Product Line Testing

KV Product Line Engineering (343.354)

Dr. Roberto Lopez-Herrejon
Dr. Rick Rabiser

JOHANNES KEPLER
UNIVERSITY LINZ

ISSE

11.5.2016

Overview of SPL Testing

- Describe the main challenges that variability adds for software testing

- Present a taxonomy of the different approaches available for SPL testing

- Illustrate with more detail Combinatorial Interaction Testing in the SPL context

Verification vs Validation

- Verification
  - Activities to assess if a program implements some specific functionality or provides a quality
  - Are we building the product right?
- Validation
  - Activities to assess if a program is what the user actually requires
  - Are we building the right product?

Some Approaches to Verification

- Testing
  - Experimenting with the behavior of a program
  - Dynamic – runs or executes a program

- Analysis
  - Deduces the correct operation of programs from static models

Software Testing in General

- Software Testing – IEEE Glossary of terms 1990
  - The process of operating a system or component ***under specified conditions***, observing or recording the results, and making an evaluation of some aspect of the system or component.
- Why is testing important?
  - Errors are unavoidable human nature and problem complexity
  - Improve product confidence in developers and users
  - Mandatory activity for some domains like safety critical systems

Goals of Testing

- Dijkstra’s quote
  - Program testing can be used to show the presence of bug, but never to show their absence

- Goals
  - What failures does my program have?
  - What causes the failures in my program?
  - How can those failures be fixed?

What is a failure?

- Failure
  - A manifest symptom of the presence of an error
- How do you know a failure is a failure?
  - Non conformance to a specification
- What is a specification?
  - Statement of user or program requirements

Some Terminology

- Defect
  - An error in a program that can cause a failure
- Fault
  - An incorrect state in a program execution that leads to a failure
- Bug
  - Synonym of defect

Testing Should Be ...

- Systematic
  - Planed activity
- Repeatable
  - Same tests should produce same results
- Accurate
  - Concise information on testing conditions and expected outcomes

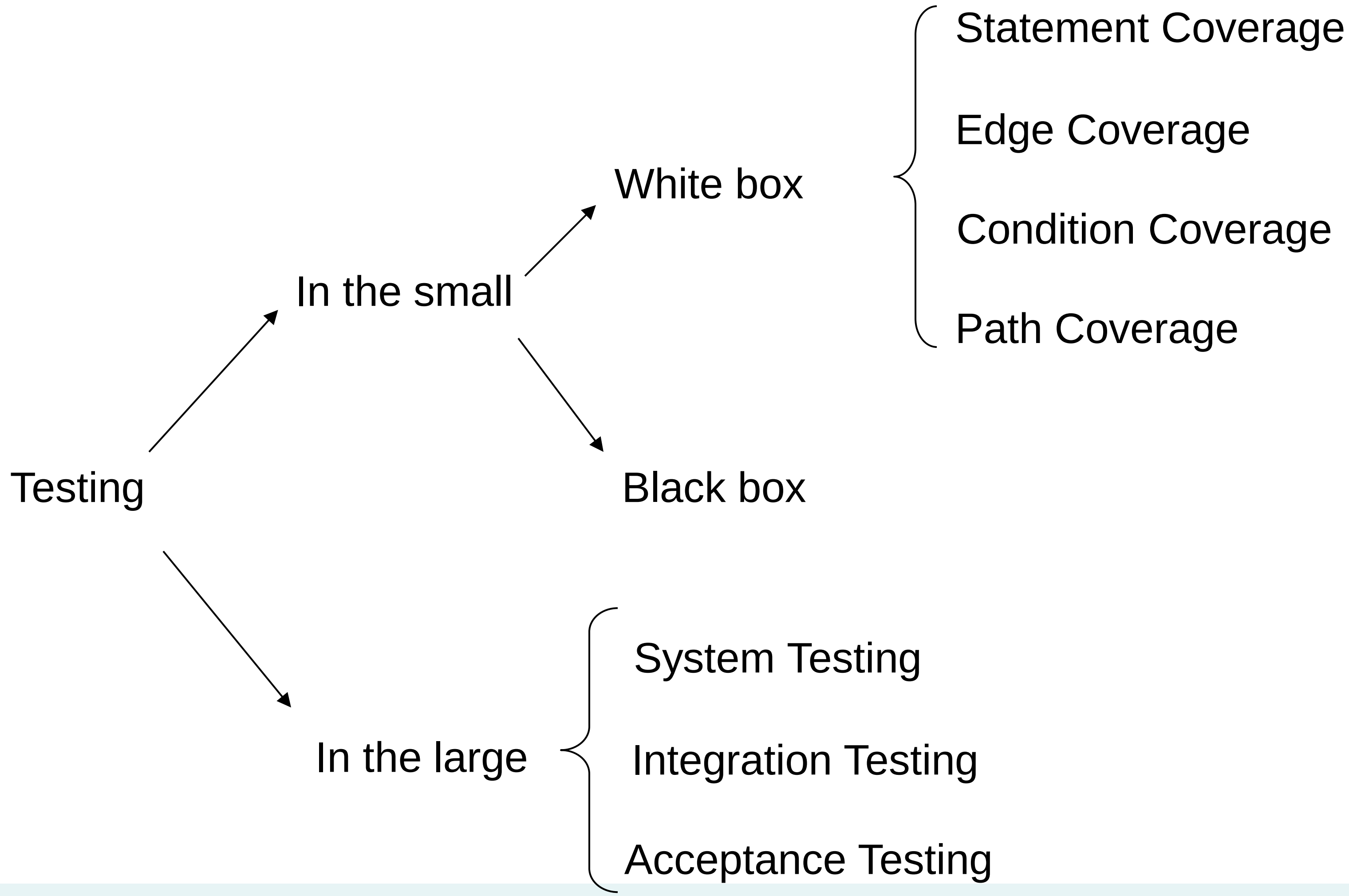
When to do testing?

- As early as possible
  - In the development life cycle
    - regardless of model (waterfall, spiral, etc.)
- As frequent as possible
  - Incremental development approach
- As exhaustive as possible
  - Test everything that is of interest
    - User interface, functionality, response time

Testing Myths

- Programs can be tested completely
  - Possible to achieve 100% failure free programs
- Testing is a fully automated activity
  - No need for software engineers
- The more tests used the more reliable programs are
  - Quantity over quality of tests

Software Testing Panorama



What about SPL testing?

- The key different is the large number of products that form a product line

- Some key questions:
  - Which testing should be done at domain engineering and which at application engineering?
  - How can testing artifacts be reused?
  - Should all products be tested? If not, which ones to select? In what order to test them?

