AGRippin: A Novel Search Based Testing Technique for Android Applications

Domenico Amalfitano,

Nicola Amatucci,

Anna Rita Fasolino,

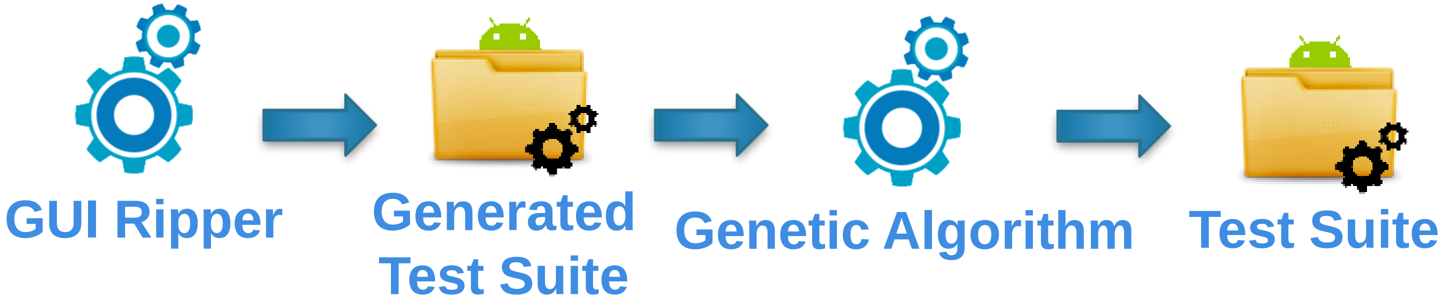
Porfirio Tramontana

Context & Motivation

- Automated GUI Testing Techniques for Mobile Applications
  - Capture&Replay, Model-Based, Model-Learning and Random Techniques
- Model-Learning and Random Techniques don't need information about the application under test
  - Fully automated generation of a Test Suite
- Model-Learning vs Random Techniques
  - Test-Suites generated by Random Techniques are able to reach an higher coverage of the application under test, but they are more complex

AGRippin

- Android Genetic Ripping
  - GUI Ripping Technique
    - Model-Learning Technique
  - Genetic Algorithm Implementation
- The purpose of the technique is to generate test suites that are more effective than the ones obtained by the Model-Learning technique, without increasing the test suite size



