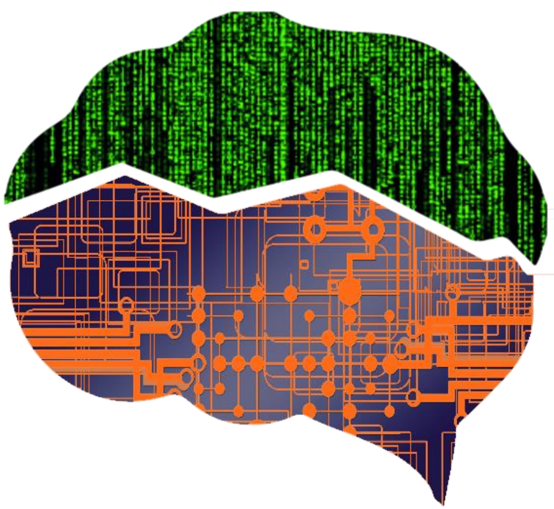
Software Product Line Engineering

Non-Functional Properties

Christian Kästner (Carnegie Mellon University)
Sven Apel (Universität Passau)
Norbert Siegmund (Bauhaus-Universität Weimar)
Gunter Saake (Universität Magdeburg)



Introduction

- Not considered so far:
  - How to configure a software product line?
  - How about non-functional properties?
  - How to measure and estimate a variant’s non-functional properties?

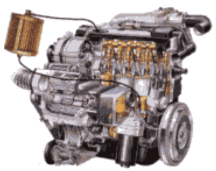
Agenda

- Configuration and non-functional properties
- Approaches for measurement and estimation
- Experience reports
- Outlook

Configuration of Software Product Lines

Recap: Configuration and Generation Process

Reusable artifacts

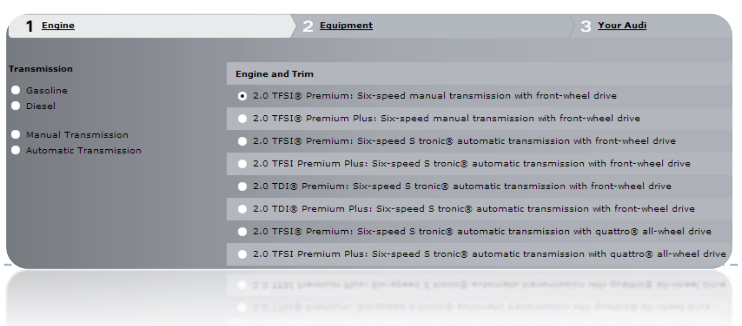




Car variants

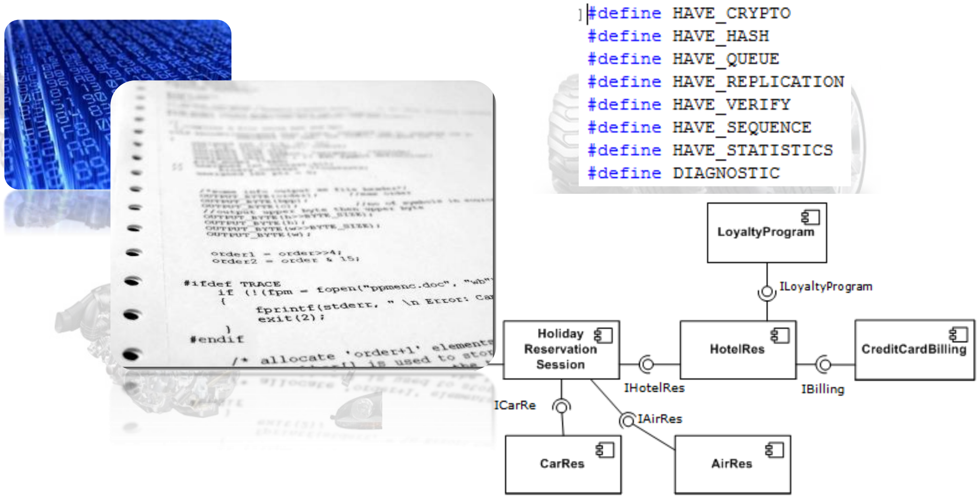
Variant generation

Configuration based on requirements

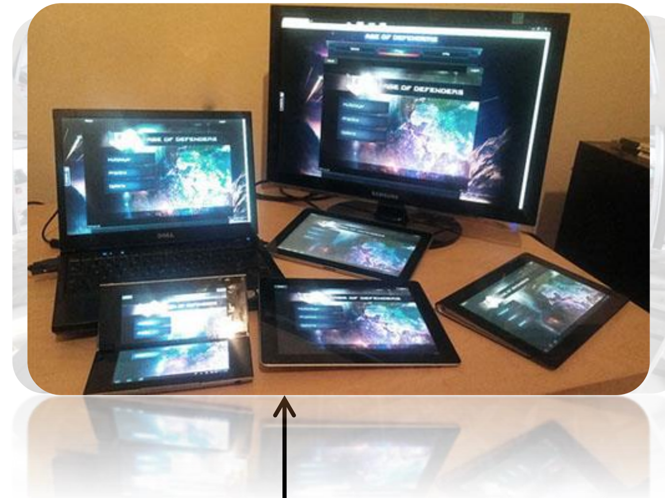


Recap: Configuration and Generation Process

Reusable artifacts (code, documentation, etc.)

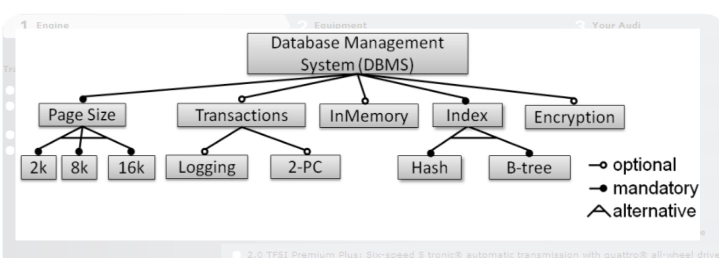


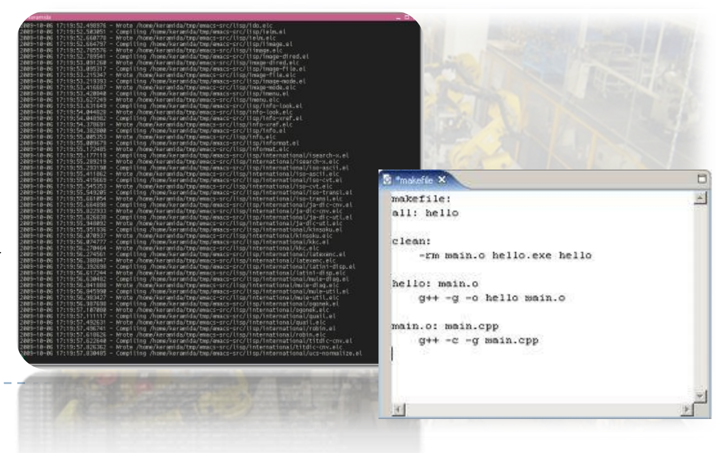
Variants



Variant generation

Configuration based on requirements





Configuration with Feature Models

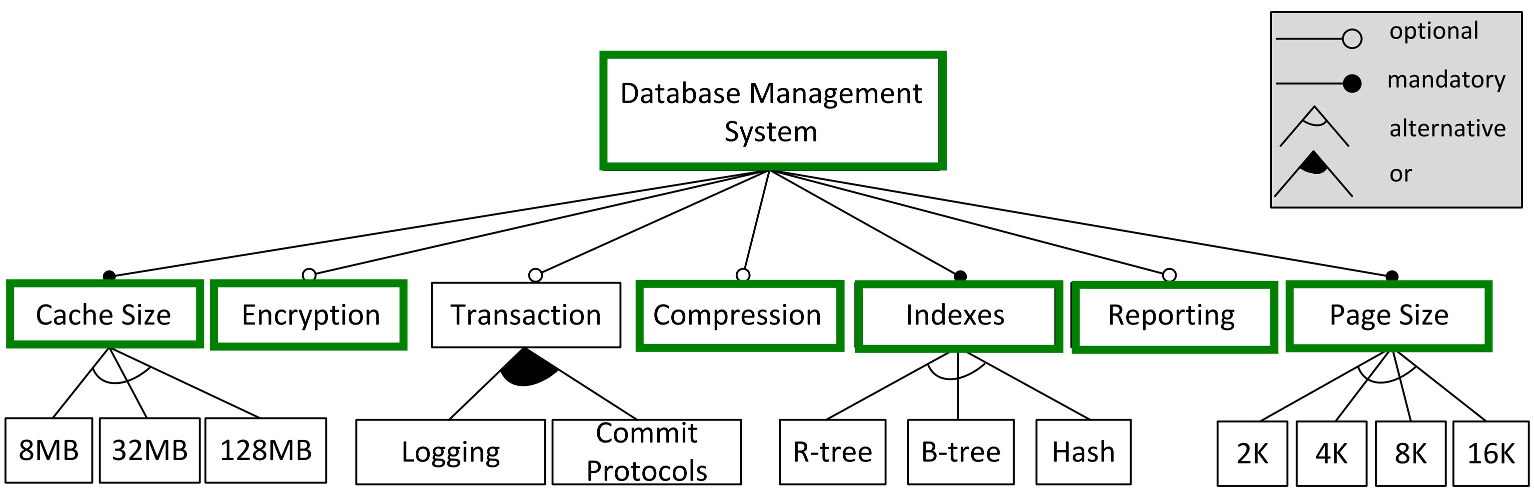
Functional requirements


Encryption


Compression


Data analysis


Reporting



Partial feature selection

Non-Functional Requirements

Non only functionality is important

Performance

Memory consumption

Footprint

Non-Functional Properties: Definition(s)

- Also known as quality attributes
- Over 25 definitions (see [6])
- In general:

  Any property of a product that is not related with functionality represents a non-functional property.

