Empirically Studying the Role of Selection Operators During Search-Based Test Suite Prioritization

Alexander P. Conrad[‡], Robert S. Roos[†], and Gregory M. Kapfhammer[†]

[‡] Department of Computer Science
University of Pittsburgh

[†] Department of Computer Science
Allegheny College

Genetic and Evolutionary Computation Conference
Search-Based Software Engineering Track
July 2010

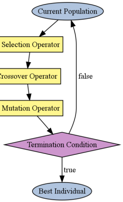
Important Contributions

Search-Based Prioritizers

Empirical Results

Genetic algorithm-based test prioritizer that uses many mutation, crossover, selection, and transformation operators

Important Contributions

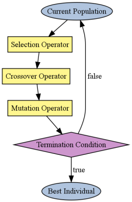


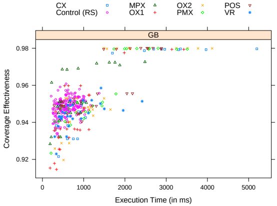
Search-Based Prioritizers

Empirical Results

Genetic algorithm-based test prioritizer that uses many mutation, crossover, selection, and transformation operators

Important Contributions



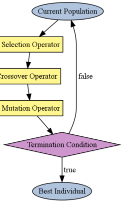


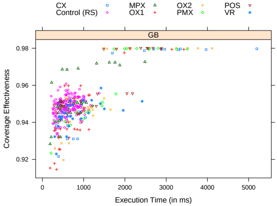
Search-Based Prioritizers

Empirical Results

Genetic algorithm-based test prioritizer that uses many mutation, crossover, selection, and transformation operators

Important Contributions



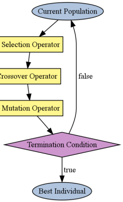


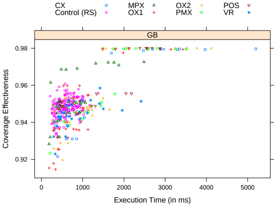
Search-Based Prioritizers

Empirical Results

Genetic algorithm-based test prioritizer that uses many mutation, crossover, selection, and transformation operators

Important Contributions



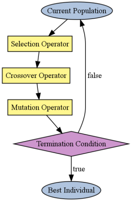


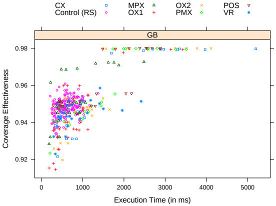
Search-Based Prioritizers

Empirical Results

Automatically constructed **tree models** highlight the unique role that the **selection** operator plays during prioritization

Important Contributions



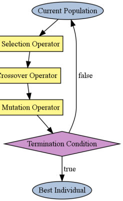


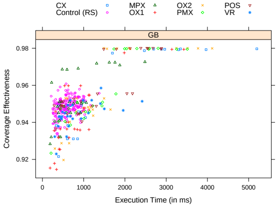
Search-Based Prioritizers

Empirical Results

Genetic algorithm is superior to **random search** and **hill climbing** and often suitable for many testing environments

Important Contributions



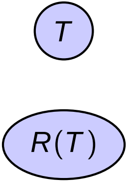


Search-Based Prioritizers

Empirical Results

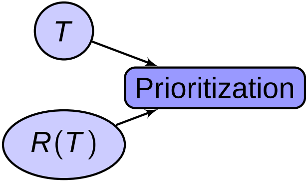
Complete **genetic algorithm**-based prioritization framework is available from **http://gelations.googlecode.com/**

Process of Regression Testing



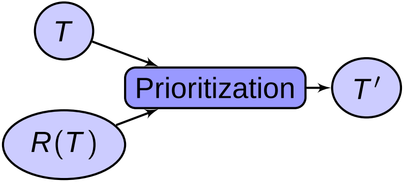
Reorder the test suite in order to **improve** effectiveness

Process of Regression Testing



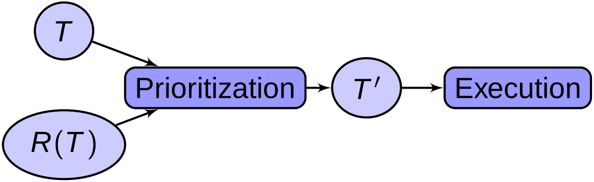
Reorder the test suite in order to **improve** effectiveness

Process of Regression Testing



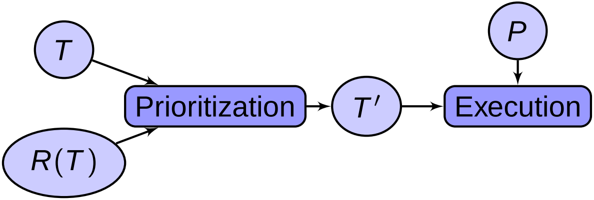
Reorder the test suite in order to **improve** effectiveness

Process of Regression Testing



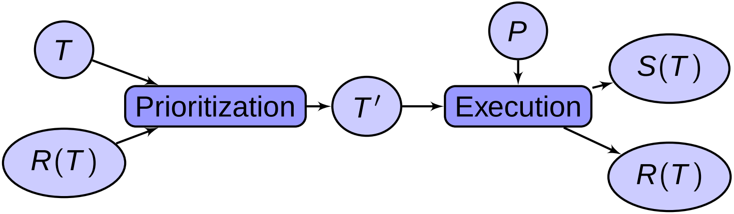
Reorder the test suite in order to **improve** effectiveness

Process of Regression Testing



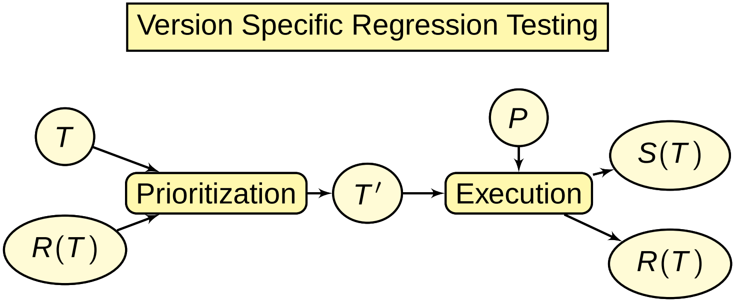
Reorder the test suite in order to **improve** effectiveness

Process of Regression Testing



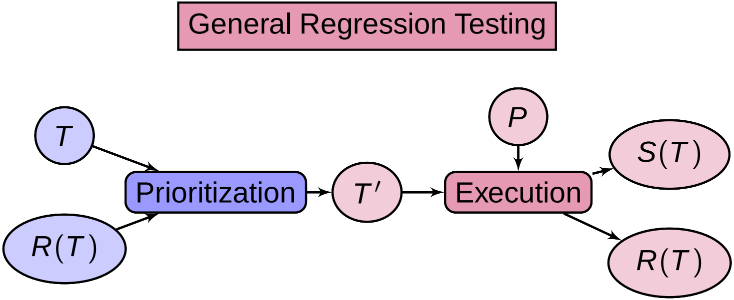
Reorder the test suite in order to **improve** effectiveness

Process of Regression Testing



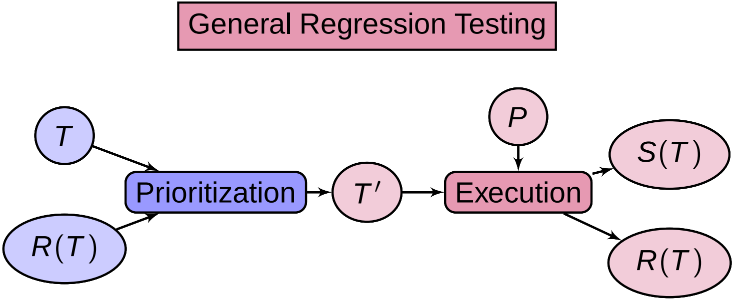
Re-prioritize each time the suite or program **changes**

Process of Regression Testing



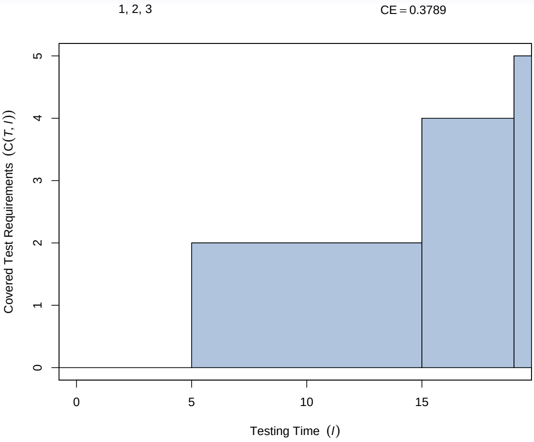
Use the **same** suite for **multiple** rounds of test execution

Process of Regression Testing



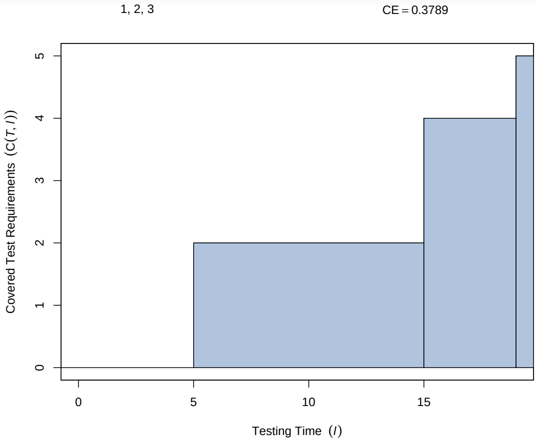
Do et al. "the worst thing that JUnit users can do is not practice some form of prioritization" (ISSRE 2004)