Taxonomy of SPL Testing Approaches (1) – [Neto11]

- **Testing strategy**
  - What is the process to test the products?
- Possibilities
  - Test product by product
    - Do not take advantage of reuse
    - May not be feasible for SPL with many products
  - Incremental testing
    - Test a first product and use regression testing techniques for the next products
  - Opportunistic reuse
    - Ad hoc reuse of testing artifacts and effort
  - Design of test assets for reuse
    - Ideal case

Taxonomy of SPL Testing Approaches (2)

- **Type of analysis techniques**
  - What kinds of analysis can be used for testing?

- Two alternatives
  - Dynamic
    - Based on the execution of the products
  - Static
    - Examples: walkthrouhgs, model checking, type checking

Taxonomy of SPL Testing Approaches (3)

- **Testing Level**
  - Level of granularity and development activity
- Possibilities
  - At Domain Engineering
    - Unit testing: smallest unit of software implementation. Usually at the code level.
    - Integration testing: applied when modules (features) are integrated to form the common platform
  - At Application Engineering
    - System testing: does the product meet the required features?
    - Acceptance testing: is the customer satisfied with the product?
    - Integration testing: applied when modules (features) are integrated to create a product.

Other Important SPL Testing Issues

- Testability
  - Making a system more suitable for testing
  - Using additional information such as control flow to reduce the number of tests to apply

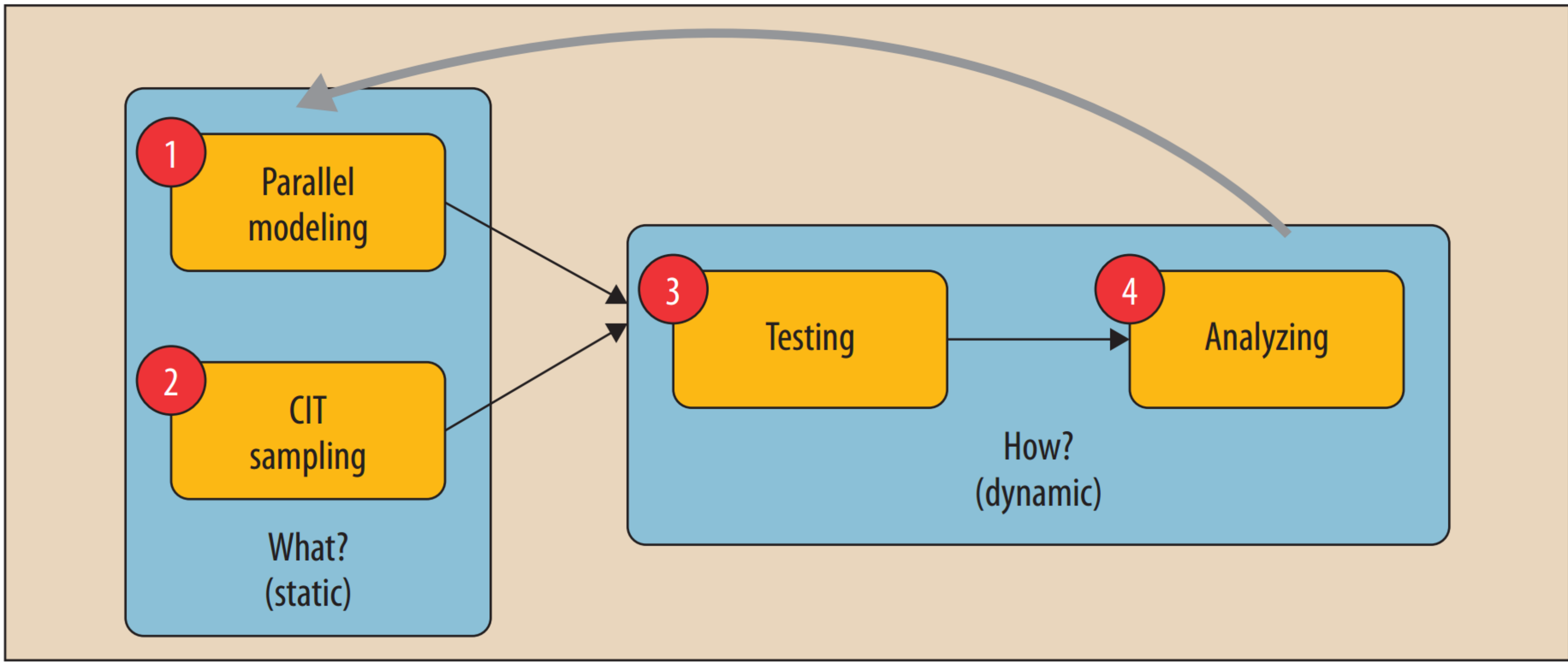
- Test coverage
  - Extensions of the coverage criteria of one-off programs (e.g. branch coverage, statement coverage, etc.)

Combinatorial Interaction Testing for SPL

Combinatorial Interaction Testing (CIT)

- Combinatorial interaction testing
  - Basic idea: select a representative subset of program elements where interaction errors are more likely to occur.
- Applied to SPL testing consist of the following steps
  - Select a set of products from the product line
  - Configure and implement the selected products
  - Apply standard testing techniques to each individual selected product

