Suite Prioritization



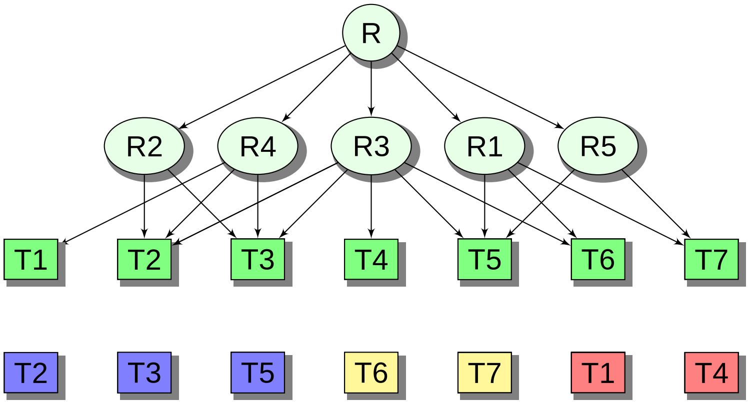
Prioritized test suites cover requirements **faster** thus enabling the **rapid** detection of defects

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?

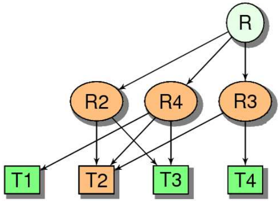
Existing Prioritization Techniques

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

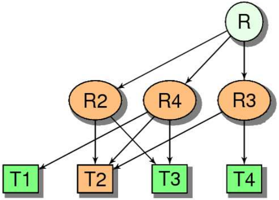


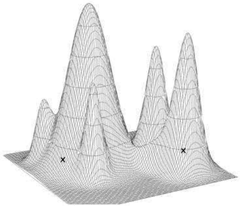
Greedy approaches select the next best test case

Hill climbers search the state space for improved orderings

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

Existing Prioritization Techniques





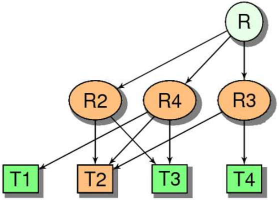
http://artedi.ebc.uu.se/course/Embo01/Phylogeny/phylogeny_readme.html

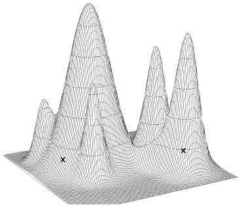
Greedy approaches select the next best test case

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 out performs hill climbing in terms of both efficiency and effectiveness

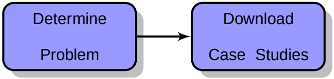
Conducting an Empirical Evaluation

Determine

Problem

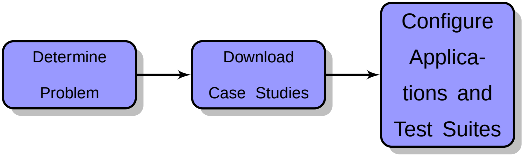
These highlighted tasks are **manual**, **expensive**, and **prone to error**

Conducting an Empirical Evaluation



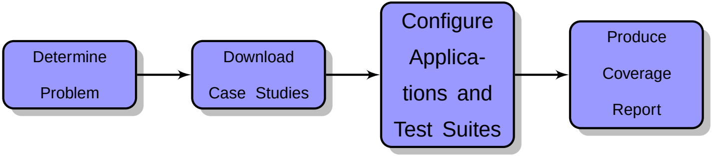
These highlighted tasks are **manual**, **expensive**, and **prone to error**

Conducting an Empirical Evaluation



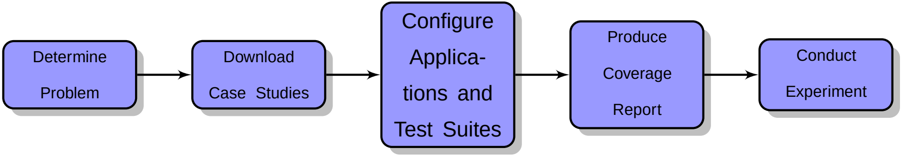
These highlighted tasks are **manual**, **expensive**, and **prone to error**

