Recent Research on Search Based Software Testing: Part 2

Francisco Chicano

University of Málaga, Spain (assistant professor)

Colorado State University, USA (faculty affiliate)

Test Suite Minimization in Regression Testing

F. Arito et al., SSBSE 2012

Test Suite Minimization

Given:

- **A set of test cases $T = \{t_1, t_2, ..., t_n\}$**
- **A set of program elements to be covered (e.g., branches) $E = \{e_1, e_2, ..., e_k\}$**
- **A coverage matrix**

M=

	e_1	e_2	e_3	...	e_k
t_1	1	0	1	…	1
t_2	0	0	1	…	0
…	…	…	…	…	…
t_n	1	1	0	…	0

$$m_{ij} = \begin{cases} 1 & \text{if element } e_j \text{ is covered by test } t_i \\ 0 & \text{otherwise} \end{cases}$$

Find a subset of tests $X \subseteq T$ maximizing coverage and minimizing the testing cost

$$minimize \quad cost(X) = \sum_{\substack{i=1 \\ t_i \in X}}^{n} c_i$$

Yoo & Harman

$$maximize \quad cov(X) = |\{e_j \in \mathcal{E} | \exists t_i \in X \text{ with } m_{ij} = 1\}|$$

NP-hard Problems

In many papers we can read…

"Our optimization problem is NP-hard, and for this reason we use…

- **Metaheuristic techniques**
- **Heuristic algorithms**
- **Stochastic algorithms**

… which do not ensure an optimal solution but they are able to find good solutions in a reasonable time."

As far as we know: no efficient (polynomial time) algorithm exists for solving NP-hard problems

But we know "inefficient" algorithms (exponential time in the worst case)

The SATisfiability Problem

Can we find an assignment of boolean values (true and false) to the variables such that all the formulas are satisfied?

$$\neg A \wedge (B \vee C)$$
$$(A \vee B) \wedge (\neg B \vee C \vee \neg D) \wedge (D \vee \neg E)$$
$$A \vee B$$

The first NP-complete problem (Stephen Cook, 1971)

If it can be solved efficiently (polynomial time) then P=NP

The known algorithms solve this problem in exponential time (worst case)

State-of-the-art algorithms in SAT

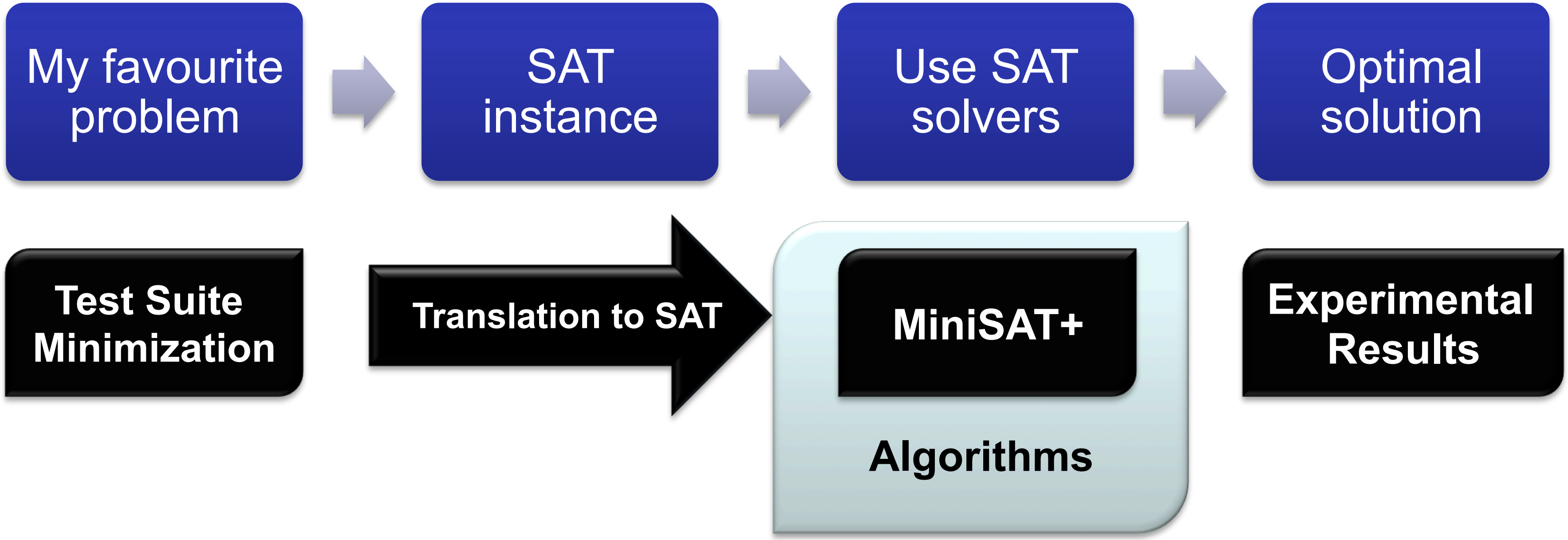
Nowadays, SAT solvers can solve instances with 500 000 boolean variables

This means a search space of $2^{500\,000} \approx 10^{150514}$

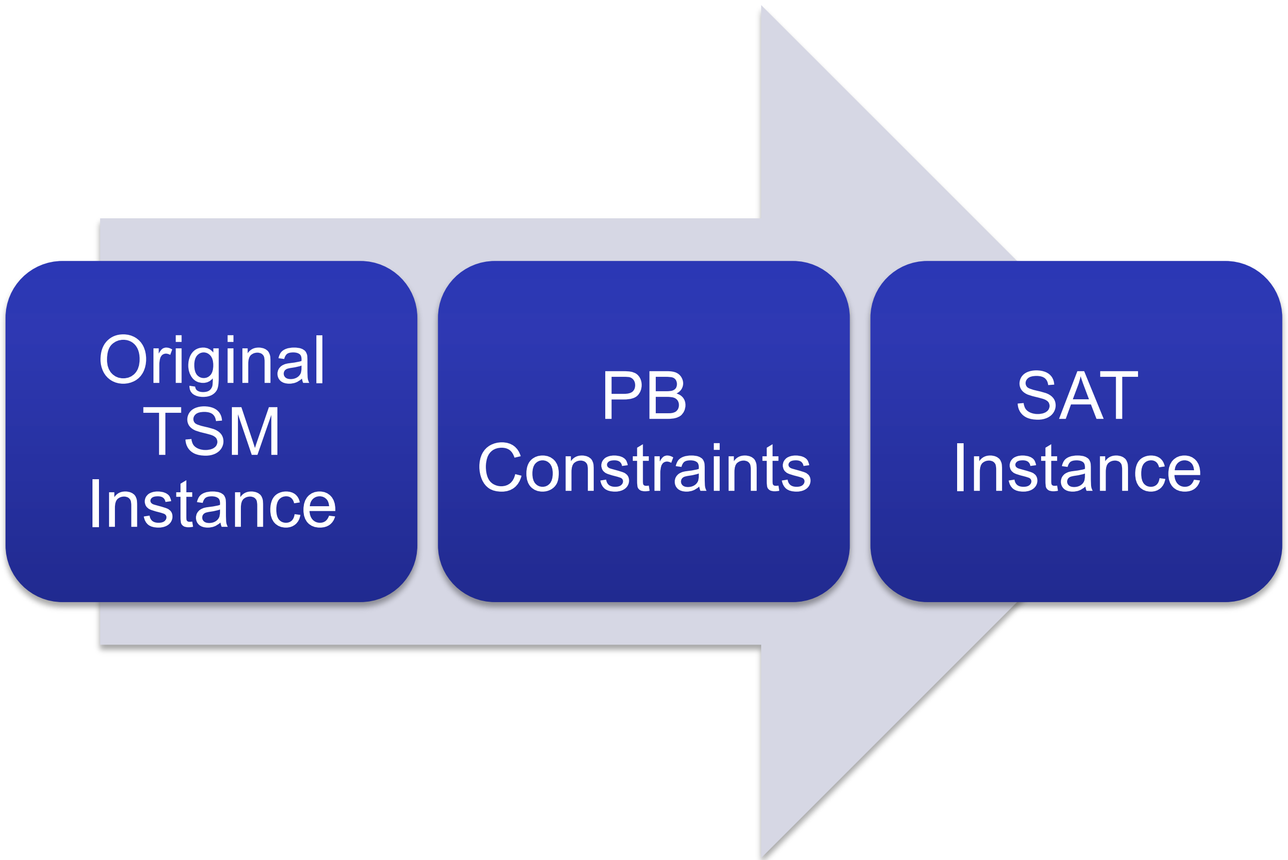
The SATisfiability Problem

Main research question:

Can we use the advances of SAT solvers to solve optimization algorithms up to optimality?



Outline



Pseudo-Boolean Constraints

A Pseudo-Boolean (PB) constraint has the form:

$$\sum_{i=1}^{n} a_i x_i \odot B$$

where

$$\odot \in \{<, \leq, =, \neq, >, \geq\}$$

$$a_i, B \in \mathbb{Z} \quad x_i \in \{0, 1\}$$

Can be translated to SAT instances (usually efficient)

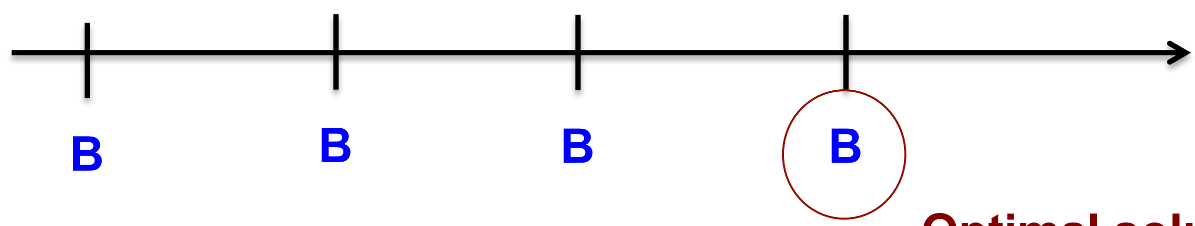
Are a higher level formalism to specify a decision problem

Can be the input for MiniSAT+

Translating Optimization to Decision Problems

Let us assume we want to minimize f(x)

Check	**Check**	**Check**	**Check**
$f(x) \leq B$	$f(x) \leq B$	$f(x) \leq B$	$f(x) \leq B$
no	**no**	**no**	**yes**



Optimal solution found

The same can be done with multi-objective problems, but we need more PB constraints

$$f_1(y) \leq B_1 \qquad f_2(y) \leq B_2 \qquad \cdots \qquad f_m(y) \leq B_m$$

PB Constraints for the TSM Problem

M=

	e_1	e_2	e_3	…	e_m
t_1	1	0	1	…	1
t_2	0	0	1	…	0
…	…	…	…	…	…
t_n	1	1	0	…	0

$$m_{ij} = \begin{cases} 1 & \text{if element } e_j \text{ is covered by test } t_i \\ 0 & \text{otherwise} \end{cases}$$

$$e_j \leq \sum_{i=1}^{n} m_{ij} t_i \leq n \cdot e_j \qquad 1 \leq j \leq m$$

Cost

$$\sum_{i=1}^{n} c_i t_i \leq B$$

Coverage

$$\sum_{j=1}^{m} e_j \geq P$$

Example

	e_1	e_2	e_3	e_4
t_1	1	0	1	0
t_2	1	1	0	0
t_3	0	0	1	0
t_4	1	0	0	0
t_5	1	0	0	1
t_6	0	1	1	0