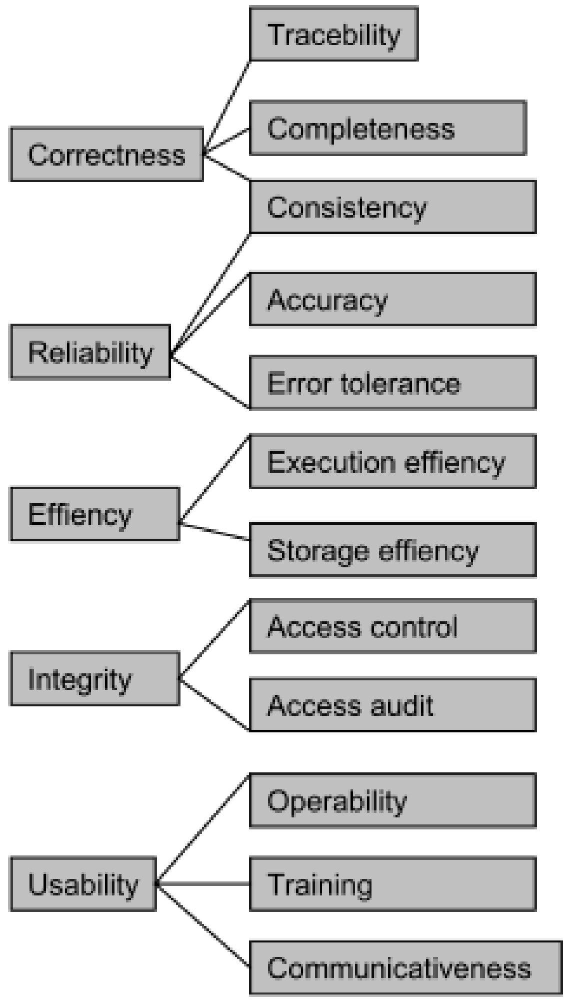
- Different models describe relationships among non-functional properties

McCall‘s Quality Model I [7]

- Modelling of quality attributes and factors to simplify communcation between developers and users
- Hierarchical model:
  - 11 factors (specify product; external user view)
  - 23 quality criteria (for development; internal developer view)
  - Metrics (to control and evaluate results)

McCall's Quality Model I [7]

External View



Internal View

ISO Standard 9126 + SO/IEC 25010:2011

Quelle: Wikipedia

SO/IEC 25010:2011 defines:

1. A quality in use model composed of five characteristics (some of which are further subdivided into subcharacteristics) that relate to the outcome of interaction when a product is used in a particular context of use. This system model is applicable to the complete human-computer system, including both computer systems in use and software products in use.

2. A product quality model composed of eight characteristics (which are further subdivided into subcharacteristics) that relate to static properties of software and dynamic properties of the computer system. The model is applicable to both computer systems and software products.

Categorization

- Quantitative
  - Response time (performance), throughput, etc.
  - Energy- and memory consumption
  - Measurable properties, metric scale
  - Easy to evaluate

- Qualitative
  - Extensibility
  - Error freeness
  - Robustness
  - Security
  - No direct measurement (often, no suitable metric)

How to configure with non-functional properties in mind?

Non-functional requirements

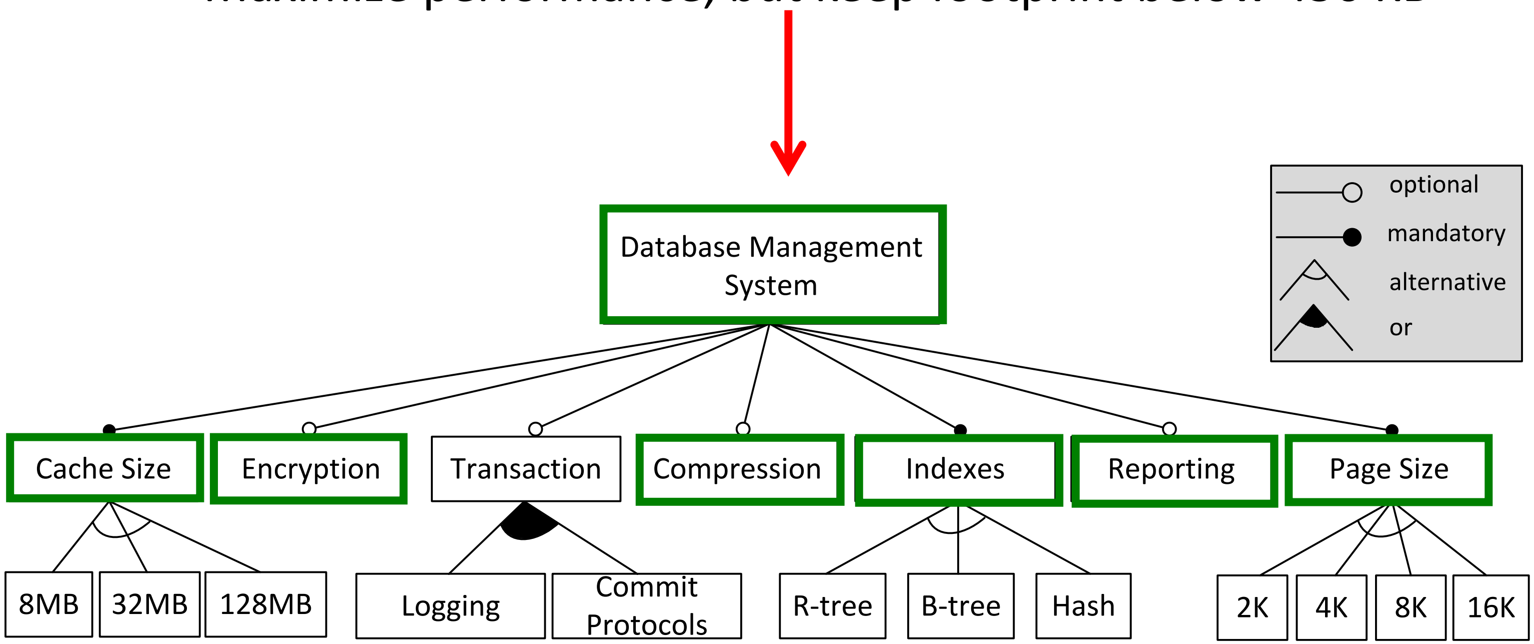

Energy consumption


Performance


Memory consumption

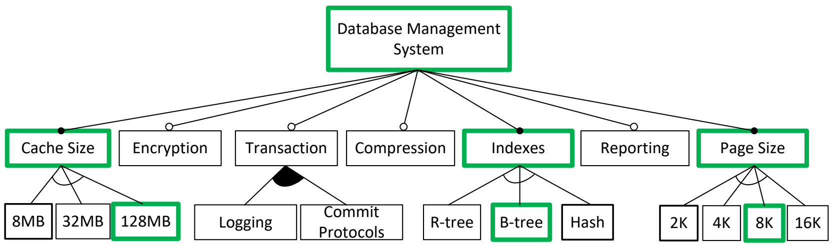

Footprint

Maximize performance, but keep footprint below 450 KB



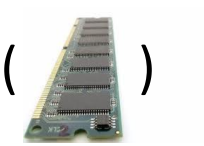
Motivating Questions of Practical Relevance

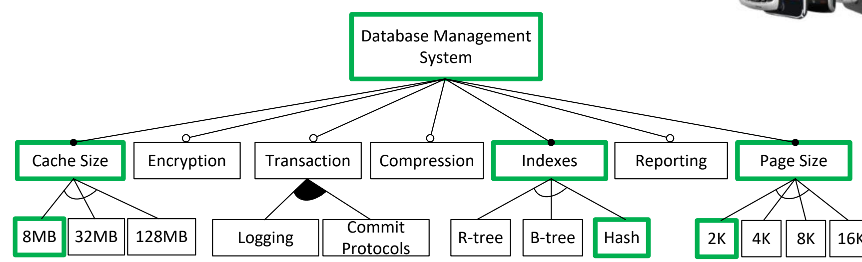
- What is the footprint of a variant for a given feature selection?



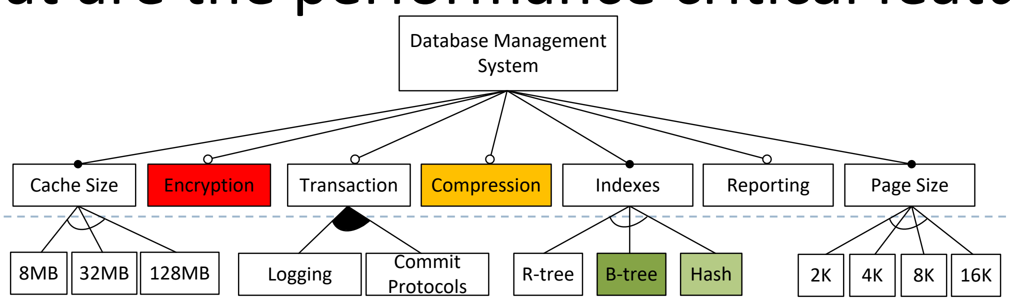
425 KB

- What is the best feature selection to minimize memory consumption?

Min()

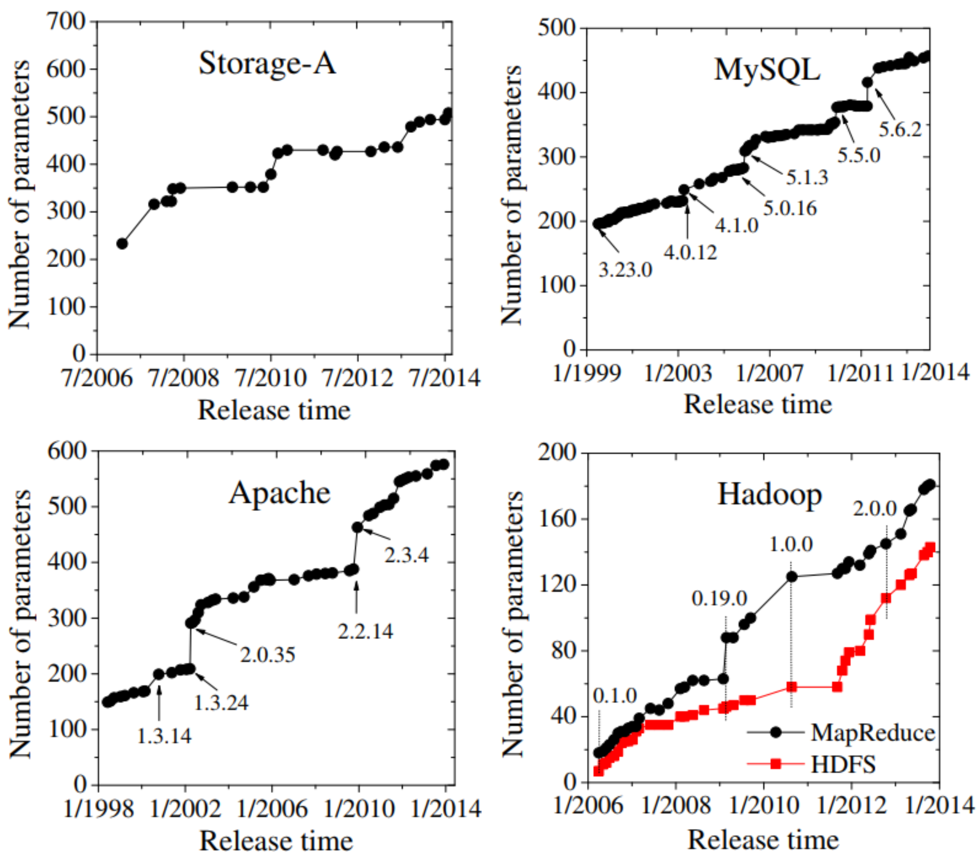


- What are the performance critical features?