Conducting an Empirical Evaluation



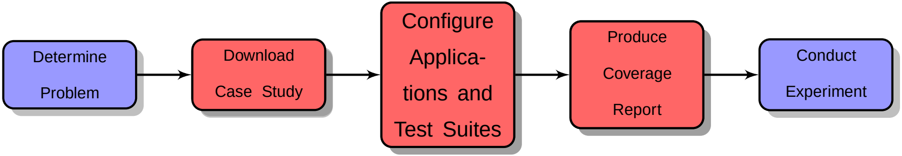
These highlighted tasks are **manual**, **expensive**, and **prone to error**

Conducting an Empirical Evaluation



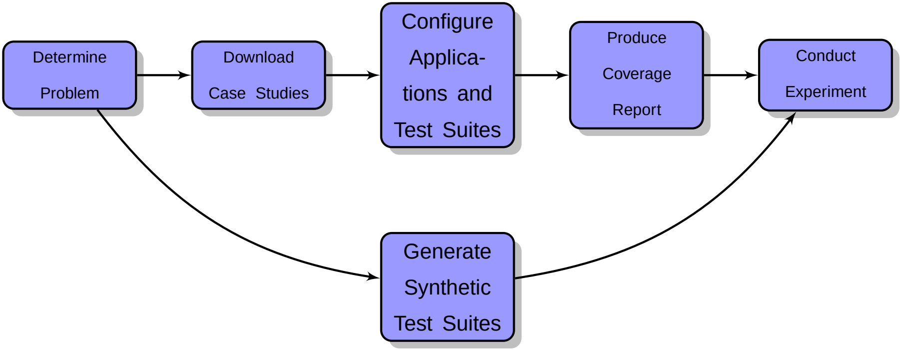
These highlighted tasks are **manual**, **expensive**, and **prone to error**

Conducting an Empirical Evaluation



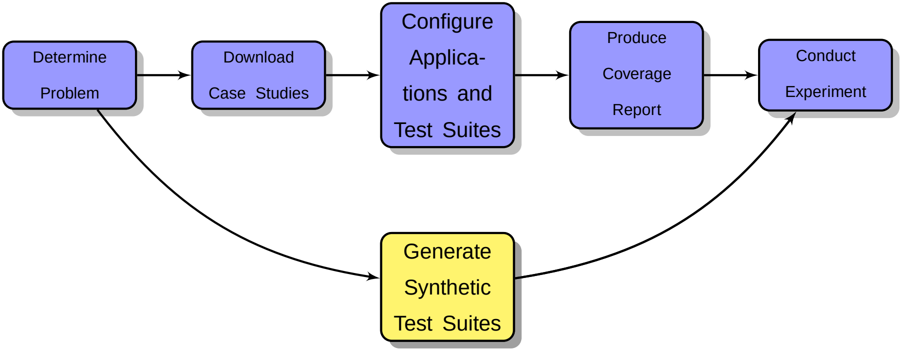
These highlighted tasks are **manual**, **expensive**, and **prone to error**

Conducting an Empirical Evaluation



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

Conducting an Empirical Evaluation

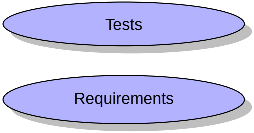


Synthetically generating a test suite is **automated**, **effective**, and **efficient**

Generating Synthetic Test Suite

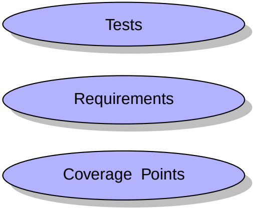
The **total number of tests** controls how many tests the suite will contain

Generating Synthetic Test Suite



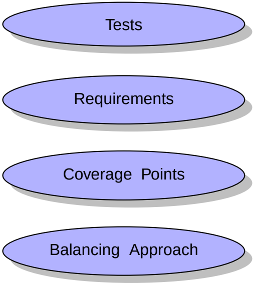
The **total number of requirements** governs how many requirements the test suite will cover

Generating Synthetic Test Suite



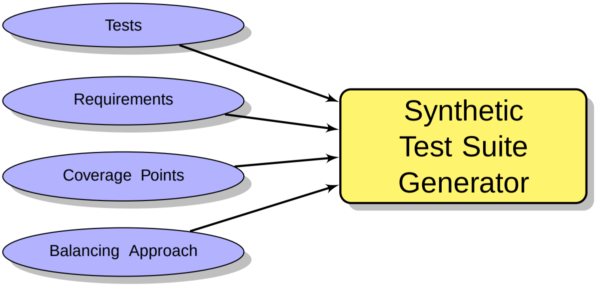
The **total number of coverage points** controls how many unique test-requirement pairs the test suite will contain