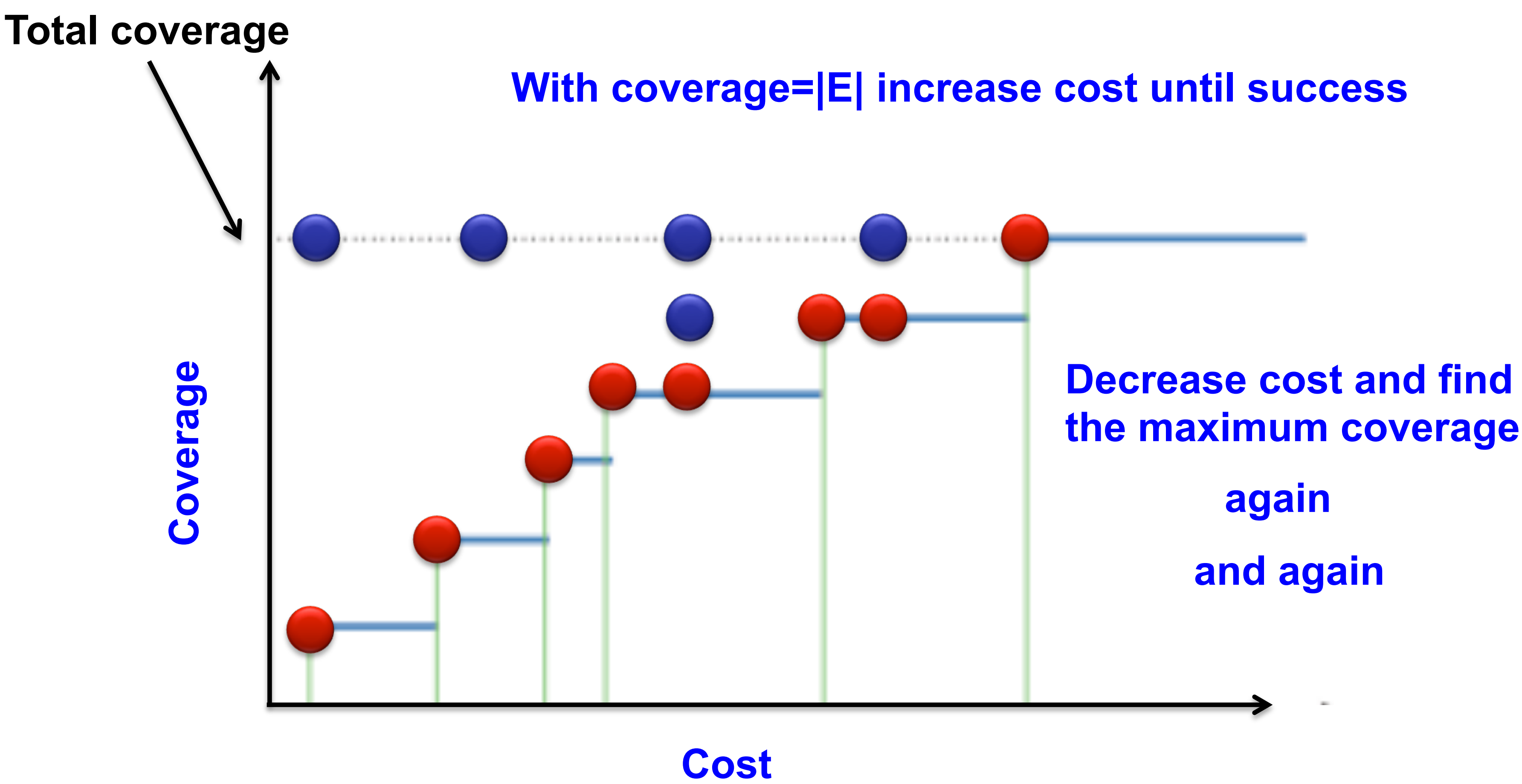
Bi-objective problem

$$\begin{cases} e_1 \leq t_1 + t_2 + t_4 + t_5 \leq 6e_1 \\ e_2 \leq t_2 + t_6 \leq 6e_2 \\ e_3 \leq t_1 + t_3 + t_6 \leq 6e_3 \\ e_4 \leq t_5 \leq 6e_4 \\ t_1 + t_2 + t_3 + t_4 + t_5 + t_6 \leq B \\ e_1 + e_2 + e_3 + e_4 \geq P \end{cases}$$

Single-objective problem (total coverage)

$$\begin{cases} t_1 + t_2 + t_4 + t_5 \geq 1 \\ t_2 + t_6 \geq 1 \\ t_1 + t_3 + t_6 \geq 1 \\ t_5 \geq 1 \\ t_1 + t_2 + t_3 + t_4 + t_5 + t_6 \leq B \end{cases}$$

Algorithm for Solving the 2-obj TSM



TSM Instances

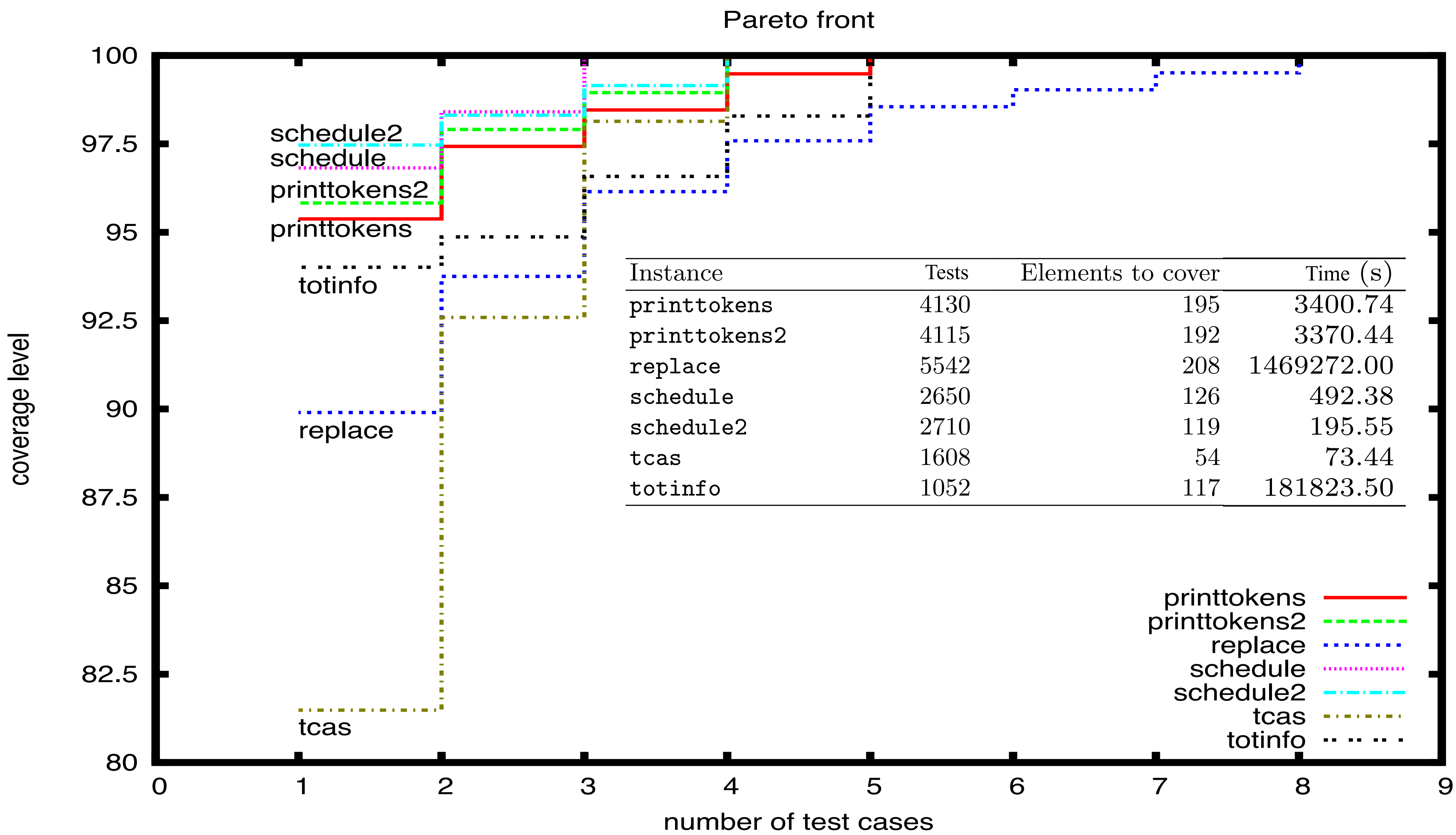
Instances from the Software-artifact Infrastructure Repository (SIR)

`http://sir.unl.edu/portal/index.php`

Instance	Tests	Elements to cover
printtokens	4130	195
printtokens2	4115	192
replace	5542	208
schedule	2650	126
schedule2	2710	119
tcas	1608	54
totinfo	1052	117

Cost of each test: 1

Pareto Front



Instance	Tests	Elements to cover	Time (s)
printtokens	4130	195	3400.74
printtokens2	4115	192	3370.44
replace	5542	208	1469272.00
schedule	2650	126	492.38
schedule2	2710	119	195.55
tcas	1608	54	73.44
totinfo	1052	117	181823.50

Pareto Front

Instance	Elements	Tests	Coverage	Solution
printtokens	195	5	100%	$(t_{2222}, t_{2375}, t_{3438}, t_{4100}, t_{4101})$
	194	4	99.48%	$(t_{1908}, t_{2375}, t_{4099}, t_{4101})$
	192	3	98.46%	$(t_{1658}, t_{2363}, t_{4072})$
	190	2	97.43%	(t_{1658}, t_{3669})
	186	1	95.38%	(t_{2597})
printtokens2	192	4	100%	$(t_{2521}, t_{2526}, t_{4085}, t_{4088})$
	190	3	98.95%	$(t_{457}, t_{3717}, t_{4098})$
	188	2	97.91%	(t_{2190}, t_{3282})
	184	1	95.83%	(t_{3717})
replace	208	8	100%	$(t_{306}, t_{410}, t_{653}, t_{1279}, t_{1301}, t_{3134}, t_{4057}, t_{4328})$
	207	7	99.51%	$(t_{309}, t_{358}, t_{653}, t_{776}, t_{1279}, t_{1795}, t_{3248})$
	206	6	99.03%	$(t_{275}, t_{290}, t_{1279}, t_{1938}, t_{2723}, t_{2785})$
	205	5	98.55%	$(t_{426}, t_{1279}, t_{1898}, t_{2875}, t_{3324})$
	203	4	97.59%	$(t_{298}, t_{653}, t_{3324}, t_{5054})$
	200	3	96.15%	$(t_{2723}, t_{2901}, t_{3324})$
	195	2	93.75%	(t_{358}, t_{5387})
	187	1	89.90%	(t_{358})
schedule	126	3	100%	$(t_{1403}, t_{1559}, t_{1564})$
	124	2	98.41%	(t_{1570}, t_{1595})
	122	1	96.82%	(t_{1572})
schedule2	119	4	100%	$(t_{2226}, t_{2458}, t_{2462}, t_{2681})$
	118	3	99.15%	$(t_{101}, t_{1406}, t_{2516})$
	117	2	98.31%	(t_{2461}, t_{2710})
	116	1	97.47%	(t_{1584})
tcas	54	4	100%	$(t_{5}, t_{1191}, t_{1229}, t_{1608})$
	53	3	98.14%	$(t_{13}, t_{25}, t_{1581})$
	50	2	92.59%	(t_{72}, t_{1584})
	44	1	81.48%	(t_{217})
totinfo	117	5	100%	$(t_{62}, t_{118}, t_{218}, t_{1000}, t_{1038})$
	115	4	98.29%	$(t_{62}, t_{118}, t_{913}, t_{1016})$
	113	3	96.58%	$(t_{65}, t_{216}, t_{913})$
	111	2	94.87%	(t_{65}, t_{919})
	110	1	94.01%	(t_{179})