CIT Phases

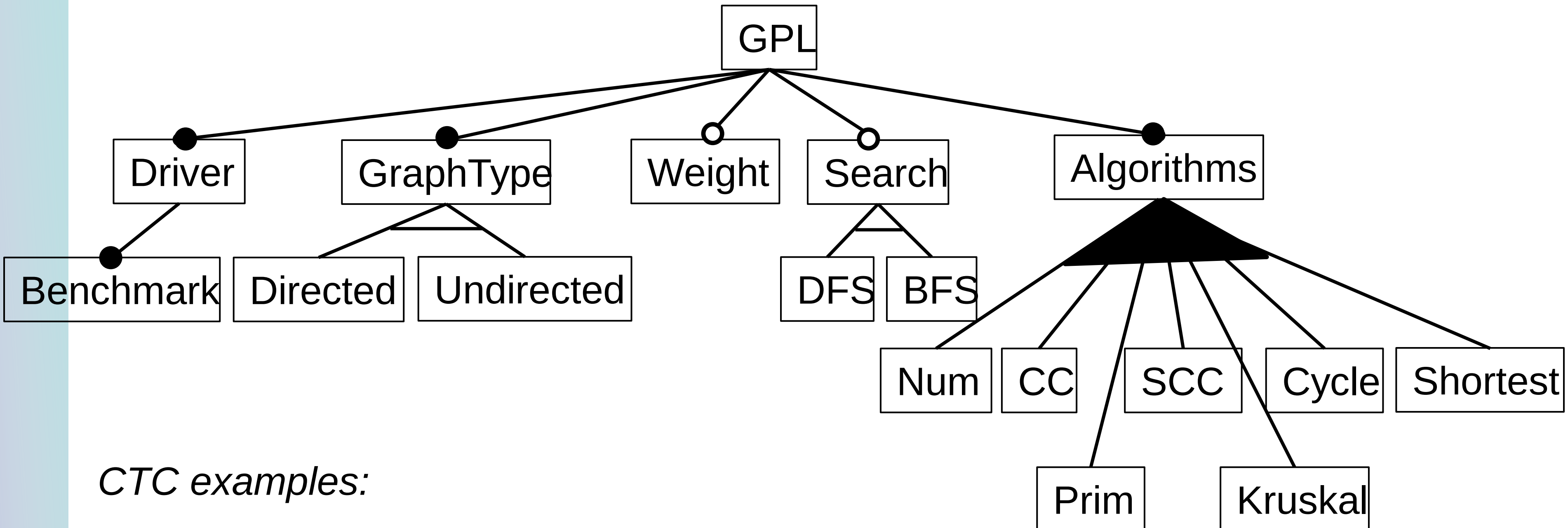


C. Yilmaz, S. Fouche', M. B. Cohen, A. A. Porter, G. Demiröz, and U. Koc, "Moving forward with combinatorial interaction testing," *IEEE Computer*, vol. 47, no. 2, pp. 37–45, 2014.

CIT Phases

- ***What*** products have to be tested?
  - To sufficiently test the SPL
- ***How*** should the products be tested?
  - Do we need to run all tests for all the products?
  - Which tests execute which features?
  - Prioritize test execution (e.g. importance, time, …)
- Feedback analysis information to what-phase
  - Execution time for certain tests -> reduce testing of this feature if possible
  - Features with more faults -> test more thoroughly

Running Example – Graph Product Line (GPL)



CTC examples:

Num requires Search

Cycle requires DFS

Prim requires Weight

Prim requires Undirected

Kruskal excludes Prim

Terminology (1)

- **Feature List (FL)** is the list of features in a feature model.
  - For GPL feature model the feature list FL is [GPL, Driver, Benchmark, GraphType, Directed, Undirected, Weight, Search, DFS, BFS, Algorithms, Num, CC, SCC, Cycle, Shortest, Prim, Kruskal].

Terminology (2)

- **Feature Set (FS)** is a 2-tuple [sel,⌐sel] where sel and ⌐sel are respectively the set of selected and not-selected features of a member product.

- Let FL be a feature list, thus

  sel $\subseteq FL$, ⌐sel $\subseteq FL$

  sel ∩ ⌐ sel = ∅

  sel ∪ ⌐ sel = FL

- The terms p.sel and p.⌐sel respectively refer to the set of selected and not-selected features of product p.

Terminology (3)

- **Valid feature set**
  - A feature set fs is valid in feature model fm, i.e. valid(fs, fm) holds, if it does not contradict any of the constraints introduced by fm.
  - Recall, we have seen how this operation is implemented in the previous class

In GPL, some valid feature sets

FS	GPL	Dri	Gtp	W	Se	Alg	B	D	U	DFS	BFS	N	CC	SCC	Cyc	Sh	Prim	Kru
fs0	✓	✓	✓	✓		✓	✓		✓								✓	
fs1	✓	✓	✓	✓	✓	✓	✓		✓	✓			✓					✓
fs2	✓	✓	✓		✓	✓	✓	✓		✓		✓			✓			
fs3	✓	✓	✓	✓	✓	✓	✓	✓			✓	✓				✓		
fs4	✓	✓	✓	✓	✓	✓	✓	✓		✓		✓		✓	✓	✓		
fs5	✓	✓	✓	✓	✓	✓	✓		✓	✓		✓	✓		✓		✓	
fs6	✓	✓	✓	✓	✓	✓	✓		✓		✓	✓	✓					✓
fs7	✓	✓	✓	✓	✓	✓	✓		✓	✓		✓	✓		✓			

Driver (Dri), GraphType (Gtp), Weight (W), Search (Se), Algorithms (Alg), Benchmark (B), Directed (D), Undirected (U), Num (N), Cycle (Cyc), Shortest (Sh), Kruskal (Kr).

Terminology (4)

- Definition of t-set
  - A t-set ts is a 2-tuple [sel,⌐sel] representing a partially configured product, defining the selection of t features of feature list FL:

    ts.se ∪ ts.⌐sel ⊆ FL ^

    ts.sel ∩ ts.⌐sel =∅ ^

    |ts.se ∪ ts. ⌐sel| = t.

- We say t-set ts is covered by feature set fs iff

  ts.sel ⊆ fs.sel ^ ts. ⌐sel ⊆ fs. ⌐ sel.

- A t-set ts is valid in a feature model fm if there exists a valid feature set fs that covers ts.

Example of 2-wise sets – pairwise testing (1)

FS	GPL	Dri	Gtp	W	Se	Alg	B	D	U	DFS	BFS	N	CC	SCC	Cyc	Sh	Prim	Kru
fs0	✓	✓	✓	✓		✓	✓		✓								✓	
fs1	✓	✓	✓	✓	✓	✓	✓		✓	✓			✓					✓
fs2	✓	✓	✓		✓	✓	✓	✓		✓		✓			✓			
fs3	✓	✓	✓	✓	✓	✓	✓	✓			✓	✓				✓		
fs4	✓	✓	✓	✓	✓	✓	✓	✓		✓		✓		✓	✓	✓		
fs5	✓	✓	✓	✓	✓	✓	✓		✓	✓		✓	✓		✓		✓	
fs6	✓	✓	✓	✓	✓	✓	✓		✓		✓	✓	✓					✓
fs7	✓	✓	✓	✓	✓	✓	✓		✓	✓		✓	✓		✓			

Driver (Dri), GraphType (Gtp), Weight (W), Search (Se), Algorithms (Alg), Benchmark (B), Directed (D), Undirected (U), Num (N), Cycle (Cyc), Shortest (Sh), Kruskal (Kr).