Android Ripper

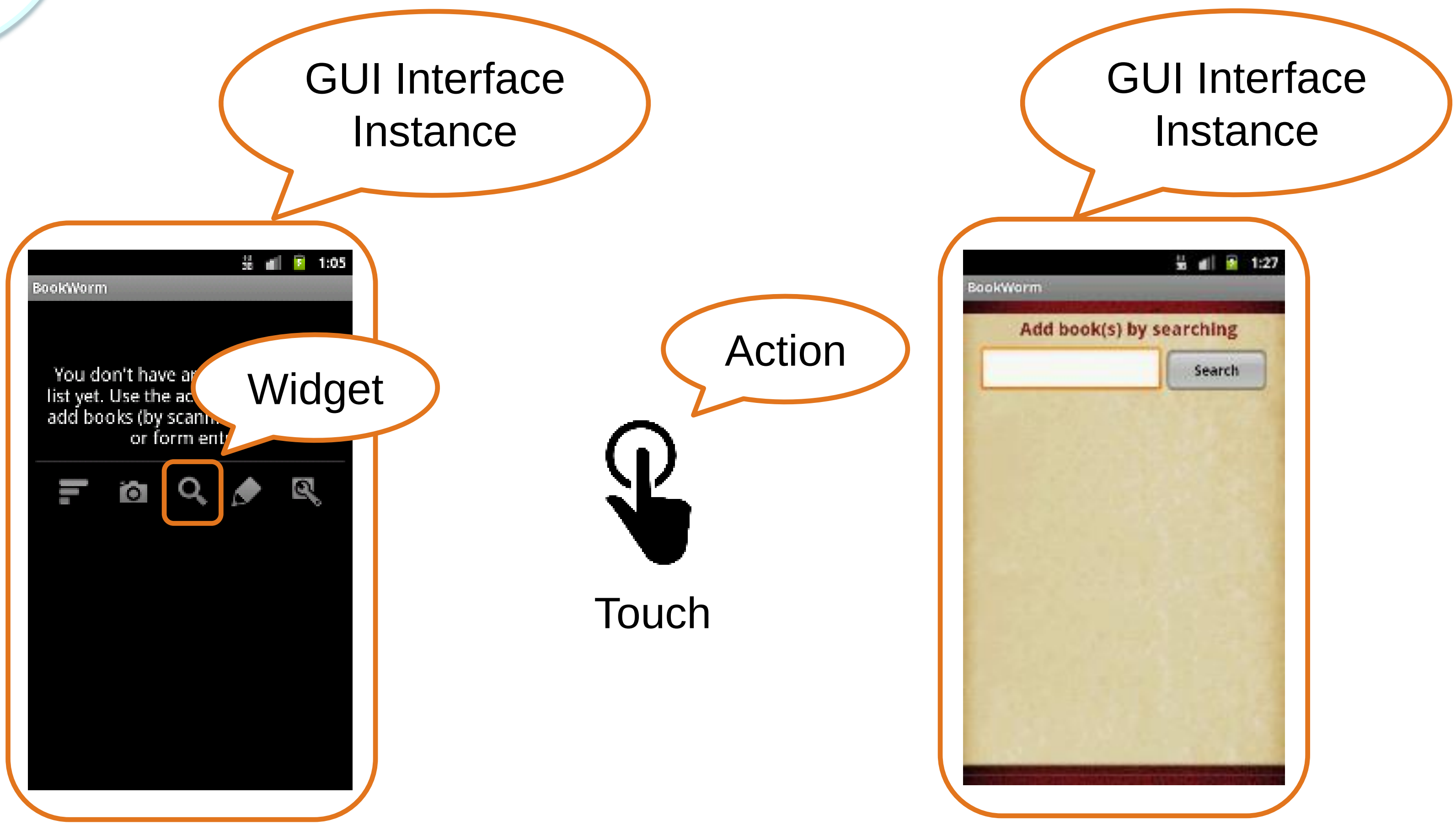
- Implements a set of GUI Ripping Techniques
- The GUI showing on the screen is analyzed and a list of fireable User and System events is built
  - After the execution of each event the GUI shown is analyzed and a new list of events is built
  - Each event is performed according to a Breadth-First, Depth-First or Random strategy
    - The tool creates sequences of events by visiting the GUI
    - The GUIs visited are added to a list
- A Test Suite is built reproducing to the sequences of executed events