Reduction in the Number of Test Cases

Since we are considering cost 1 for the tests, we can apply an a priori reduction in the original test suite

	e_1	e_2	e_3	…	e_m
t_1	1	0	0	…	1
t_2	1	0	1	…	1
…	…	…	…	…	…
t_n	1	1	0	…	0

Test t_1 can be removed

Instance	Original Size	Reduced Size	Elements to cover
`printtokens`	4130	40	195
`printtokens2`	4115	28	192
`replace`	5542	215	208
`schedule`	2650	4	126
`schedule2`	2710	13	119
`tcas`	1608	5	54
`totinfo`	1052	21	117

Results with the Reduction

The optimal Pareto Front for the reduced test suite can be found from 200 to 180 000 times faster

	Original (s)	Reduced (s)
printtokens	3400.74	2.17
printtokens2	3370.44	1.43
replace	1469272.00	345.62
schedule	492.38	0.24
schedule2	195.55	0.27
tcas	73.44	0.33
totinfo	181823.50	0.96

Software Product Lines Testing

R. Lopez-Herrejon et al., ICSM 2013

Software Product Lines

A product line is a set of related products developed from a shared set of assets

- **The products have similar characteristics**
- **The products have unique characteristics**

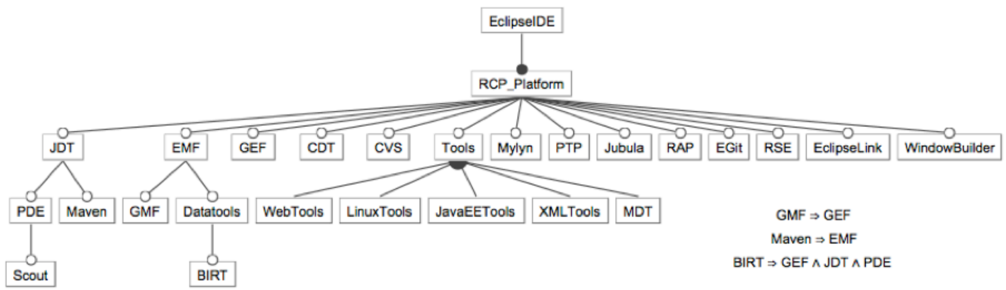
Advantages

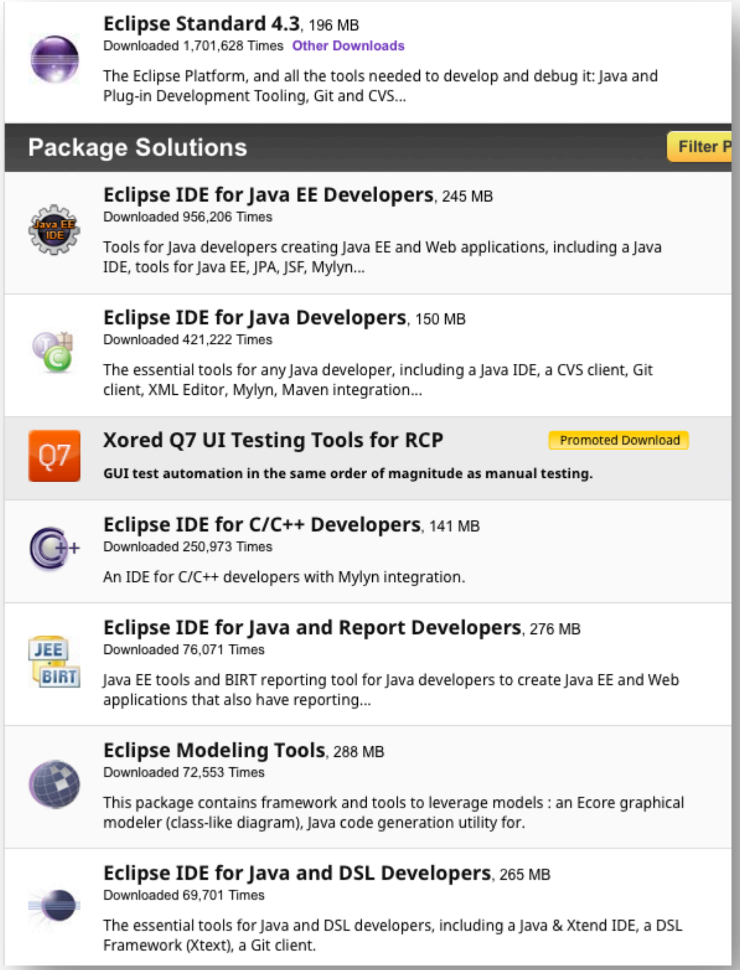
- **Support customization**
- **Improves reuse**
- **Reduce time to market**

Software Product Lines

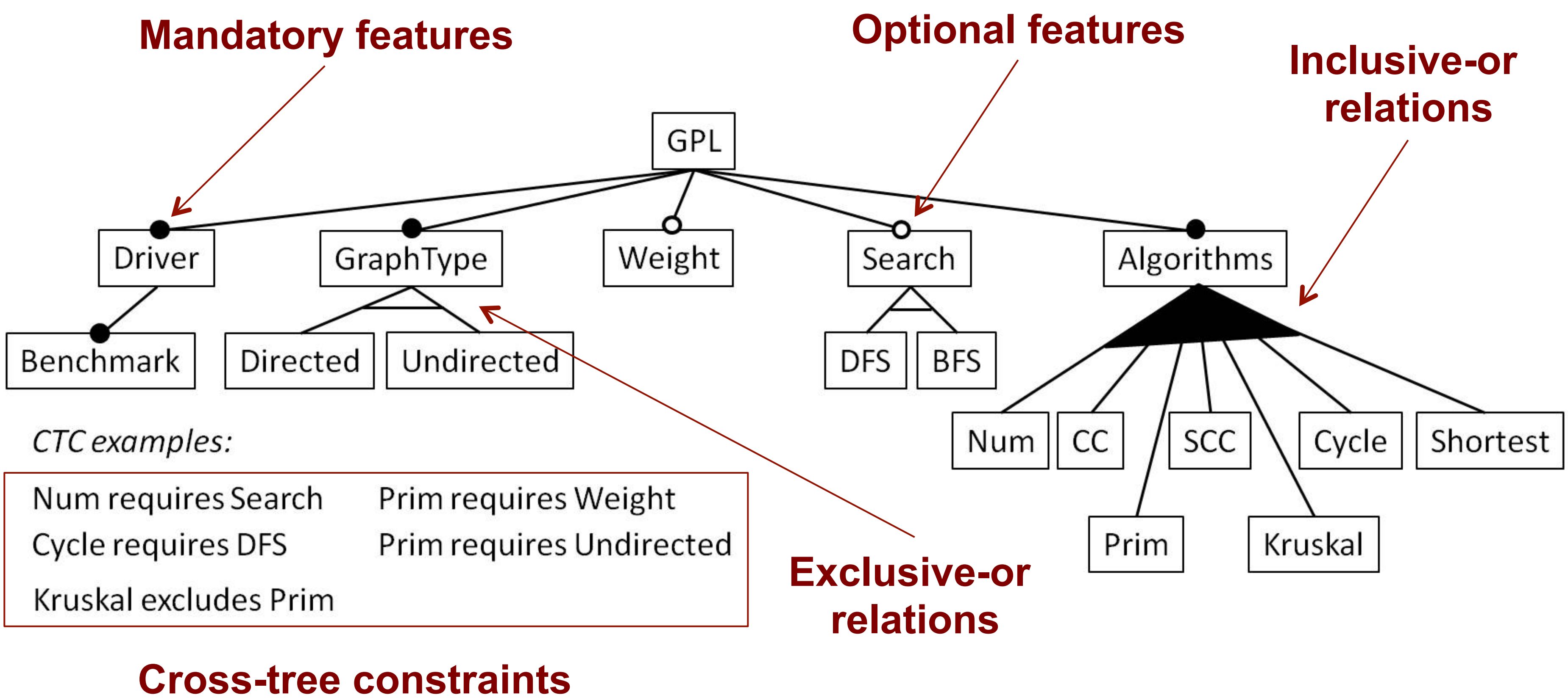
In Software Product Lines the product is Software