Practical Relevance

Configuration complexity: [1] Xu et al. FSE’15: Developers and users are overwhelmed with configuration options



Substantial increase in configurability

Unused optimization (up to 80% of options ignored)

Parameter:	optimizer_prune_level (Boolean)	/*MySQL*/
Desc.:	Controls the heuristics applied during query optimization to prune less-promising partial plans from the optimizer search space.	
Values:	0 or 1	
Usage:	No user set the parameter in our dataset.	

(a) **Empirical, heuristic usages**

Parameter:	key_cache_block_size (Numeric)	/*MySQL*/
Desc.:	The size in bytes of blocks in the key cache.	
Values:	[512, 16384]	
Usage:	All the users stay with the default value 1024 in our dataset.	

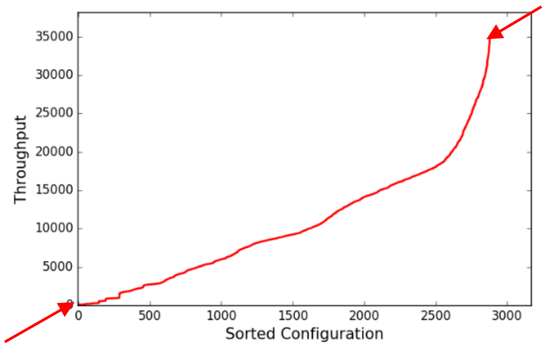
Why Should We Care?

Outdated default configurations: [2] Van Aken et al. ICMD’17: Default configuration assumes 160MB RAM

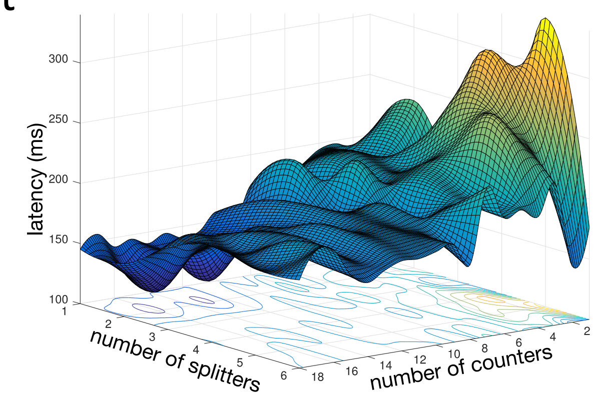
Non-optimal default configurations: [4] Herodotuo et al. CIDSR’11: Default configuration results in worst-case execution time

Non-optimal default configurations: [3] Jamshidi et al., MASCOTS’16: Changing configuration is key to tailor the system to the use case

Best



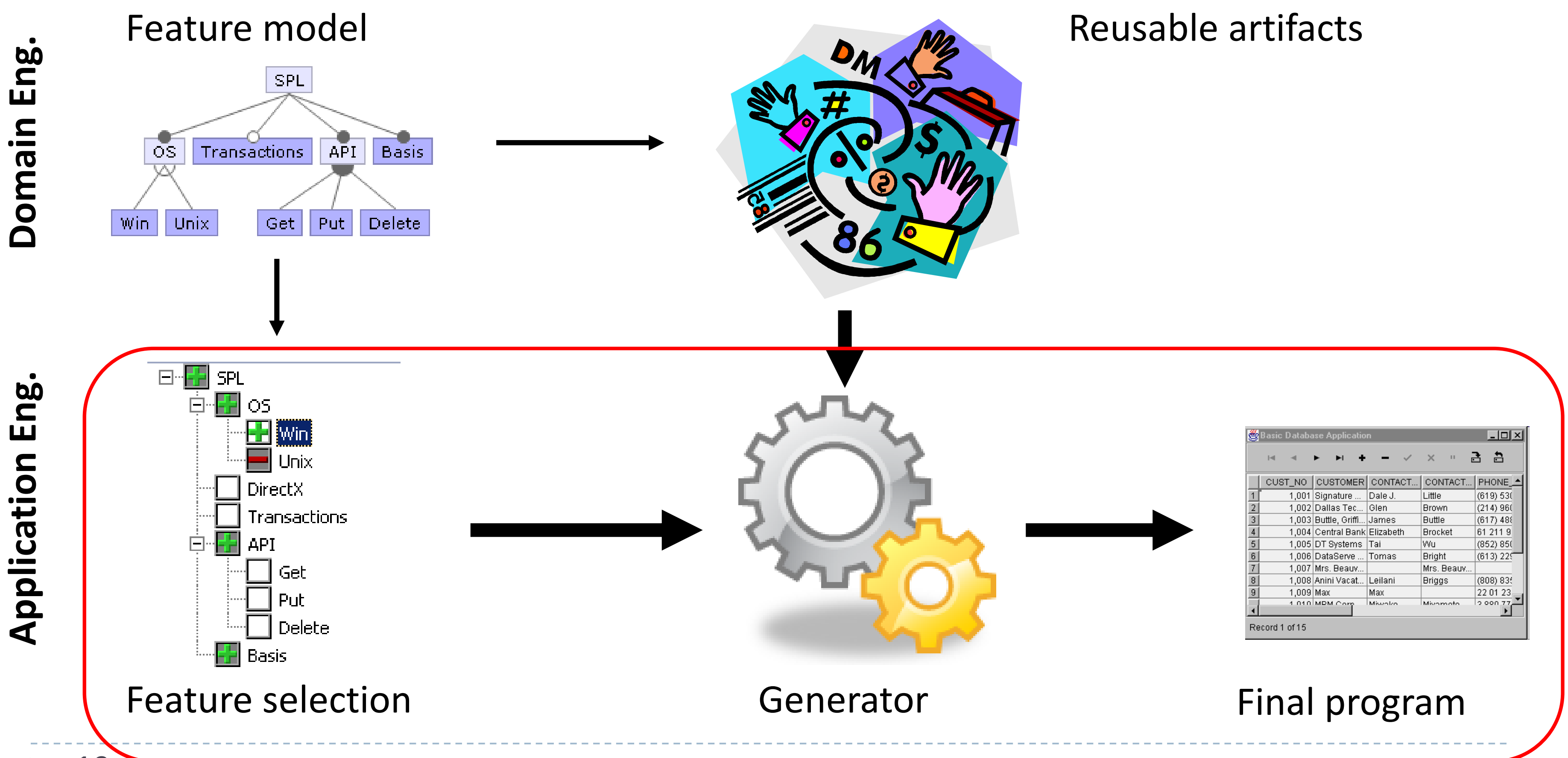
Worst



Best configuration is 480 times better than **Worst** configuration

Only by tweaking 2 options out of 200 in Apache Storm - observed **~100%** change in latency

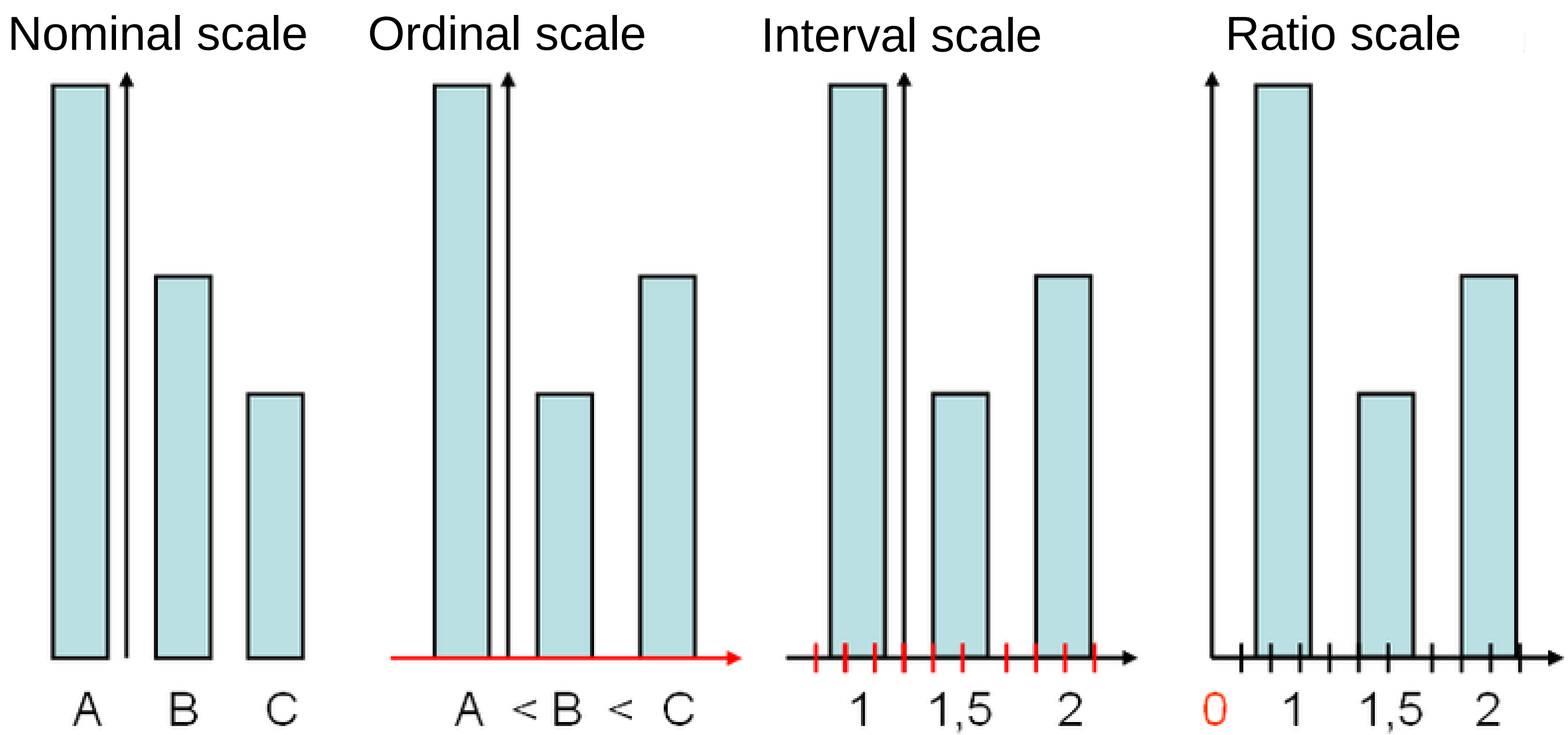
Relation


Domain Eng.
Feature model
SPL
OS
Transactions
API
Basis
Win
Unix
Get
Put
Delete
Reusable artifacts
Application Eng.
SPL
OS
Win
Unix
DirectX
Transactions
API
Get
Put
Delete
Basis
Feature selection
Generator
Final program