Importance of Test Suite Prioritization



Prioritize to **increase** the CE of a test suite $CE = \frac{\text{Actual}}{\text{Ideal}} \in [0, 1]$

Importance of Test Suite Prioritization

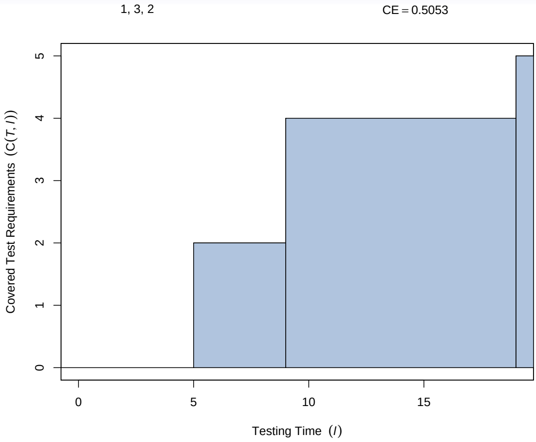


Test Orderings

1,2,3

Original ordering exhibits poor effectiveness score - **CE = 0.3789**

Importance of Test Suite Prioritization



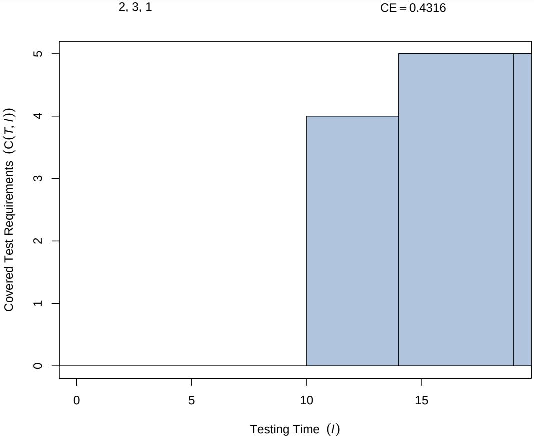
Test Orderings

1,2,3

1,3,2

Different ordering improves the effectiveness score - **CE = 0.5053**

Importance of Test Suite Prioritization



Test Orderings

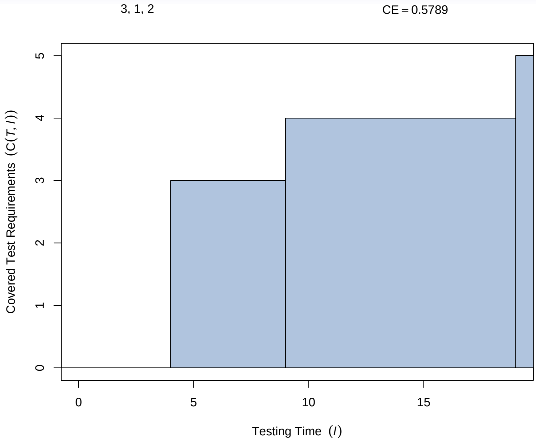
1,2,3

1,3,2

2,3,1

Some orderings have less improved scores - **CE = 0.4316**

Importance of Test Suite Prioritization



Test Orderings

1,2,3

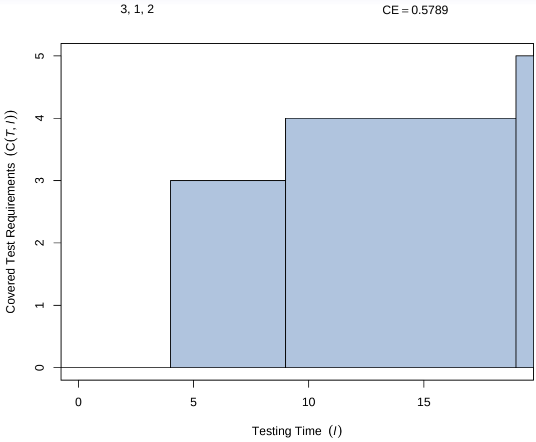
1,3,2

2,3,1

3,1,2

Best ordering shows a higher effectiveness scores - **CE = 0.5789**

Importance of Test Suite Prioritization



Test Orderings

1,2,3

1,3,2

2,3,1

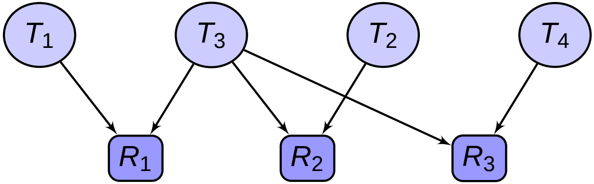
3,1,2

Greedy methods often produce high effectiveness orderings

Limitations of Greedy Methods

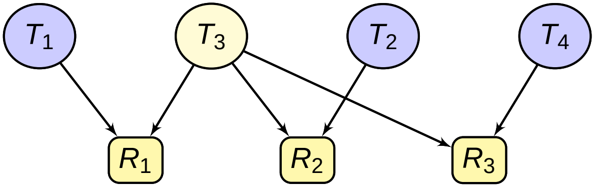
Possible configuration of the **coverage report**

Limitations of Greedy Methods



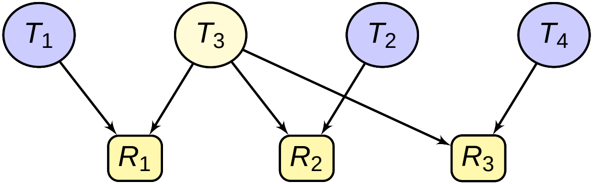
Possible configuration of the **coverage report**

Limitations of Greedy Methods



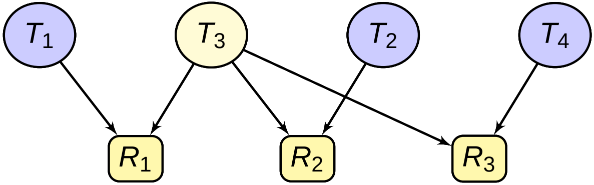
Possible configuration of the **coverage report**

Limitations of Greedy Methods



Possible configuration of the **coverage report**

Limitations of Greedy Methods



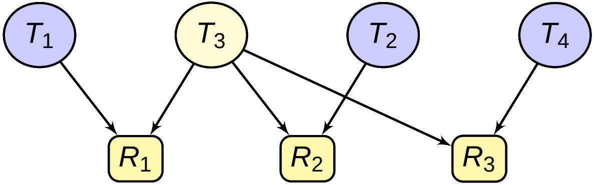
$time(T_1) = 1$

$time(T_2) = 1$

$time(T_3) = 1$

Execution time of the test cases may mislead greedy

Limitations of Greedy Methods



$time(T_1) = 1$

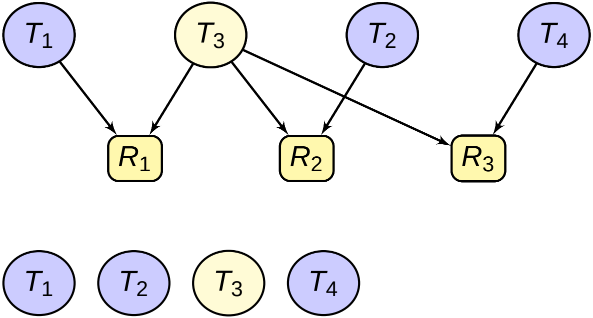
$time(T_2) = 1$

$time(T_3) = 1$

$time(T_4) = 2.45$

Execution time of the test cases may mislead greedy

Limitations of Greedy Methods



$time(T_1) = 1$

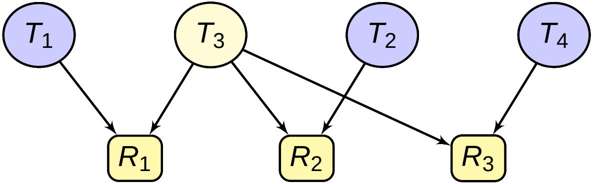
$time(T_2) = 1$

$time(T_3) = 1$

$time(T_4) = 2.45$

Original ordering has low effectiveness score

Limitations of Greedy Methods



$time(T_1) = 1$

$time(T_2) = 1$

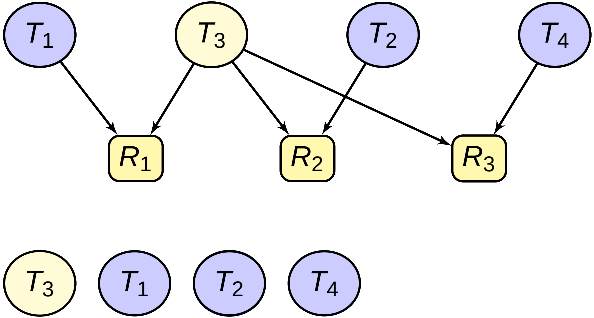
$time(T_3) = 1$

$time(T_4) = 2.45$

T_1 T_2 T_3 T_4 $CE(T) = 0.54$

Original ordering has low effectiveness score

Limitations of Greedy Methods



$time(T_1) = 1$

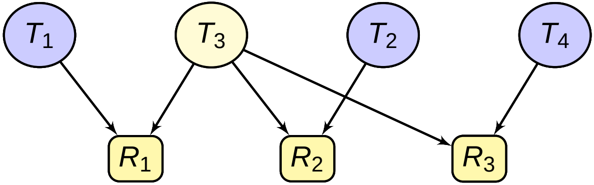
$time(T_2) = 1$

$time(T_3) = 1$

$time(T_4) = 2.45$

Greedy method constructs suite with marginal improvement

Limitations of Greedy Methods



$time(T_1) = 1$

$time(T_2) = 1$

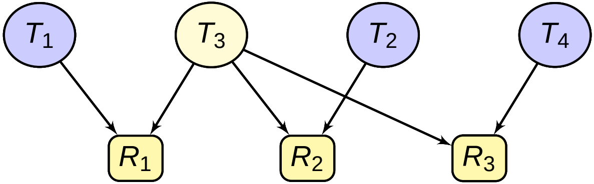
$time(T_3) = 1$

$time(T_4) = 2.45$

T_3 T_1 T_2 T_4 $CE(T') = 0.55$

Greedy method constructs suite with marginal improvement

Limitations of Greedy Methods



$time(T_1) = 1$

$time(T_2) = 1$

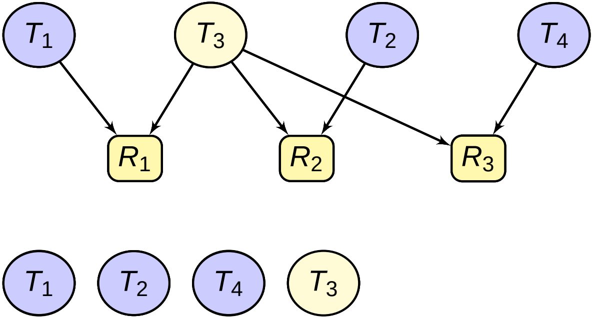
$time(T_3) = 1$

$time(T_4) = 2.45$

T_3 T_1 T_2 T_4 $CE(T') = 0.55$

Greedy can exhibit high run-times (Jiang et al. ASE 2009)