Generating Synthetic Test Suite



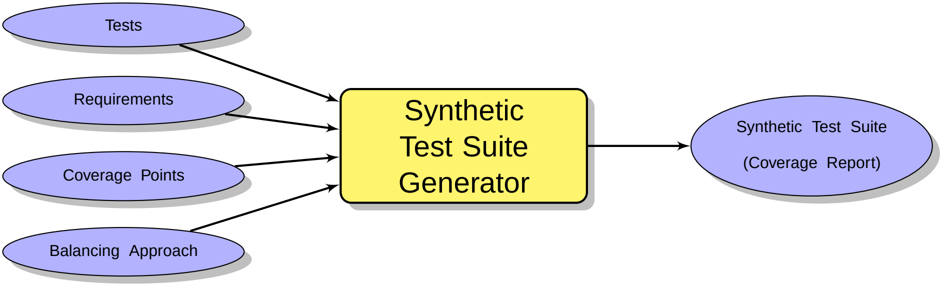
The **balancing configuration** dictates how the coverage points will be distributed in the synthetic test suite

Generating Synthetic Test Suite



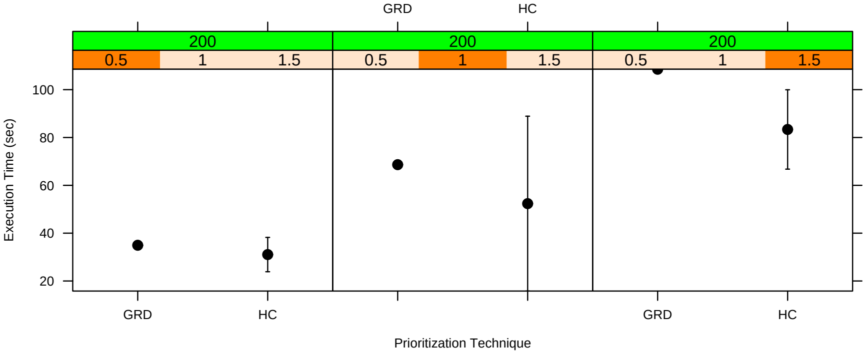
Our empirical results show that synthetic generation takes less then **0.2 seconds** for extremely large test suites

Generating Synthetic Test Suite



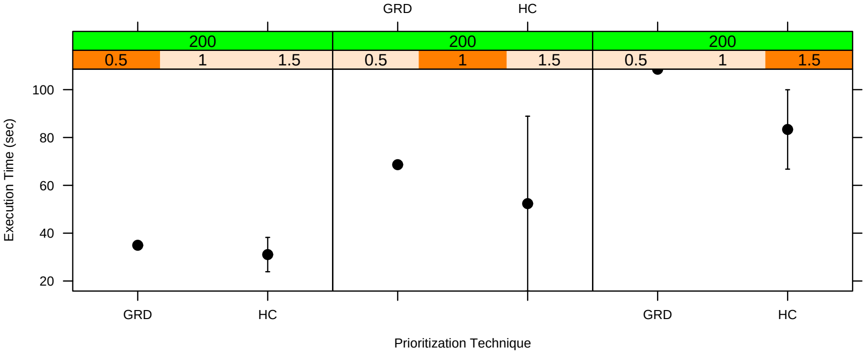
Contains information concerning the **requirements covered** and the **execution time** of each test

Empirical Results – Prioritizer Efficiency



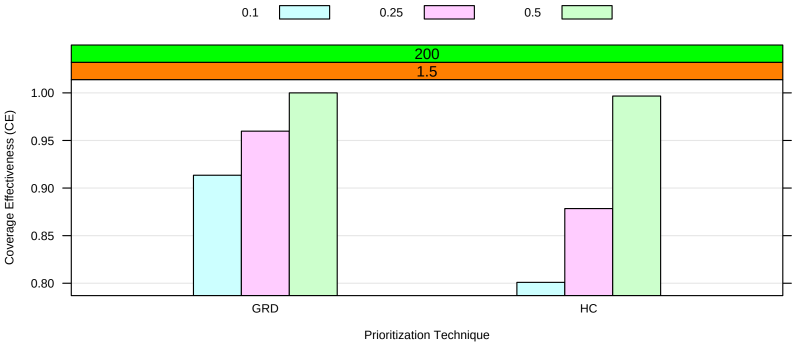
HC demonstrated to be more efficient then GRD for large test suites

