On the application of SAT solvers for Search Based Software Testing

LENGUAJES Y CIENCIAS DE LA COMPUTACIÓN
UNIVERSIDAD DE MÁLAGA

UNIVERSIDAD DE MÁLAGA

Francisco Chicano

Departamento de Lenguajes y Ciencias de la Computación

Universidad de Málaga

Spain

Optimization Problem

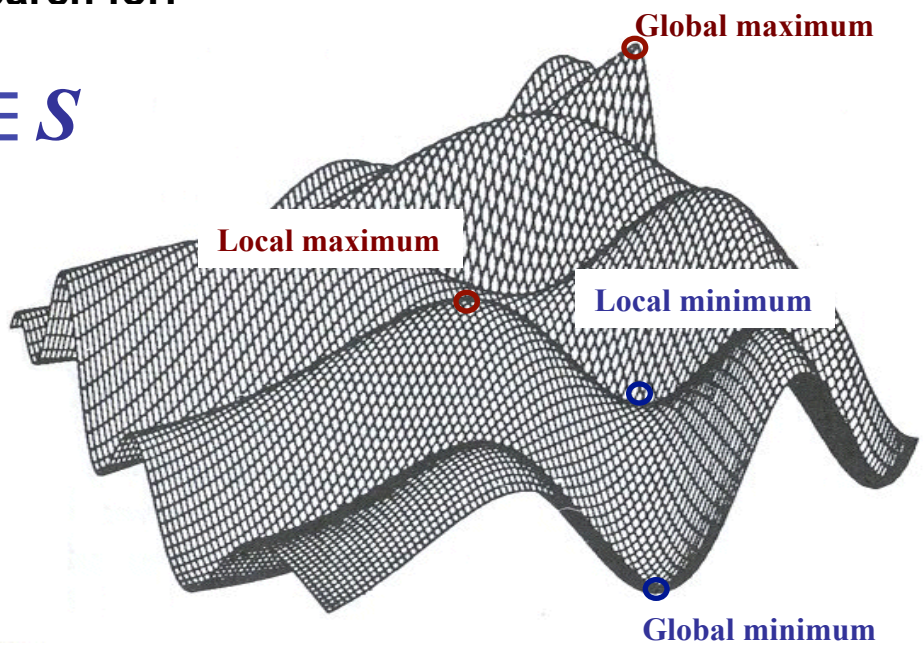
An optimization problem is a pair: $P = (S,f)$ where:

S is a set of solutions (solution or search space)

$f: S \rightarrow R$ is an objective function to minimize or maximize

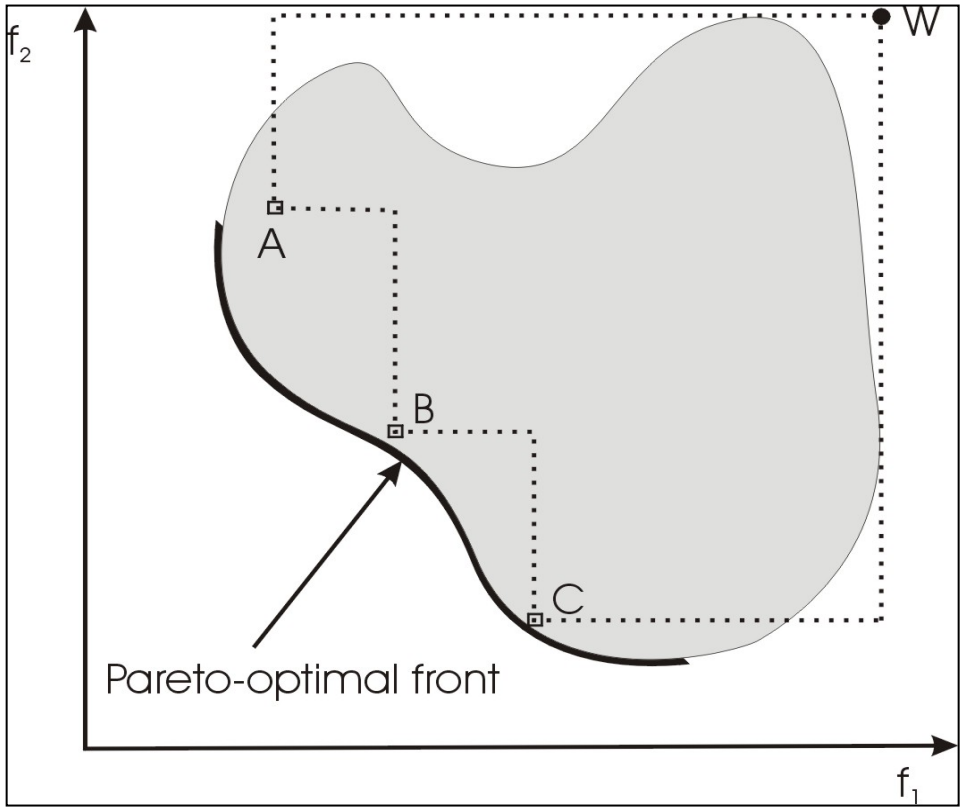
If our goal is to minimize the function we search for:

$$s' \in S \mid f(s') \leq f(s), \forall s \in S$$

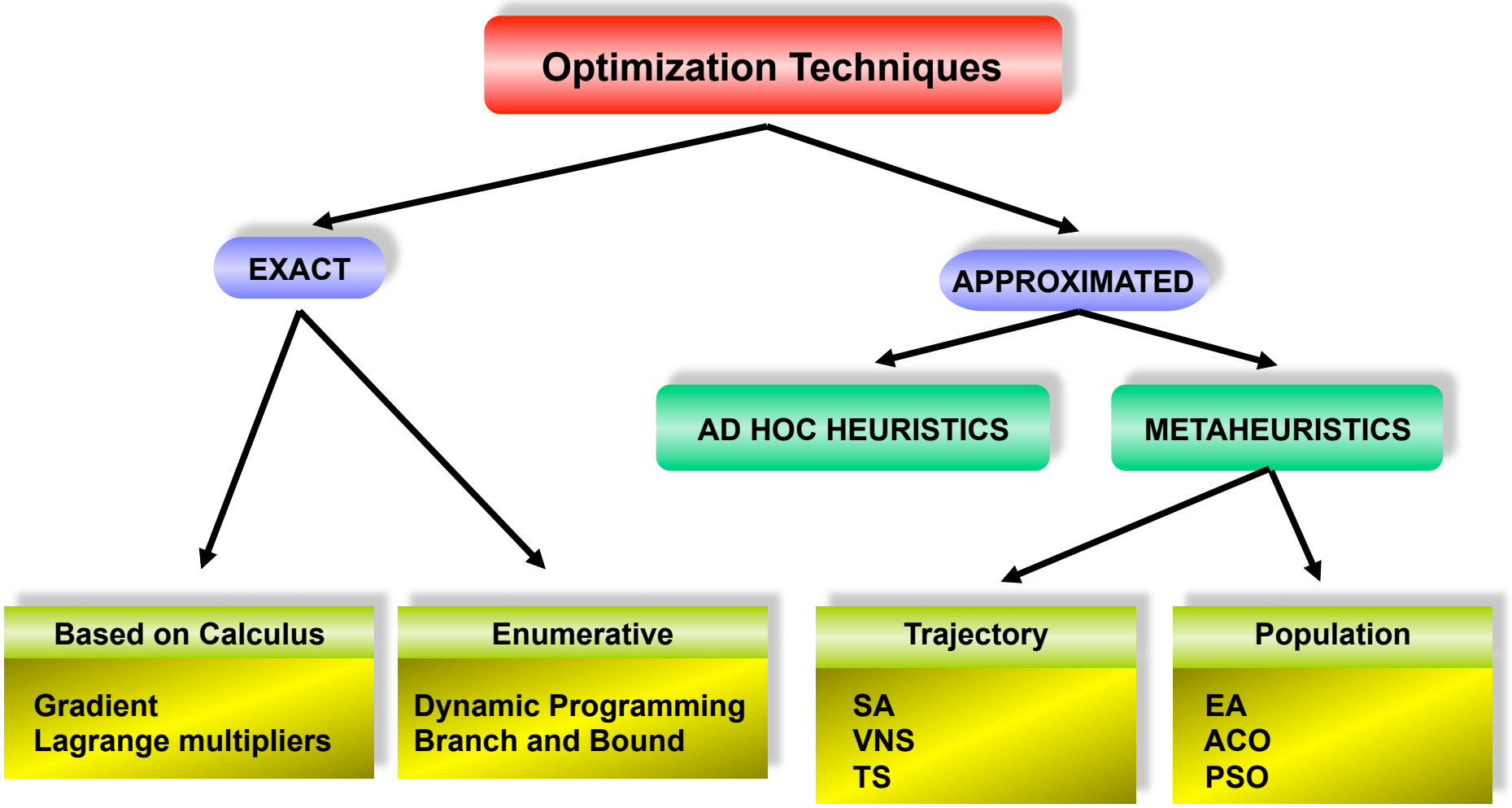


Multi-Objective Optimization Problem

In a MO problem there are several objectives (functions) we want to optimize



Optimization Techniques



Evolutionary Algorithm

Pseudocode of a simple EA

```
P = generateInitialPopulation ();
evaluate (P);
while not stoppingCondition () do
        P´ = selectParents (P);
        P´ = applyVariationOpterators (P´);
        evaluate(P´);
        P = selectNewPopulation (P,P´);
end while
return the best solution found
```

Three main steps: selection, reproduction, replacement

- Variation operators → Make the population to evolve
  - Recombination: exchange of features
  - Mutation: generation of new features

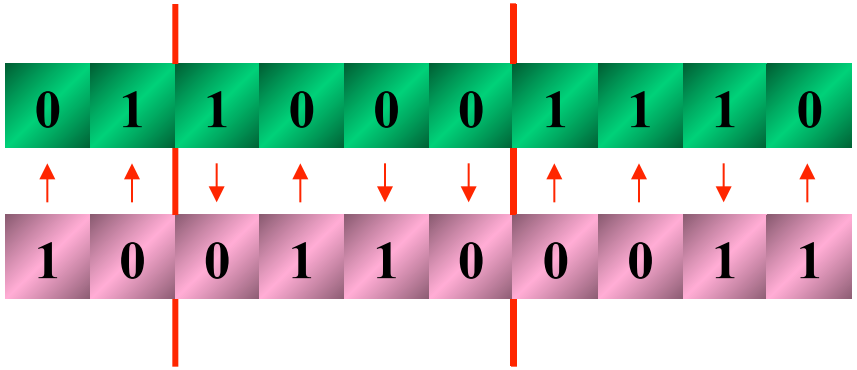
Evolutionary Algorithm

Genetic Algorithms

- **Individuals**

Binary Chromosome

- **Recombination**
  - One point
  - Two points
  - Uniform



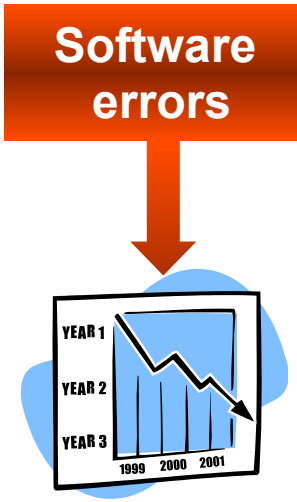
- **Mutation → bit flips**

Software Testing: Definition and Goal

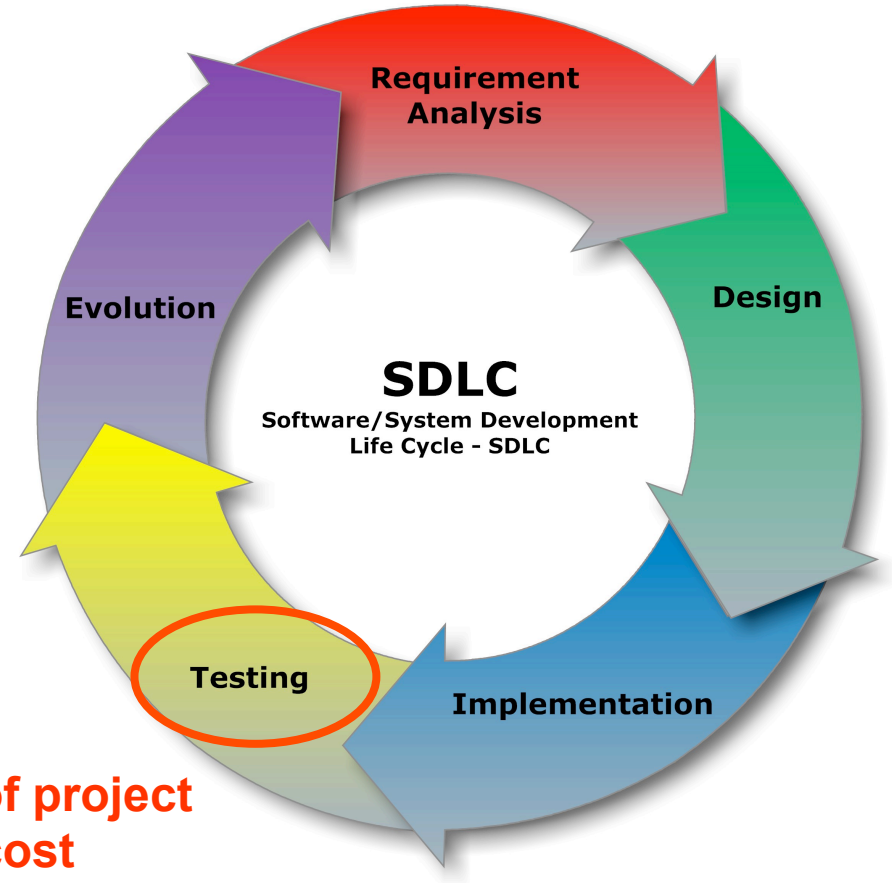
- What is software testing?
  - It is the process of running a software product or a portion of it in a controlled environment with a given input followed by the collection and analysis of the output and/or other relevant information of the execution.
- What is the goal of software testing?
  - To find out errors in a portion or the complete software product and/or to assure with a high probability that the software is correct (according to the requirements).