Limitations of Greedy Methods



$time(T_1) = 1$

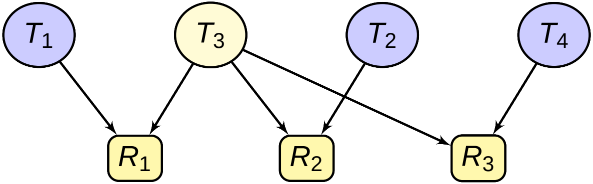
$time(T_2) = 1$

$time(T_3) = 1$

$time(T_4) = 2.45$

Genetic algorithm finds a **higher quality** ordering

Limitations of Greedy Methods



$time(T_1) = 1$

$time(T_2) = 1$

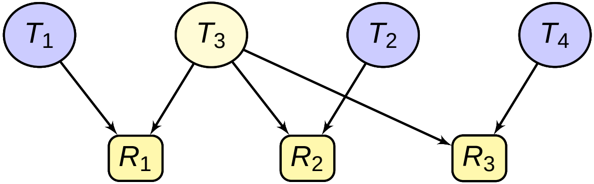
$time(T_3) = 1$

$time(T_4) = 2.45$

T_1 T_2 T_4 T_3 $\quad CE(T') = 0.63$

Genetic algorithm finds a **higher quality** ordering

Limitations of Greedy Methods



$time(T_1) = 1$

$time(T_2) = 1$

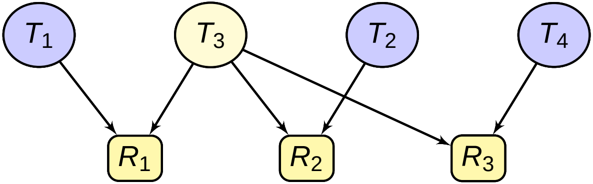
$time(T_3) = 1$

$time(T_4) = 2.45$

T_1 T_2 T_4 T_3 $CE(T') = 0.63$

Genetic algorithm is amenable to **parallelization**

Limitations of Greedy Methods



$time(T_1) = 1$

$time(T_2) = 1$

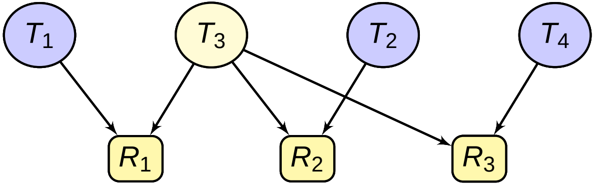
$time(T_3) = 1$

$time(T_4) = 2.45$

T_1 T_2 T_4 T_3 $CE(T') = 0.63$

Genetic algorithm supports **“human in the loop”**

Limitations of Greedy Methods



$time(T_1) = 1$

$time(T_2) = 1$

$time(T_3) = 1$

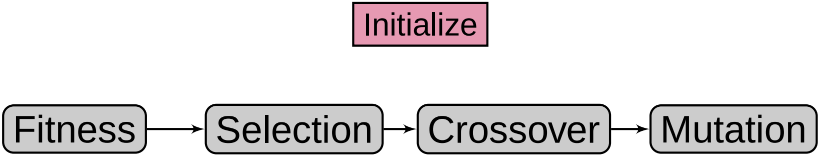
$time(T_4) = 2.45$

T_1 T_2 T_4 T_3

$CE(T') = 0.63$

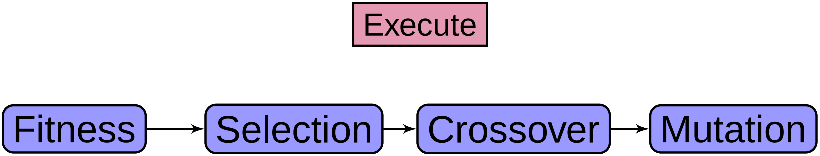
Genetic algorithm constructs **diverse** test orderings

Test Prioritization with Genetic Algorithms



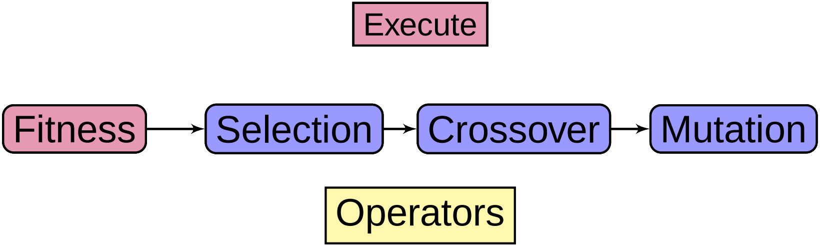
Randomly create suites by repeatedly **shuffling** $\langle T_1, \ldots, T_n \rangle$

Test Prioritization with Genetic Algorithms



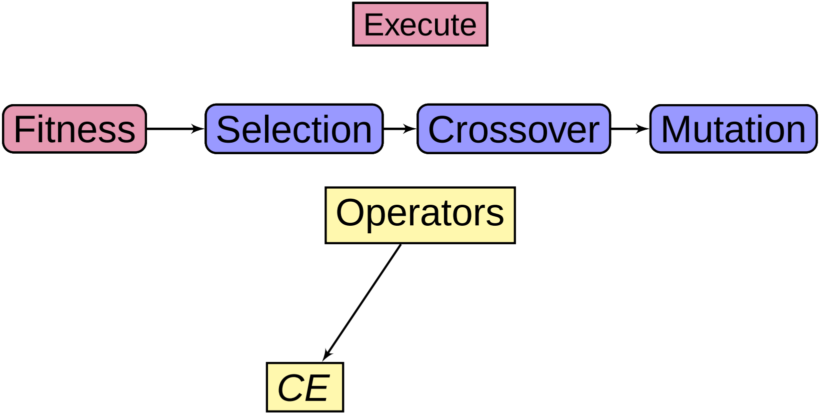
Execute the phases until the genetic algorithm **stagnates**

Test Prioritization with Genetic Algorithms



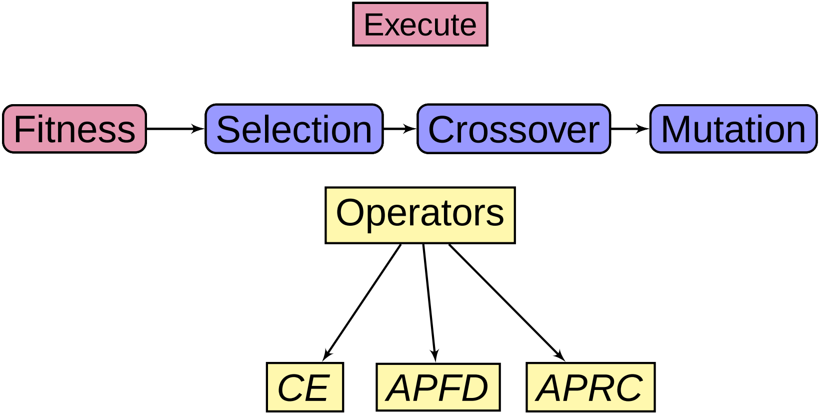
Use **coverage effectiveness** in this study - **others possible**

Test Prioritization with Genetic Algorithms



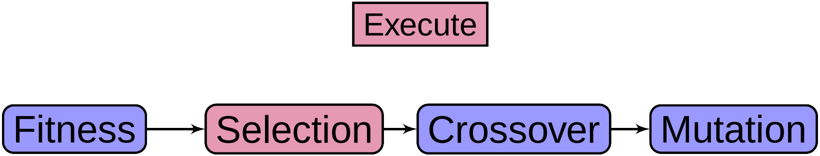
Use **coverage effectiveness** in this study - **others possible**

Test Prioritization with Genetic Algorithms



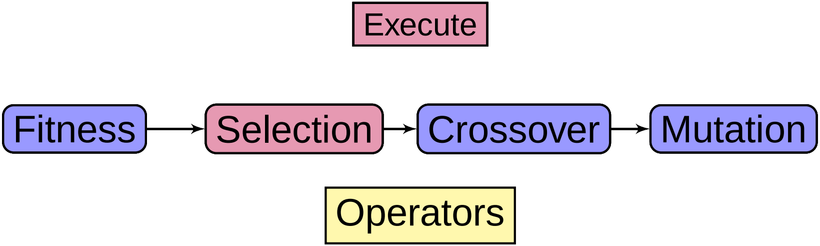
Use **coverage effectiveness** in this study - **others possible**