2-wise set [{Driver},{Prim}] covered by fs1-fs4, fs6-fs7

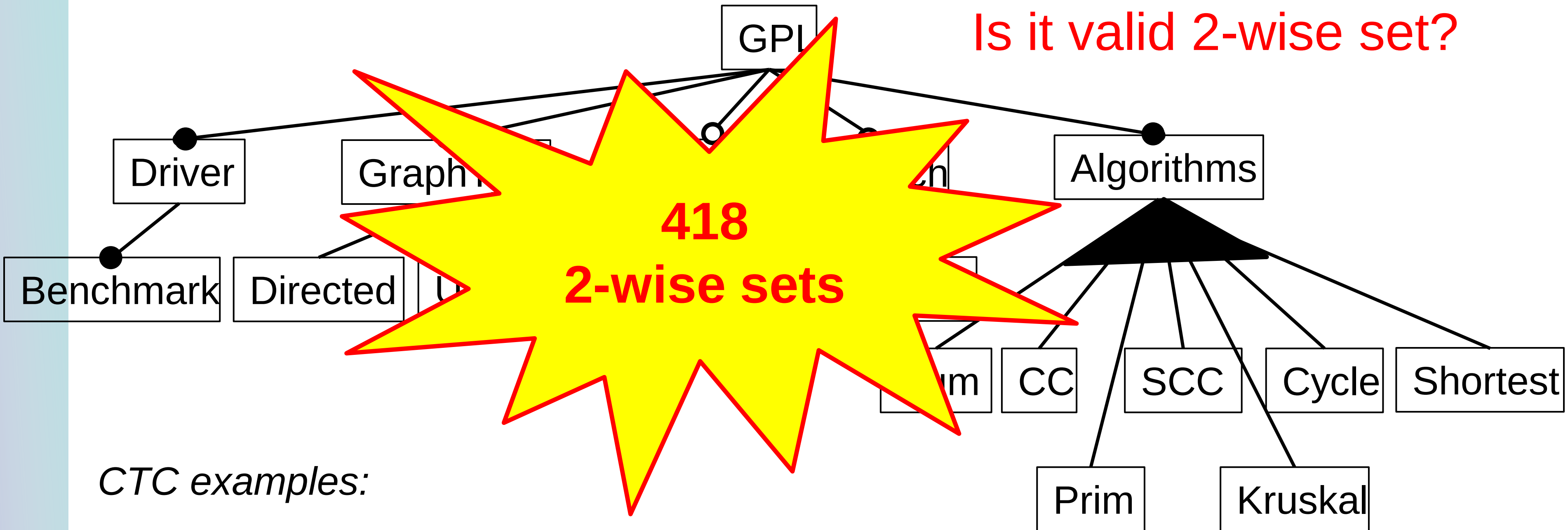
Example of 2-wise sets – pairwise testing (2)

FS	GPL	Dri	Gtp	W	Se	Alg	B	D	U	DFS	BFS	N	CC	SCC	Cyc	Sh	Prim	Kru
fs0	✓	✓	✓	✓		✓	✓		✓								✓	
fs1	✓	✓	✓	✓	✓	✓	✓		✓	✓			✓					✓
fs2	✓	✓	✓		✓	✓	✓	✓		✓		✓			✓			
fs3	✓	✓	✓	✓	✓	✓	✓	✓			✓	✓				✓		
fs4	✓	✓	✓	✓	✓	✓	✓	✓		✓		✓		✓	✓	✓		
fs5	✓	✓	✓	✓	✓	✓	✓		✓	✓		✓	✓		✓		✓	
fs6	✓	✓	✓	✓	✓	✓	✓		✓		✓	✓	✓					✓
fs7	✓	✓	✓	✓	✓	✓	✓		✓	✓		✓	✓		✓			

Driver (Dri), GraphType (Gtp), Weight (W), Search (Se), Algorithms (Alg), Benchmark (B), Directed (D), Undirected (U), Num (N), Cycle (Cyc), Shortest (Sh), Kruskal (Kr).

2-wise set [{Kruskal, DFS}, ø] covered by fs1

Example of 2-wise sets – pairwise testing (3)



Is it valid 2-wise set?

CTC examples:

Num requires Search
Cycle requires DFS
Kruskal excludes Prim
Prim requires Weight
Prim requires Undirected

2-wise set [ø, {Directed, Undirected}]

Terminology (5)

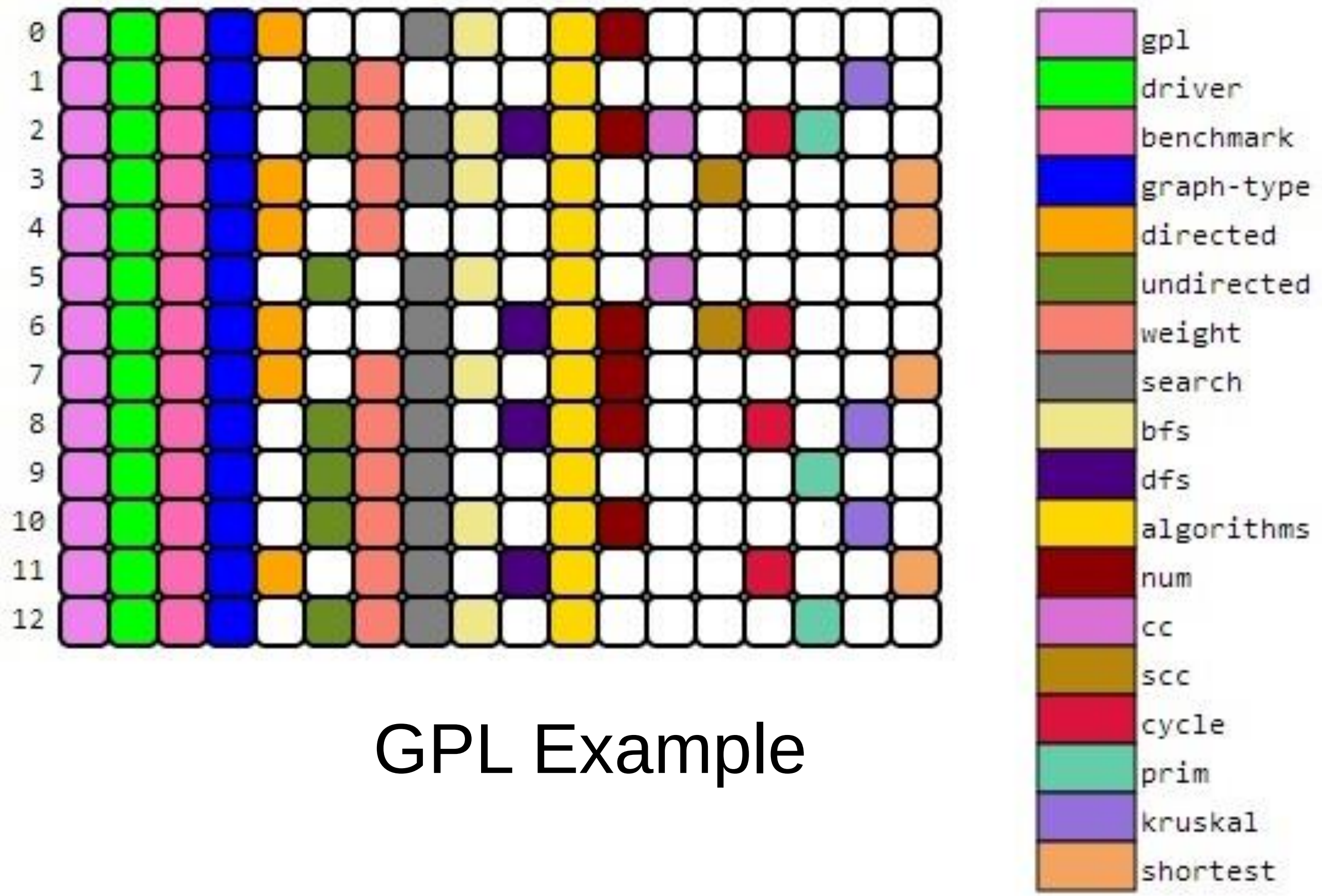
- **Definition t-wise covering array**
  - A tCA for a feature model fm is a set of valid feature sets that covers all valid t-sets denoted by fm.
- Questions:
  - How can t-wise covering arrays be computed?
  - Can a minimum covering array be found?
- Answer
  - Complex problem – NP complete