Amalfitano, D.; Fasolino, A.R.; Tramontana, P.; De Carmine, S.; Memon, A.M., "Using GUI ripping for automated testing of Android applications," *Automated Software Engineering (ASE), 2012 Proceedings of the 27th IEEE/ACM International Conference on* , vol., no., pp.258,261, 3-7 Sept. 2012

Genetic Algorithm

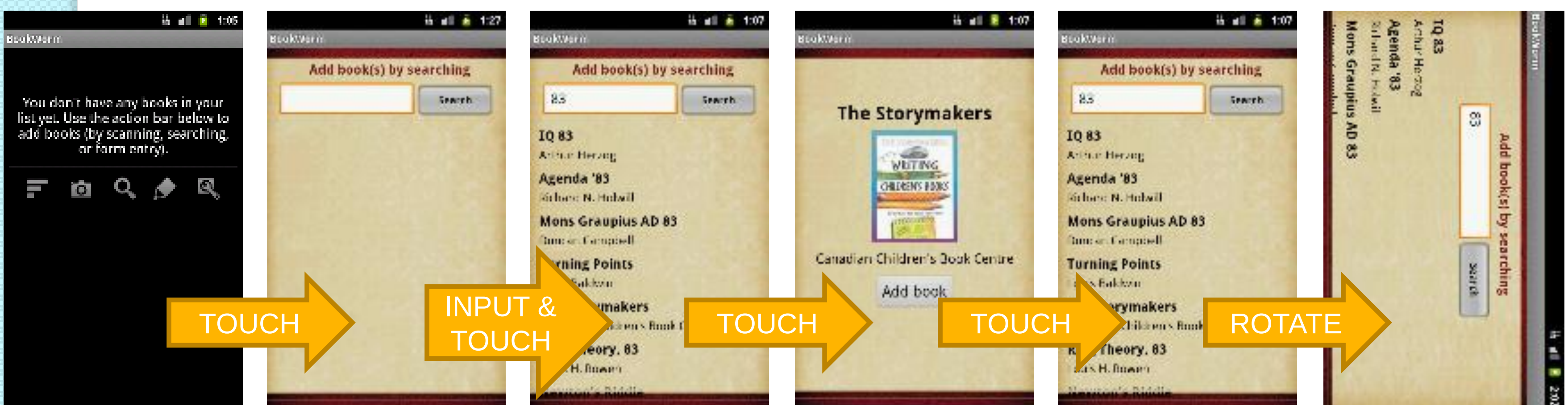
- Representation
  - Test Case Representation
- Crossover Technique
  - Single-Point Crossover
- Mutation Technique
- Fitness Evaluation
  - Global & Local Fitness
- Selection Technique
  - Rank-Based Selection
- Combination Technique
  - Further exploration of unvisited GUI Interfaces