Test Prioritization with Genetic Algorithms



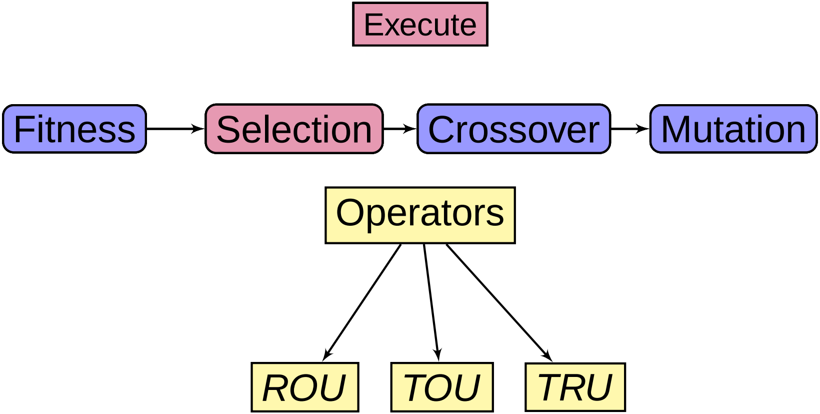
Choose orderings to become **parents** of next generation

Test Prioritization with Genetic Algorithms



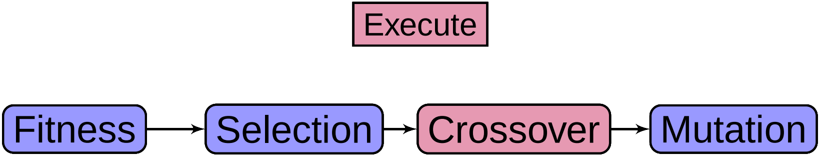
Choose orderings to become **parents** of next generation

Test Prioritization with Genetic Algorithms



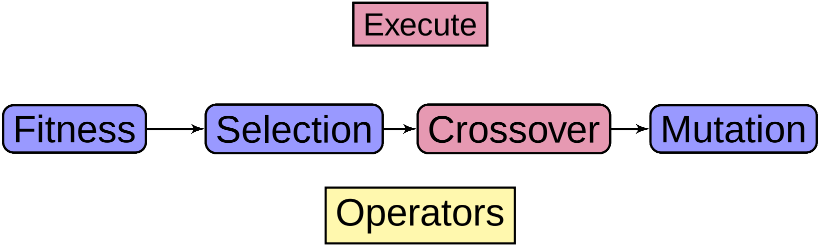
Choose orderings to become **parents** of next generation

Test Prioritization with Genetic Algorithms



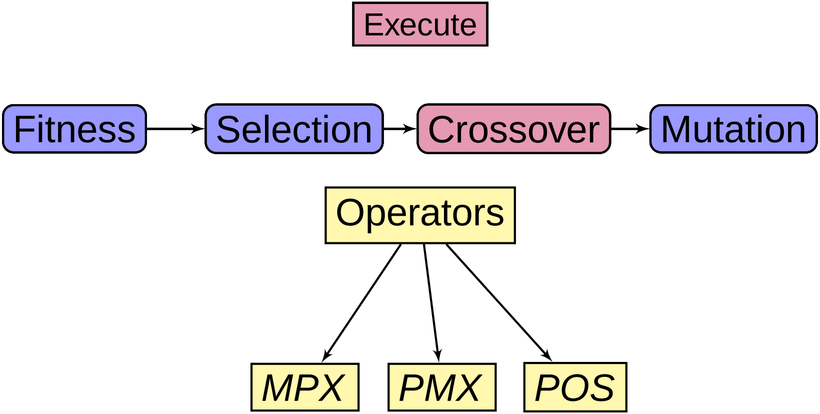
Seven possible operators **combine** parents to produce children

Test Prioritization with Genetic Algorithms



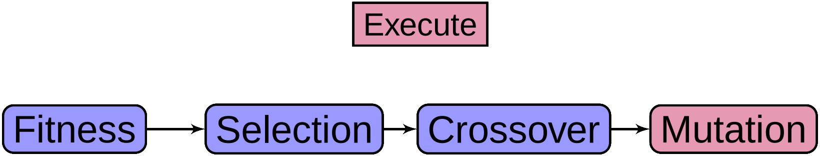
Seven possible operators **combine** parents to produce children

Test Prioritization with Genetic Algorithms



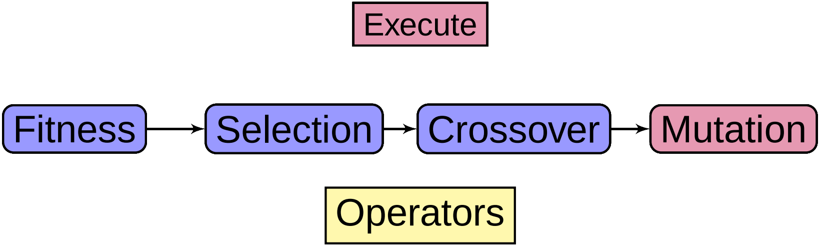
Seven possible operators **combine** parents to produce children

Test Prioritization with Genetic Algorithms



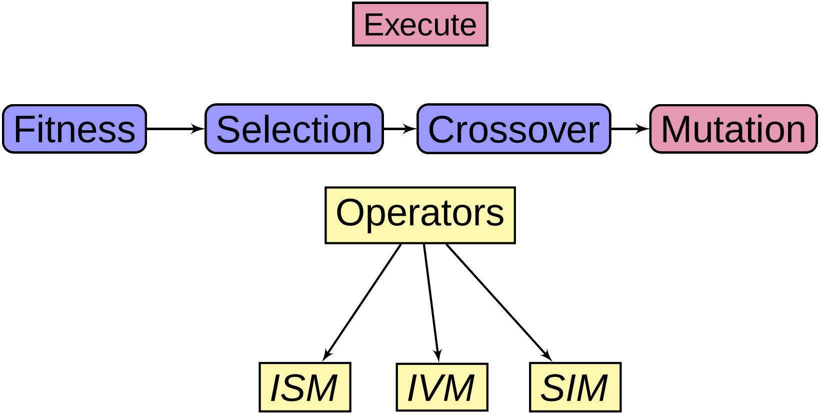
Six possible operators make random **changes** to orderings

Test Prioritization with Genetic Algorithms



Six possible operators make random **changes** to orderings

Test Prioritization with Genetic Algorithms



Six possible operators make random **changes** to orderings

Configuration of the Genetic Algorithm

Possible Configurations

Mutation Rate

Child Density

Population

Stagnancy

Explored a **wide variety** of genetic algorithm **configurations**

Configuration of the Genetic Algorithm

Possible Configurations

Mutation Rate

Child Density

Population

Stagnancy

How frequently do we **modify** individual test **orderings**?

Configuration of the Genetic Algorithm

Possible Configurations

Mutation Rate

Child Density

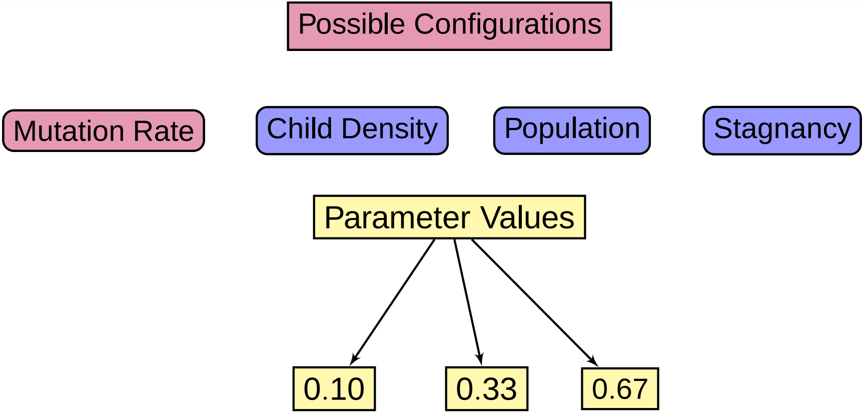
Population

Stagnancy

Parameter Values

How frequently do we **modify** individual test **orderings**?

Configuration of the Genetic Algorithm



How frequently do we **modify** individual test **orderings**?

Configuration of the Genetic Algorithm

Possible Configurations

Mutation Rate | Child Density | Population | Stagnancy

How many **children** should be in the **next population**?

Configuration of the Genetic Algorithm

Possible Configurations

Mutation Rate

Child Density

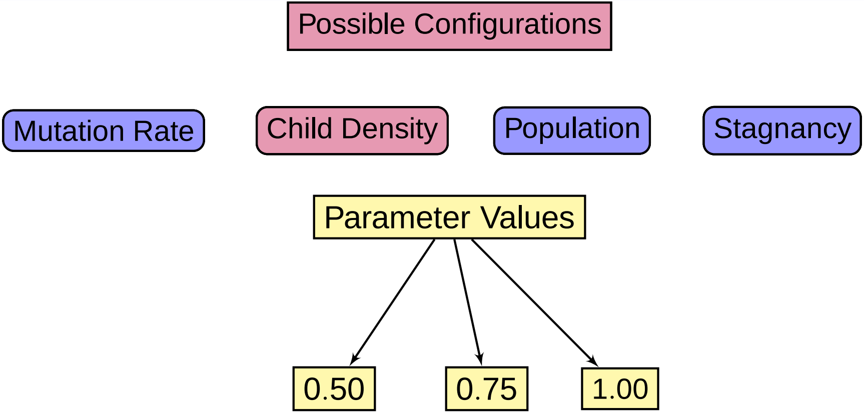
Population

Stagnancy

Parameter Values

How many **children** should be in the **next population**?