Measuring Non-Functional Properties

Side Note: Theory of Measurement

- Stevens defines different levels of measurement [4]



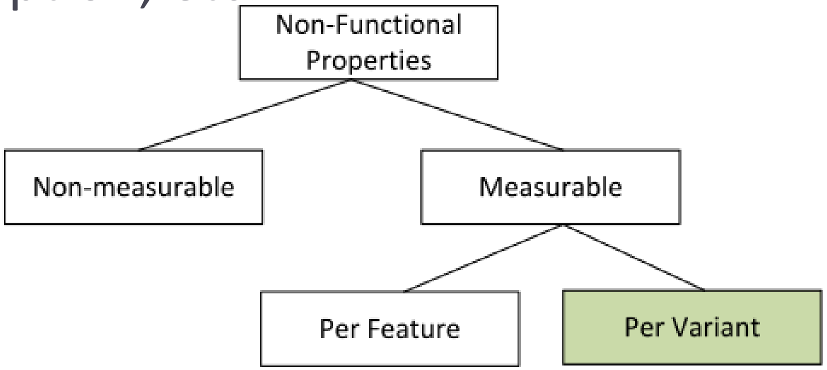
Quelle: Wikipedia

Examples:

Sex | Grades | Time (date) | Age

Classification of Non-Functional Properties for Software Product Lines

- Not measurable properties:
  - Qualitative properties
  - Properties without a sensible metric (maintainability?)
- Measurable per feature
  - Properties exist for individual features
  - Source code properties, footprint, etc.
- Measurable per variant
  - Properties exist only in final (running) variants
  - Performance, memory consumption, etc.



Methods for Measuring Product Lines

- How to measure non-functional properties of variants and whole product lines?

- Artifact-based
- Family-based
- Variant-based

Measurement: Artifact-based

- Artifact-based
  - Features are measured in isolation from other features
  - Linear effort with respect to the number of features
  - Robust against changes of the product line

- Drawbacks:
  - Not all properties are measurable (performance?)
  - Requirements specific implementation techniques (#ifdef?)
  - No black-box systems, since code is required
  - No feature interactions considered (accuracy?)
  - Requires artificial measurement environment

Effort	Accuracy	Applicability	Generality	Environment
+	-	-	-	-

Measurement: Family-based

- Family-based
  - Measurement of all features and their combinations at the same time
  - Requires feature model to derive influence of individual features on the measurement output
  - Effort: O(1) if there are no constraints
- Drawbacks:
  - Not all properties measurable; artificial measurement setting
  - Inaccurate with respect to feature interactions
  - Requires tracing information from features to code

Effort	Accuracy	Applicability	Generality	Environment
++	-	-	-	-

Measurement: Variant-based

- Variant-based
  - Measure each individual variant
  - Every property can be measured
  - Works for black-box systems
  - Independent of the implementation technique
  - Interactions between features can be measured

- Drawback:
  - Huge measurement effort $O(2^n)$

Effort	Accuracy	Applicability	Generality	Environment
--	+	+	+	+

Example: SQLite

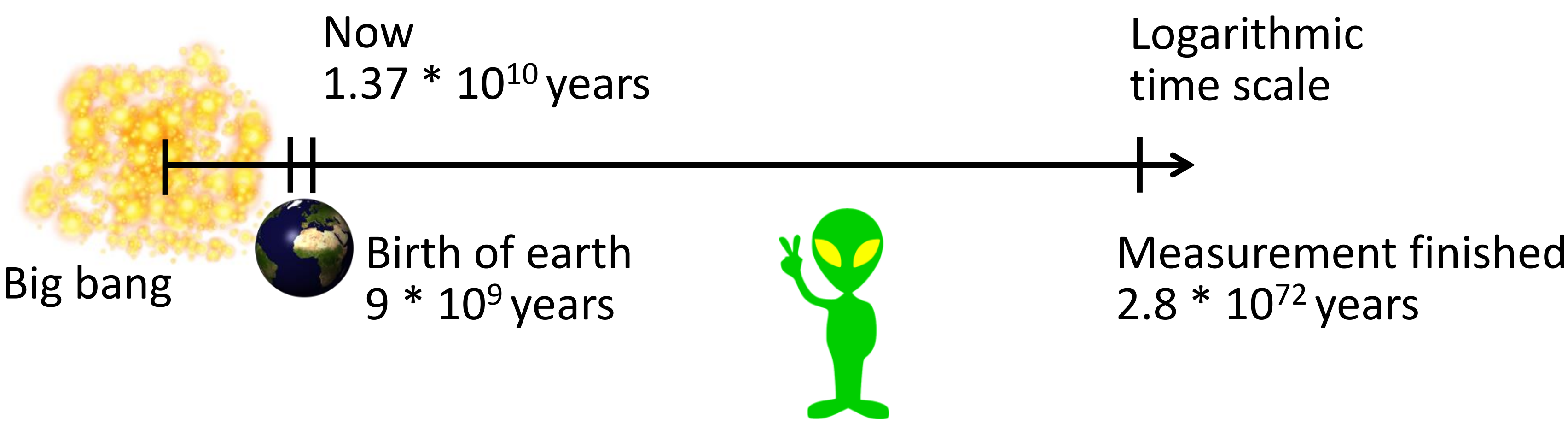
- Exclusive Locking
- Case Sensitivity
- Thread Safety
- Atomic Write
- …

Varianten:

260,532,200,783,961,400,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000

Approach 0: Brute Force

- SQL data:
  - $3*10^{77}$ varaints
  - 5 minutes per measurement (compilation + benchmark)
- $3*10^{77}$ * 5min = 2,853,881,278,538,812,785,388,127,853,881,300,000,000,000,000,000,000,000,000,000,000,000,000,000 years!

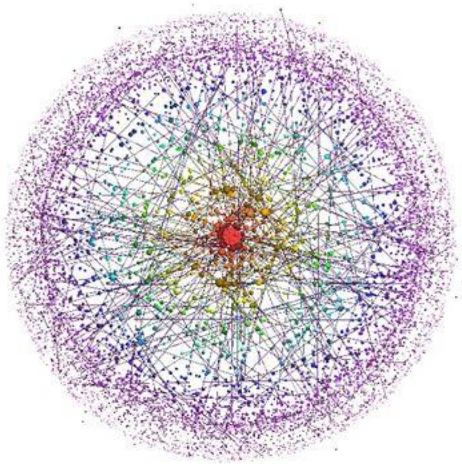


Approach 1: Sampling

- Measure only few, specific variants
  - Predict properties of unseen configurations
  - State-of-the-art approaches use machine-learning techniques for learning a prediction model

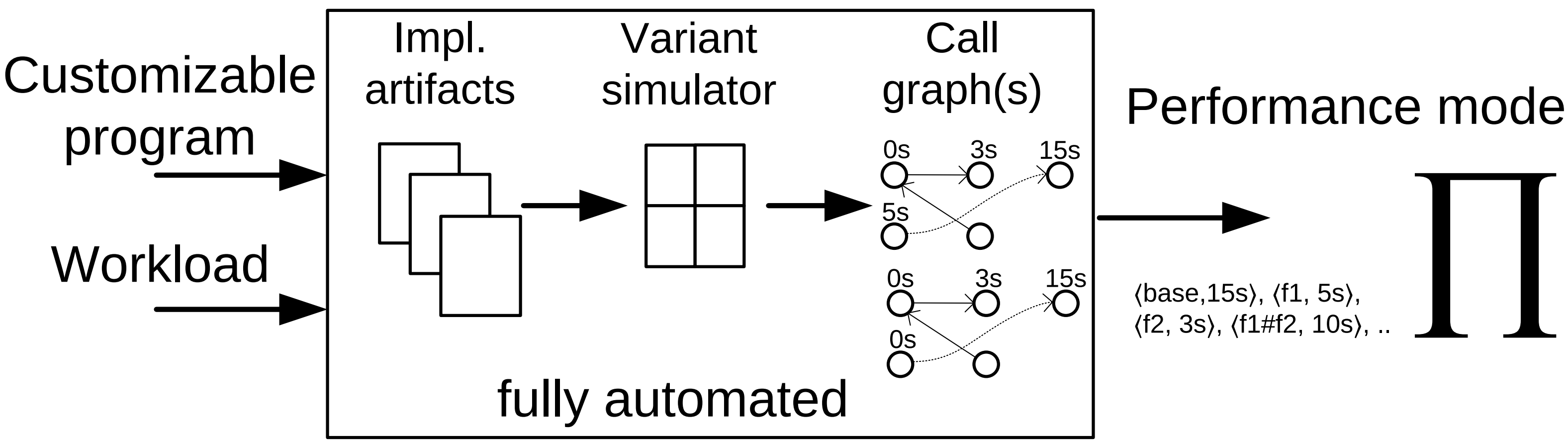
- Problem: Feature interactions
  - We need to measure many combinations of features to identify and quantify the influence of interactions
  - Order-6 interaction:

13,834,413,152 = 131,605 years!