Test Cases Representation



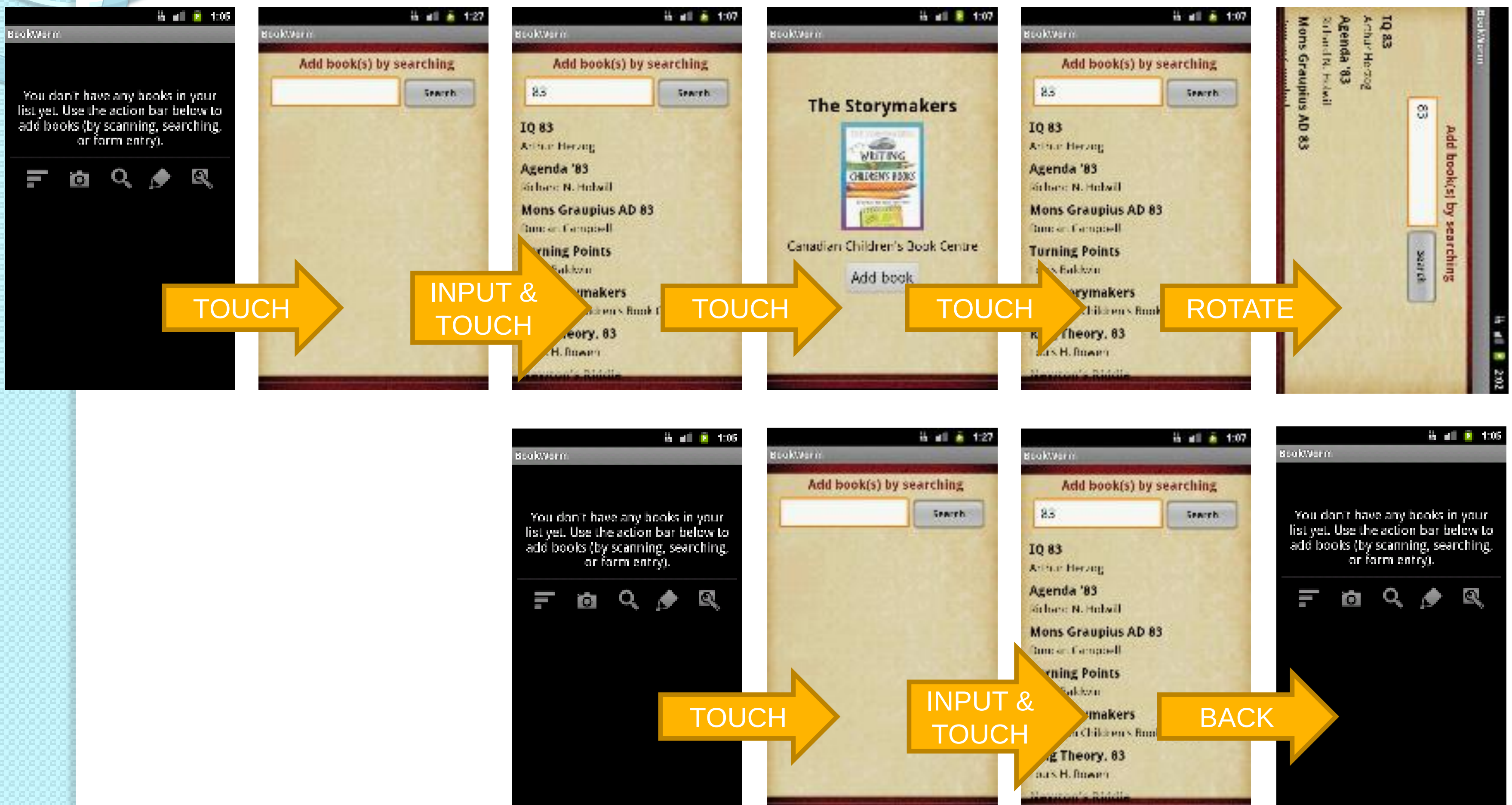
Test Cases Representation

- A Test Case is a sequence of GUI Interface Instances and Actions

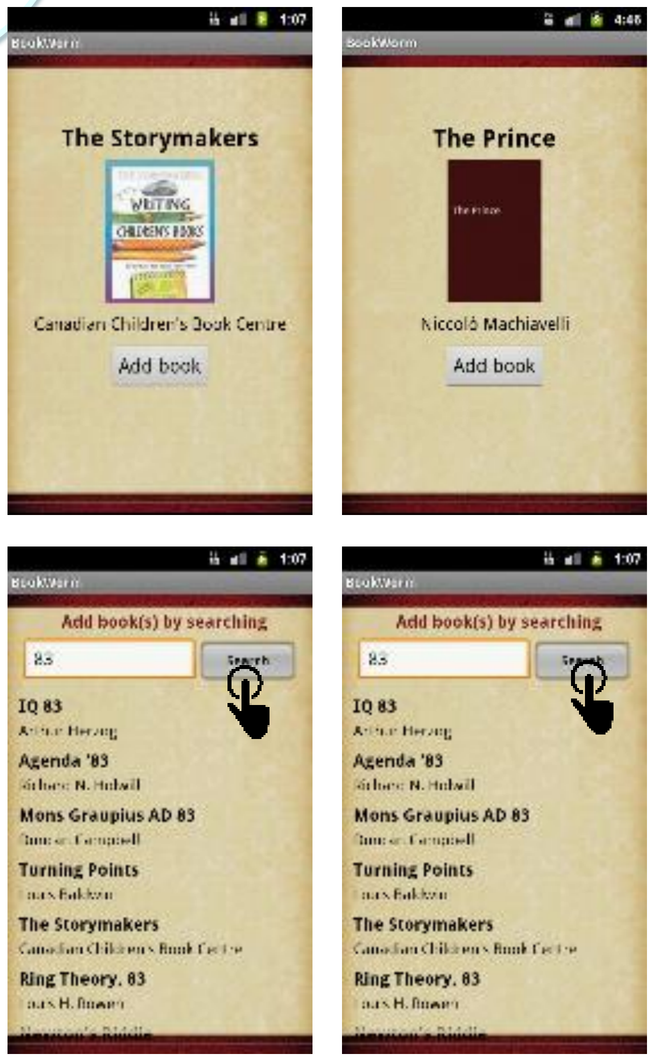


Crossover Technique

Single-Point Crossover!