Configuration of the Genetic Algorithm



How many **children** should be in the **next population**?

Configuration of the Genetic Algorithm

Possible Configurations

Mutation Rate | Child Density | Population | Stagnancy

How many **test suites** should exist in the population?

Configuration of the Genetic Algorithm

Possible Configurations

Mutation Rate | Child Density | Population | Stagnancy

Parameter Values

How many **test suites** should exist in the population?

Configuration of the Genetic Algorithm

Possible Configurations

Mutation Rate | Child Density | Population | Stagnancy

Parameter Values

75 | 150 | 225

How many **test suites** should exist in the population?

Configuration of the Genetic Algorithm

Possible Configurations

Mutation Rate

Child Density

Population

Stagnancy

How many **generations** without fitness **improvement**?

Configuration of the Genetic Algorithm

Possible Configurations

Mutation Rate | Child Density | Population | Stagnancy

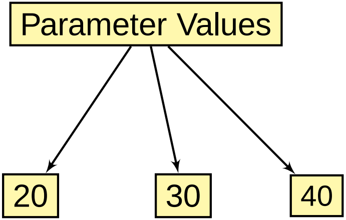
Parameter Values

How many **generations** without fitness **improvement**?

Configuration of the Genetic Algorithm

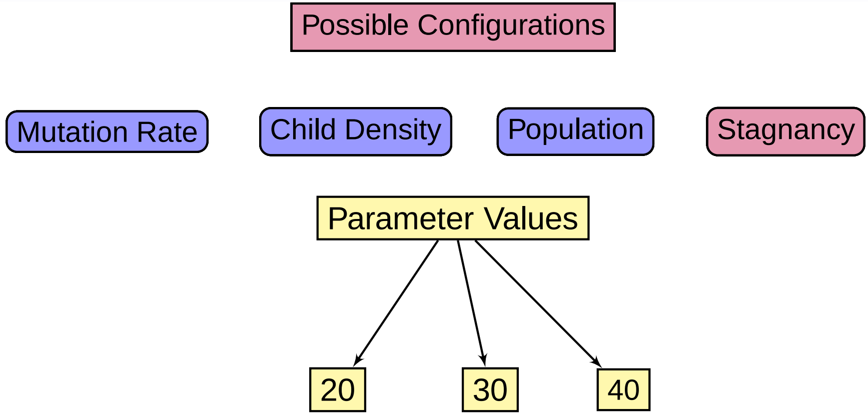
Possible Configurations

Mutation Rate | Child Density | Population | Stagnancy



How many **generations** without fitness **improvement**?

Configuration of the Genetic Algorithm



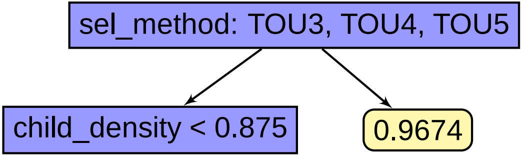
See the **paper** for further **operator** and **configuration** details

Analysis Techniques: Tree Models

sel_method: TOU3, TOU4, TOU5

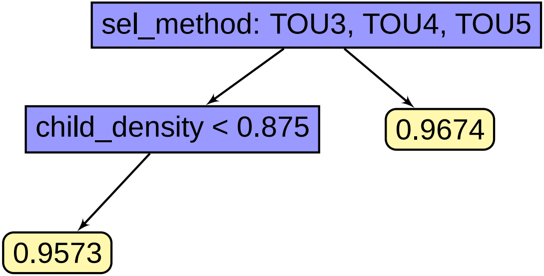
Tree Models: Recursive partitioning creates hierarchical view of data

Analysis Techniques: Tree Models



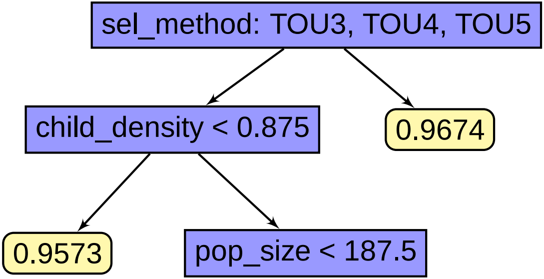
Tree Models: Recursive partitioning creates hierarchical view of data

Analysis Techniques: Tree Models



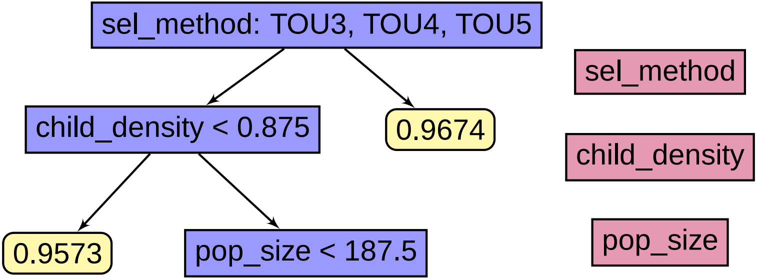
Tree Models: Recursive partitioning creates hierarchical view of data

Analysis Techniques: Tree Models



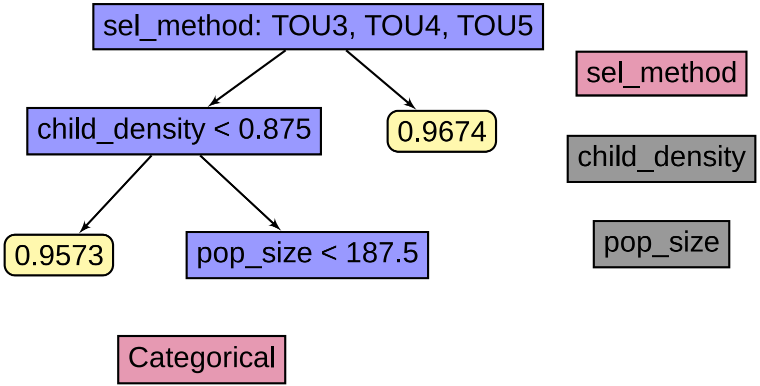
Tree Models: Recursive partitioning creates hierarchical view of data

Analysis Techniques: Tree Models



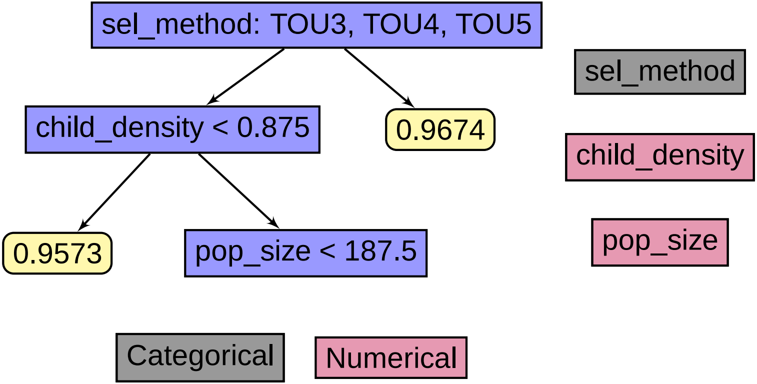
Explanatory Variable: Configuration of the genetic algorithm

Analysis Techniques: Tree Models



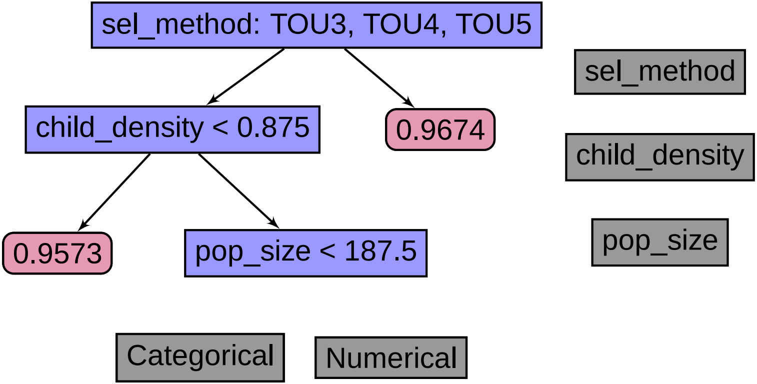
Non-parametric techniques that handles **different** variable types

Analysis Techniques: Tree Models



Non-parametric techniques that handles **different** variable types

Analysis Techniques: Tree Models



Response Variable: Fitness of the final test ordering (CE score)

Experimental Goals and Design

Name	$\|T\|$	$\|\mathcal{R}(T)\|$	CCN	NCSS
DS	110	40	1.35	1243.00
GB	51	88	2.60	1455.00
JD	54	783	1.64	2716.00
LF	13	6	1.40	215.00
RM	13	19	2.13	569.00
SK	27	117	2.00	628.00
TM	27	46	2.21	748.00
RP	76	221	2.65	6822.00

Several applications and test suites - **coverage** reports derived from **call-tree** based adequacy (McMaster and Memon ICSM 2005)

Experimental Goals and Design

Name	$\lvert T \rvert$	$\lvert \mathcal{R}(T) \rvert$	CCN	NCSS
DS	110	40	1.35	1243.00
GB	51	88	2.60	1455.00
JD	54	783	1.64	2716.00
LF	13	6	1.40	215.00
RM	13	19	2.13	569.00
SK	27	117	2.00	628.00
TM	27	46	2.21	748.00
RP	76	221	2.65	6822.00

Several applications and test suites - **coverage** reports derived from **call-tree** based adequacy (McMaster and Memon ICSM 2005)