Approach 2: Family-Based Measurement

- Create a *variant simulator*
- Execute simulator and measurement the property
- Compute the influences of each feature based on the execution of the simulator

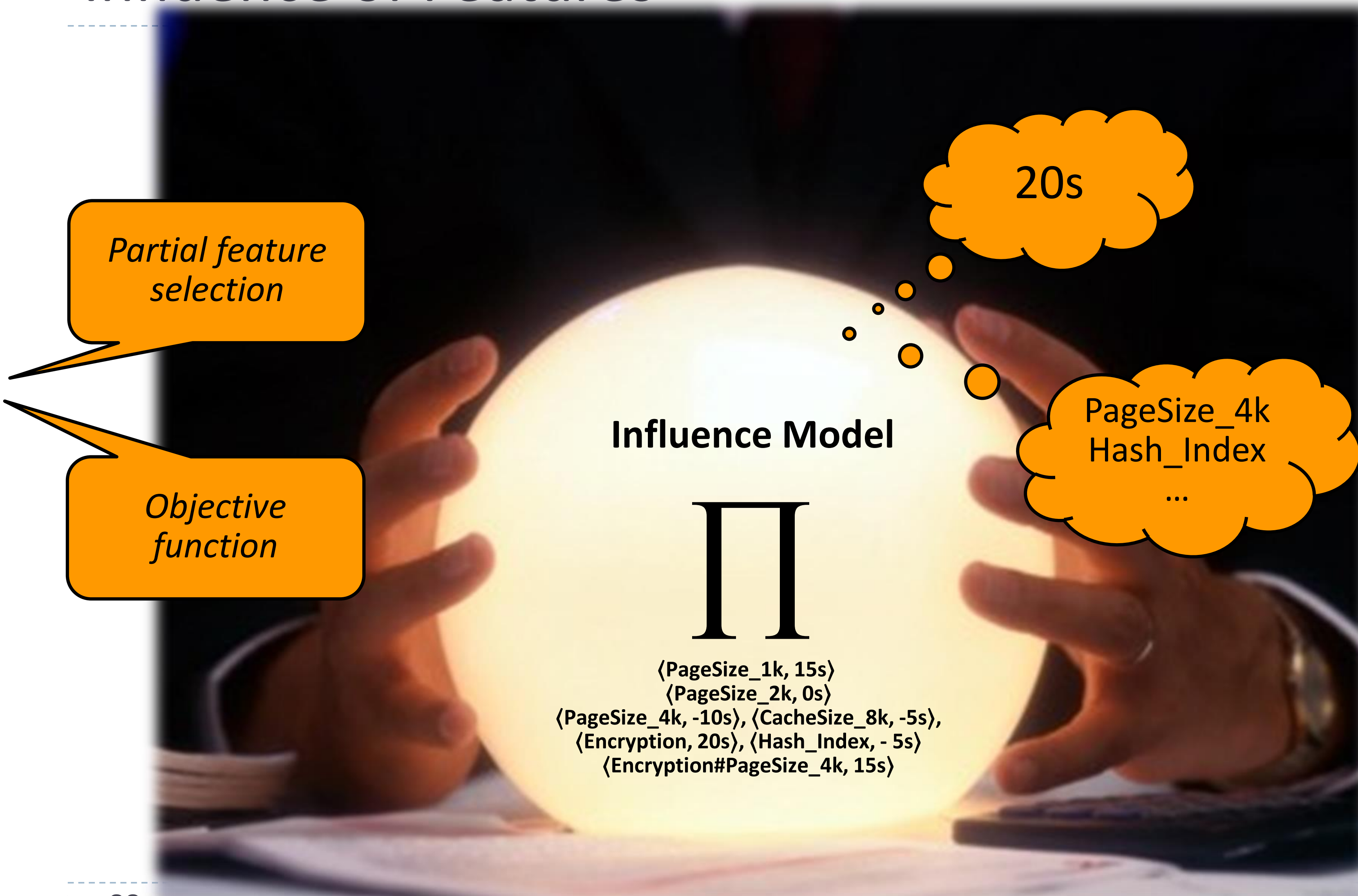


Prediction of Non-Functional Properties

Learning Techniques

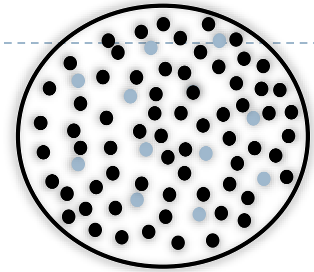
- Regression
- Neuronal networks
- CART
- Bayse Nets
- MARS
- M5
- Cubist
- Principal Component Analysis
- Evolutionary algorithms
- …

Goal: Prediction of Properties based on the Influence of Features



Overview

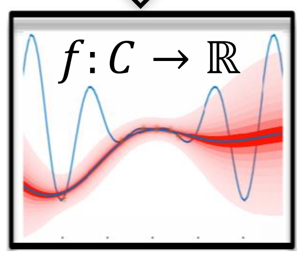
Configuration space
Size:~ $2^{\#options}$



(1) Sampling

Cohen et al. TSE'08; Siegmund et al. SPLC'11, SQJ'12, ICSE'12, FSE'15; Sarkar et al. ASE'15; Henard et al. TSE'14, ICSE'15; Oh et al. FSE'17; Johansen et al. SPLC'12; Medeiros et al. ICSE'16; Dechter et al. AAAI'02; Gogate and Dechter CP'06; Chakraborty et al. AAAI'14; …
Key domains: Combinatorial testing, artificial intelligence, search-based software engineering, design of experiments

Performance model



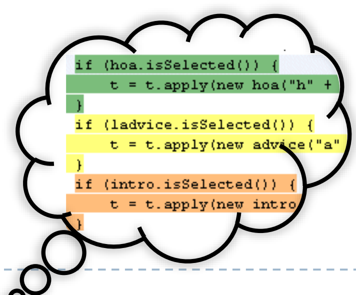
(2) Learning

Guo et al. ASE'13; Siegmund et al. ICSE'12, FSE'15; Sakar et al. ASE'15; Oh et al. FSE'17; Zhang et al. ASE'15; Nair et al. FSE'17,arXiv'17; Jamshidi et al. SEAMS'17; Xi et al. WWW'04,…
Key domains: machine learning, statistics

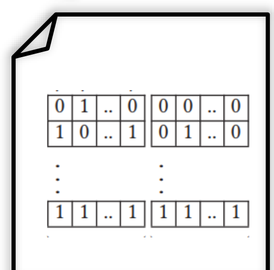
Goal

(4) Analysis

Not covered here



System understanding



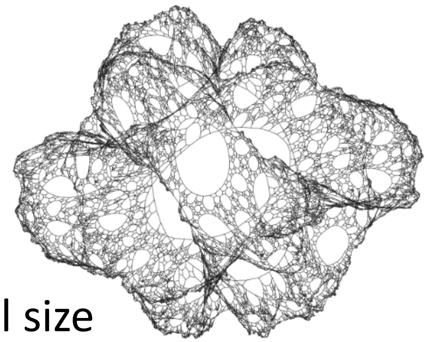
Optimal configuration(s)

(3) Optimization

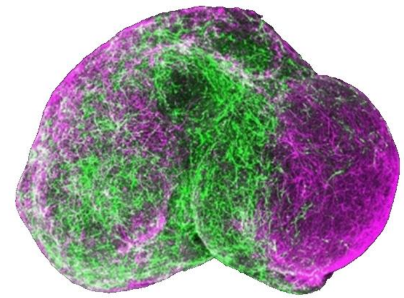
Sayyad et al. ICSE'13, ASE'13; Henard et al. ICSE'15; White et al. JSS'09; Guo et al. JSS'12; Kai Shi ICSME'17; Olaechea et al. SPLC'14; Hierons et al. TOSEM'16; Tan et al. ISSTA'15; Siegmund et al. SQJ'12; Benavides et al. CAiSE'05; Zheng et al. OSR'07; Jamshidi et al. MASCOTS'16; Osogami und Kato SIGMETRICS'07; Filieri et al. FSE'15
Key domains: search-based software engineering, meta-heuristics, machine learning, artificial intelligence, mathematical optimization

Sampling – Overview

Challenges:

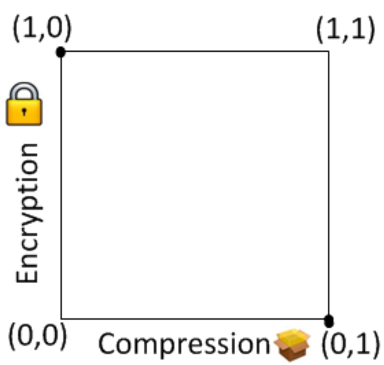


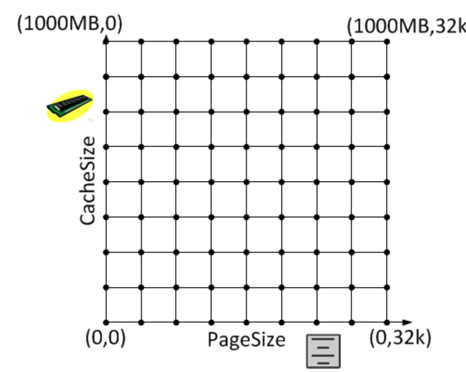
Exponential size configuration space



Find only relevant configurations for measurement

Binary configuration options





Numeric configuration options

Random Sampling

Or how to obtain randomness in the presence of **constraints**?