Software Testing: Impact

Software testing is important because…



60.000 M$ annually (0,6% GDP) in USA



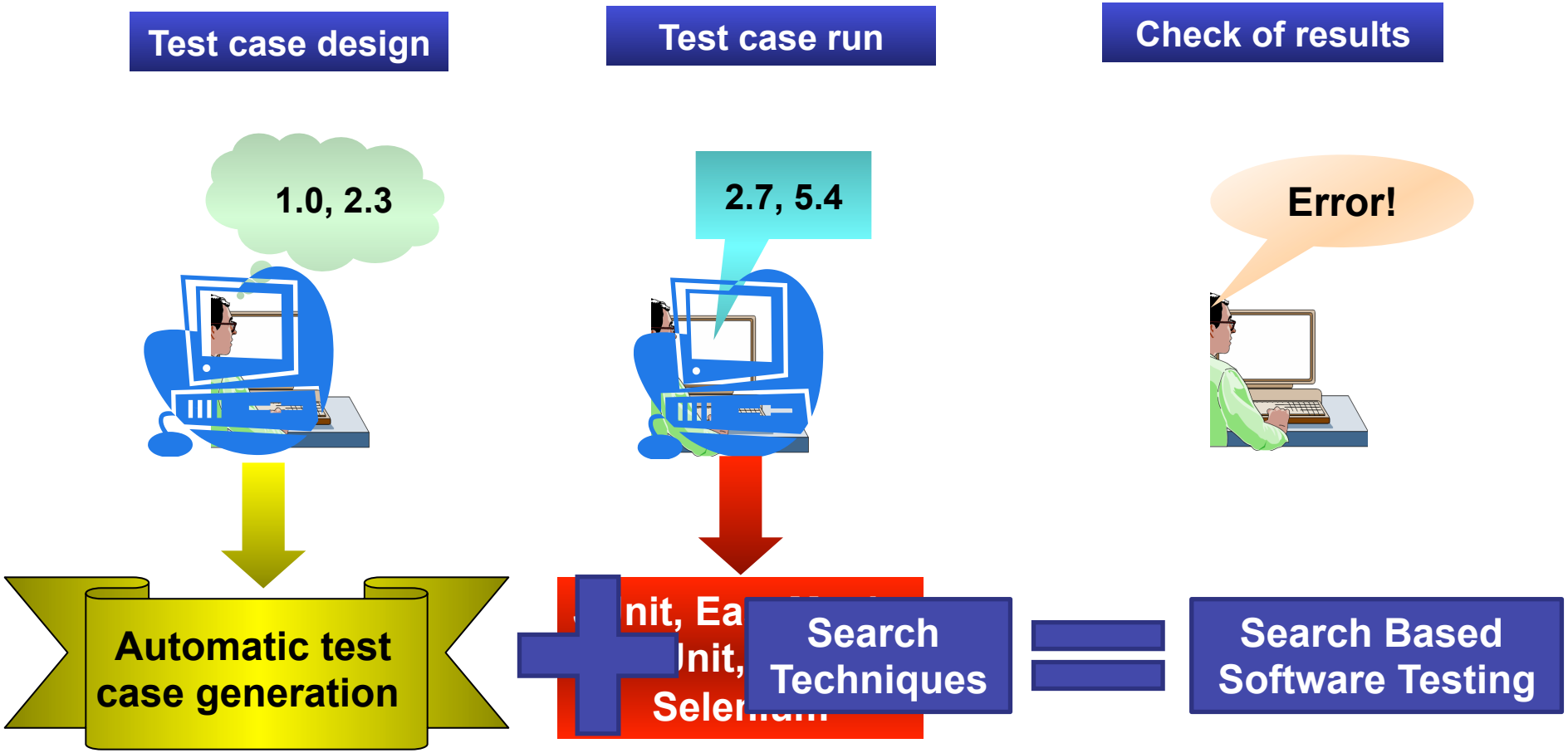
60 % of project cost

Software Testing: Classification

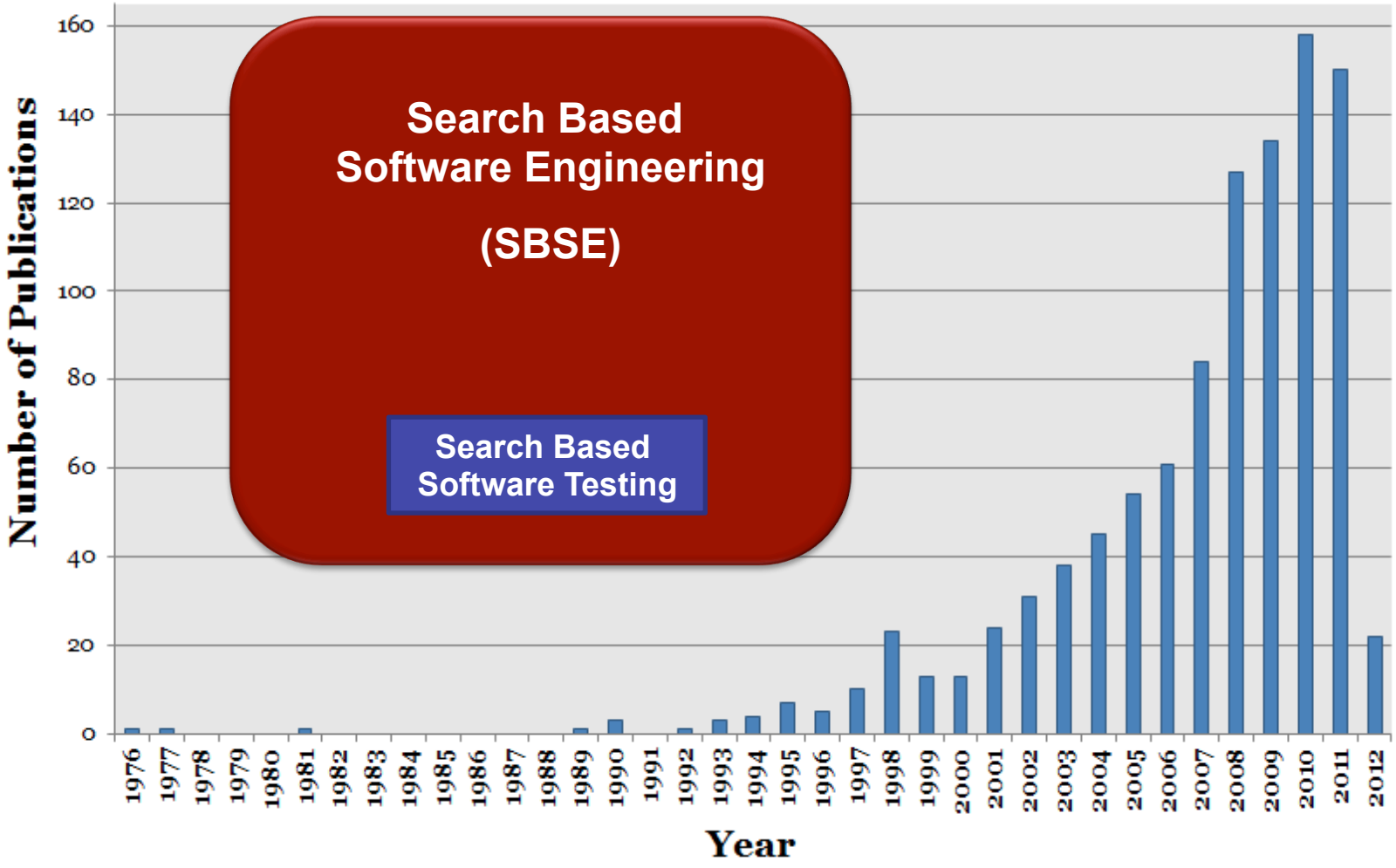
Classification of testing techniques (by goal)

- Unit testing: test one module of the software.
- Integration testing: test the interfaces between different modules in the software.
- System testing: test the complete system.
- Validation testing: test if the software system fulfills the requirements.
- Acceptance testing: the client test whether the system is what s/he wants.
- Regression testing: after a change in the software test whether a new error has been introduced.
- Stress testing: test the system under a high load
- Load testing: test the response of the system under a normal load of work.