They are modelled using Feature Models





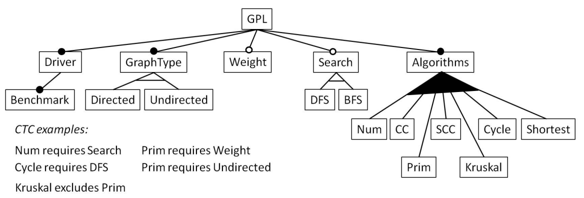
Feature Models



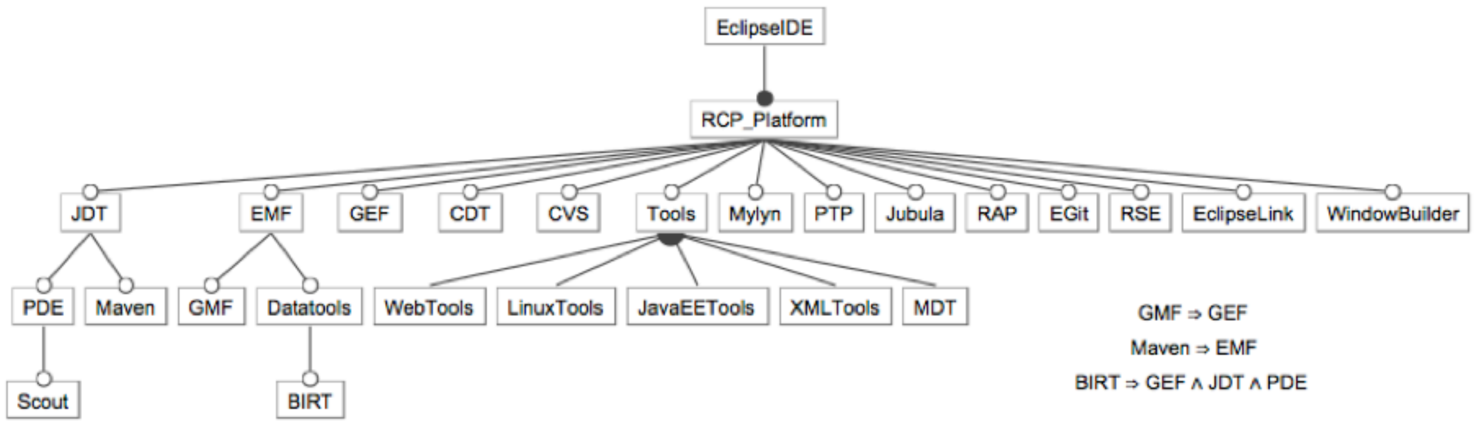
Graph Product Line Feature Model

Testing of Software Product Lines

The GPL Feature Model is small: 73 distinct products



But the number of products grows exponentially with the number of features…



… and testing each particular product is not viable

Testing of SPLs: Combinatorial Interaction Testing

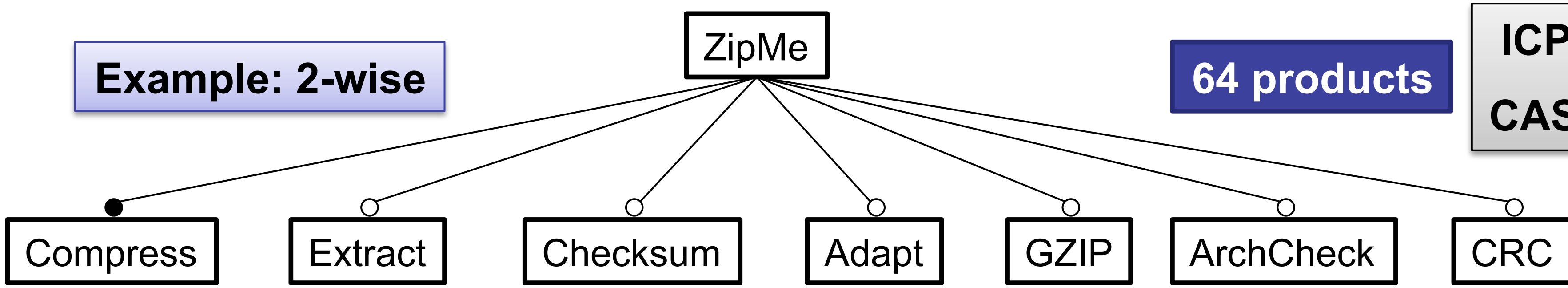
Assuming each feature has been tested in isolation, most of the defects come from the interaction between features

Combinatorial Interaction Testing consists in selecting the minimum number of products that covers all *t*-wise interactions (*t*-wise coverage).

Example: 2-wise

64 products

ICPL
CASA



	Compress	Extract	Checksum	Adapt	GZIP	ArchCheck	CRC
1	✔		✔	✔	✔	✔	
2	✔	✔	✔				
3	✔	✔			✔	✔	
4	✔	✔		✔			✔
5	✔					✔	✔
6	✔		✔		✔		✔

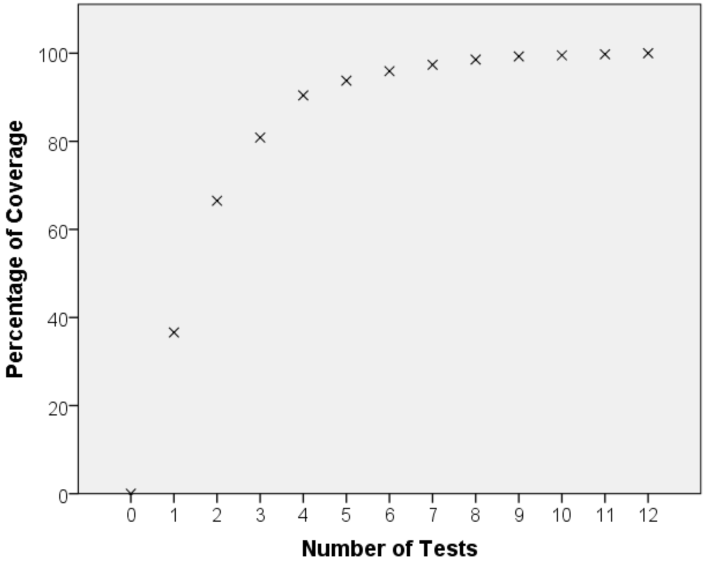
Testing of SPLs: Multi-Objective Formulation

If we don't have the resources to run all the tests, which one to choose?

Multi-objective formulation:

- **minimize the number of products**
- **maximize the coverage (t-wise interactions)**

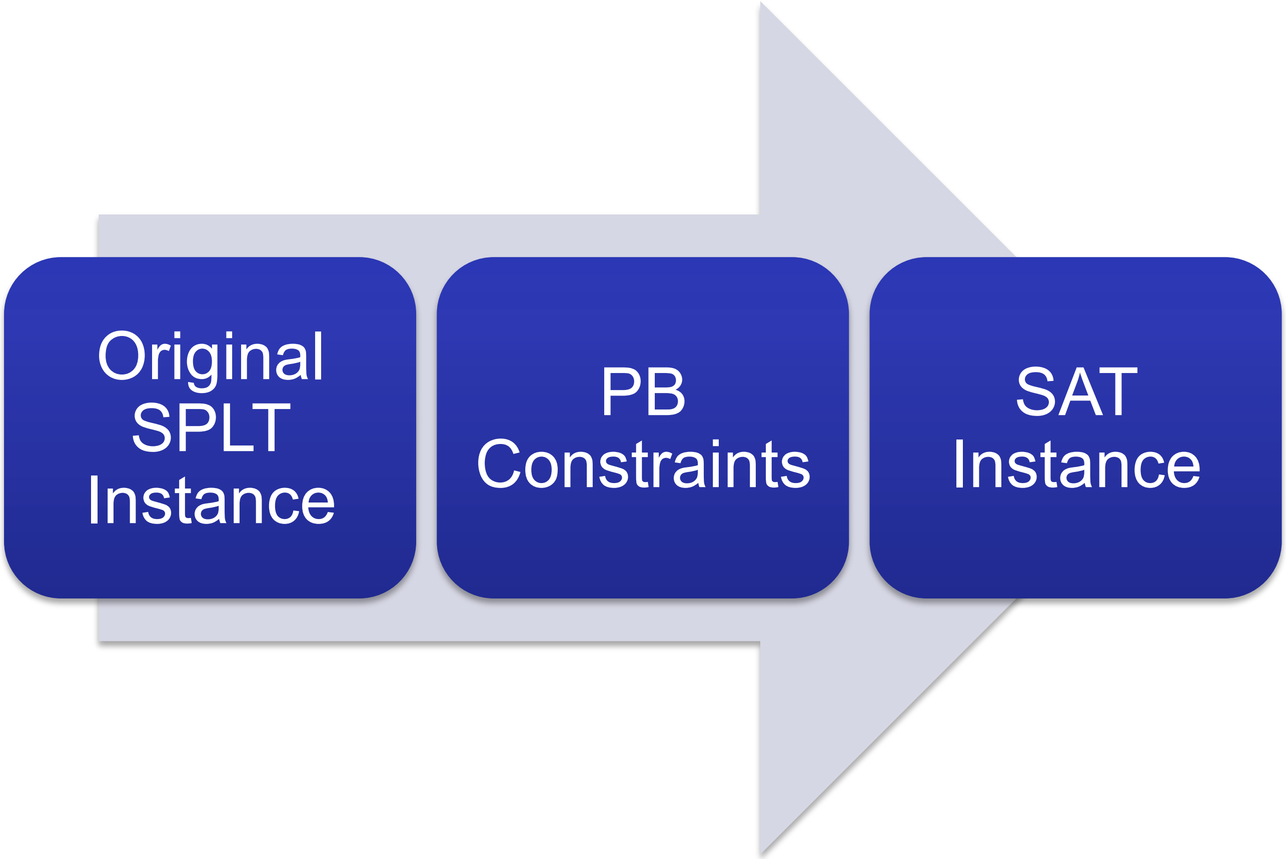
The solution is not anymore a table of products, but a Pareto set



GPL

2-wise interactions

Testing of SPLs: Approach



Testing of SPLs: Approach

Modelling SPLT using PseudoBoolean constraints

Variable	Meaning
$x_{p,i}$	Presence of feature i in product p
$c_{p,i,j,k}$	Product p covers the pair (i,j) with signature k
$d_{i,j,k}$	The pair (i,j) with signature k is covered by some product

k takes values 0, 1, 2 and 3.

All the variables are boolean {0,1}

The values of the signature are:

- **00 (both unselected)**
- **10 (only first selected)**
- **01 (only second selected)**
- **11 (both selected)**

Testing of SPLs: Approach

Equations of the model

- **For each product *p***
  - **Constraints imposed by the Feature Model**
- **For each product *p* and pair of features *i* and *j***

$$2c_{p,i,j,3} \leq x_{p,i} + x_{p,j} \leq 1 + c_{p,i,j,3}$$

$$2c_{p,i,j,2} \leq x_{p,i} + (1 - x_{p,j}) \leq 1 + c_{p,i,j,2}$$

$$2c_{p,i,j,1} \leq (1 - x_{p,i}) + x_{p,j} \leq 1 + c_{p,i,j,1}$$

$$2c_{p,i,j,0} \leq (1 - x_{p,i}) + (1 - x_{p,j}) \leq 1 + c_{p,i,j,0}$$

Testing of SPLs: Approach

Equations of the model (cont.)

- **For each pair of features *i* and *j* and signature *k***

$$d_{i,j,k} \leq \sum_{p} c_{p,i,j,k} \leq n d_{i,j,k}$$

  - ***n* is the number of products**

- **Objective: maximize coverage**

$$max : \sum_{i,j,k} d_{i,j,k}$$

Testing of SPLs: Approach

Algorithm 1 Algorithm for obtaining the optimal Pareto set.