Empirical Results – Prioritizer Efficiency



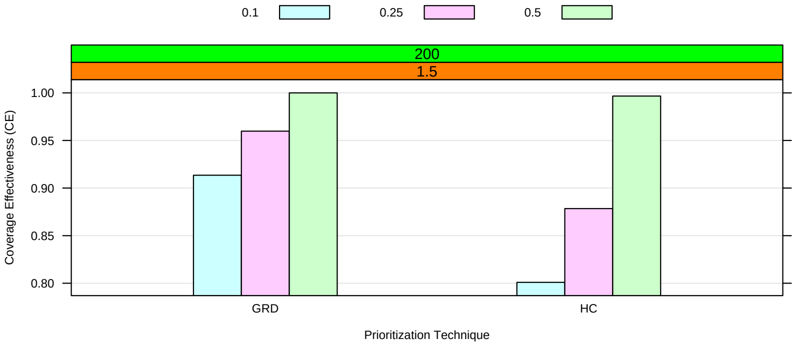
HC demonstrated to be more efficient then GRD for large test suites

Empirical Results – Prioritizer Effectiveness



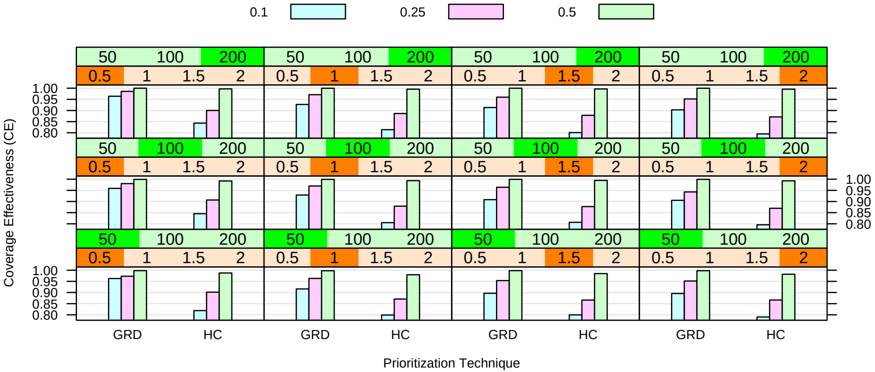
As the amount of coverage points in the test suite increases the performance of HC becomes comparable to that of GRD

Empirical Results – Prioritizer Effectiveness



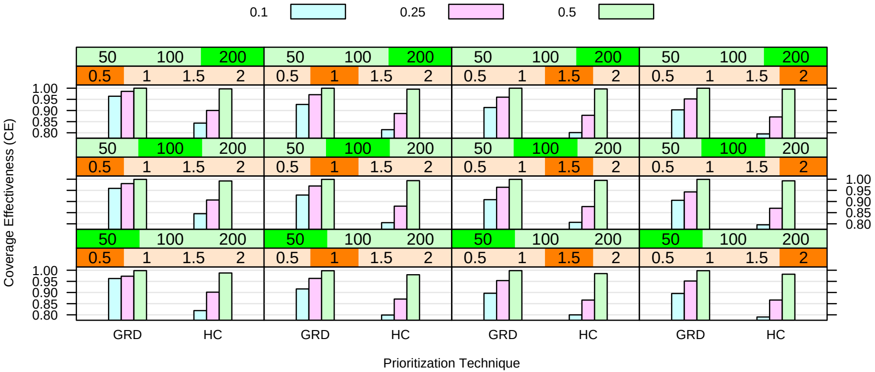
As the amount of coverage points in the test suite increases the performance of HC becomes comparable to that of GRD

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 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 reorder 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.2.5 [**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. Regression testing methods establish 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 $T = \langle t_1, t_2, t_3, \ldots, t_n \rangle$ covers the set of requirements $\mathcal{R}(T) = \{r_1, r_2, r_3, \ldots, r_m\}$. Each test case $t_i \in T$ is associated with the non-empty set $\mathcal{R}(t_i) \subseteq \mathcal{R}(T)$ [1, 5]. During the empirical study of search-based

GECCO'10, July 7–11, 2010, Portland, Oregon, USA.

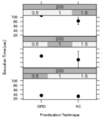


Figure 1: Execution Time - Fully Random.

and greedy test suite prioritizers, researchers often use the T and $\mathcal{R}(T)$ associated with real-world case study applications. Yet, this practice can be difficult and time consuming because of the need to tailor prototype test reordering tools for complex real-world programs. Furthermore, some small case study applications may not be representative of all real-world programs, thus hindering 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 $\mathcal{R}(T)$. 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 used 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_r controls how many requirements the generated test suite will have, such that $|\mathcal{R}(T)| = F_r \times |T|$. 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 fraction of the total number of possible coverage points, so that $C = F_c \times |\mathcal{R}(T)| \times |T|$.