Software Testing: Automatization



Search Based Software Engineering



Our Research on SBSE

- Software Project Scheduling
- White-box Software Testing
- Testing of Concurrent Systems (based on Model Checking)
- Test Sequences for Functional Testing
- Test Suite Minimization in Regression Testing
- Software Product Lines Testing
- Prioritized Pairwise Combinatorial Interaction Testing
- Testing Complexity

Test Suite Minimization in Regression Testing

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 (at least exponential time)

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

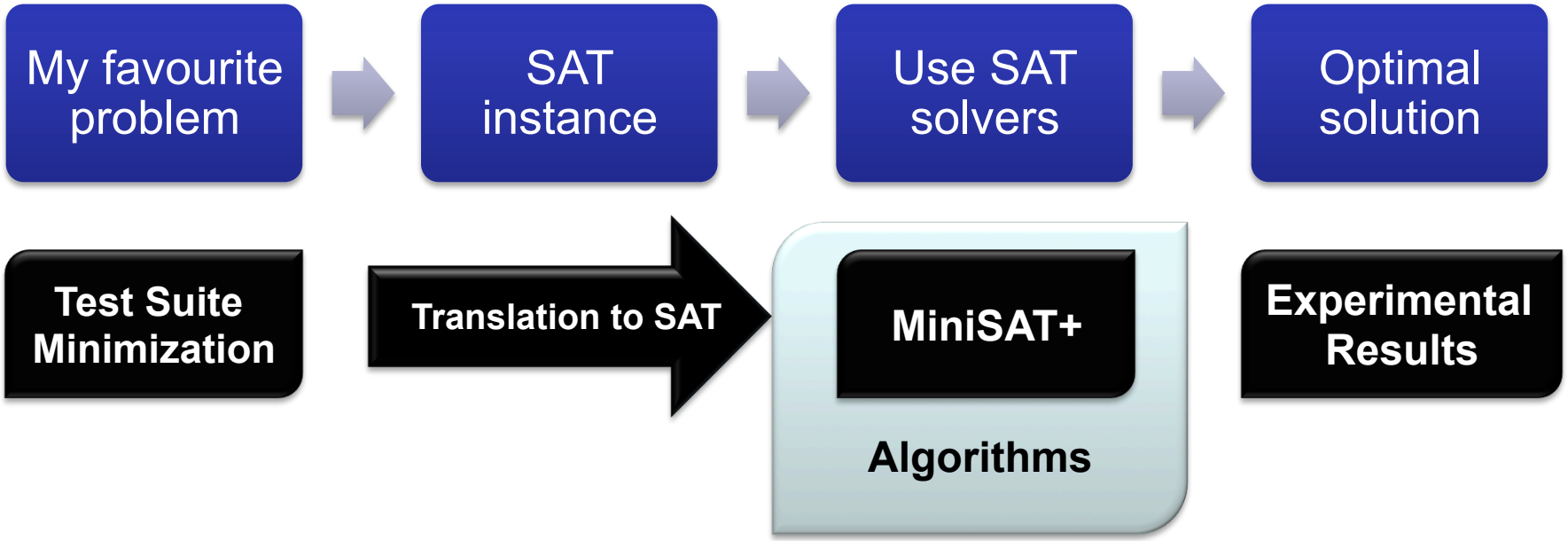
Nowadays, SAT solvers can solve instances with 500 000 boolean variables

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

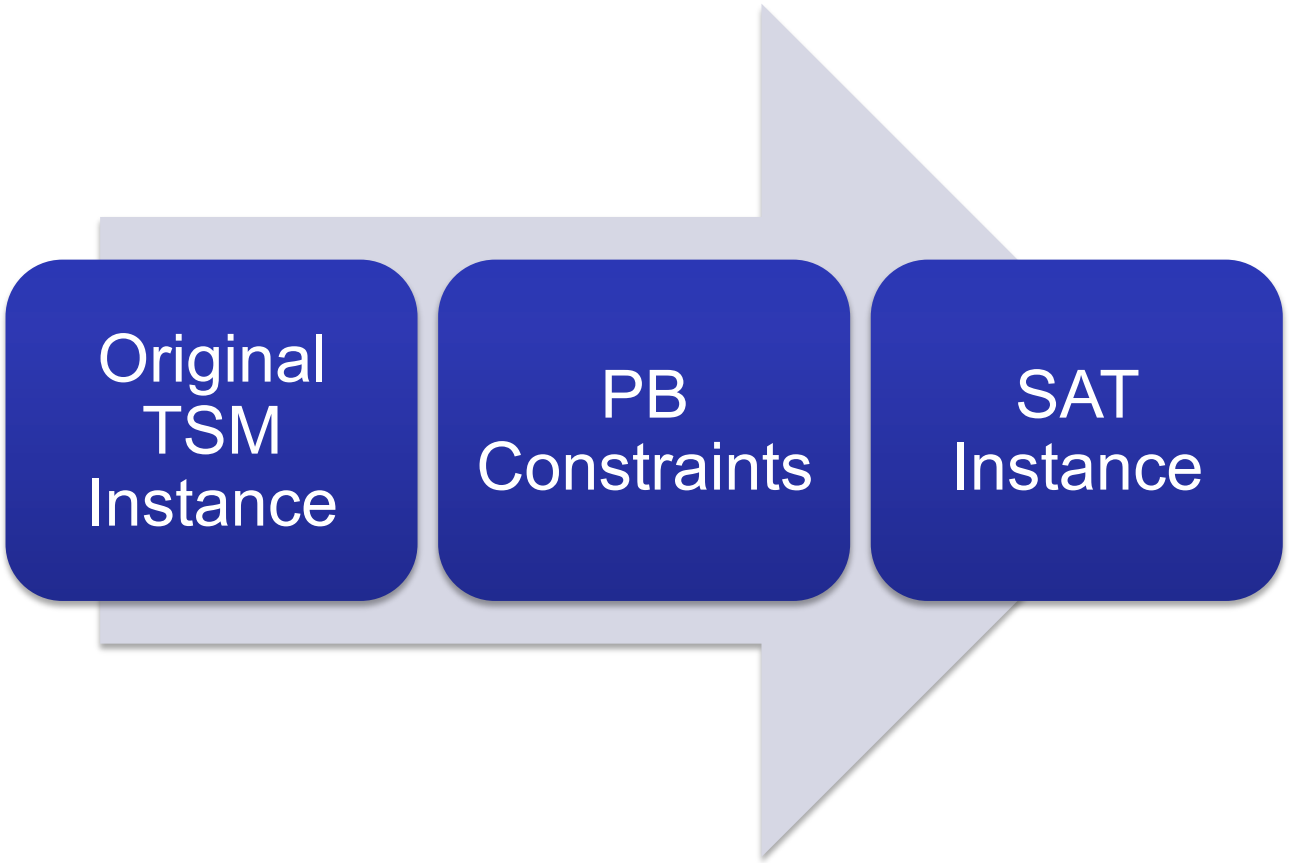
The SATisfiability Problem

Main research question:

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



Outline



F. Arito et al., SSBSE 2012

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+

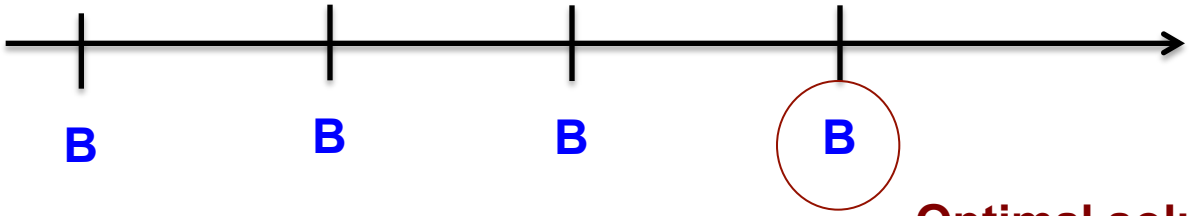
Translating Optimization to Decision Problems

Let us assume we want to minimize f(x)

Check **Check** **Check** **Check**

$$f(x) \leq B \quad f(x) \leq B \quad f(x) \leq B \quad f(x) \leq B$$

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

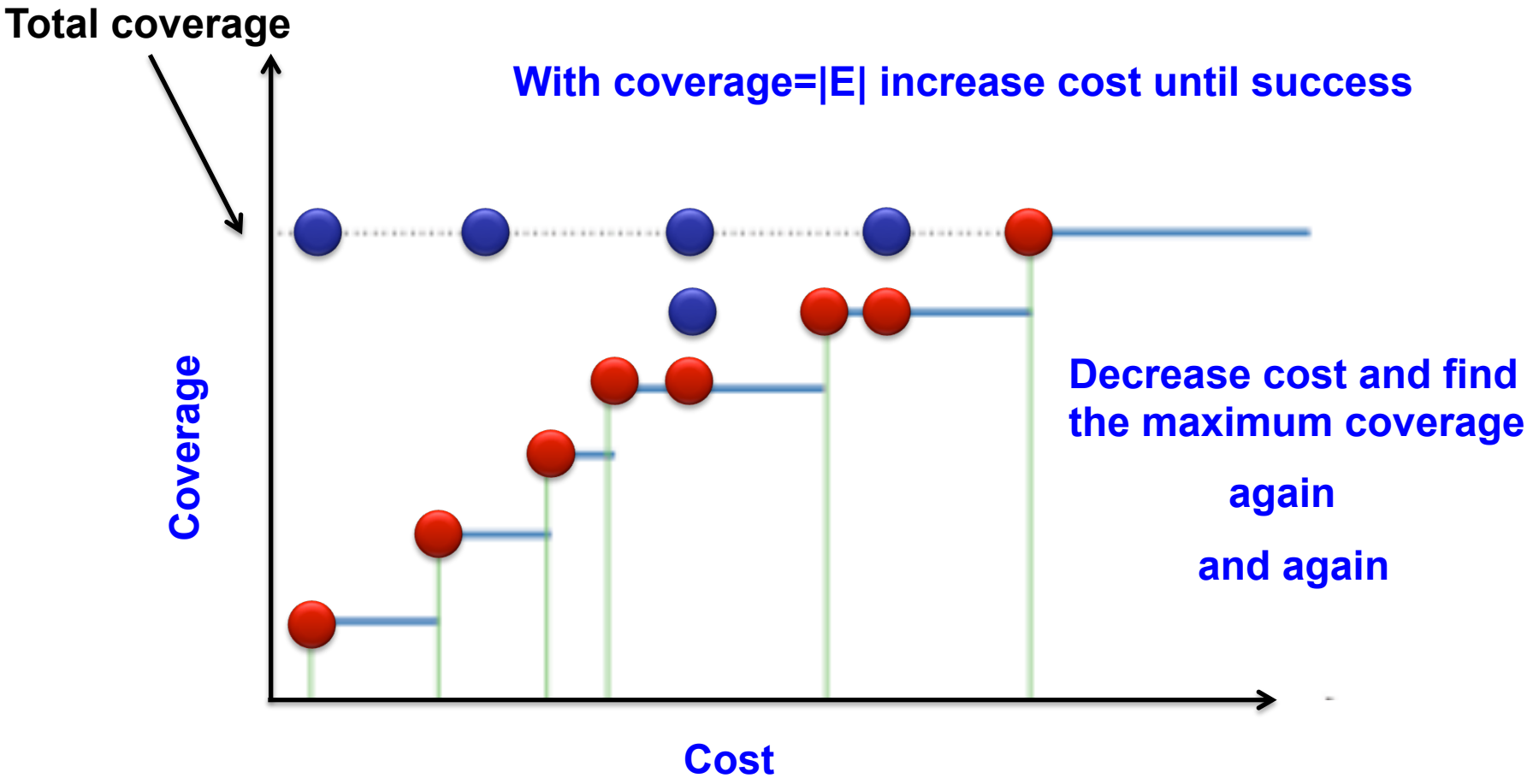