Equivalence Criteria

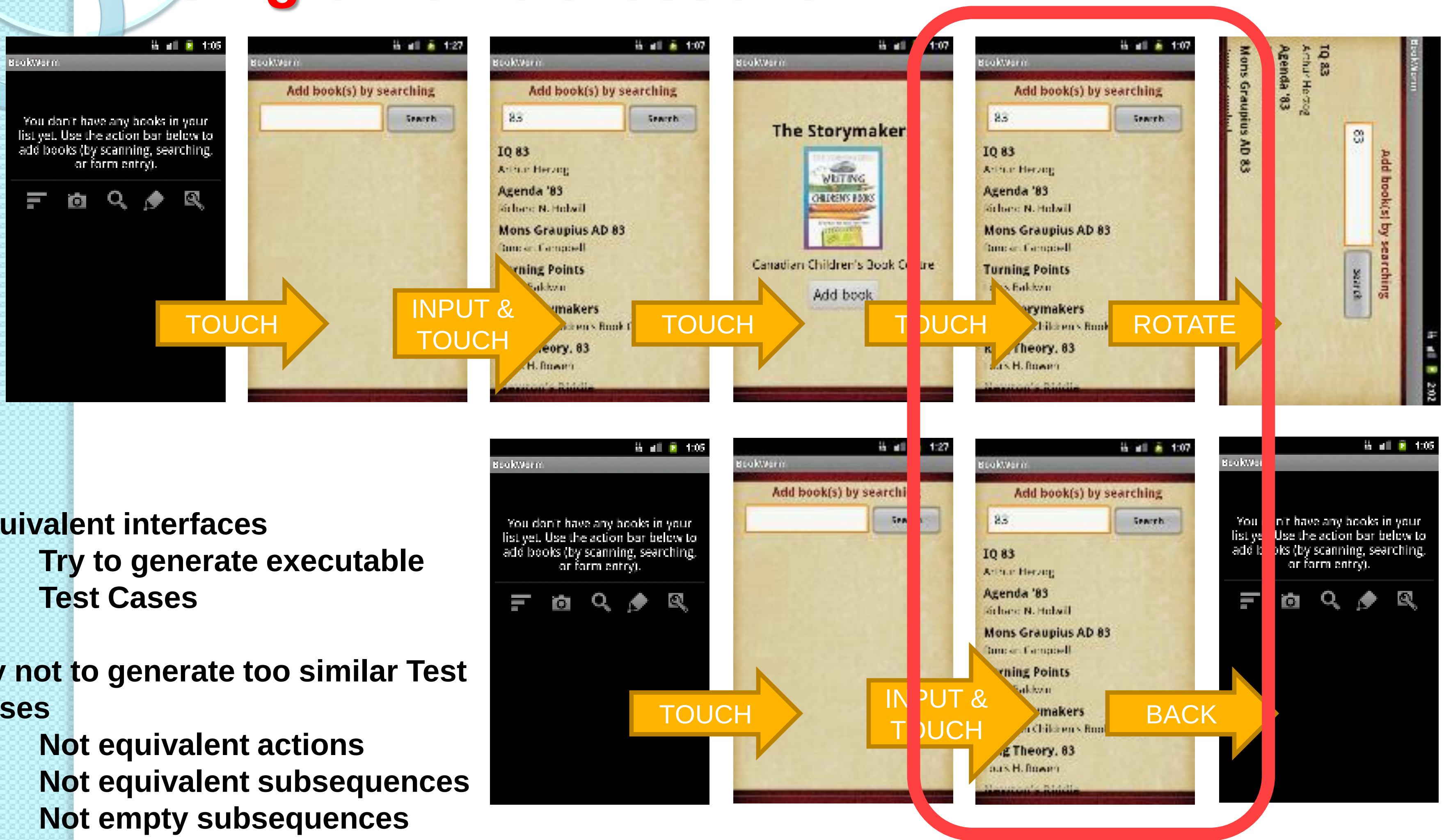


- **Two GUI interfaces are considered equivalent if they include the same set of widgets and they define the same set of event handlers.**

- **Two actions are considered equivalent if they are associated to the same user actions and the same event.**