Experimental Goals and Design

Name	$\|T\|$	$\|\mathcal{R}(T)\|$	CCN	NCSS
DS	110	40	1.35	1243.00
GB	51	88	2.60	1455.00
JD	54	783	1.64	2716.00
LF	13	6	1.40	215.00
RM	13	19	2.13	569.00
SK	27	117	2.00	628.00
TM	27	46	2.21	748.00
RP	76	221	2.65	6822.00

Several applications and test suites - **coverage** reports derived from **call-tree** based adequacy (McMaster and Memon ICSM 2005)

Experimental Goals and Design

Name	$\lvert T \rvert$	$\lvert \mathcal{R}(T) \rvert$	CCN	NCSS
DS	110	40	1.35	1243.00
GB	51	88	2.60	1455.00
JD	54	783	1.64	2716.00
LF	13	6	1.40	215.00
RM	13	19	2.13	569.00
SK	27	117	2.00	628.00
TM	27	46	2.21	748.00
RP	76	221	2.65	6822.00

Use **additional** case study **applications** and adequacy **criteria** as future work in order to **control** threats to external **validity**

Experimental Goals and Design

Name	$\lvert T \rvert$	$\lvert \mathcal{R}(T) \rvert$	CCN	NCSS
DS	110	40	1.35	1243.00
GB	51	88	2.60	1455.00
JD	54	783	1.64	2716.00
LF	13	6	1.40	215.00
RM	13	19	2.13	569.00
SK	27	117	2.00	628.00
TM	27	46	2.21	748.00
RP	76	221	2.65	6822.00

Use **random** and **hill climbing** (first and steepest ascent) as control methods for comparison to the genetic algorithm prioritizer

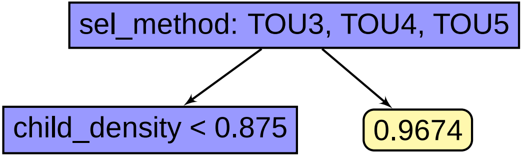
Experimental Goals and Design

Name	$\|T\|$	$\|\mathcal{R}(T)\|$	CCN	NCSS
DS	110	40	1.35	1243.00
GB	51	88	2.60	1455.00
JD	54	783	1.64	2716.00
LF	13	6	1.40	215.00
RM	13	19	2.13	569.00
SK	27	117	2.00	628.00
TM	27	46	2.21	748.00
RP	76	221	2.65	6822.00

See the **paper** for more details about the **design** of the empirical study (e.g., configuration of random and hill climbing prioritizers)

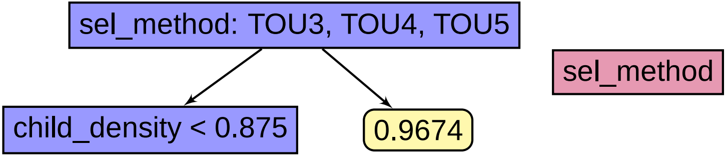
Results: Selection Method Importance

RM



Results: Selection Method Importance

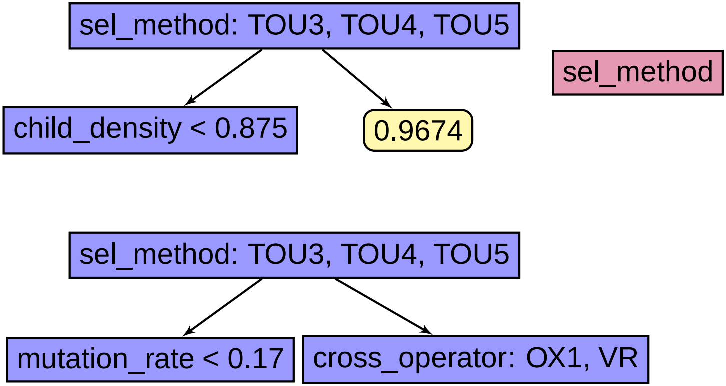
RM



Results: Selection Method Importance

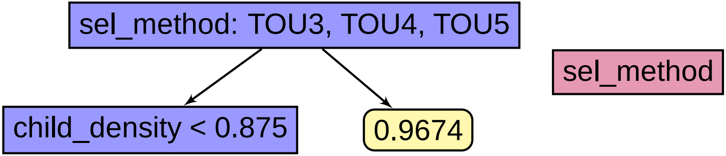
RM

GB

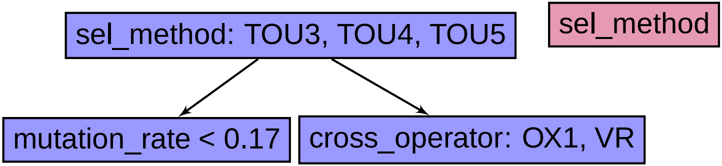


Results: Selection Method Importance

RM

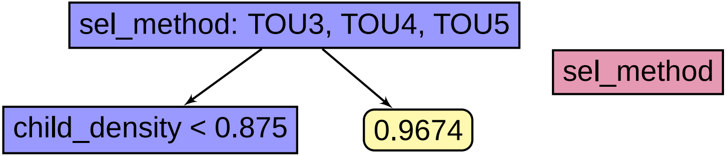


GB

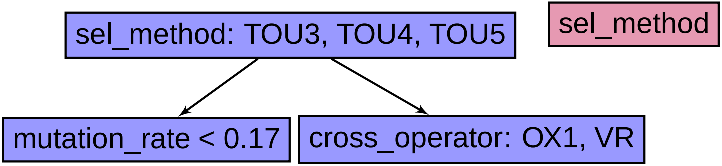


Results: Selection Method Importance

RM

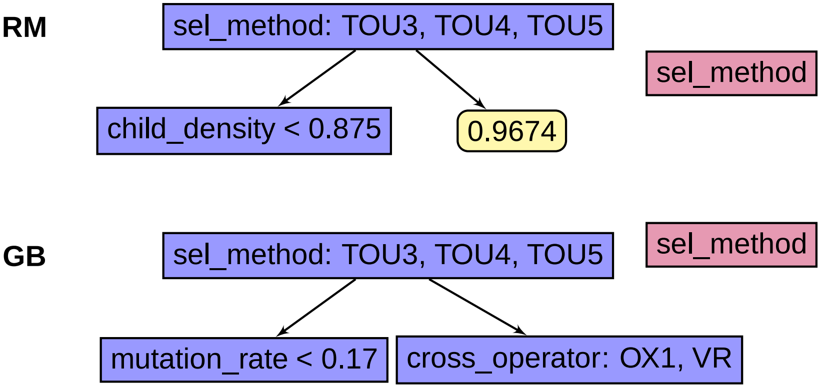


GB



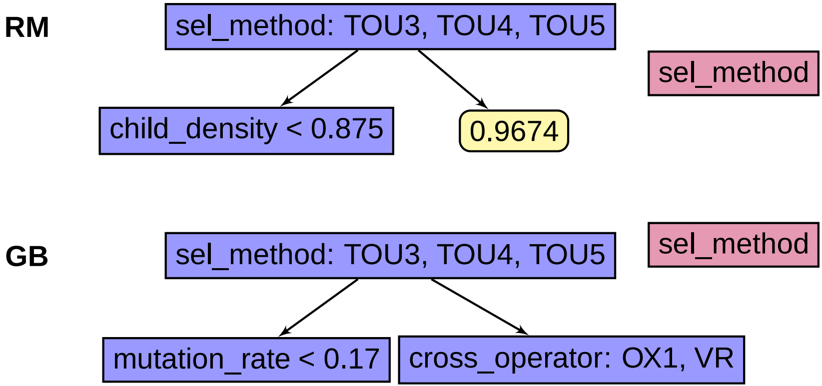
The sel_method variable is always the **most important** parameter

Results: Selection Method Importance



Importance of sel_method holds for **all** case study **applications**

Results: Selection Method Importance



How does the selection method impact the genetic algorithm?

Results: Selection Intensity

Name	ROUE	ROUL	TRU60	TRU40	TOU2	TOU5
DS	0.9742	0.9837	0.9893	0.9915	0.9514	0.9706
GB	0.9500	0.9572	0.9668	0.9700	0.9062	0.9402
JD	0.9247	0.9328	0.9431	0.9451	0.8993	0.9192
LF	0.9903	0.9903	0.9903	0.9903	0.9903	0.9903
RM	0.9665	0.9670	0.9681	0.9682	0.9328	0.9475
RP	0.9774	0.9824	0.9868	0.9879	0.9570	0.9705
SK	0.9859	0.9878	0.9911	0.9915	0.9667	0.9763
TM	0.9585	0.9605	0.9662	0.9672	0.9503	0.9579
Avg.	0.9659	0.9702	0.9752	0.9765	0.9443	0.9591

Except for the smallest application (LF), the CE scores of the **evolved** orderings are **better** than the **initial** and **reverse** test suites

Results: Selection Intensity

Name	ROUE	ROUL	TRU60	TRU40	TOU2	TOU5
DS	0.9742	0.9837	0.9893	0.9915	0.9514	0.9706
GB	0.9500	0.9572	0.9668	0.9700	0.9062	0.9402
JD	0.9247	0.9328	0.9431	0.9451	0.8993	0.9192
LF	0.9903	0.9903	0.9903	0.9903	0.9903	0.9903
RM	0.9665	0.9670	0.9681	0.9682	0.9328	0.9475
RP	0.9774	0.9824	0.9868	0.9879	0.9570	0.9705
SK	0.9859	0.9878	0.9911	0.9915	0.9667	0.9763
TM	0.9585	0.9605	0.9662	0.9672	0.9503	0.9579
Avg.	0.9659	0.9702	0.9752	0.9765	0.9443	0.9591