Bi-objective problem

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

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

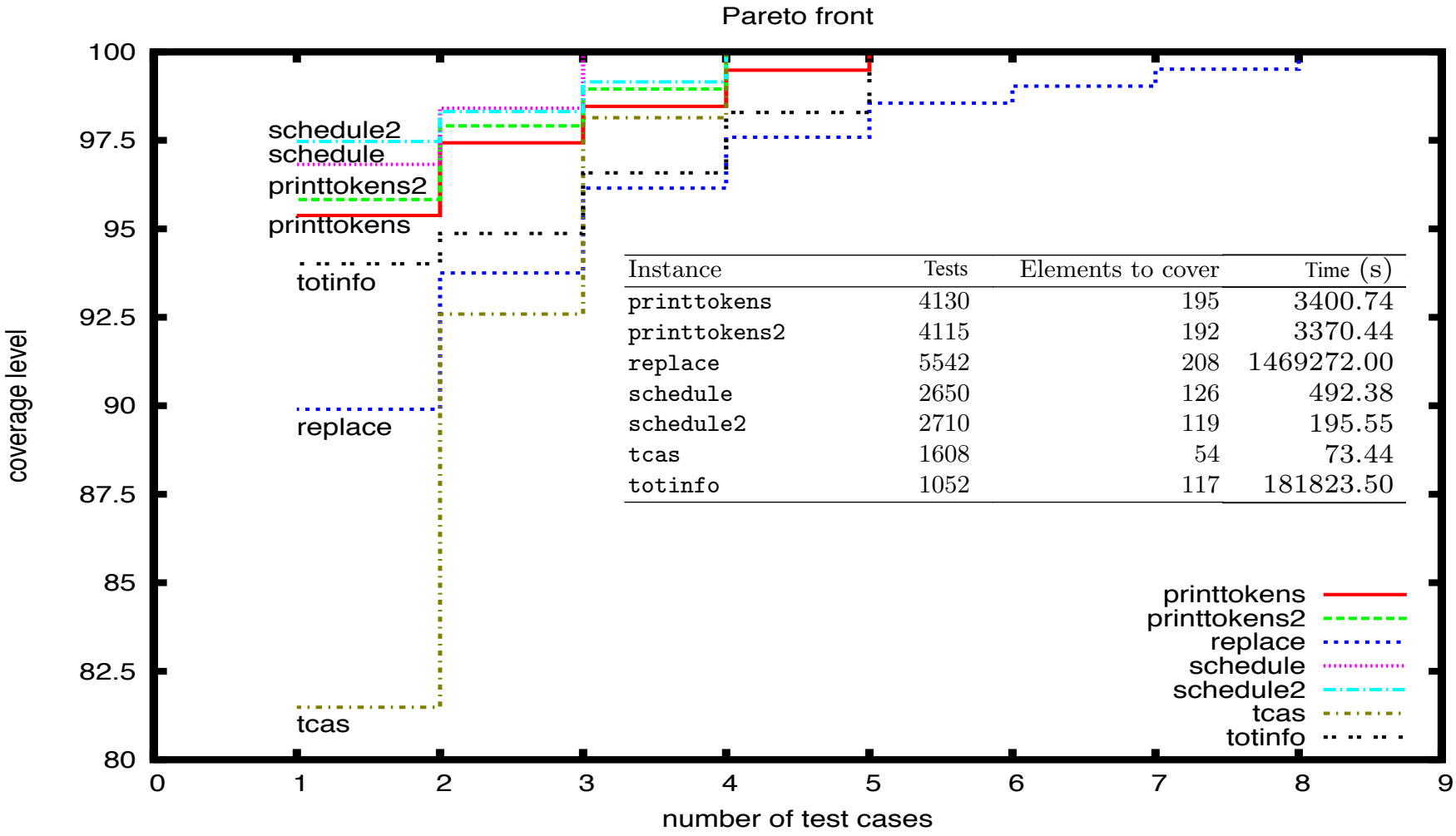
Instances from the Software-artifact Infrastructure Repository (SIR)

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**

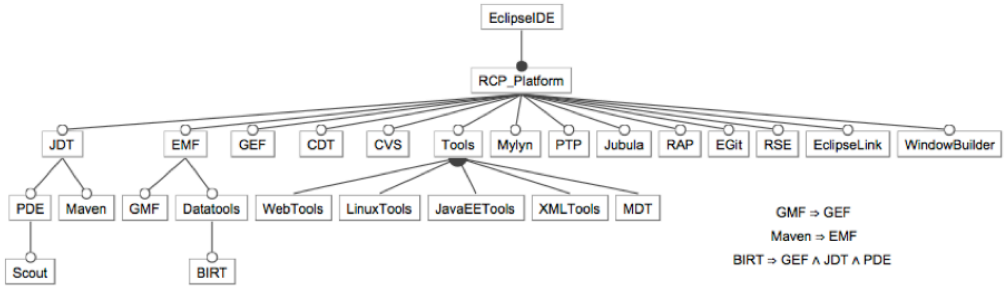
Advantages

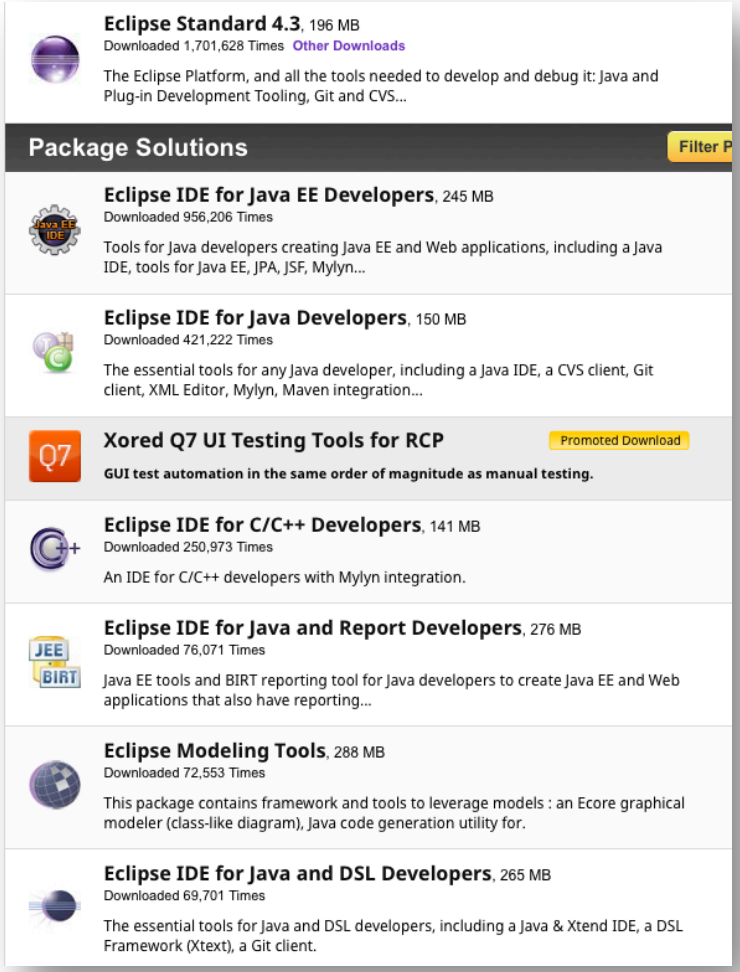
- **Support customization**
- **Improves reuse**
- **Reduce time to market**