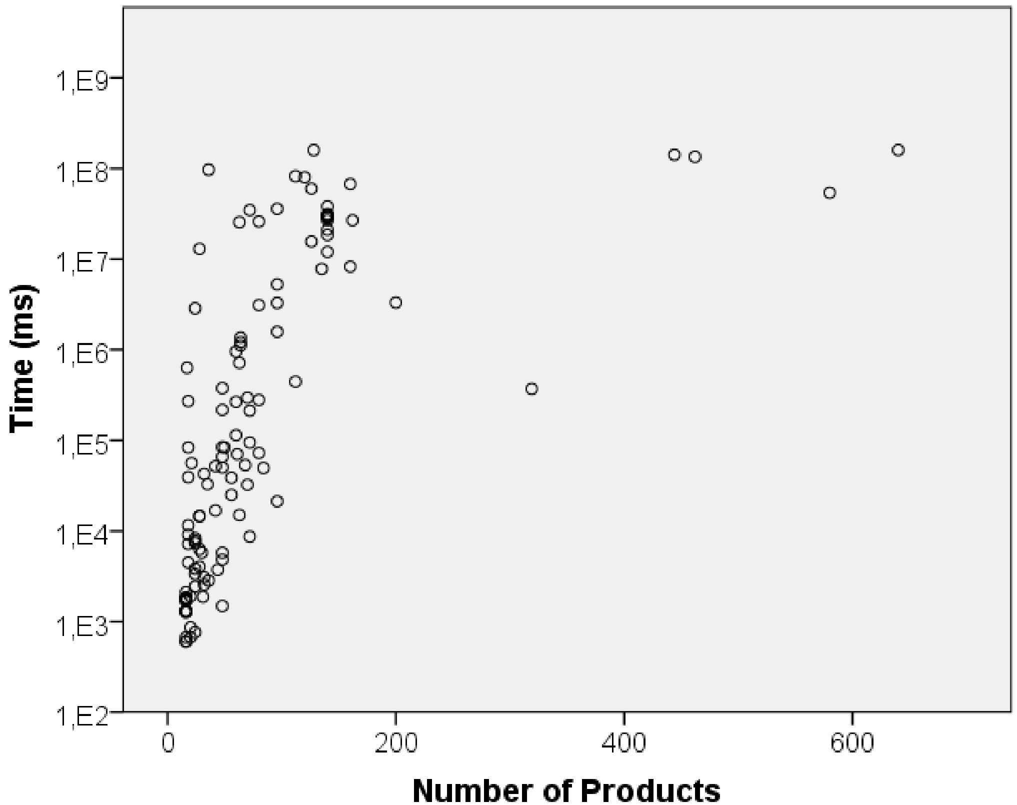
$optimal\_set \leftarrow \{\emptyset\}$;
$cov[0] \leftarrow 0$;
$cov[1] \leftarrow C_2^f$;
$sol \leftarrow$arbitraryValidSolution(fm);
$i \leftarrow 1$;
while $cov[i] \neq cov[i-1]$ **do**
  $optimal\_set \leftarrow optimal\_set \cup \{sol\}$;
  $i \leftarrow i + 1$;
  $m \leftarrow$prepareMathModel(fm,i);
  $sol \leftarrow$solveMathModel(m);
  $cov[i] \leftarrow |sol|$;
end while

Testing of SPLs: Results

Experiments on 118 feature models taken from

SPLOT repository (http://www.splot-research.org)

SPL Conqueror (http://wwwiti.cs.uni-magdeburg.de/~nsiegmun/SPLConqueror/)



16 to 640 products

Intel Core2 Quad Q9400

2.66 GHz, 4 GB

Prioritized Pairwise Testing in Software Product Lines

R. Lopez-Herrejon et al., GECCO 2014

Our contributions

- Formalization of prioritization testing scheme proposed by Johansen et al.
- Implementation with the **P**arallel **P**rioritized product line **G**enetic **S**olver (PPGS)
- Comprehensive evaluation and comparison against greedy approach.

Prioritization Motivation

- Key ideas
  - Each feature combination represents an important product of the SPL
  - For each relevant product give a positive integer value that reflects the priority of the product
    - Market importance
    - Implementation costs
    - ...

Feature List and Feature Set

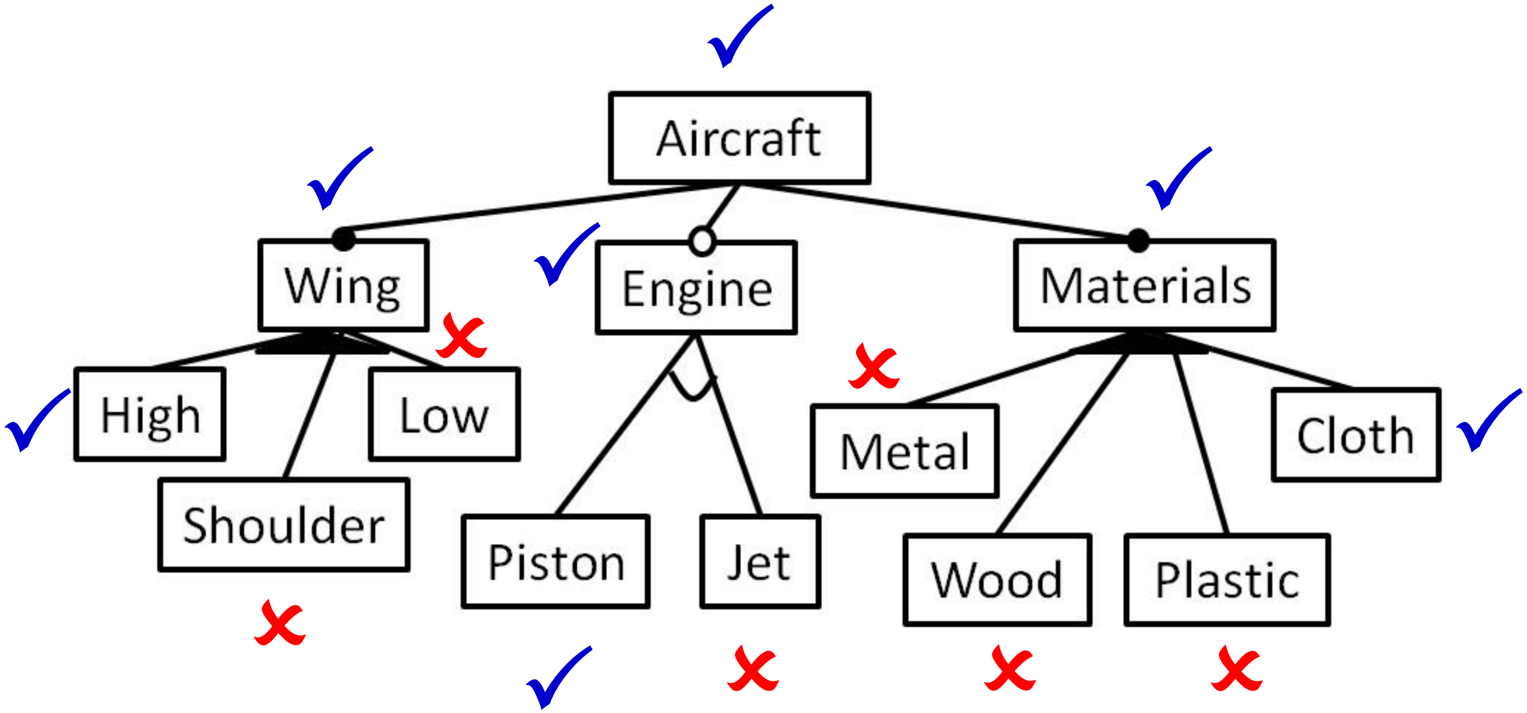
Definition 1. *Feature List (FL) is the list of features in a feature model.*

Definition 2. *Feature Set (FS) is a 2-tuple* $[sel, \overline{sel}]$ *where* sel *and* $\overline{sel}$ *are respectively the set of selected and not-selected features of a member product. Let FL be a feature list, thus* $sel, \overline{sel} \subseteq FL$, $sel \cap \overline{sel} = \emptyset$, *and* $sel \cup \overline{sel} = FL$. *The terms* $p.sel$ *and* $p.\overline{sel}$ *respectively refer to the set of selected and unselected features of product* p.

- Example Feature List (FL)

  Aircraft, Wing, Engine, Materials, High, Shoulder, Low, Piston, Jet, Metal, Wood, Plastic, Cloth

Feature Set Example



Selected = {Aircraft, Wing, High, Engine, Piston, Materials, Cloth}

Unselected = {Shoulder, Low, Jet, Metal, Wood, Plastic}

Terminology (3)

Definition 3. *A feature set fs is valid in feature model fm, i.e.* `valid(fs, fm)` *holds, iff fs does not contradict any of the constraints introduced by fm.*

- Examples of valid feature sets
  - **A**ircraft, **Wi**ng, **En**gine, **Ma**terials, **H**igh, **S**houlder, **L**ow, **Pi**ston, **J**et, **Me**tal, **Wo**od, **Pl**astic, **C**loth

Prod	A	Wi	E	Ma	H	S	L	Pi	J	Me	Wo	Pl	C
p0	✓	✓	✓	✓	✓			✓				✓	
p1	✓	✓	✓	✓	✓			✓					✓
p2	✓	✓	✓	✓	✓				✓			✓	
p3	✓	✓	✓	✓	✓				✓				✓
p4	✓	✓		✓	✓						✓	[illegible]	[illegible]
p5	✓	✓	✓	✓	✓			✓			✓	[illegible]	[illegible]
p6	✓	✓	✓	✓	✓			✓		✓	[illegible]	[illegible]	[illegible]
p7	✓	✓	✓	✓	✓				✓	✓	[illegible]	[illegible]	[illegible]

315 valid feature sets

Prioritized Product

Definition 4. *A prioritized product* **pp** *is a 2-tuple [***fs***, w], where* **fs** *represents a valid feature set in feature model* **fm** *and $w \in \mathbb{R}$ represents its weight.* Let pp_i and pp_j be two prioritized products. We say that pp_i has higher priority than pp_j for test-suite generation iff pp_i's weight is greater than pp_j's weight, that is $pp_i.w > pp_j.w$.

- Example

Prod	A	Wi	E	Ma	H	S	L	Pi	J	Me	Wo	Pl	C
p0	✓	✓	✓	✓	✓			✓				✓	
p1	✓	✓	✓	✓	✓			✓					✓

pp1 = [p1, 17]

Pairwise configuration

Definition 5. *A pairwise configuration pc is a 2-tuple [sel, $\overline{sel}$] representing a partially configured product, defining the selection of 2 features of feature list FL, i.e.* $pc.sel \cup pc.\overline{sel} \subseteq FL \wedge pc.sel \cap pc.\overline{sel} = \emptyset \wedge |pc.sel \cup pc.\overline{sel}| = 2$. *We say a pairwise configuration pc is covered by feature set fs iff* $pc.sel \subseteq fs.sel \wedge pc.\overline{sel} \subseteq fs.\overline{sel}$.