Crossover Technique

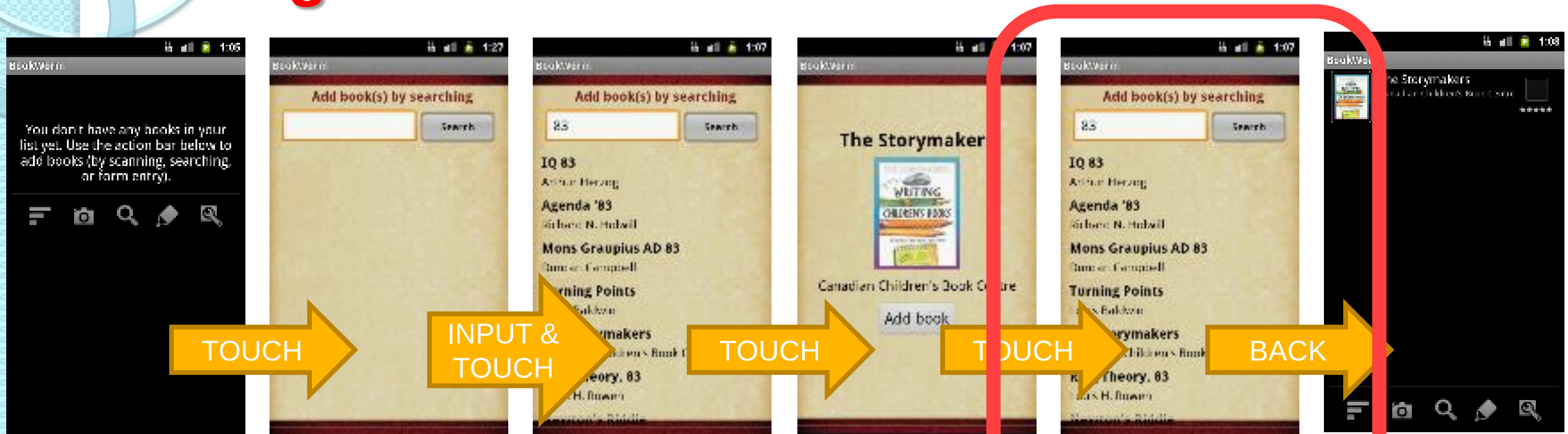
Single-Point Crossover!

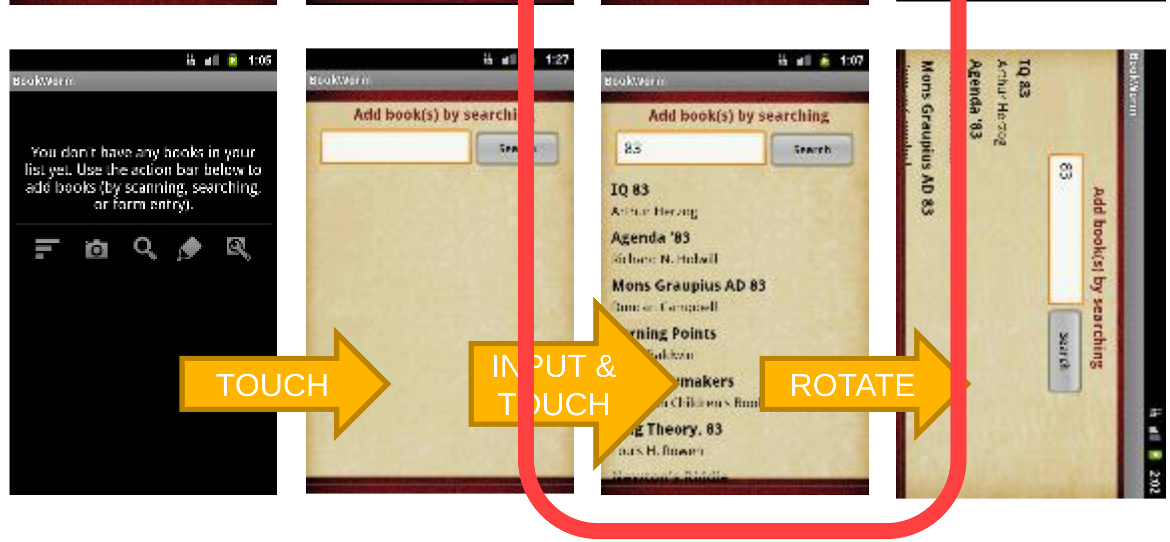


- **Equivalent interfaces**
  - **Try to generate executable Test Cases**
- **Try not to generate too similar Test Cases**
  - **Not equivalent actions**
  - **Not equivalent subsequences**
  - **Not empty subsequences**

Crossover Technique

Single-Point Crossover!