Trivial approach: Enumerate all configurations and randomly draw one

Not scalable

Easy to implement

True randomness

[12] Temple et al. TR'17; [13] Guo et al. ASE'13; [14] Nair et al. FSE'15; [15] Zhang et al. ASE'15;

SAT approach: Manipulate a SAT/CSP solver:

[5] Henard et al. ICSE'15: Randomly permute constraint and literal order and phase selection (order true - false)

[17] Siegmund et al. FSE'17: Specify distribution of config. as constraints

No guaranteed uniformity

Limited scalability

Easy to implement

Better distribution

BDD approach: Create a counting BDD to enumerate all configurations: [6] Oh et al. FSE'17

BDD creation can be expensive

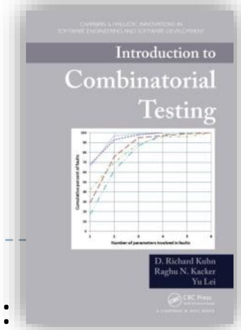
Scales up to 2,000 options

True randomness

Beyond SE: Tailored algorithms: [7] Chakraborty et al. AAAI'14: Hash the configuration space

[8] Gogate and Dechter CP'06 and [9] Dechter et al. AAAI'02: Consider CSP output as probability distribution

Sampling with Coverage I



Survey: [10] Medeiros et al. ICSE'16

Interaction coverage: t-wise, (e.g., 2-wise = pair-wise)

Kuhn et al.:

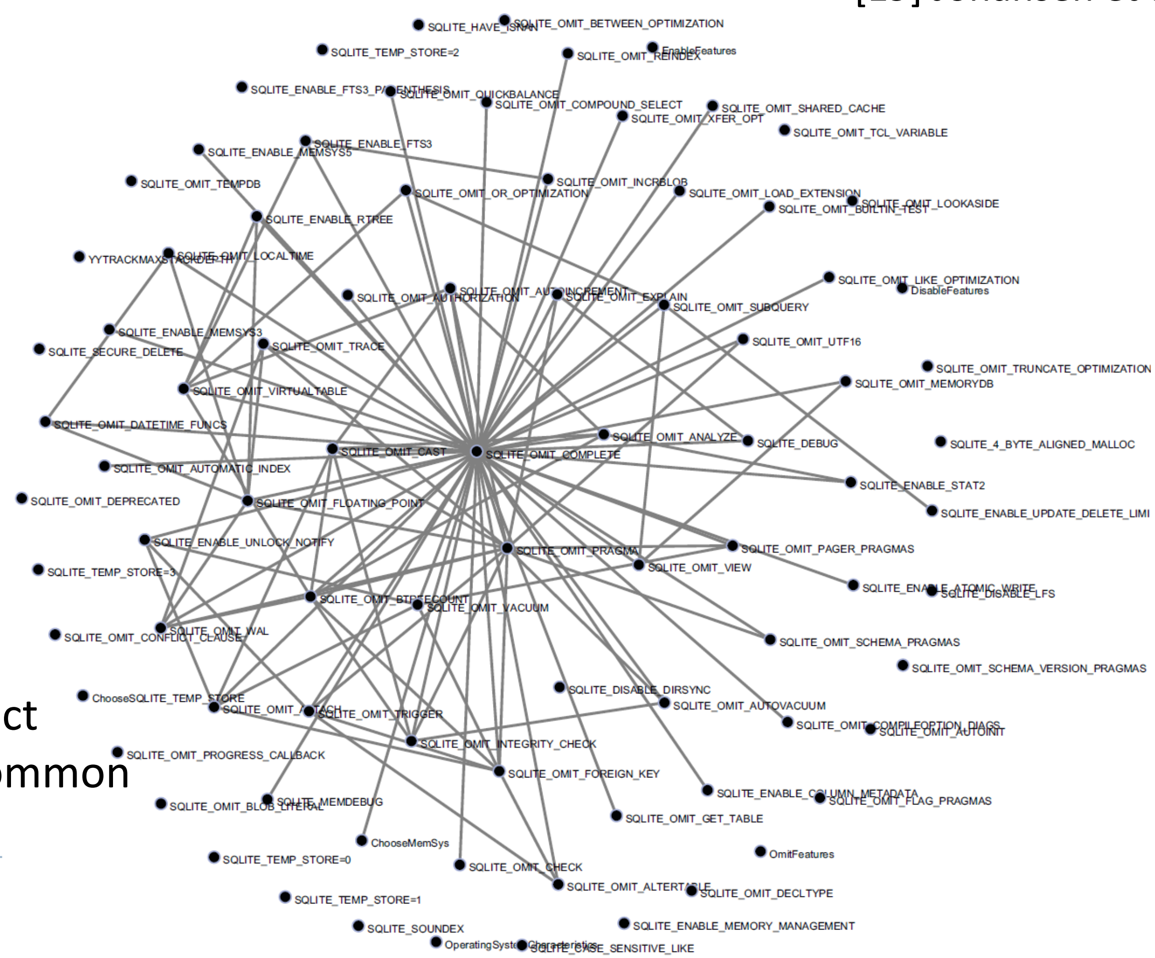
[11] Henard et al. TSE'14

[18] Cohen et al. TSE'08

[19] Johansen et al. SPLC'12

[20] Siegmund et al. SPLC'11

[21] Siegmund et al. ICSE'12



Insights:

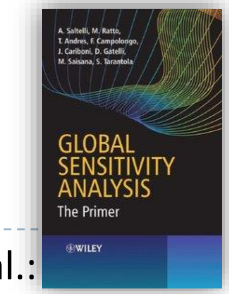
Many options do not interact

2-wise interactions most common

Hot-spot options

Sampling with Coverage II

Saltellie et al.:



Option coverage: Cover all options either by minimizing or maximizing interactions

```
#ifdef A
  // code 1
#endif

#ifdef B
  // code 2
#else
  // code 3
#endif

#ifdef C
  // code 4
#endif
```

pair-wise			
config-1:	!A	!B	C
config-2:	!A	B	!C
config-3:	A	!B	!C
config-4:	A	B	C

one-enabled			
config-1:	A	!B	!C
config-2:	!A	B	!C
config-3:	!A	!B	C

one-disabled			
config-1:	!A	B	C
config-2:	A	!B	C
config-3:	A	B	!C

most-enabled-disabled			
config-1:	A	B	C
config-2:	!A	!B	!C

statement-coverage			
config-1:	A	B	C
config-2:	A	!B	C

Leave-one-out /one disabled sampling: [10] Medeiros et al. ICSE'16

Option-wise sampling: [20,24] Siegmund et al. SPLC'11, IST'13

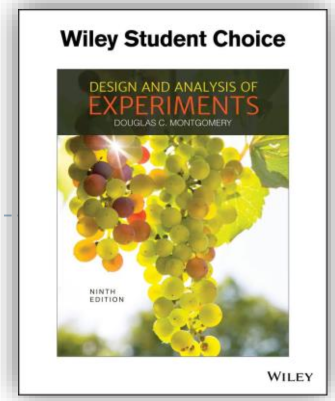
Negative option-wise sampling: [22] Siegmund et al. FSE'15

Option-frequency sampling: [23] Sakar et al. ASE'15

	x_1	x_2	x_3	..	x_i	..	..	x_N
1	0	1	1	0	0	0	1	1
2	0	0	1	1	1	0	0	0
3	1	1	0	1	0	1	1	1
4	0	1	0	1	0	1	0	0
5	1	1	0	0	0	1	0	1
6	0	0	0	1	1	1	0	0
7	1	1	0	1	0	0	0	1
8	1	0	0	0	1	0	0	1

	x_1	x_2	x_3	..	x_i	..	..	x_N
selected	4	5	2	..	3	..	..	5
deselected	4	3	6	..	5	..	..	3

Sampling Numeric Options



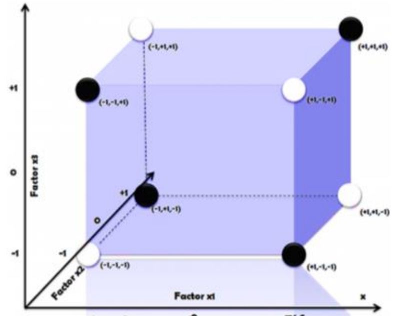
Identification of Main Effects with all other interactions negligible

PLACKETTE BURMAN

	X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8	X_9	X_{10}	X_{11}
1	+1	+1	+1	+1	+1	+1	+1	+1	+1	+1	+1
2	-1	+1	-1	+1	+1	+1	-1	-1	-1	+1	-1
3	-1	-1	+1	-1	+1	+1	+1	-1	-1	-1	+1
4	+1	-1	-1	+1	-1	+1	+1	+1	-1	-1	-1
5	-1	+1	-1	-1	+1	-1	+1	+1	+1	-1	-1
6	-1	-1	+1	-1	-1	+1	-1	+1	+1	+1	-1
7	-1	-1	-1	+1	-1	-1	+1	-1	+1	+1	+1
8	+1	-1	-1	-1	+1	-1	-1	+1	-1	+1	+1
9	+1	+1	-1	-1	-1	+1	-1	-1	+1	-1	+1
10	+1	+1	+1	-1	-1	-1	+1	-1	-1	+1	-1
11	-1	+1	+1	+1	-1	-1	-1	+1	-1	-1	+1
12	+1	-1	+1	+1	+1	-1	-1	-1	+1	-1	-1

Identification of Main Effects confounded with 2 way interactions when resources are limited

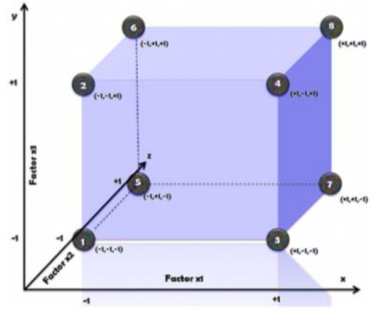
FRACTIONAL FACTORIAL



- ***Number of Runs***: L^{F-f}

Identification of Main Effects **WITH** 2 way interactions

FULL FACTORIAL

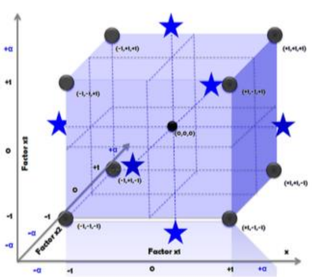


- ***Number of Runs***: L^{F} .

- The goal is **OPTIMIZATION of critical factors**
- **Emphasis is on the fitted surface representing true behavior**
- Detect non-linear **significant curvature in response surface** to investigate **full quadratic relationship**

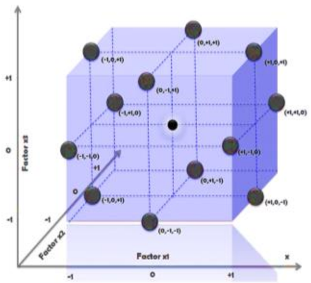
Response Surface

CENTRAL COMPOSITE



- ***Levels:*** **5** ['- α', '-1', '0' and '+1', '+ α']
- ***No of Runs:*** $2^{f-p} + 2SP + CP$

BOX BEHNKEN



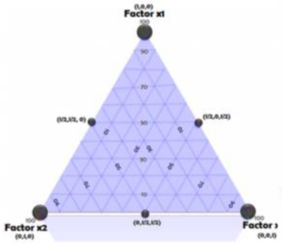
- ***Levels:*** **3** levels per factor
- ***Design Points:*** at the "**mid**" points of **edges** of the process space & center

- **Factors** are **COMPONENTS of a MIXTURE** &
- Components must **TOTAL TO A CONSTANT** i.e. **1 (100%)**.
- **Response** is a **function of proportion of mixture components.**

Mixture

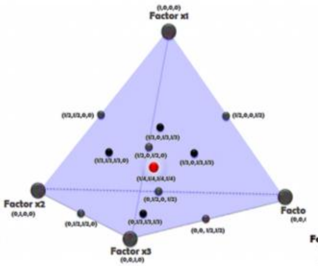
All **components** have the **same range**

SIMPLEX LATTICE



Number of design points *in the {q, m } simplex-lattice* ***(q+m-1)! /(m!(q-1)!).***

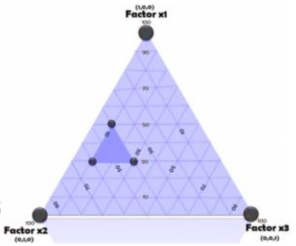
SIMPLEX CENTROID



In the q-component, the ***number of distinct points is $2^q - 1$.***

Upper and/or lower bound constraints.