Prod	A	Wi	E	Ma	H	S	L	Pi	J	[illegible]	[illegible]	[illegible]	[illegible]
p0	✓	✓	✓	✓	✓			✓			[illegible]	[illegible]	[illegible]
p1	✓	✓	✓	✓	✓			✓			[illegible]	[illegible]	[illegible]
p2	✓	✓	✓	✓	✓				✓		[illegible]	✓	
p3	✓	✓	✓	✓	✓				✓				✓
p4	✓	✓		✓	✓						✓		
p5	✓	✓	✓	✓	✓			✓			✓		
p6	✓	✓	✓	✓	✓			✓		✓			
p7	✓	✓	✓	✓	✓				✓	✓			

240 pairwise configurations

pc1=[{Plastic},{Cloth}]

pc2=[{High, Wood},{}]

Weighted Pairwise Configuration

Definition 6. *A weighted pairwise configuration wpc is a 2-tuple [pc,w] where pc is a pairwise configuration and* $w \in \mathbb{R}$ *represents its weight computed as follows. Let PP be a set of prioritized products and* PP_{pc} *be a subset,* $PP_{pc} \subseteq PP$*, such that* PP_{pc} *contains all prioritized products in PP that cover pc of wpc, i.e.* $PP_{pc} = \{pp \in PP | pp.fs \text{ covers } wpc.pc\}$*. Then* $w = \sum_{p \in PP_{pc}} p.w$

pc1=[{Plastic},{Cloth}]

Prod	A	Wi	E	Ma	H	S	L	Pi	J	Me	Wo	Pl	C	weights
p0	✓	✓	✓	✓	✓			✓				✓		17
p1	✓	✓	✓	✓	✓			✓					✓	17
p2	✓	✓	✓	✓	✓				✓			✓		15
p3	✓	✓	✓	✓	✓				✓				✓	15
p4	✓	✓		✓	✓						✓			13
p5	✓	✓	✓	✓	✓			✓			✓			13
p6	✓	✓	✓	✓	✓			✓		✓				6
p7	✓	✓	✓	✓	✓				✓	✓				6

wpc1.w= pp0.w + pp2.w = 17 + 15 = 32

Prioritized Pairwise Covering Array

Definition 7. *A prioritized pairwise covering array ppCA for a feature model fm and a set of weighted pairwise configurations WPC is a set of valid feature sets FS that covers all weighted pairwise configurations in WPC whose weight is greater than zero:* $\forall wpc \in WPC\ (wpc.w > 0 \Rightarrow \exists fs \in ppCA$ *such that fs covers wpc.pc).*

- Example of ppCA

A	Wi	E	Ma	H	S	L	Pi	J	Me	Wo	Pl	C
✓	✓	✓	✓	✓			✓					✓
✓	✓	✓	✓	✓				✓			✓	
✓	✓	✓	✓	✓			✓			✓		
✓	✓	✓	✓	✓				✓	✓	✓		✓
✓	✓	✓	✓		✓		✓		✓		✓	
✓	✓		✓	✓						✓		

p1, p2, p5 (rows 1–3)

new products (rows 4–6)

Challenge: Find a ppCA with the minimum number of feature sets

PPGS Algorithm

Algorithm 1: Pseudocode of PPGS.

```
proc Input:feature model FM, prioritized products prods
TS ← ∅    // Initialize the test suite
RP ← weighted_pairs_to_cover(prods)
while not empty(RP) do
   t=0
   P(t) ← Create_Population() // P = population
   while  evals < totalEvals do
      Q ← ∅    // Q = auxiliary population
      for i ← 1 to (PPGS.popSize / 2) do
         parents←Selection(P(t))
         offspring←Recombination(PPGS.Pc,parents)
         offspring←Mutation(PPGS.Pm,offspring)
         Fix(offspring)
         ParallelEvaluator.addSolution(offspring)
      end for
      solutions←ParallelEvaluator.evaluate();
      Insert(solutions,Q)
      P(t+1) := Replace (Q,P(t))
      t= t + 1
   end while   //internal loop
   TS ← TS ∪ best_solution(P(t))
   RemovePairs(RP, best_solution(P(t)))
end while   //external loop
return TS
end_proc
```

Parameter setting

Parameter	Setting
Crossover type	one-point
Crossover probability	0.8
Selection strategy	binary tournament
Population size	10
Mutation probability	0.1
Termination condition	1000 evaluations

Implemented in jMetal framework

Evaluation

- Compared against Prioritized-ICPL (pICPL)
  - Proposed by Johansen et al. (2012)
  - Uses data parallelization
- Three different weight priority assignment methods
- Different percentages of selected products
  - Ranging from 5% upto 50%

Weight priority assignment methods

1. Measured values
   - 16 real SPL examples
   - Code and feature model available
   - Non-functional properties measured (e.g. footprint)
2. Ranked-based values
   - Based on how dissimilar two products are
   - More dissimilar higher chances of covering more pairs
3. Random values
   - [Min..Max] range

SPL Name	Prop	NF	NP	NC	PP%
Prevayler	F	6	32	24	75.0
LinkedList	F	26	1440	204	14.1
ZipMe	F	8	64	64	100.0
PKJab	F	12	72	72	100.0
SensorNetwork	F	27	16704	3240	19.4
BerkeleyDBF	F	9	256	256	100.0
Violet	F	101	≈ 1E20	101	≈ 0.0
Linux subset	F	25	≈ 3E24	100	≈ 0.0
LLVM	M	12	1024	53	5.1
Curl	M	14	1024	68	6.6
x264	M	17	2048	77	3.7
Wget	M	17	8192	94	1.15
BerkeleyDBM	M	19	3840	1280	33.3
SQLite	M	40	≈ 5E7	418	≈ 0.0
BerkeleyDBP	P	27	1440	180	12.50
Apache	P	10	256	192	75.0

Footprint, **M**ain memory consumption, **P**erformance, **N**umber of **F**eatures, **N**umber of **P**roducts, **N**umber of **C**onfigurations, **P**ercentage of **P**rioritized products.

Experimental corpus

	G1	G2	G3	Summary
Number Feature Models	160	59	16	235
Number Products	16-1K	1K-80K	32-≈3E24	16-≈3E24
Number Features	10-56	14-67	6-101	6-101
Weight Priority Assignment RK Ranked-Based, RD Random, M Measured	RK,RD	RK,RD	M	
Prioritized Products Percentage	20,30,50	5,10,20	≈0.0 - 100	
Problem Instances	960	354	16	1330

Problem instances G1 = 160 fm X 2 priority assig. X 3 percentages = 960

Problem instances G2 = 59 fm X 2 priority assig. X 3 percentages = 354

Problem instances G3 = 16 fm X 1 priority assig. = 16

Total independent runs = 1330 X 2 algorithms x 30 indep. runs = 79,800

Wilcoxon Test (1)

- Confidence level 95%
- We show the mean and standard deviation of number of products required to cover 50% upto 100% of the total weighted coverage
- We highlight where the difference is statistically significant

Group G1 – less than 1000 products

Cov.	PPGS	pICPL	Cov.	PPGS	pICPL
50%	$1.20_{0.40}$	$1.20_{0.40}$	96%	$4.00_{1.23}$	$4.37_{1.42}$
75%	$1.92_{0.51}$	$1.98_{0.58}$	97%	$4.38_{1.32}$	$4.71_{1.54}$
80%	$2.15_{0.59}$	$2.25_{0.68}$	98%	$4.83_{1.46}$	$5.18_{1.74}$
85%	$2.47_{0.72}$	$2.58_{0.81}$	99%	$5.58_{1.71}$	$5.87_{1.99}$
90%	$2.88_{0.86}$	$3.13_{1.03}$	100%	$7.56_{2.85}$	$7.56_{3.03}$
95%	$3.72_{1.14}$	$4.06_{1.33}$	TIME	23897_{28669}	10116_{18842}

PPGS smaller size

pICPL faster

Wilcoxon Test (2)

Group G2 – from 1,000 to 80,000 products

Cov.	PPGS	pICPL	Cov.	PPGS	pICPL
50%	$1.16_{0.36}$	$1.36_{0.83}$	96%	$4.98_{0.97}$	$5.83_{3.14}$
75%	$2.09_{0.42}$	$2.47_{1.65}$	97%	$5.55_{1.10}$	$6.43_{3.27}$
80%	$2.39_{0.52}$	$2.86_{1.79}$	98%	$6.34_{1.34}$	$7.23_{3.48}$
85%	$2.73_{0.59}$	$3.27_{2.08}$	99%	$7.66_{1.88}$	$8.59_{4.11}$
90%	$3.36_{0.76}$	$3.98_{2.38}$	100%	$14.57_{10.65}$	$13.79_{9.98}$
95%	$4.59_{0.90}$	$5.42_{3.12}$	TIME	$273728_{7.2E+5}$	$638164_{2.1E+6}$

- PPGS yields test suites of smaller sizes
- PPGS performs faster than pICPL

Wilcoxon Test (3)

Group G3 – Measured Values, 32 to ≈3E24 products

Model	Alg.	50%	75%	80%	85%	90%	95%	96%	97%	98%	99%	100%	TIME
Apache	PPGS	2	3	3	4	4	6	6	6	7	7	7	10394
	pICPL	2	3	3	4	5	6	7	7	7	8	8	7582
Berk.DBF	PPGS	2	4	4	5	5.97	6.97	6.97	6.97	7.97	8	8.17	11213
	pICPL	2	4	5	6	7	8	8	8	8	9	9	8152
Berk.DBM	PPGS	2	3	3	4	4.73	6.87	7.80	8.77	9.97	11.90	23.33	117607
	pICPL	2	3	3	4	6	7	8	8	10	11	21	94512
Berk.DBP	PPGS	1	2	2	3	3	4	4.83	5	5.93	7	10.60	47361
	pICPL	1	2	3	3	4	6	6	6	6	7	12	57291
Curl	PPGS	2	3	3	3.97	4.03	5.83	6	6.50	7.37	8.07	9.63	17454
	pICPL	2	3	3	4	4	6	6	6	7	7	8	6382
LinkedList	PPGS	1	2	2	2	3	4.23	5	5	6.13	7.79	13.37	60684
	pICPL	1	2	2	3	3	4	4	5	7	11	14	71151
Linux	PPGS	2	4	4	5	6	7	7.67	8	8.37	9.40	11.10	49385
	pICPL	2	4	5	5	6	8	8	8	8	9	10	30522
LLVM	PPGS	2	3	3.03	4	5	6	6	6.07	7	8	8.17	12805
	pICPL	2	3	4	4	5	6	7	7	7	8	8	9032
PKJab	PPGS	1	2	2	3	3.07	4	5	5	5	6	7	11439
	pICPL	1	2	3	3	3	5	5	6	7	8	8	4661
Prevayler	PPGS	2	3	3	3	4	5	5	5.60	6	6	6	8091
	pICPL	2	3	3	3	4	5	5	5	6	6	6	2412
S.Network	PPGS	1	3	3	3	4	5.03	5.47	6	6.97	7.87	13.97	71971
	pICPL	1	3	4	5	6	8	9	9	10	11	17	74181
SQL.Mem	PPGS	1	2.17	2.90	3.23	4.07	6.14	6.97	7.93	9.23	11.70	31.53	903118
	pICPL	1	3	4	4	5	8	8	9	11	14	28	407991
Violet	PPGS	1	1	1	2	2	2.93	3	3.07	3.30	4.53	12.83	31376054
	pICPL	1	1	1	2	2	3	3	4	4	6	15	2471691
Wget	PPGS	2	2.13	3	3.07	4	5.43	6	6.40	7	8.03	11.37	31525
	pICPL	2	3	3	4	4	6	6	7	7	9	11	19612
x264	PPGS	1.23	2.23	3	3.07	4	5.30	6	6.50	7.23	8.47	12.10	37368
	pICPL	1	2	3	3	4	5	6	7	7	9	13	13441
ZipMe	PPGS	2	3	3	4	5	6	6	7	7	7	7.03	13035
	pICPL	2	3	3	4	5	6	6	6	7	7	7	6142

PPGS

smaller size

pICPL

faster

Â12 measure

- $\hat{A}_{12}$ is an effect size measure
  - i.e. value 0.3 means that an algorithm A would obtain lower values than algorithm B for a measure M in 70% of the times
- Lower values, PPGS obtains smaller test suites

Group	50%	75%	80%	85%	90%	95%
G1	0.4985	0.4729	0.4511	0.4473	0.3785	0.3501
G2	0.4529	0.4193	0.3760	0.3726	0.3436	0.2887
[illegible]	0.5104	0.4562	0.2844	0.3563	0.3198	0.3239