Terminology (6) – Pairwise Testing SPL

- Pairwise Covering Array
  - A test suite whose products contain ***all*** the pairwise (i.e. t=2) feature combinations denoted by the corresponding feature model

- For example, GPL
  - Has 73 different products
  - Has 418 valid pairs of features to cover

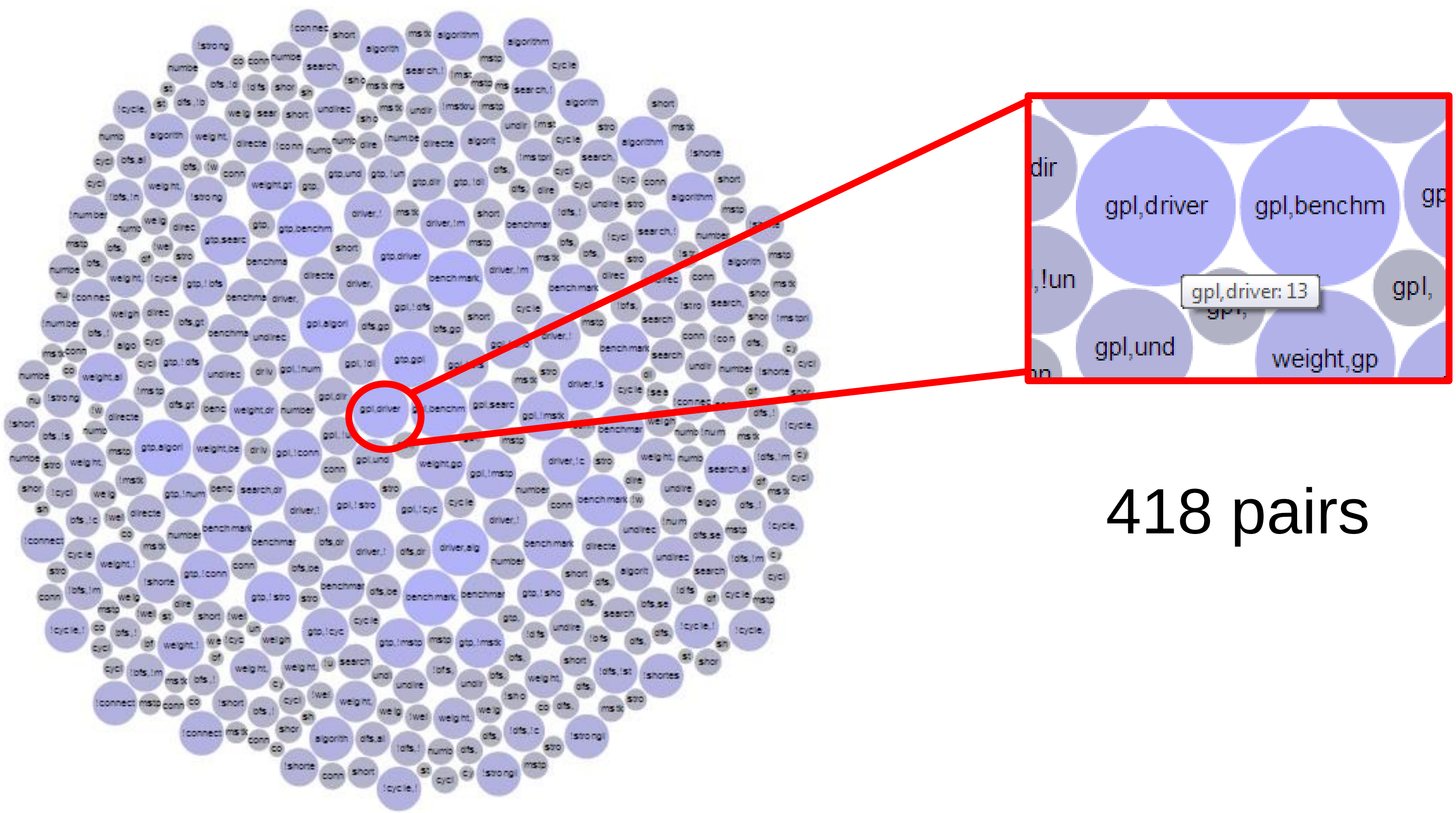
Covering Array Visualization
VISSOFT13



GPL Example

JOHANNES KEPLER
UNIVERSITY LINZ

Bubble and Heat Map Coverage – GPL Example



418 pairs

SPL Pairwise CIT

- Question:
  - How to compute the pairwise covering arrays from the feature models?

- Multiple approaches:
  - Perrouin et al. – Alloy based
  - MoSo-Polite – CSP based technique
  - PACOGEN – constraint programming
  - ICPL – greedy algorithm
  - CASA – simulated annealing
  - Ensan – genetic algorithm with cyclomatic complexity metric
  - Henard et al. (2012) search based on similarity metrics
  - …..

Search-Based Software Engineering (SBSE)

Basics – Interlude

Evolutionary Computation

- Evolutionary Computation
  - Includes several stochastic search methods which computationally simulate the natural evolutionary process

- Example techniques
  - Genetic algorithms
    - Individuals are typically represented as binary strings, commonly used in numerical optimization problems
  - Genetic programming
    - Individuals encode programs typically represented as tree-structures whose fitness function evaluate how well the programs execute a computational task

Evolutionary Computation Illustration

- Randomly creates an initial population
- Evaluates the initial population
- At each generation
  1. select the individuals with best fitness
  2. mutate their characteristics
  3. re-evaluate them

Algorithm 1 Basic Evolutionary Algorithm

$t \leftarrow 0$
initialize $P(t)$
evaluate $P(t)$
while not *termination − condition* **do**
  $t \leftarrow t + 1$
  select $P(t)$ *from* $P(t-1)$
  mutate $P(t)$
  evaluate $P(t)$
end while

Search-Based Software Engineering (SBSE)

- Search-Based Software Engineering focuses on the application of search-based optimization techniques to problems in software engineering [Harman10]