- **Generated test cases are added to the Test Suite, if they are actually executable**
- **Using the given equivalence criteria we try to ensure the generation of executable sequences of events**

Mutation Technique

- Input values generated by fully automated techniques are usually random or constants
- Our mutation operator changes these values randomly
  - Addresses, Names, Numbers, Paths, Email, Dates, Time Values, …
- Test cases are added to the test suite
  - The original test cases are kept into the test suite

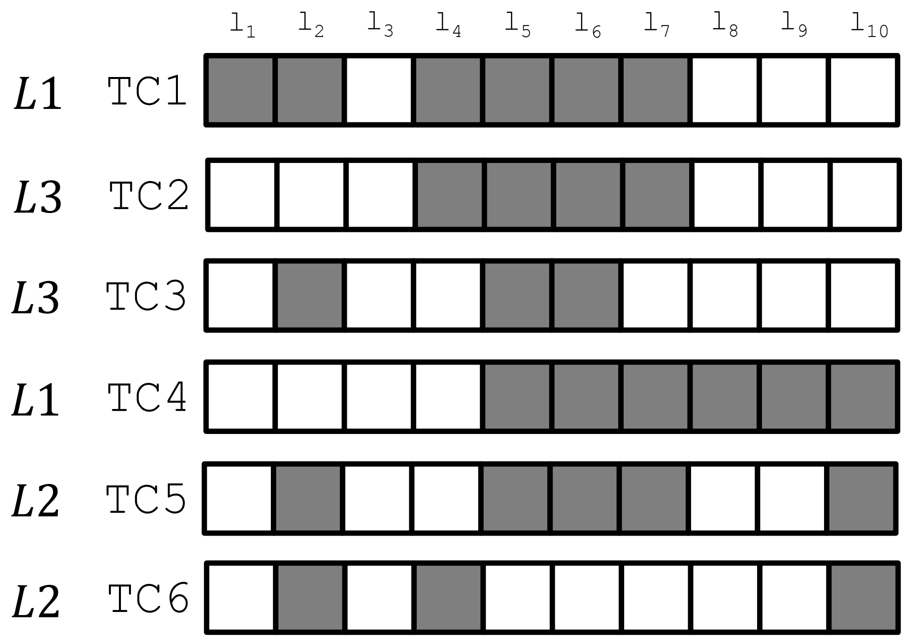
Fitness Evaluation

- The Test Suite is executed and the code coverage of each Test Case is evaluated
- Global Fitness Function
  - Test Suite Fitness
    - Percentage of LOCs covered by the Test Suite
- Local Fitness Function
  - Test Case Fitness
    - Diversity between the sets of LOCs covered by each Test Case

Local Fitness Evaluation

TCx **Test Case**
li **Line of Code**

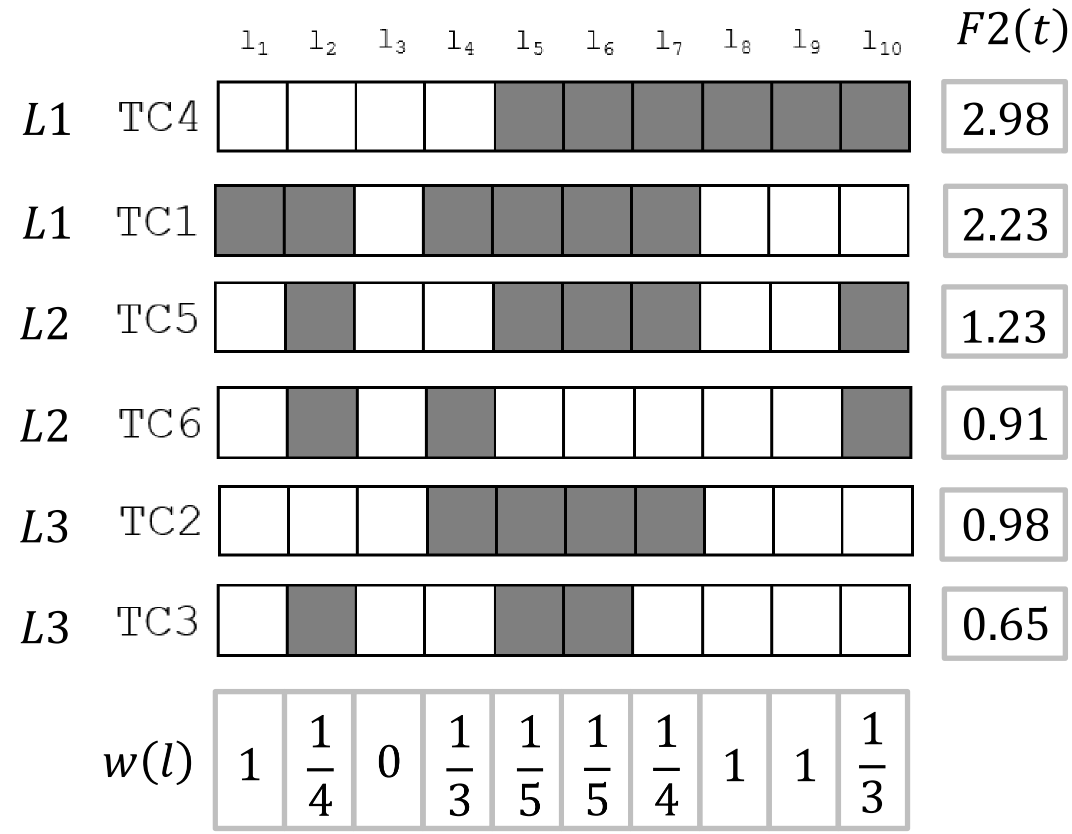
L1, test cases that cover at least a line not covered by other test cases
L2, test cases that cover a set of lines also covered by the union of the coverage sets of other test cases
L3, test cases that cover a set of lines completely covered by at least one different test case