D- OPTIMAL CONSTRAINED



Number of Design points

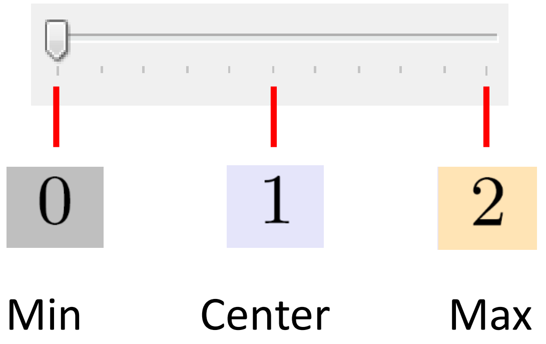
$x_i = L_i + (1 - L)\, x_i^*$

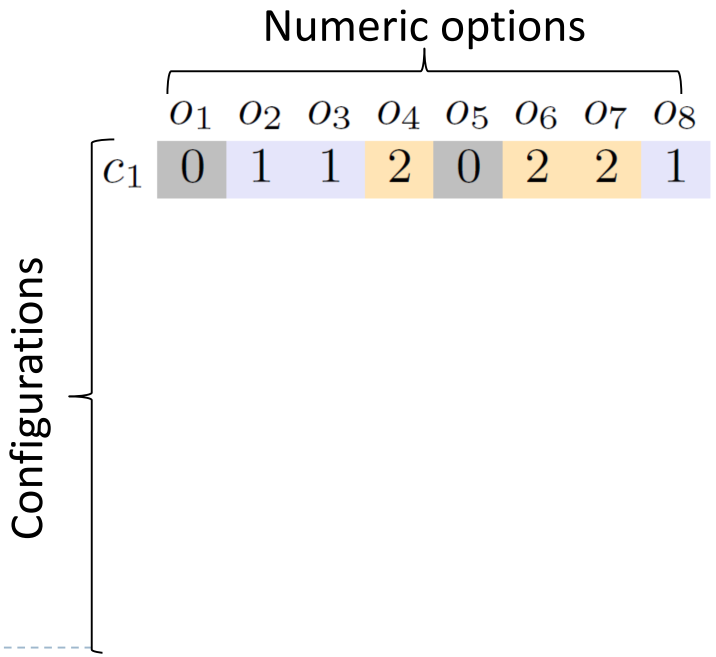
L = sum of all lower bounds.

Plackett-Burman Design (PBD)

- Minimizes the variance of the estimates of the independent variables (numeric options)
- ...while using a limited number of measurements
- Design specifies *seeds* depending on the number of experiments to be conducted (i.e., configurations to be measured)

Value range of a numeric option

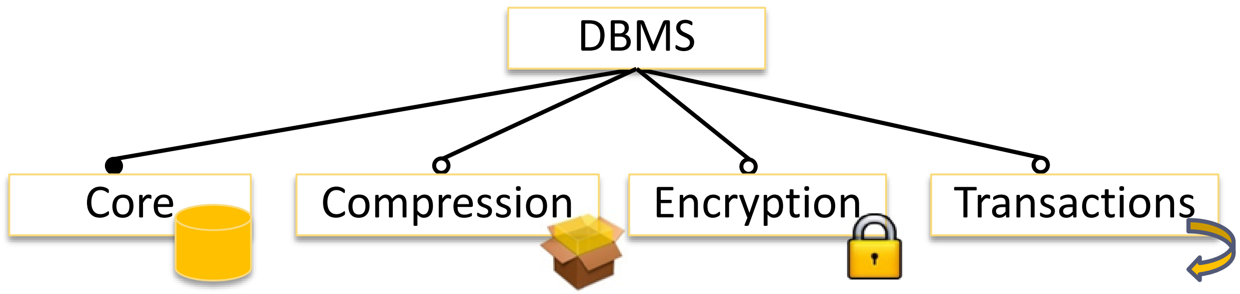


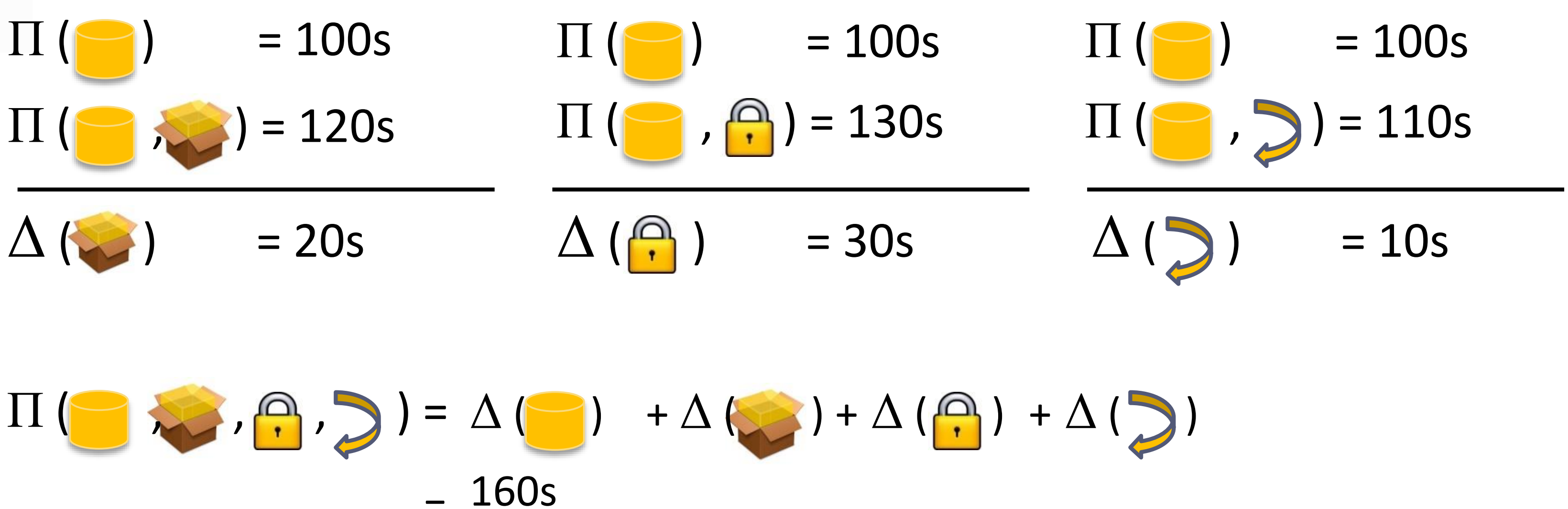


In Detail: Feature-wise Sampling

Determine the Influence of Individual Features

- How shall we approach?





Experience with Feature-wise Sampling

Footprint

- Material:

Product Line	Domain	Origin	Language	Features	Variants	LOC
Prevayler	Database	Industrial	Java	5	24	4 030
ZipMe	Compression	Academic	Java	8	104	4 874
PKJab	Messenger	Academic	Java	11	72	5 016
SensorNet	Simulation	Academic	C++	26	3240	7 303
Violet	UML editor	Academic	Java	100	ca. 10^{20}	19 379
Berkeley DB	Database	Industrial	C	8	256	209 682
SQLite	Database	Industrial	C	85	ca. 10^{23}	305 191
Linux kernel	Operating system	Industrial	C	25	ca. $3 * 10^7$	13 005 842

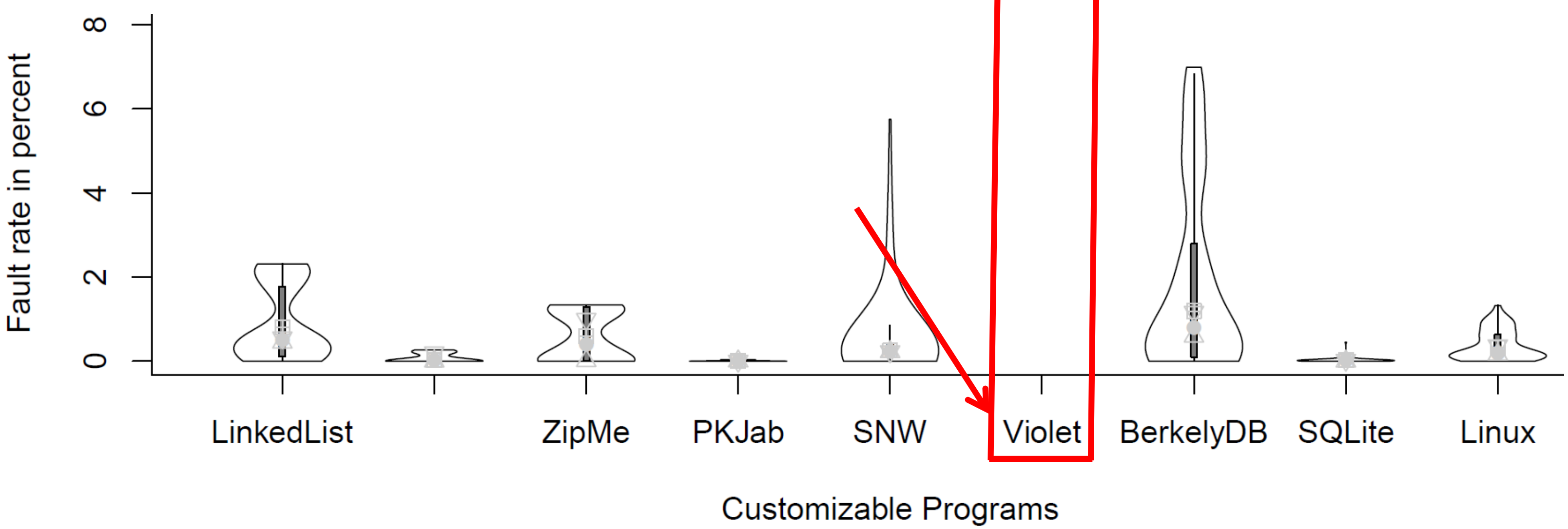
Results: Footprint

- Average error rate of 5.5% without Violet
- With Violet: 21.3%

Why this error?