Software Product Lines

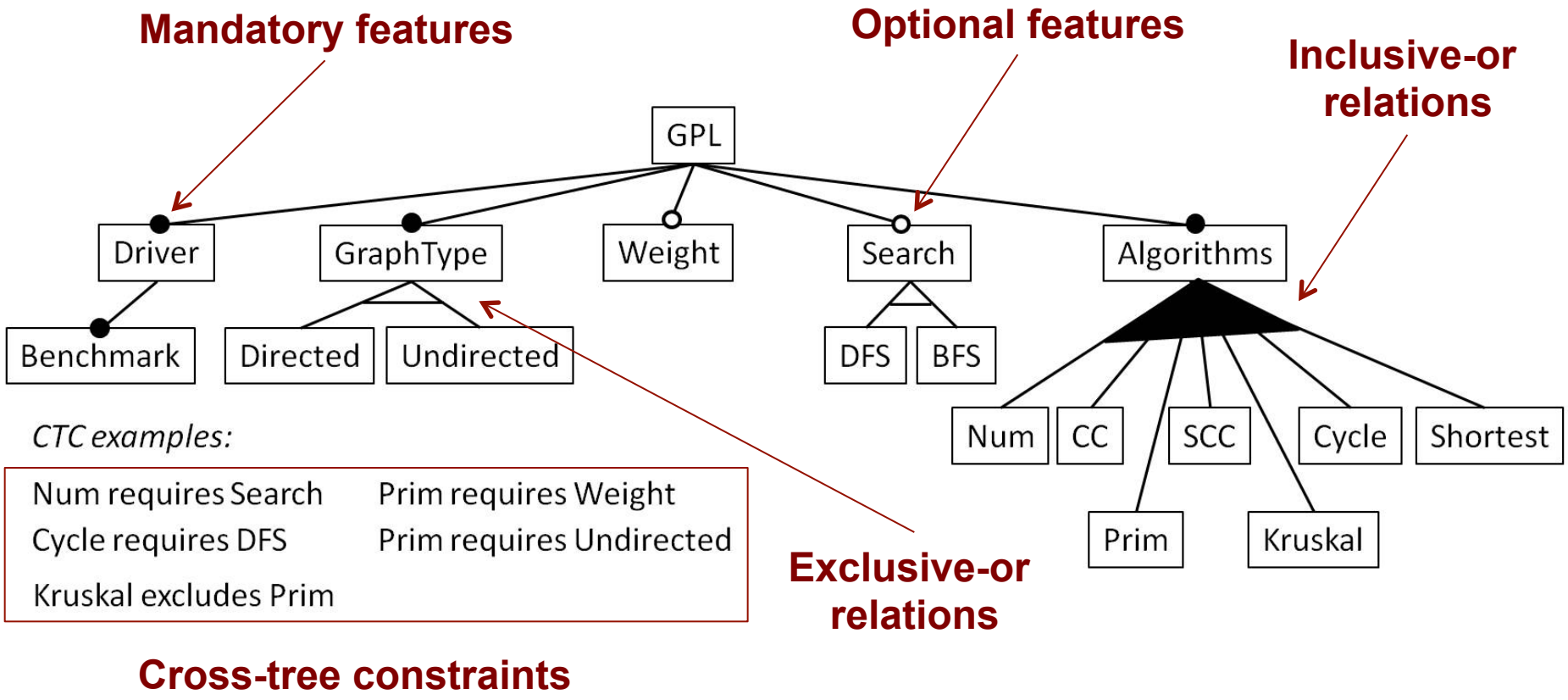
In Software Product Lines the product is Software

They are modelled using Feature Models





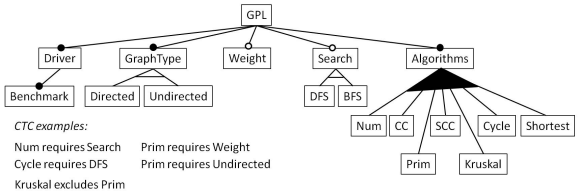
Feature Models



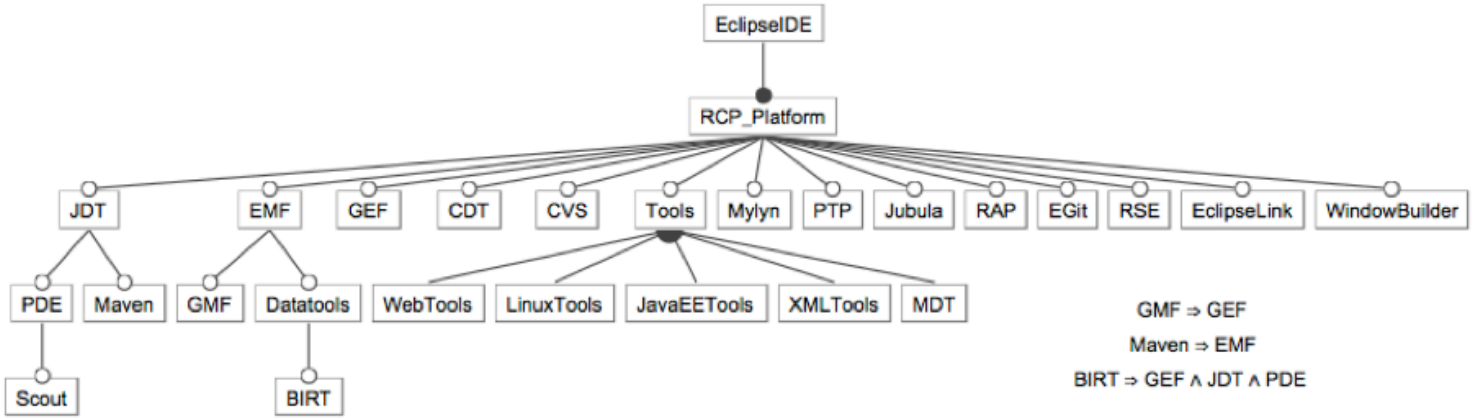
Graph Product Line Feature Model

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…



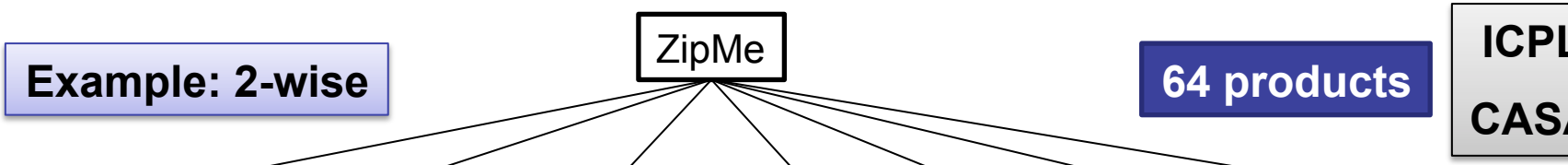
… and testing each particular product is not viable

Testing of SPLs: Combinatorial Interaction Testing

Assuming each features 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

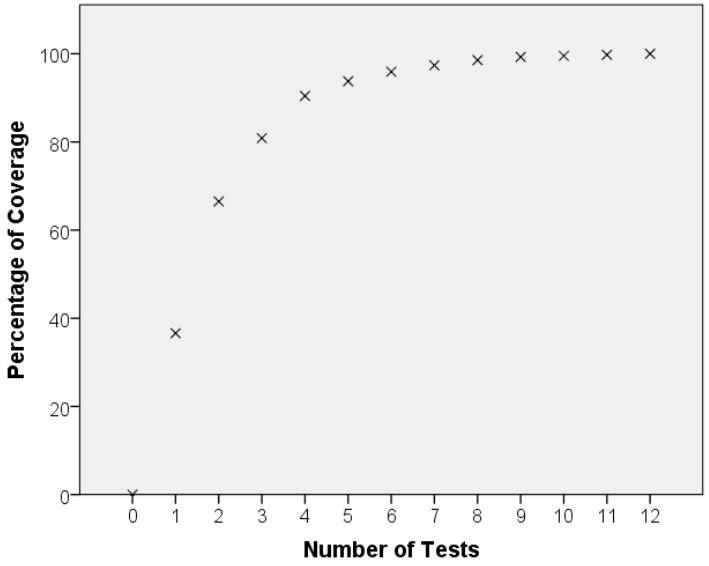
If we don't have the resources to run all the tests, which one to choose?