- Typical techniques are:
  - Basic searches, e.g. hill-climbing, simulated annealing
  - Techniques based on evolutionary computation

Hill Climbing Illustration

- Looks at a neighborhood of SampleSize states and selects the one with best fitness
- Main problem
  - Can get stuck in a local maxima

Algorithm 1 Steepest Ascent Hill Climbing

```
X ← random initial state
I ← 0
Best ← X
while (I<MaxIter) ∧ (evaluate(Best) ≠ BestFitness) do
    S ← 0
    while S<SampleSize do
        X′ ← move(Best)
        if evaluate(X′) better than evaluate(X) then
            X ← X′
        end if
        S ← S + 1
    end while
    if evaluate(X) better than evaluate(Best) then
        Best ← X
    end if
    I ← I + 1
end while
return Best
```

A Genetic Algorithm for CIT SPL Testing

Overview

Prioritized Generic Solver (PGS)

- A constructive genetic algorithm for computing a pairwise covering array from a feature model

- Salient characteristics
  - An individual represents a sets of products
  - PGS adds a new product to the partial solution in each iteration until all pairwise combinations are covered
  - In each iteration the algorithm tries to find the product that adds the most coverage to the partial solution
  - Provides a *fix* operation in case the new offspring are not valid products

PGS Algorithm Overview

Algorithm 1. Pseudocode of PGS.

```
proc Input:(PGS)      //Algorithm parameters in 'PGS'
TS ← ∅    // Empty the test suite
RP ← pairs_to_cover(FM)    // Initialize the pairwise configurations
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 (PGS.popSize / 2) do
            parents←Selection(P(t))
            offspring←Recombination(PGS.Pc,parents)
            offspring←Mutation(PGS.Pm,offspring)
            Fix(offspring)
            Evaluate_Fitness(offspring)
            Insert(offspring,Q)
        end for
        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
end_proc
```

PGS comparison

- We select 19 realistic feature models
  - There exist an available implementation of the SPLs they model
- Compared against
  - ICPL – greedy approach
  - CASA – simulated annealing approach
- Preliminary results
  - Fixing operation imposes a heavy performance penalty
  - PGS obtains comparable results in size with CASA

Multi-Objective Evolutionary Algorithms for Pairwise Testing of SPLs

Multi-Objective Optimization (MOO) for SPL Testing [CEC14]

- Some recent work
  - Wang, Ali, Gotlieb – GECCO13
  - Henard, Papadakis, Perrouin, Klein, Traon – SPLC13

- Common thread
  - Use a linearization approach where each optimization objective is given a weight and later added

$$\sum_{i=1..n} w_i \; x \; Obj_i$$

  - Coverage and test suite size are the recurrent optimization objectives

Our Two Objectives

For pairwise testing:

1. Minimize test suite size

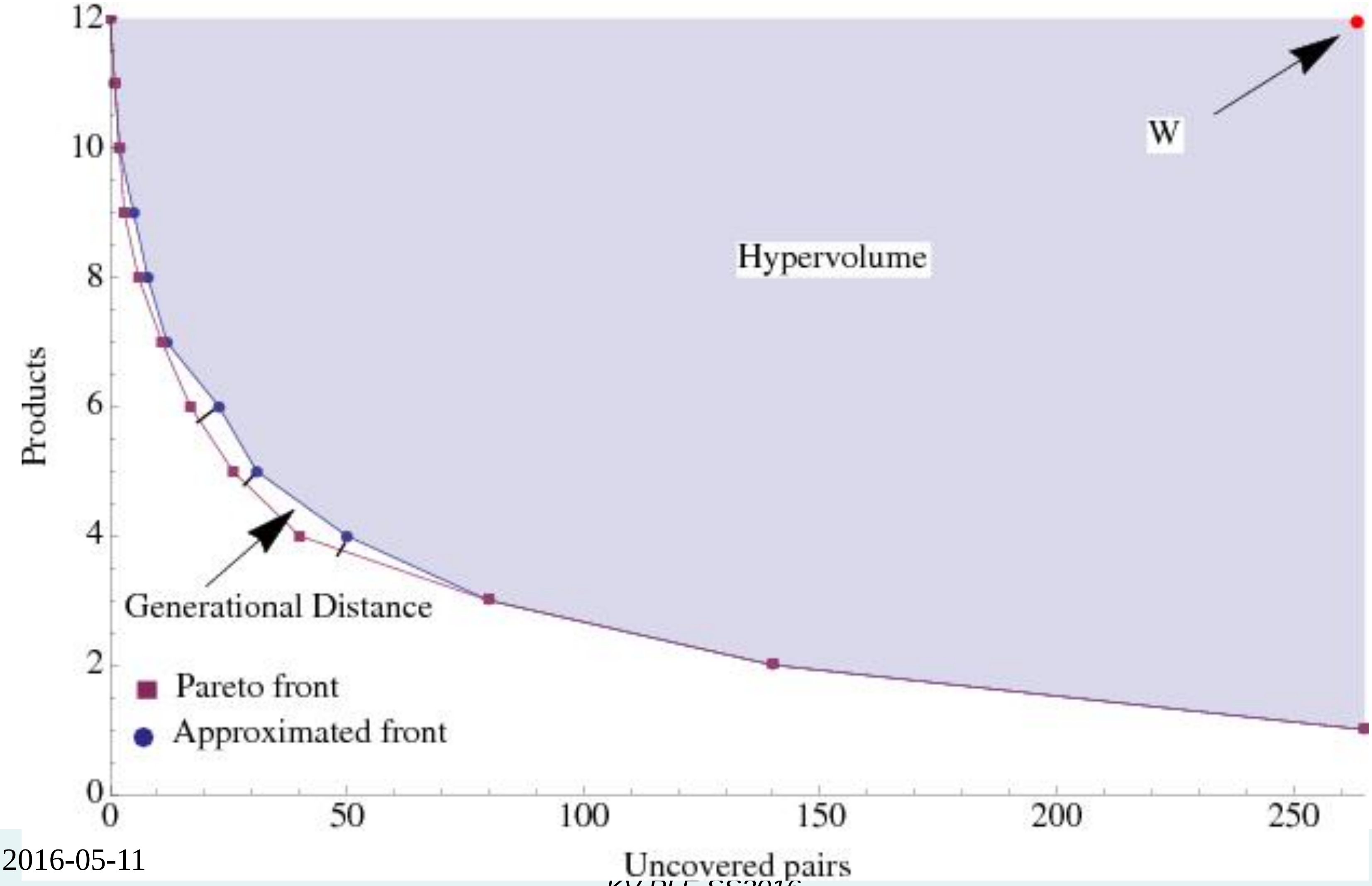
2. Minimize pairs to uncover
   - Out of the total of valid pairs denoted by a feature model

Our work

- Uses *classical* MOO algorithms
  - NSGA-II – crowding distance and ranking
  - MOCell – cellular GA, based on neighbourhood
  - SPEA2 – population and archive
  - PAES – evolution strategy

- Use standard comparison MOO metrics
  - Hypervolume
    - Volume in the objective space covered by a non-dominated set of solutions
  - Generational distance
    - Summarizes the distance between the points in a front and the Pareto front.