Group	96%	97%	98%	99%	100%
G1	0.3410	0.3703	0.3634	0.4000	0.5157
G2	0.2847	0.2647	0.2497	0.2595	0.4945
G3	0.3312	0.313[illegible]	0.3927	0.3068	0.4166

pICPL smaller test suites

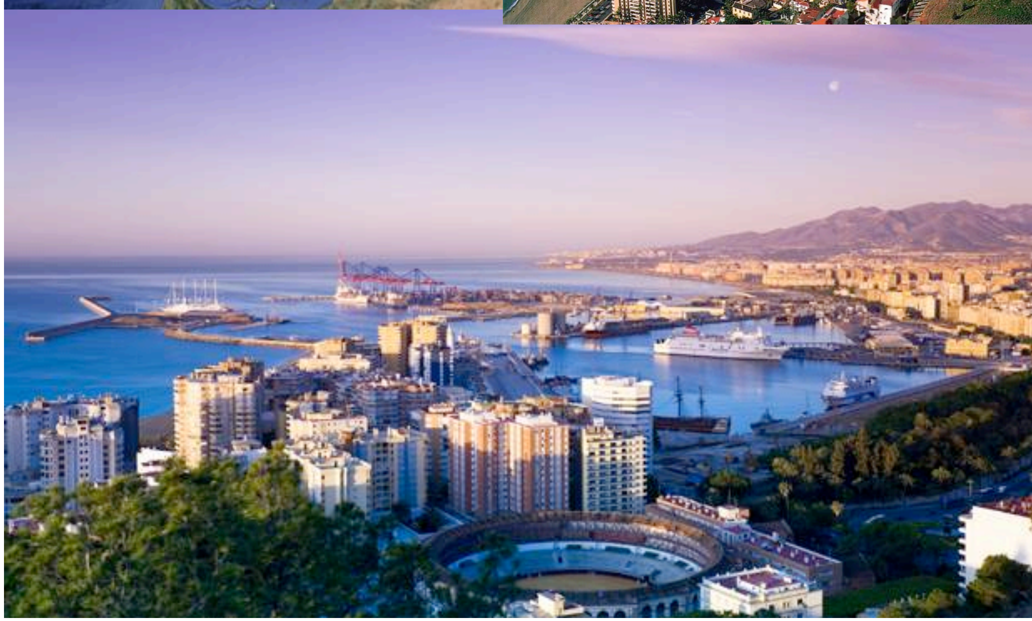
pICPL smaller test suites

PPGS best performance

PPGS obtains smaller size test suites most of the times

Recent Research on Search Based software Testing: Part 2

Thanks for your attention !!!



Test Suite Minimization in Regression Testing

(Landscape Theory)

Binary Search Space

- **The set of solutions is the set of binary strings with length *n***

0	1	0	0	1	0	1	1	1	0

- **Neighborhood used: one-change neighborhood**
  - **Two solutions *x* and *y* are neighbors iff *Hamming(x,y)=1***

0	1	0	0	1	0	1	1	1	0

1	1	0	0	1	0	1	1	1	0
0	**0**	0	0	1	0	1	1	1	0
0	1	**1**	0	1	0	1	1	1	0
0	1	0	**1**	1	0	1	1	1	0
0	1	0	0	**0**	0	1	1	1	0

0	1	0	0	1	**1**	1	1	1	0
0	1	0	0	1	0	**0**	1	1	0
0	1	0	0	1	0	1	**0**	1	0
0	1	0	0	1	0	1	1	**0**	0
0	1	0	0	1	0	1	1	1	**1**

Elementary Landscapes: Characterizations

- **An elementary landscape is a landscape for which**

$$\underset{y\in N(x)}{\operatorname{avg}}\{f(y)\} = \alpha f(x) + \beta \quad \forall x \in X$$

Depend on the problem/instance

Linear relationship

where

$$\underset{y\in N(x)}{\operatorname{avg}}\{f(y)\} \stackrel{\text{def}}{=} \frac{1}{d} \sum_{y\in N(x)} f(y)$$

$$\bar{f} = \frac{1}{|X|} \sum_{y\in X} f(y)$$

- **Grover's wave equation**

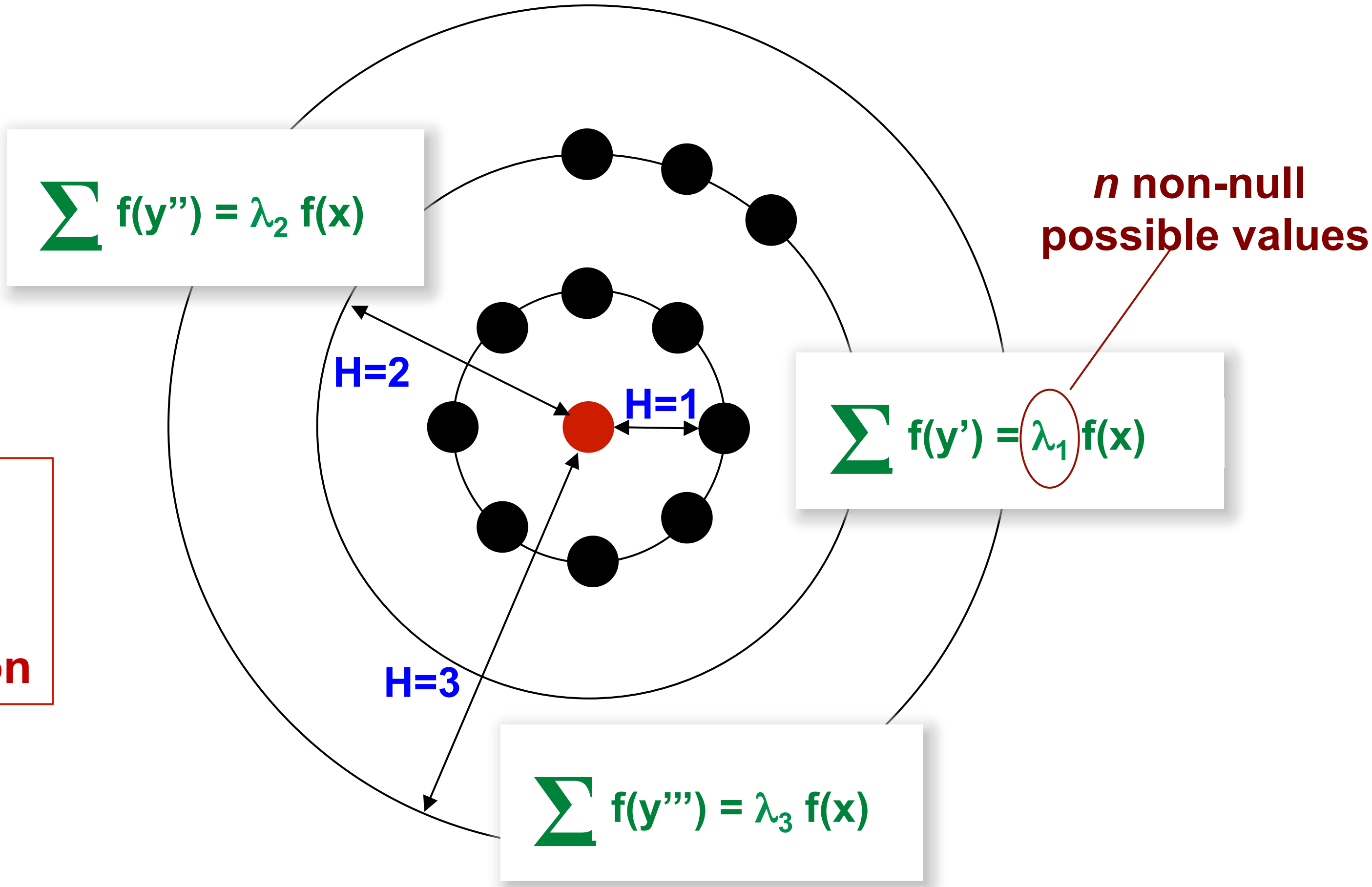
$$\underset{y\in N(x)}{\operatorname{avg}}\{f(y)\} = f(x) + \frac{\lambda}{d}\left(\bar{f} - f(x)\right)$$

Eigenvalue

$$\alpha = 1 - \frac{\lambda}{d} \qquad \beta = \frac{\lambda}{d}\bar{f}$$

Spheres around a Solution

- If *f* is elementary, the average of *f* in any sphere and ball of any size around *x* is a linear expression of f(x)!!!



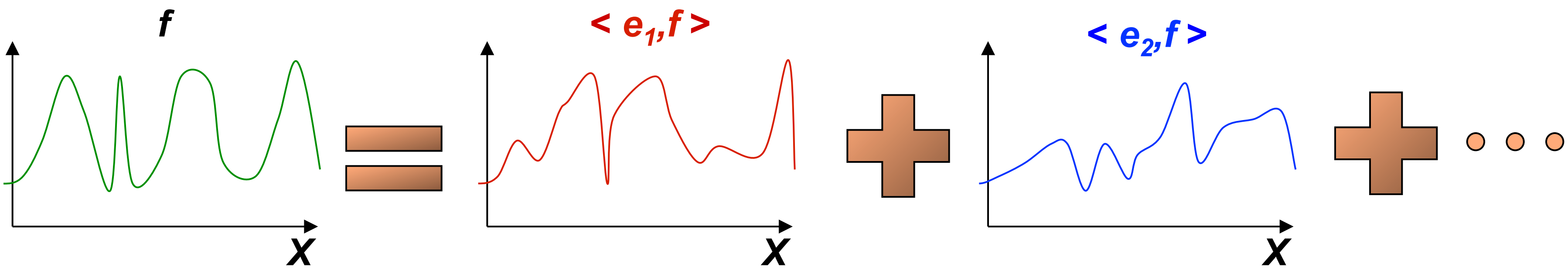
Sutton

Whitley

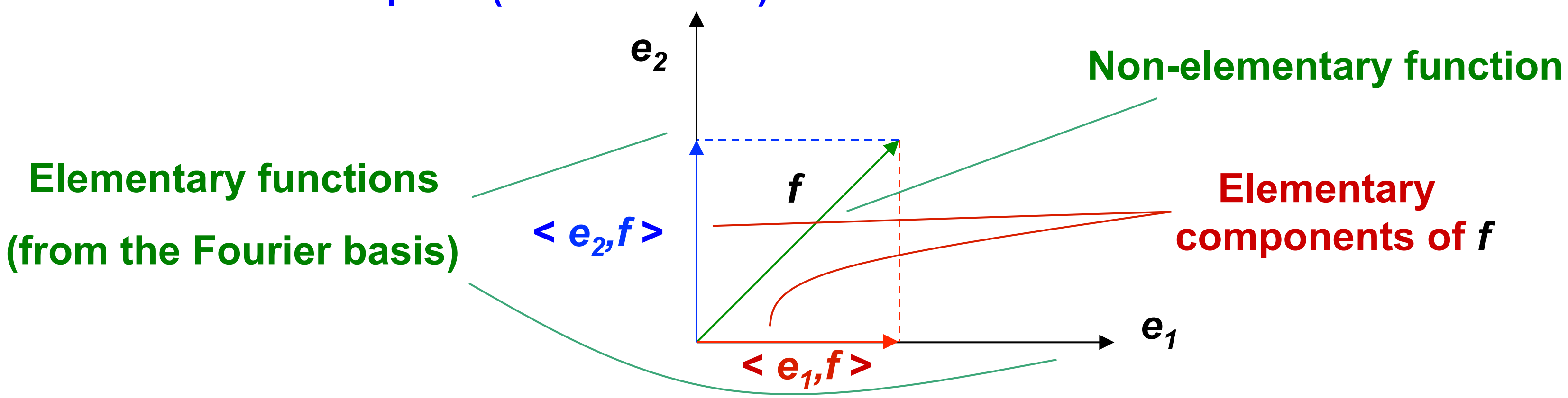
Langdon

Landscape Decomposition

- **What if the landscape is not elementary?**
- **Any landscape can be written as the sum of elementary landscapes**