Conclusion

- Synthetic test suite generation is efficient
- Enable the identification of fundemental 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 reorder 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.2.5 [**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. Regression testing methods establish 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 $T = \langle t_1, t_2, t_3, \ldots, t_n \rangle$ covers the set of requirements $\mathcal{R}(T) = \{r_1, r_2, r_3, \ldots, r_m\}$. Each test case $t_i \in T$ is associated with the non-empty set $\mathcal{R}(t_i) \subseteq \mathcal{R}(T)$ [1, 5]. During the empirical study of search-based and greedy test suite prioritizers, researchers often use the T and $\mathcal{R}(T)$ associated with real-world case study applications. Yet, this practice can be difficult and time consuming because of the need to tailor prototype test reordering tools for complex real-world programs. Furthermore, some small case study applications may not be representative of all real-world programs, thus hindering 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 $\mathcal{R}(T)$. 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 used 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_r controls how many requirements the generated test suite will have, such that $|\mathcal{R}(T)| = F_r \times |T|$. 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 fraction of the total number of possible coverage points, so that $C = F_c \times |\mathcal{R}(T)| \times |T|$.

Figure 1: Execution Time - Fully Random.

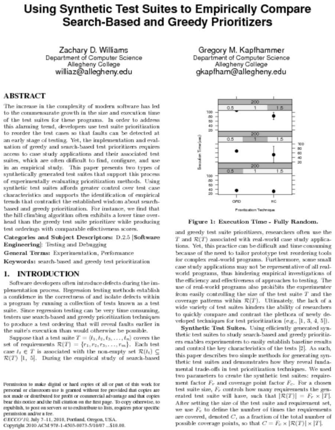
Conclusion

- Synthetic test suite generation is efficient
- Enable the identification of fundemental 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/



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 reorder 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.2.5 [**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. Regression testing methods establish 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 $T = \langle t_1, t_2, t_3, \ldots, t_n \rangle$ covers the set of requirements $\mathcal{R}(T) = \{r_1, r_2, r_3, \ldots, r_m\}$. Each test case $t_i \in T$ is associated with the non-empty set $\mathcal{R}(t_i) \subseteq \mathcal{R}(T)$ [1, 5]. During the empirical study of search-based and greedy test suite prioritizers, researchers often use the T and $\mathcal{R}(T)$ associated with real-world case study applications. Yet, this practice can be difficult and time consuming because of the need to tailor prototype test reordering tools for complex real-world programs. Furthermore, some small case study applications may not be representative of all real-world programs, thus hindering 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 $\mathcal{R}(T)$. 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 used 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_r controls how many requirements the generated test suite will have, such that $|\mathcal{R}(T)| = F_r \times |T|$. 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 fraction of the total number of possible coverage points, so that $C = F_c \times |\mathcal{R}(T)| \times |T|$.

Figure 1: Execution Time - Fully Random.

Conclusion

- Synthetic test suite generation is efficient
- Enable the identification of fundemental 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 reorder 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.2.5 [**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. Regression testing methods establish 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 $T = \langle t_1, t_2, t_3, \ldots, t_n \rangle$ covers the set of requirements $\mathcal{R}(T) = \{r_1, r_2, r_3, \ldots, r_m\}$. Each test case $t_i \in T$ is associated with the non-empty set $\mathcal{R}(t_i) \subseteq \mathcal{R}(T)$ [1, 5]. During the empirical study of search-based and greedy test suite prioritizers, researchers often use the T and $\mathcal{R}(T)$ associated with real-world case study applications. Yet, this practice can be difficult and time consuming because of the need to tailor prototype test reordering tools for complex real-world programs. Furthermore, some small case study applications may not be representative of all real-world programs, thus hindering 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 $\mathcal{R}(T)$. 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 used 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_r controls how many requirements the generated test suite will have, such that $|\mathcal{R}(T)| = F_r \times |T|$. 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 fraction of the total number of possible coverage points, so that $C = F_c \times |\mathcal{R}(T)| \times |T|$.

Figure 1: Execution Time - Fully Random.

GECCO'10, July 7–11, 2010, Portland, Oregon, USA.