Study a type of operator as it **increases** in **intensity**, or the change in average fitness due to selection (Blickle & Thiele, *Evol Comp* 1997)

Results: Selection Intensity

Name	ROUE	ROUL	TRU60	TRU40	TOU2	TOU5
DS	0.9742	0.9837	0.9893	0.9915	0.9514	0.9706
GB	0.9500	0.9572	0.9668	0.9700	0.9062	0.9402
JD	0.9247	0.9328	0.9431	0.9451	0.8993	0.9192
LF	0.9903	0.9903	0.9903	0.9903	0.9903	0.9903
RM	0.9665	0.9670	0.9681	0.9682	0.9328	0.9475
RP	0.9774	0.9824	0.9868	0.9879	0.9570	0.9705
SK	0.9859	0.9878	0.9911	0.9915	0.9667	0.9763
TM	0.9585	0.9605	0.9662	0.9672	0.9503	0.9579
Avg.	0.9659	0.9702	0.9752	0.9765	0.9443	0.9591

Increasing selection intensity **improves** the CE scores of test orderings, even though it does **not** cause more rapid **convergence**

Results: Selection Intensity

Name	ROUE	ROUL	TRU60	TRU40	TOU2	TOU5
DS	0.9742	0.9837	0.9893	0.9915	0.9514	0.9706
GB	0.9500	0.9572	0.9668	0.9700	0.9062	0.9402
JD	0.9247	0.9328	0.9431	0.9451	0.8993	0.9192
LF	0.9903	0.9903	0.9903	0.9903	0.9903	0.9903
RM	0.9665	0.9670	0.9681	0.9682	0.9328	0.9475
RP	0.9774	0.9824	0.9868	0.9879	0.9570	0.9705
SK	0.9859	0.9878	0.9911	0.9915	0.9667	0.9763
TM	0.9585	0.9605	0.9662	0.9672	0.9503	0.9579
Avg.	0.9659	0.9702	0.9752	0.9765	0.9443	0.9591

Increasing selection intensity **improves** the CE scores of test orderings, even though it does **not** cause more rapid **convergence**

Results: Selection Intensity

Name	ROUE	ROUL	TRU60	TRU40	TOU2	TOU5
DS	0.9742	0.9837	0.9893	0.9915	0.9514	0.9706
GB	0.9500	0.9572	0.9668	0.9700	0.9062	0.9402
JD	0.9247	0.9328	0.9431	0.9451	0.8993	0.9192
LF	0.9903	0.9903	0.9903	0.9903	0.9903	0.9903
RM	0.9665	0.9670	0.9681	0.9682	0.9328	0.9475
RP	0.9774	0.9824	0.9868	0.9879	0.9570	0.9705
SK	0.9859	0.9878	0.9911	0.9915	0.9667	0.9763
TM	0.9585	0.9605	0.9662	0.9672	0.9503	0.9579
Avg.	0.9659	0.9702	0.9752	0.9765	0.9443	0.9591

Increasing selection intensity **improves** the CE scores of test orderings, even though it does **not** cause more rapid **convergence**

Results: Selection Intensity

Name	ROUE	ROUL	TRU60	TRU40	TOU2	TOU5
DS	0.9742	0.9837	0.9893	0.9915	0.9514	0.9706
GB	0.9500	0.9572	0.9668	0.9700	0.9062	0.9402
JD	0.9247	0.9328	0.9431	0.9451	0.8993	0.9192
LF	0.9903	0.9903	0.9903	0.9903	0.9903	0.9903
RM	0.9665	0.9670	0.9681	0.9682	0.9328	0.9475
RP	0.9774	0.9824	0.9868	0.9879	0.9570	0.9705
SK	0.9859	0.9878	0.9911	0.9915	0.9667	0.9763
TM	0.9585	0.9605	0.9662	0.9672	0.9503	0.9579
Avg.	0.9659	0.9702	0.9752	0.9765	0.9443	0.9591

Low intensity selection causes search to **meander** around low quality test suite prioritizations, making fitness **stagnate** and the GA terminate

Results: Selection Intensity

Name	ROUE	ROUL	TRU60	TRU40	TOU2	TOU5
DS	0.9742	0.9837	0.9893	0.9915	0.9514	0.9706
GB	0.9500	0.9572	0.9668	0.9700	0.9062	0.9402
JD	0.9247	0.9328	0.9431	0.9451	0.8993	0.9192
LF	0.9903	0.9903	0.9903	0.9903	0.9903	0.9903
RM	0.9665	0.9670	0.9681	0.9682	0.9328	0.9475
RP	0.9774	0.9824	0.9868	0.9879	0.9570	0.9705
SK	0.9859	0.9878	0.9911	0.9915	0.9667	0.9763
TM	0.9585	0.9605	0.9662	0.9672	0.9503	0.9579
Avg.	0.9659	0.9702	0.9752	0.9765	0.9443	0.9591

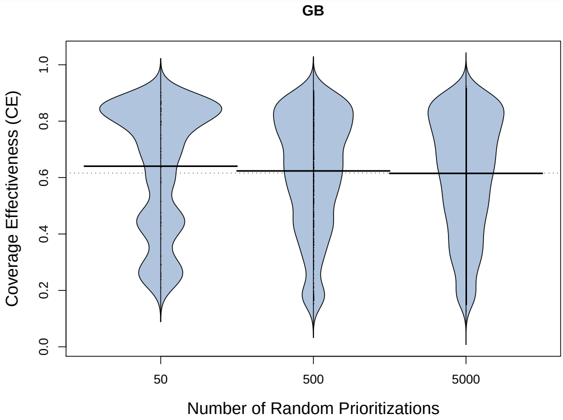
High intensity selection focuses on a **local** optimum of **high quality** instead of **hunting** for hard-to-find **global** optimum

Results: Selection Intensity

Name	ROUE	ROUL	TRU60	TRU40	TOU2	TOU5
DS	0.9742	0.9837	0.9893	0.9915	0.9514	0.9706
GB	0.9500	0.9572	0.9668	0.9700	0.9062	0.9402
JD	0.9247	0.9328	0.9431	0.9451	0.8993	0.9192
LF	0.9903	0.9903	0.9903	0.9903	0.9903	0.9903
RM	0.9665	0.9670	0.9681	0.9682	0.9328	0.9475
RP	0.9774	0.9824	0.9868	0.9879	0.9570	0.9705
SK	0.9859	0.9878	0.9911	0.9915	0.9667	0.9763
TM	0.9585	0.9605	0.9662	0.9672	0.9503	0.9579
Avg.	0.9659	0.9702	0.9752	0.9765	0.9443	0.9591

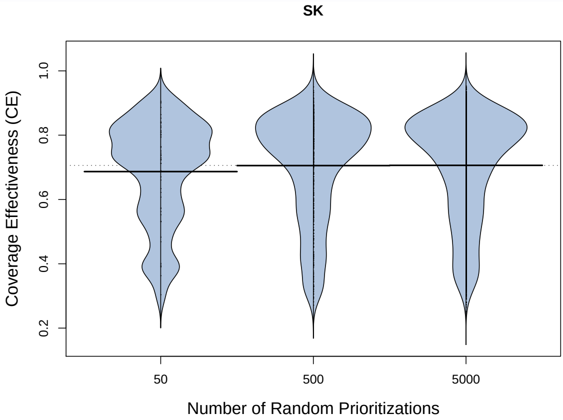
One Explanation: the fitness landscape for coverage effectiveness contains many local optima that are good test orderings

Results: Comparison to Random



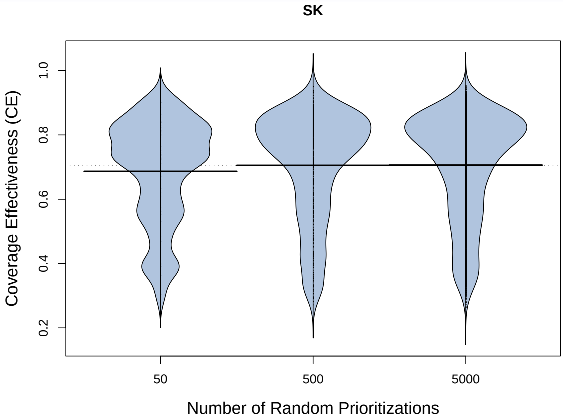
GB: Random orderings have average CE scores around 0.6

Results: Comparison to Random



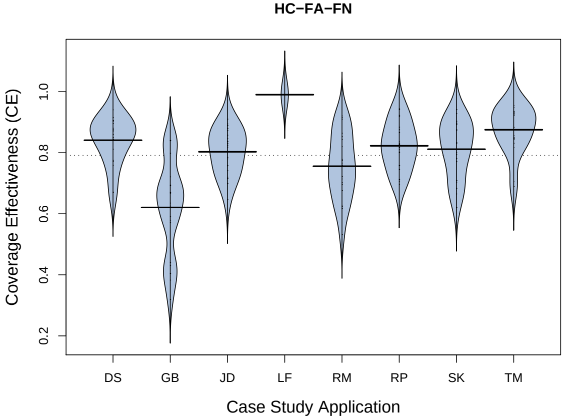
SK: Random orderings have average CE scores around 0.7

Results: Comparison to Random



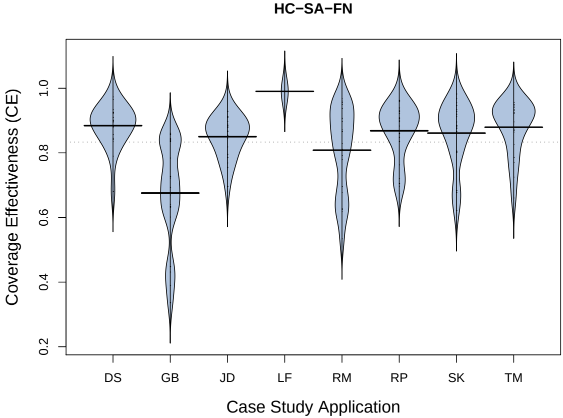
Conclusion: Random is not as effective as the genetic algorithm