Local Fitness Evaluation

TCx Test Case
li Line of Code

L1, test cases that cover at least a line not covered by other test cases
L2, test cases that cover a set of lines also covered by the union of the coverage sets of other test cases
L3, test cases that cover a set of lines completely covered by at least one different test case



- $F2(t) = \sum_{l \in Cov(t)} w(l)$
  - $Cov(t)$ the set of lines l covered by t
  - $w(l) = \frac{1}{\sum_{u \in T} c(u)}$ the relative weight of the coverage of the line l
  - $c(u) \in \{0, 1\}$, 0 if $l \notin Cov(u)$

Selection Technique

- Test Cases are ranked using the Local Fitness Function

RANK	t	F1	F2
1	TC4	L1	2.98
2	TC1	L1	2.23
3	TC5	L2	1.23
4	TC6	L2	0.91
5	TC2	L3	0.98
6	TC3	L3	0.65

turnover ratio = 1/3

- A set of Test Cases is deleted according to a «turnover ratio»

Combination Technique

- GUI Interface Instances not visited by the GUI Ripping Technique can be discovered by test cases
  - It’s probably possibile to generate a new set of Test Cases
- The discovered GUIs are used as starting points for the GUI Ripping Technique
  - The generated Test Cases are added to the Test Suite

Experimentation

- RQ: Are the test suites generated by the proposed technique more effective that the ones generated by the considered GUI Ripping Technique?

- Effectiveness
  - $\eta(T) = 100 * \frac{\left|\bigcup_{t \in T} Cov(t)\right|}{|LOC|}$

Subjects & Configuration

	Application	LOCs	Activities
AUT1	AardDict 1.4.1	2308	7
AUT2	TomDroid 0.7.1	4167	10
AUT3	OmniDroid 0.2.1	6770	16
AUT4	AlarmClock 1.7	2320	5
AUT5	BookWorm 1.0.18	3190	10

Parameter	Value
Crossover ratio	20%
Mutation Ratio	5%
Number of Iterations	30

Results

	GUI Ripper	AGRippin		
	$\eta(t)$	$\mu(\eta(t))$	$\sigma(\eta(t))$	Discovered Interfaces
AUT1	43.07%	67.10%	0.26%	1
AUT2	28.08%	32.61%	2.45%	0
AUT3	51.58%	58.31%	2.28%	0
AUT4	66.90%	68.00%	1.21%	0
AUT5	40.34%	47.22%	0.45%	1

- We repeated each experiment with AGrippin six times
  - Six different random seeds
- For each application we noticed an increase in the effectiveness $\eta(T)$

Conclusions

- We proposed a novel search based testing technique
- We evaluated its effectiveness by carrying out a case study involving five Android Applications comparing it to a GUI Ripping Technique

Future Works

- Extend the experimentation with a larger set of applications and techniques
- Study the influence of the configuration parameters on the effectiveness of the generated Test Suite
- Implement variants of the Genetic Algorithm Implementation we proposed

Thank you !!!