Conclusion

- Synthetic test suite generation is efficient
- Enable the identification of fundemental 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 reorder 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.2.5 [**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. Regression testing methods establish 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 $T = \langle t_1, t_2, t_3, \ldots, t_n \rangle$ covers the set of requirements $\mathcal{R}(T) = \{r_1, r_2, r_3, \ldots, r_m\}$. Each test case $t_i \in T$ is associated with the non-empty set $\mathcal{R}(t_i) \subseteq \mathcal{R}(T)$ [1, 5]. During the empirical study of search-based

GECCO'10, July 7–11, 2010, Portland, Oregon, USA.

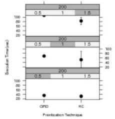


Figure 1: Execution Time - Fully Random.

and greedy test suite prioritizers, researchers often use the T and $\mathcal{R}(T)$ associated with real-world case study applications. Yet, this practice can be difficult and time consuming because of the need to tailor prototype test reordering tools for complex real-world programs. Furthermore, some small case study applications may not be representative of all real-world programs, thus hindering 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 $\mathcal{R}(T)$. 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 used 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_r controls how many requirements the generated test suite will have, such that $|\mathcal{R}(T)| = F_r \times |T|$. 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 fraction of the total number of possible coverage points, so that $C = F_c \times |\mathcal{R}(T)| \times |T|$.

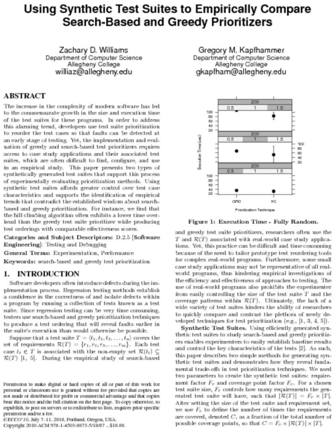
Conclusion

- Synthetic test suite generation is efficient
- Enable the identification of fundemental 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/



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 reorder 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.2.5 [**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. Regression testing methods establish 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 $T = \langle t_1, t_2, t_3, \ldots, t_n \rangle$ covers the set of requirements $\mathcal{R}(T) = \{r_1, r_2, r_3, \ldots, r_m\}$. Each test case $t_i \in T$ is associated with the non-empty set $\mathcal{R}(t_i) \subseteq \mathcal{R}(T)$ [1, 5]. During the empirical study of search-based and greedy test suite prioritizers, researchers often use the T and $\mathcal{R}(T)$ associated with real-world case study applications. Yet, this practice can be difficult and time consuming because of the need to tailor prototype test reordering tools for complex real-world programs. Furthermore, some small case study applications may not be representative of all real-world programs, thus hindering 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 $\mathcal{R}(T)$. 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 used 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_r controls how many requirements the generated test suite will have, such that $|\mathcal{R}(T)| = F_r \times |T|$. 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 fraction of the total number of possible coverage points, so that $C = F_c \times |\mathcal{R}(T)| \times |T|$.

Figure 1: Execution Time - Fully Random.

Conclusion

- Synthetic test suite generation is efficient
- Enable the identification of fundemental 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 reorder 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.2.5 [**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. Regression testing methods establish 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 $T = \langle t_1, t_2, t_3, \ldots, t_n \rangle$ covers the set of requirements $\mathcal{R}(T) = \{r_1, r_2, r_3, \ldots, r_m\}$. Each test case $t_i \in T$ is associated with the non-empty set $\mathcal{R}(t_i) \subseteq \mathcal{R}(T)$ [1, 5]. During the empirical study of search-based

GECCO'10, July 7–11, 2010, Portland, Oregon, USA.

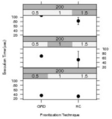


Figure 1: Execution Time - Fully Random.

and greedy test suite prioritizers, researchers often use the T and $\mathcal{R}(T)$ associated with real-world case study applications. Yet, this practice can be difficult and time consuming because of the need to tailor prototype test reordering tools for complex real-world programs. Furthermore, some small case study applications may not be representative of all real-world programs, thus hindering 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 $\mathcal{R}(T)$. 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 used 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_r controls how many requirements the generated test suite will have, such that $|\mathcal{R}(T)| = F_r \times |T|$. 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 fraction of the total number of possible coverage points, so that $C = F_c \times |\mathcal{R}(T)| \times |T|$.

Conclusion

- Synthetic test suite generation is efficient
- Enable the identification of fundemental 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/