Results: Comparison to Hill Climbing



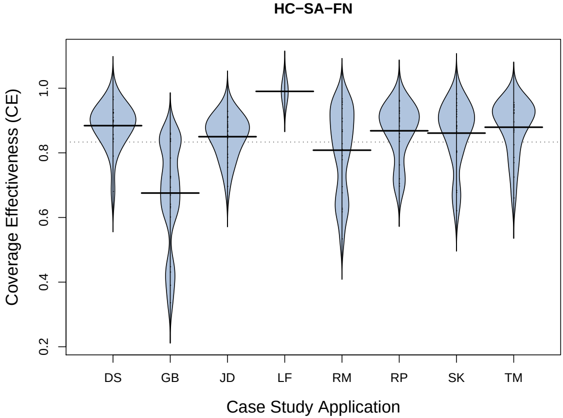
First Ascent: Across all applications, average CE score below 0.8

Results: Comparison to Hill Climbing



Steepest Ascent: Larger neighborhoods slightly improve the CE scores

Results: Comparison to Hill Climbing



Conclusion: Hill climber is not as effective as the genetic algorithm

GELATIONS Framework for Prioritization

gelations

GEnetic aLgorithm bAsed Test suIte prIOritizatioN System

Search projects

Project Home | **Downloads** | **Wiki** | **Issues** | **Source**

Summary | Updates | People

Gelations is a research prototype system for regression test suite prioritization using genetic algorithms. This system is written entirely in version 1.6 of the Java SE programming language, and is accompanied by its own regression test suite written using the JUnit unit testing framework.

Software testing is a crucial part of the software development lifecycle. Regression testing is a form of testing in which all of the old test cases written to cover different parts of a program are combined into a single test suite and executed. This form of testing helps to reveal regressions, or instances in which code that had formerly functioned correctly is broken by later changes to the system. For real-world applications, however, regression test suites can take days or even weeks to execute. One solution to this problem of execution time overhead is to reduce the suite, removing test cases that are redundant or unlikely to detect faults. This approach, however, can compromise the ability of a suite to detect faults. Another approach to this problem is test suite prioritization. Prioritization does not reduce the total execution time of a test suite, but instead reorders the test suite so that faults are more likely to be detected early in the execution of the test suite. This allows engineers to discover faults sooner and begin work to correct them earlier than would otherwise be possible, without sacrificing fault detection ability of the test suite.

This system implements a number of different selection, crossover, mutation, and fitness transformation operators, and is designed so that new or preexisting operators matching a

Activity: Low

Code license:
GNU General Public License v3

Labels:
testing, regression, genetic, java, junit, R, evolutionary, metaheuristic, softwareengineering, prioritization, geneticalgorithm

Featured downloads:
gelations-1_0.tar.gz
gelations-1_1.zip
Show all »

Feeds:
Project feeds

http://gelations.googlecode.com/ provides our framework

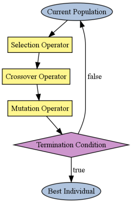
Conclusions and Future Work

Search-Based Prioritizers

Empirical Results

Developed a **genetic algorithm**-based test prioritizer supporting many evolutionary **operators** and **configurations**

Conclusions and Future Work

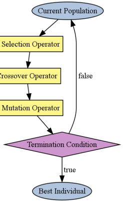


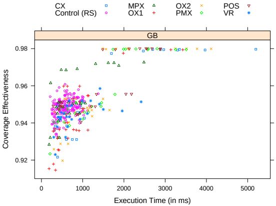
Search-Based Prioritizers

Empirical Results

Developed a **genetic algorithm**-based test prioritizer supporting many evolutionary **operators** and **configurations**

Conclusions and Future Work



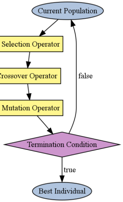


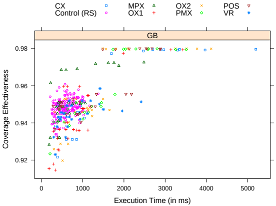
Search-Based Prioritizers

Empirical Results

Developed a **genetic algorithm**-based test prioritizer supporting many evolutionary **operators** and **configurations**

Conclusions and Future Work



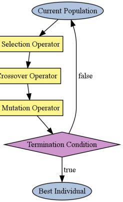


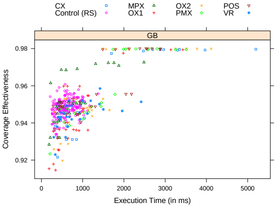
Search-Based Prioritizers

Empirical Results

Developed a **genetic algorithm**-based test prioritizer supporting many evolutionary **operators** and **configurations**

Conclusions and Future Work



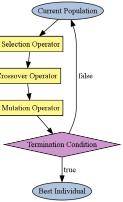


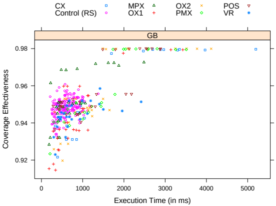
Search-Based Prioritizers

Empirical Results

Used automatically constructed **tree models** to highlight the role that the **selection** operator plays during **prioritization**

Conclusions and Future Work



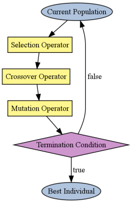


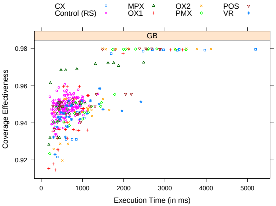
Search-Based Prioritizers

Empirical Results

Genetic algorithm is superior to **random search** and **hill climbing** and thus suitable for certain testing environments

Conclusions and Future Work



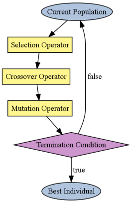


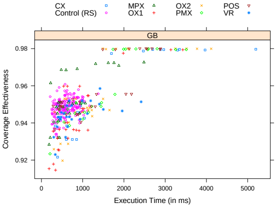
Search-Based Prioritizers

Empirical Results

Future Work: After extending the genetic algorithm, use fitness landscape analysis to understand impact of adequacy criteria

Conclusions and Future Work



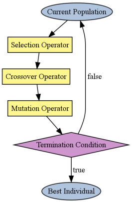


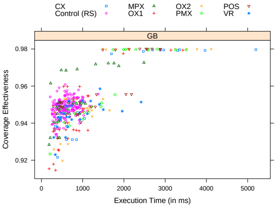
Search-Based Prioritizers

Empirical Results

Future Work: Use additional applications (e.g., SIR, XML, DBA) and test adequacy criteria (e.g., data and control flow)

Conclusions and Future Work



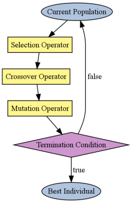


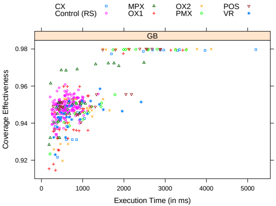
Search-Based Prioritizers

Empirical Results

Future Work: Comprehensive empirical study of all major search-based and greedy algorithms for test suite prioritization

Conclusions and Future Work



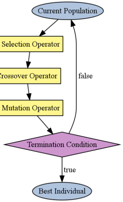


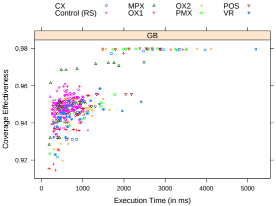
Search-Based Prioritizers

Empirical Results

Get involved and stay in touch!

Conclusions and Future Work



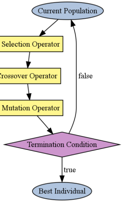


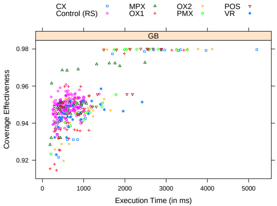
Search-Based Prioritizers

Empirical Results

Get involved and stay in touch! Read paper,

Conclusions and Future Work



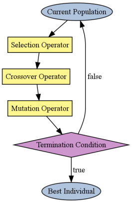


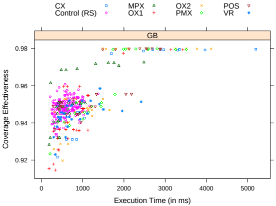
Search-Based Prioritizers

Empirical Results

Get involved and stay in touch! Read paper, download and use GELATIONS,

Conclusions and Future Work



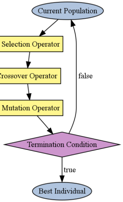


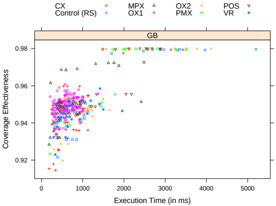
Search-Based Prioritizers

Empirical Results

Get involved and stay in touch! Read paper, download and use GELATIONS, replicate experiments,

Conclusions and Future Work



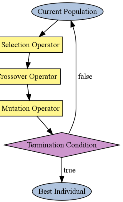


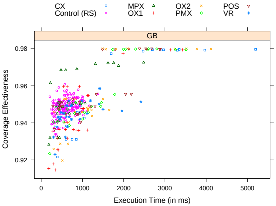
Search-Based Prioritizers

Empirical Results

Get involved and stay in touch! Read paper, download and use GELATIONS, replicate experiments, and share applications

Conclusions and Future Work





Search-Based Prioritizers

Empirical Results

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