Metrics Example – GPL



Impact of Seeding

- Seeding
  - Embed domain knowledge into the individuals of the population

- We used 3 seeding strategies for the initial population
  - Size-based Random Seeding
    - Computed a covering array with CASA and use its size to generate the population
  - Greedy Seeding
    - Greedily computes a covering array and uses its elements to generate the population
  - Single-Objective Based Seeding
    - Creates a population based on a single-objective output CASA

Experimental Setting

- We select 17 realistic feature models
  - Feature models publicly available (not reversed engineered)
  - There exist an available implementation of their SPLs
- Executed 30 independent runs
- Perfomed standard statistical analysis

Findings

- Seeding has a strong impact on the final result
  - Best strategy was Single-Objective seeding

- What is the best algorithm?
  - Statistically no best winner → NSGA-II, SPEA2, MOCell are comparable on both metrics
  - PAES offer lower quality indicators
    - possible reason – trajectory based approach

Future Work

- SBSE SPL testing
  - Extend the set of feature models case studies
  - Explore beyond pairwise testing
  - Explore beyond two objectives

Systematic Mapping Study on CIT Testing for SPLs

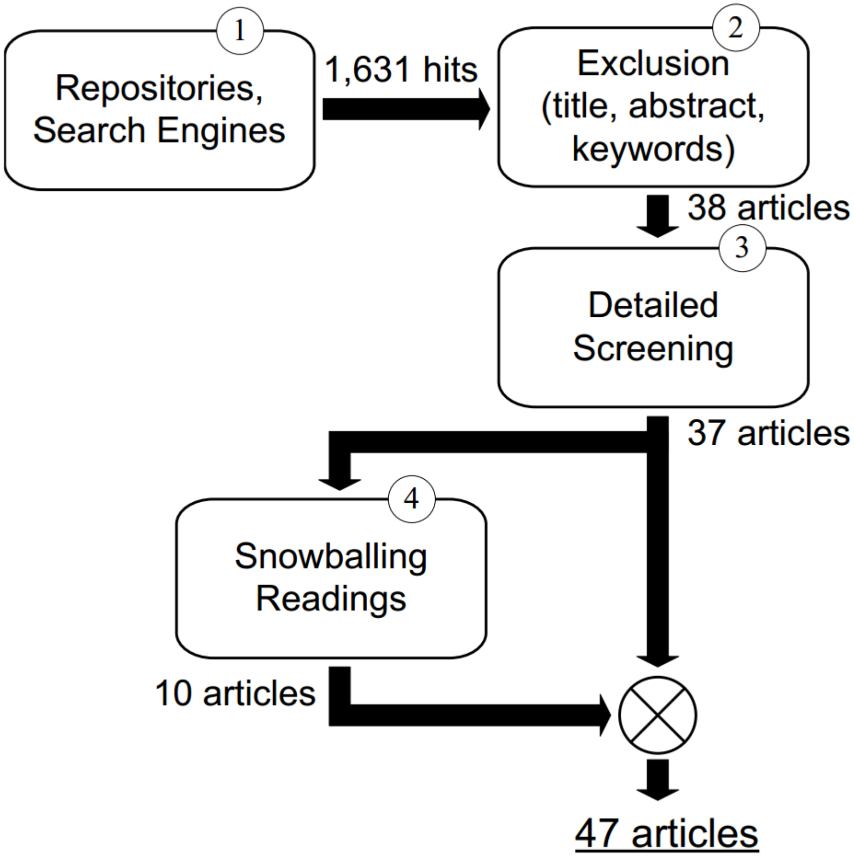
Lopez-Herrejon et al. [ICSTw2015]

Sytematic Mapping Study

- Systematic mapping study
  - One of the approaches advocated by Evidence-Based Software Engineering whose goal is to provide an overview of the results available within an area by categorizing them along criteria such as type, forum, frequency, etc

- Research questions
  - RQ1: What are the techniques that have been used for CIT in SPLs?
  - RQ2: What phases of CIT have been explored for SPLs?
  - RQ3: What are the case studies used for evaluation of the CIT approaches applied to SPLs and what is their provenance?
  - RQ4: What are the publication fora used?

Results


1
Repositories, Search Engines
1,631 hits
2
Exclusion (title, abstract, keywords)
38 articles
3
Detailed Screening
37 articles
4
Snowballing Readings
10 articles
47 articles

RQ1: Techniques Used

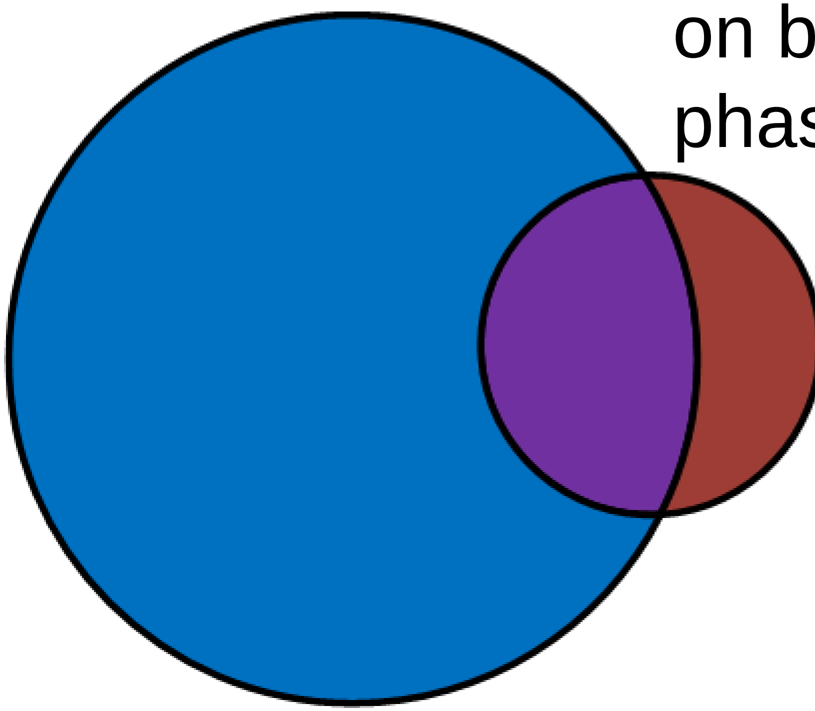
- 13 different techniques for CIT for SPLs
- Most frequent techniques
  - 13 publications greedy algorithms
  - 8 publications constraint programming