- **There exists a set of eigenfunctions of Δ that form a basis of the function space (Fourier basis)**



Elementary Landscape Decomposition of *f*

- **The elementary landscape decomposition of**

$$f(x) = cov(x) - c \cdot cost(x)$$

is

Computable in *O(nk)*

Tests that cover e_i

$$f^{(0)}(x) = \sum_{i=1}^{k} \left(1 - \frac{1}{2^{|V_i|}}\right) - c \cdot \frac{n}{2}$$

constant expression

$$f^{(1)}(x) = -\sum_{i=1}^{k} \frac{1}{2^{|V_i|}} (-1)^{n_1^{(i)}} \mathcal{K}^{|V_i|}_{|V_i|-1, n_1^{(i)}} - c \cdot \left(ones(x) - \frac{n}{2} \right)$$

Krawtchouk matrix

$$f^{(p)}(x) = -\sum_{i=1}^{k} \frac{1}{2^{|V_i|}} (-1)^{n_1^{(i)}} \mathcal{K}^{|V_i|}_{|V_i|-p, n_1^{(i)}} \quad \text{where } 1 < p \leq n$$

Tests in the solution that cover e_i

F. Chicano et al., SSBSE 2011

Elementary Landscape Decomposition of f^2

- **The elementary landscape decomposition of f^2 is**

Computable in $O(nk^2)$

$$(f^2)^{(0)}(x) = \beta^2 + \frac{c^2}{4}n - \sum_{i=1}^{k} \frac{c|V_i| + 2\beta}{2^{|V_i|}} + \sum_{i,i'=1}^{k} \frac{1}{2^{|V_i \cup V_{i'}|}}$$

$$\beta = k - cn/2$$

Number of tests that cover e_i or $e_{i'}$

$$(f^2)^{(p)}(x) = -\sum_{i=1}^{k} \left(\frac{(c|V_i| + 2\beta)(-1)^{n_1^{(i)}}}{2^{|V_i|}} \mathcal{K}^{|V_i|}_{|V_i|-p,n_1^{(i)}} \right) + \sum_{i,i'=1}^{k} \left(\frac{(-1)^{n_1^{(i \vee i')}}}{2^{|V_i \cup V_{i'}|}} \mathcal{K}^{|V_i \cup V_{i'}|}_{|V_i \cup V_{i'}|-p,n_1^{(i \vee i')}} \right) - c \sum_{i=1}^{k} \frac{(-1)^{n_1^{(i)}}}{2^{|V_i|}} \mathcal{K}^{|V_i|}_{|V_i|-p+1,n_1^{(i)}} \left(n - 2ones(x) - |V_i| + 2n_1^{(i)} \right)$$

$p > 2$

Number of tests in the solution that cover e_i or $e_{i'}$

Guarded Local Search

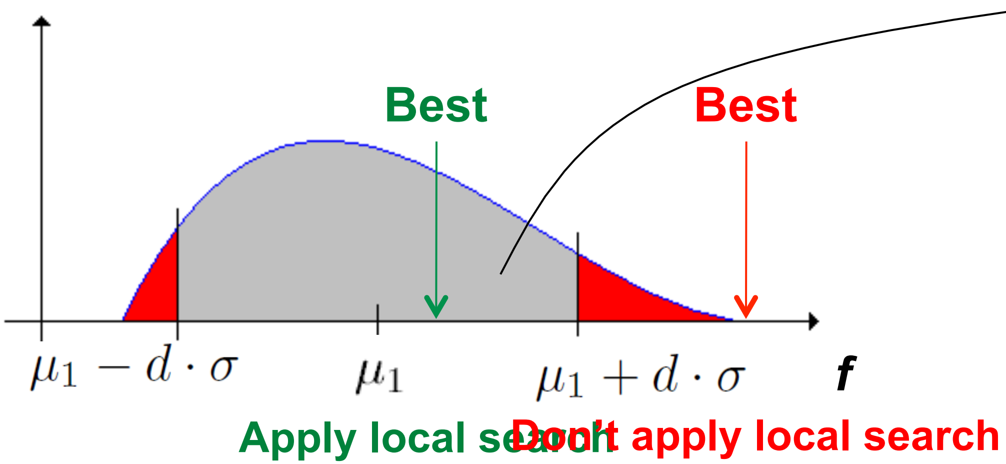
- With the Elementary Landscape Decomposition (ELD) we can compute:

$$\mu_c = \underset{y|\mathcal{H}(y,x)=r}{\text{avg}}\{f^c(y)\} = \binom{n}{r}^{-1} \sum_{p=0}^{n} \mathcal{K}_{r,p}^{(n)} \left(f^c\right)^{(p)}(x)$$

- With the ELD of *f* and *f*^{2} we can compute for any sphere and ball around a solution:

μ_1 : the average $\sigma = \sqrt{\mu_2 - \mu_1^2}$: the standard deviation

- Distribution of values around the average



Chebyshev inequality

$$100\left(1 - \frac{1}{d^2}\right)\%$$

At least 75% of the samples are in the interval

$$[\mu_1 - 2\sigma, \mu_1 + 2\sigma]$$

Guarded Local Search: Experimental Setting

- **Steady state genetic algorithm: bit-flip (*p*=0.01), one-point crossover, elitist replacement**
  - **GA (no local search)**
  - **GLS*r* (guarded local search up to radius *r*)**
  - **LS*r* (always local search in a ball of radius *r*)**

- **Instances from the Software-artifact Infrastructure Repository (SIR)**
  - **printtokens**
  - **printtokens2**
  - **schedule**
  - **schedule2**
  - **totinfo**
  - **replace**

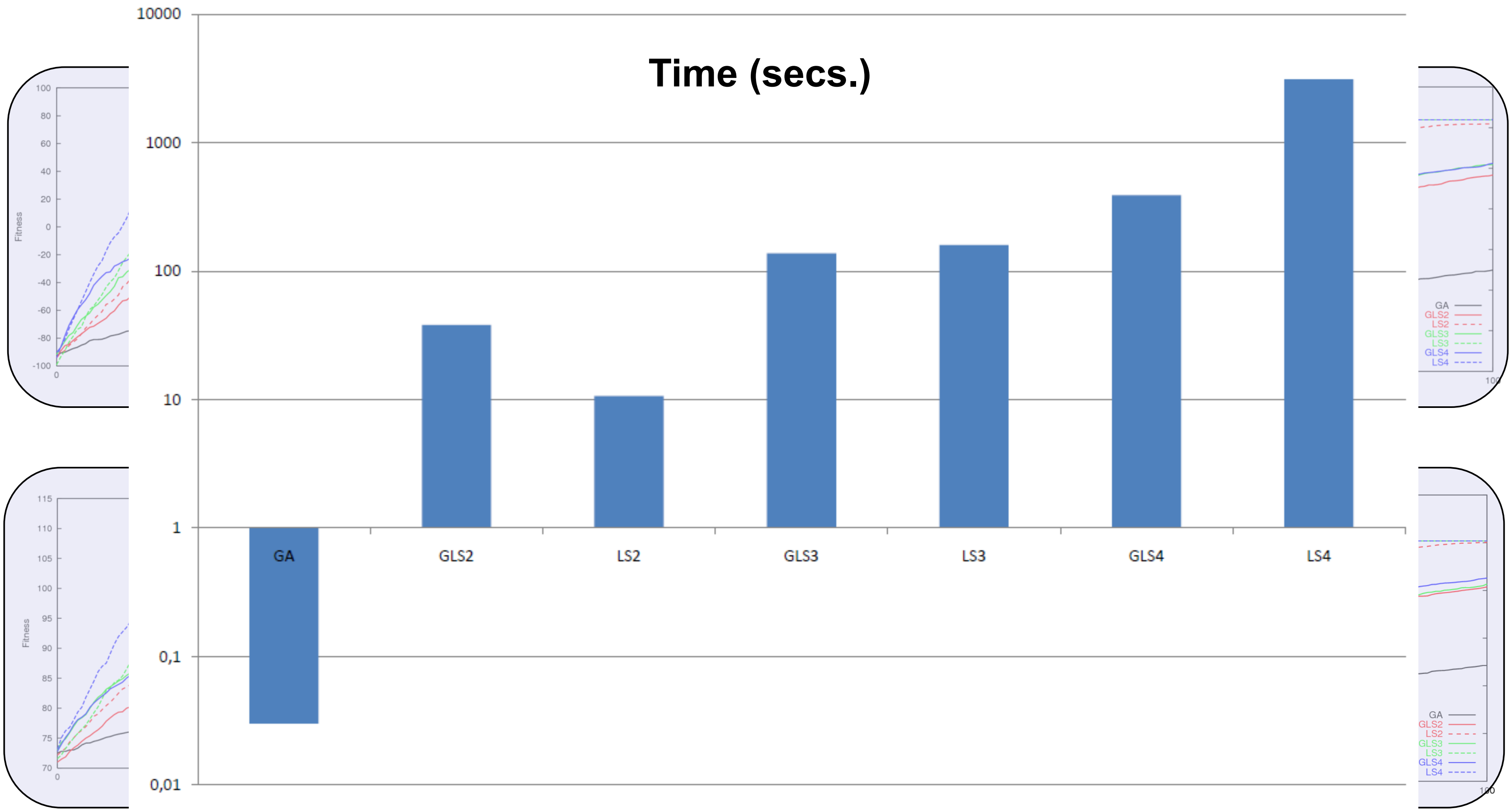
Oracle cost c=1..5

***n*=100 test cases**

***k*=100-200 items to cover**

100 independent runs

Guarded Local Search: Results



Comparison with an LS and GA

Local Search

- Best improvement

Genetic Algorithm

- 10 individuals
- 2-tournament
- Bit-flip mutation (p=0.01)
- 1-point crossover
- Steady-state

Total coverage (not Pareto front)

Instance	Ratio	Algorithm 2		Local Search		Genetic Algorithm	
		Original (s)	Reduced (s)	Avg. Cov.	Avg. Tests	Avg. Cov.	Avg. Tests
printtokens	4.61	3400.74	2.17	100.00%	6.00	99.06%	5.16
printtokens2	4.61	3370.44	1.43	100.00%	4.60	99.23%	3.56
replace	4.62	1469272.00	345.62	100.00%	10.16	99.15%	15.46
schedule	2.19	492.38	0.24	100.00%	3.00	99.84%	2.90
schedule2	4.61	195.55	0.27	100.00%	4.00	99.58%	3.70
tcas	4.61	73.44	0.33	100.00%	4.00	95.80%	3.23
totinfo	4.53	181823.50	0.96	100.00%	5.00	98.89%	5.13