Multi-objective formulation:

minimize the number of products

maximize the coverage (t-wise interactions)

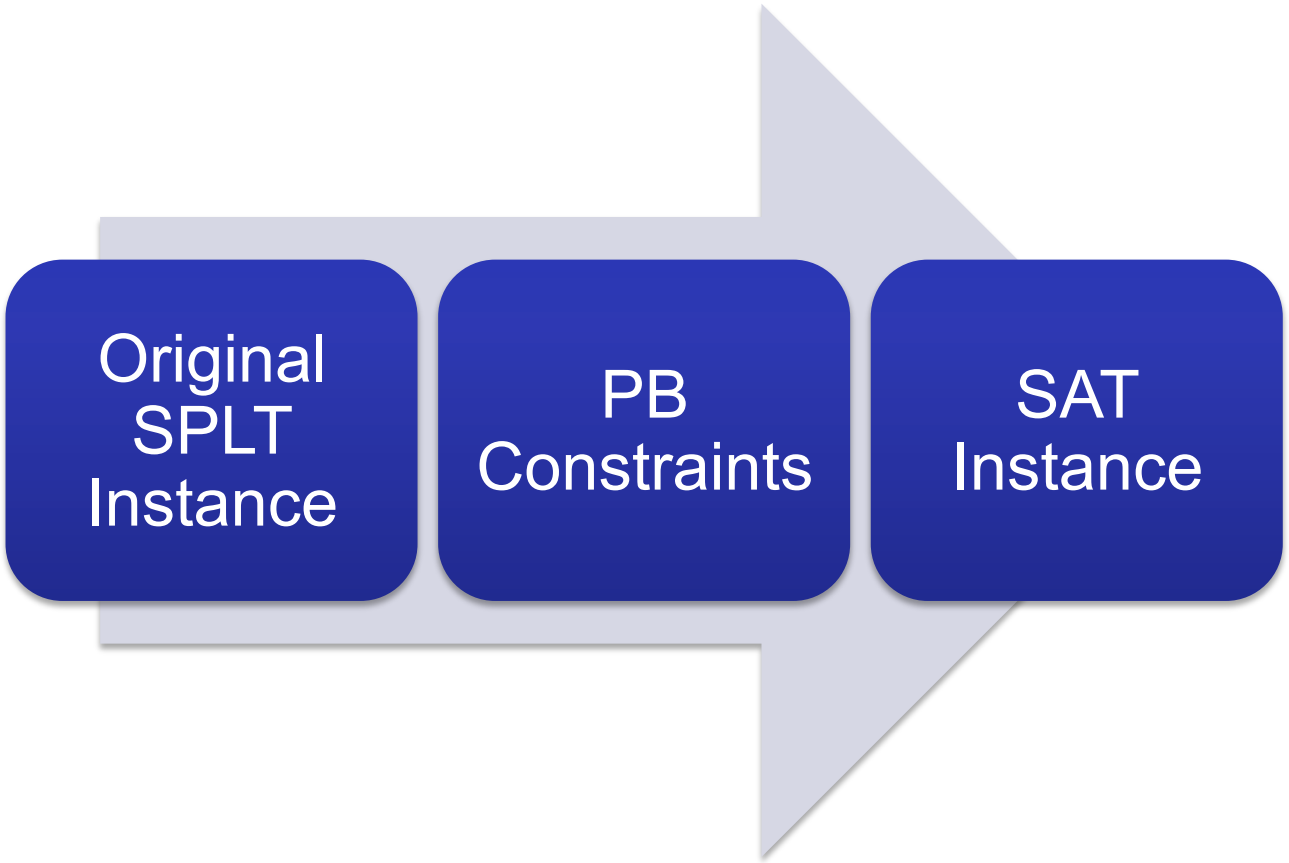
The solution is not anymore a table of products, but a Pareto set



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***

$$2c_{p,i,j,3} \leq x_{p,i} + x_{p,j} \leq 1 + c_{p,i,j,3}$$
$$2c_{p,i,j,2} \leq x_{p,i} + (1 - x_{p,j}) \leq 1 + c_{p,i,j,3}$$
$$2c_{p,i,j,1} \leq (1 - x_{p,i}) + x_{p,j} \leq 1 + c_{p,i,j,3}$$
$$2c_{p,i,j,0} \leq (1 - x_{p,i}) + (1 - x_{p,j}) \leq 1 + c_{p,i,j,3}$$

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.

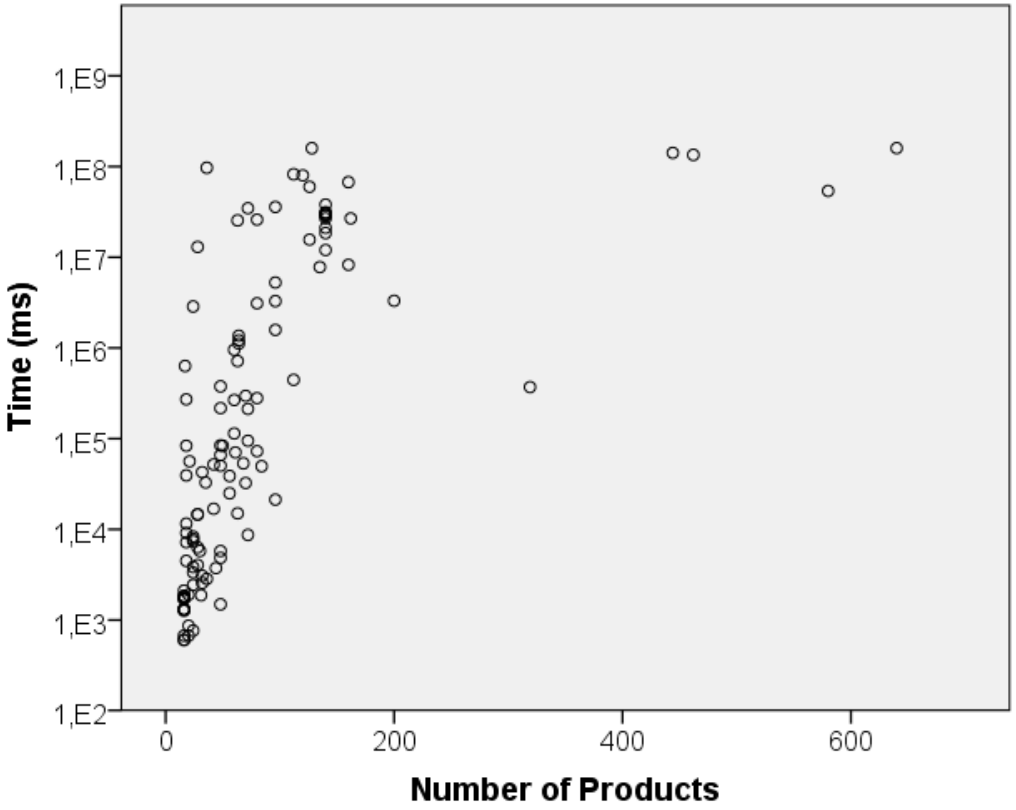
$optimal\_set \leftarrow \{\emptyset\}$;
$cov[0] \leftarrow 0$;
$cov[1] \leftarrow C_2^f$;
$sol \leftarrow$arbitraryValidSolution(fm);
$i \leftarrow 1$;
while $cov[i] \neq cov[i-1]$ **do**
  $optimal\_set \leftarrow optimal\_set \cup \{sol\}$;
  $i \leftarrow i + 1$;
  $m \leftarrow$prepareMathModel(fm,i);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

On the application of SAT solvers for Search Based Software Testing

Thanks for your attention !!!