Technique	Primary Sources Identifiers
Constraint Handling	S21
Constraint Programming	S1, S3, S13, S18, S25, S26, S30, S37
Evolutionary Algorithm (1+1)	S10, S32, S35
Exact Multi-Objective Algorithm	S19
Generic	S2, S11, S24, S27, S33, S36
Genetic Algorithm	S16, S40, S45, S47
Greedy algorithm	S4, S5, S8, S14, S15, S17, S22, S23, S28, S29, S31, S34, S38
Model-Based	S43
Multi-Objective Evolutionary Algorithm	S20, S44, S46
Random Search	S41
Simulated Annealing	S14, S23, S39, S42
Static Analysis	S7, S9, S12
Statistical Test	S6

RQ2: CIT Phases

9 papers on both phases

36 papers on *what*-phase only



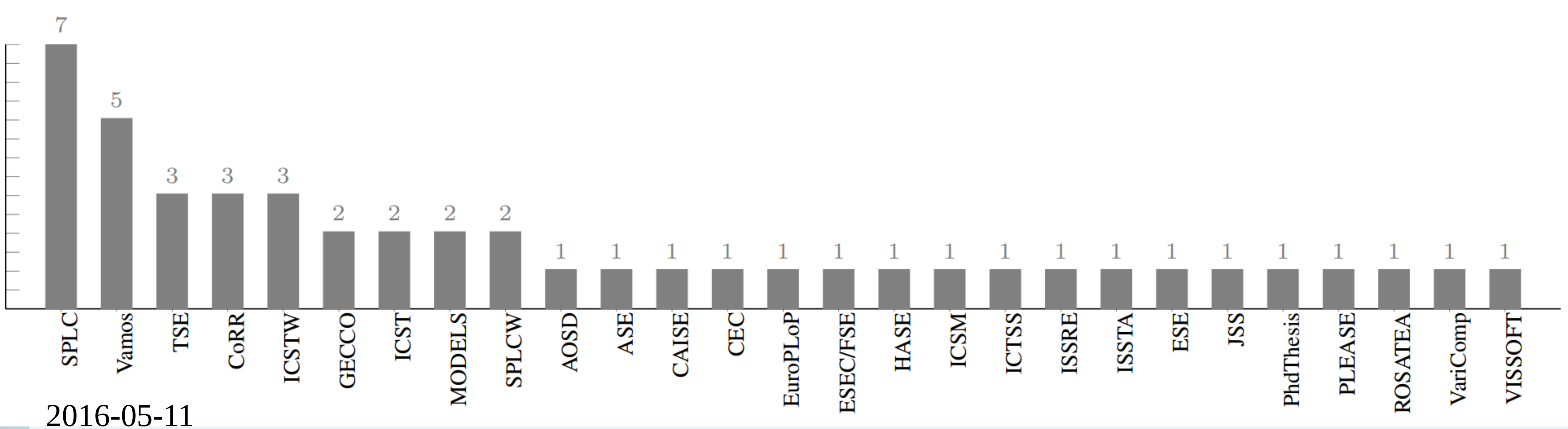
2 papers on how-phase only

RQ3: Case Studies and Provenance

- The Linux kernel is still a frequently used SPL case study
- The majority of case studies only consisted of variability models – no code available
- Most of the papers used feature models to model variability (39 papers)
- Most of them from the SPLOT repository (Software Product Line Online Tools – a Repository of Feature Models and Tools, etc.)

RQ4: Publication Fora

- 24 conference, 5 journal and 18 other (i.e. workshop, PhD thesis, …) publications
- SPLC and VaMos
  - Product Line specific conference and workshop
- ICST and ICSTW
  - Testing specific



Open Question

- How do all these approaches compare?
- Some incipient work
  - Comparison frawework applied to Alloy-Based alternative and constraint programming [Oster10]
  - Proposed by Perrouin et al. [Perrouin 12].
  - Problems
    - Limited number of feature models studied
    - Only two approaches

Bibliographic References (1)

- M. Cohen, M. Dwyer, and J. Shi. Constructing interaction test suites for highly-configurable systems in the presence of constraints: A greedy approach. IEEE Transactions on Software Engineering, 34(5):633 –650.
- B.J. Garvin, M.B. Cohen, and M.B. Dwyer. Evaluating improvements to a meta-heuristic search for constrained interaction testing. Empirical Software Engineering}, 16(1):61--102, 2011.
- Ensan, F., Bagheri, E., Gasevic, D. Evolutionary search-based test generation for software product line feature models. CAiSE. Volume 7328 of Lecture Notes in Computer Science., Springer (2012) 613—628
- Perrouin, G., Sen, S., Klein, J., Baudry, B., Traon, Y.L.. Automated and scalable t-wise test case generation strategies for software product lines. ICST, IEEE Computer Society (2010).
- Oster, S., Markert, F., Ritter, P. Automated incremental pairwise testing of software product lines. SPLC. Volume 6287 of Lecture Notes in Computer Science., Springer (2010) 196--210

Bibliographic References (2)

- Hervieu, A., Baudry, B., Gotlieb, A. Pacogen: Automatic generation of pairwise test configurations from feature models. ISSRE, IEEE (2011) 120—129.
- Lochau, M., Oster, S., Goltz, U., Schurr, A. Model-based pairwise testing for feature interaction coverage in software product line engineering. Software Quality Journal 20(3-4) (2012) 567—604.
- Cichos, H., Oster, S., Lochau, M., Schurr, A. Model-based coverage-driven test suite generation for software product lines. Models 2011. 425—439.
- Perrouin, G., Oster, S., Sen, S., Klein, J., Baudry, B., Traon, Y.L. Pairwise testing for software product lines: comparison of two approaches. Software Quality Journal 20(3-4) (2012) 605—643.
- Lee, J., Kang, S., Lee, D. A survey on software product line testing. SPLC 2012, 31—40.
- Engstrom, E., Runeson, P. Software product line testing - a systematic mapping study. Information & Software Technology 53(1) (2011) 2—13.

Bibliographic References (3)

- Da Mota Silveira Neto, P.A., do Carmo Machado, I., McGregor, J.D., de Almeida, E.S., de Lemos Meira, S.R. A systematic mapping study of software product lines testing. Information & Software Technology 53(5) (2011) 407—423.
- Johansen, M.F., Haugen, O., Fleurey, F. An algorithm for generating t-wise covering arrays from large feature models. SPLC 2012, 46—55
- R E. Lopez-Herrejon, S. Fischer, R. Ramler, and A. Egyed. A First Systematic Mapping Study on Combinatorial Interaction Testing for Software Product Lines. IEEE Eighth International Conference on Software Testing, Verification and Validation Workshops (ICSTW).