measurements

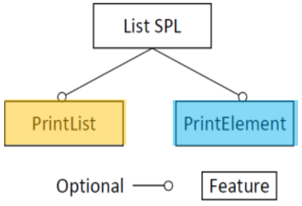
SQLite: 85 vs. 2^{88}
Linux : 25 vs. $3*10^7$



Analysis: Feature Interactions

- Two features interaction if their combined presence in a program leads to an unexpected program behavior

```
class List {
  int numberOfElements;
  Element* head;
#ifdef PrintList
  void printList() {
    cout << numberOfElements;
#ifdef PrintElement
    if(this->head != NULL)
      printElement(this->head);
#endif
  }
#endif
#ifdef PrintElement
  void printElement(Element* node){
    node->print();
    if(node->hasNext())
      this->printElement(node->getNext());}
#endif
};
```



Expected  Measured

Π (, ,) = Δ( ) + Δ ( ) + Δ ( )
= 100s + 20s + 30s
= 150s

 = 140s*

Feature Interaction: # since encrypted data has been previously compressed

Δ(#) = -10s //delta between predicted and measured performance

Experience with Pair-wise Sampling

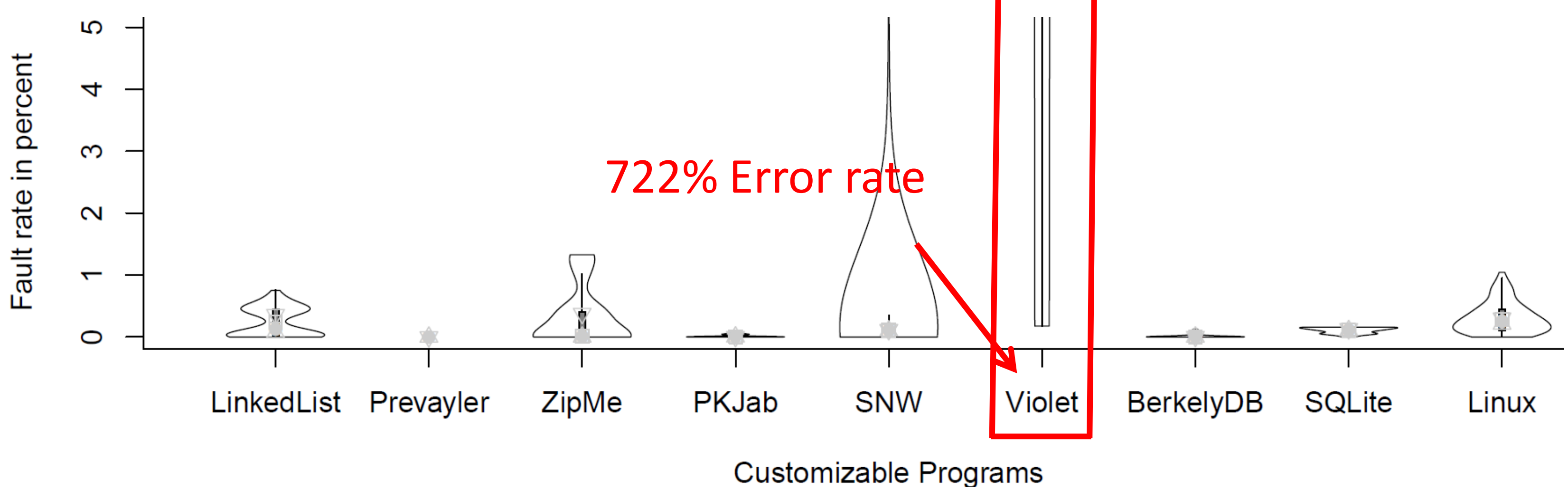
Pair-wise Measurement: Footprint

- Average error rate of 0.2% without Violet
- Reduction of 4.3 %

Partially improved, but still very bad

measurements:

SQLite: 3306 vs. 2^{85}
Linux : 326 vs. $3*10^7$



White-Box Interaction Detection: Footprint

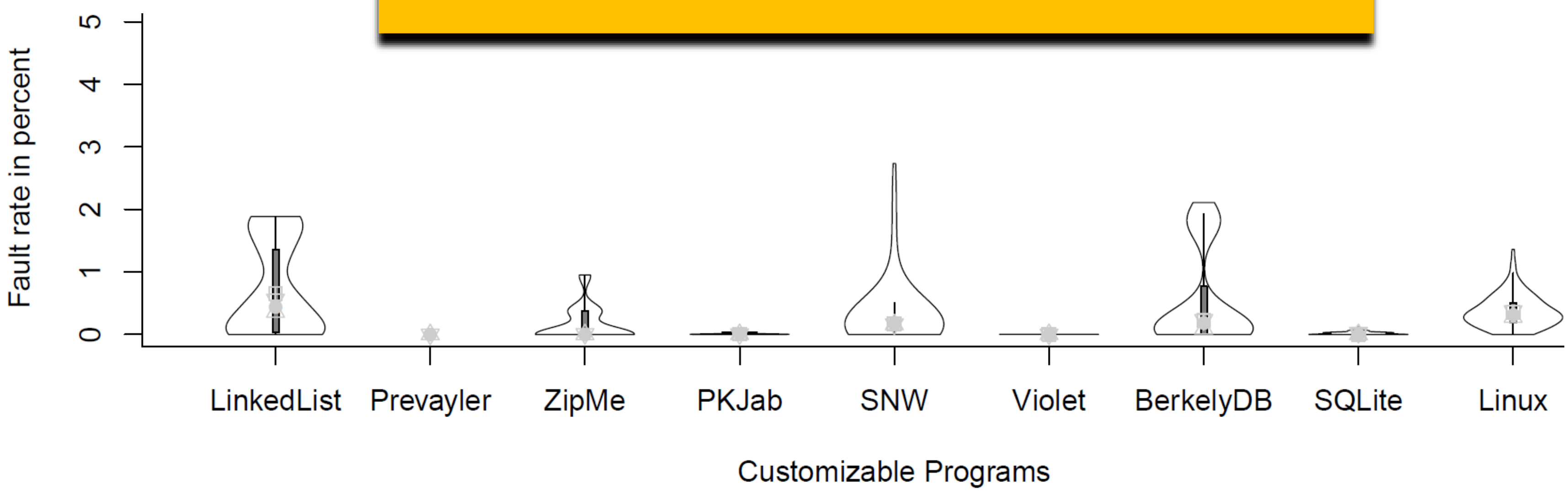
- Source code analysis revealed higher order feature interactions in Violet; these had been explicitly measured

measurements:

SQLite: 146 vs. 2^{85}

Linux : 207 vs. $3*10^7$

Average error rate of 0.2% with Violet

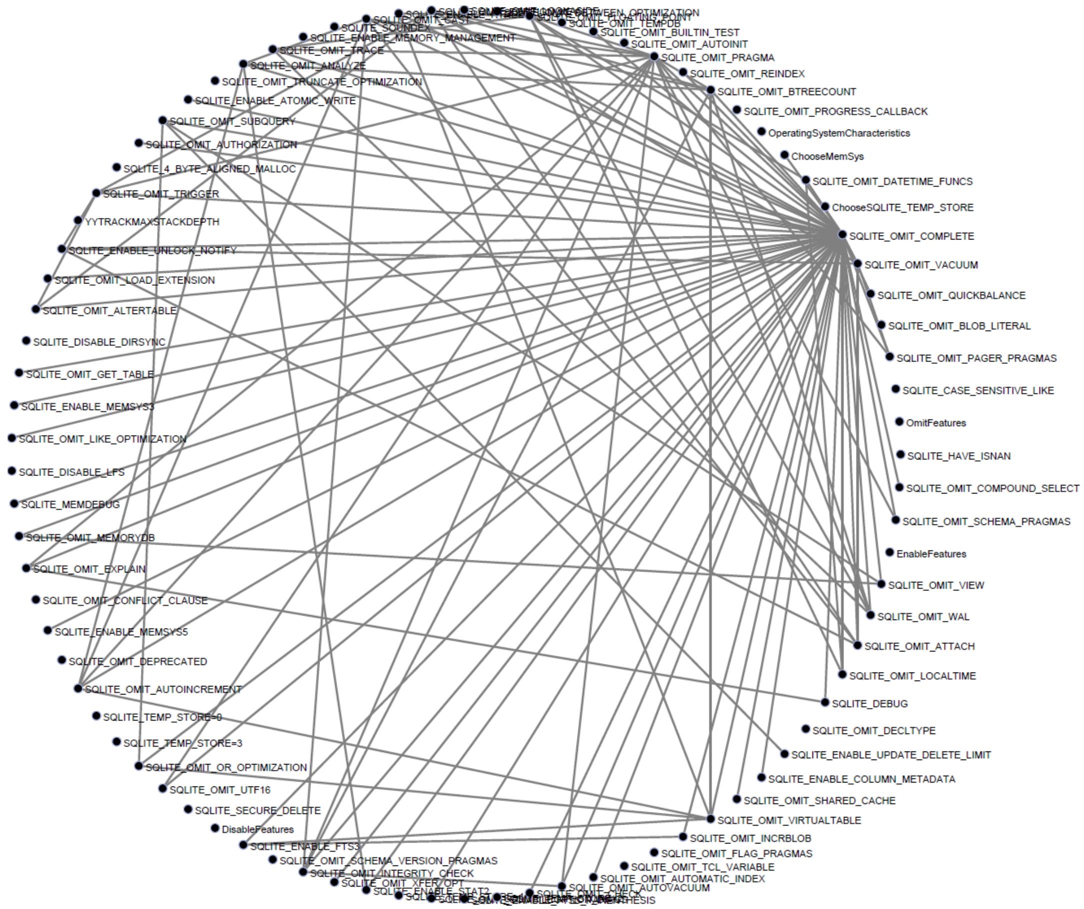


Analysis of the Results

- When learning a model, we need to consider interactions and so does the sampling approach
- In case of pair-wise sampling (2-wise)
  - High effort: O(n^2) with n features
  - Still inaccurate in presence of higher-order interactions

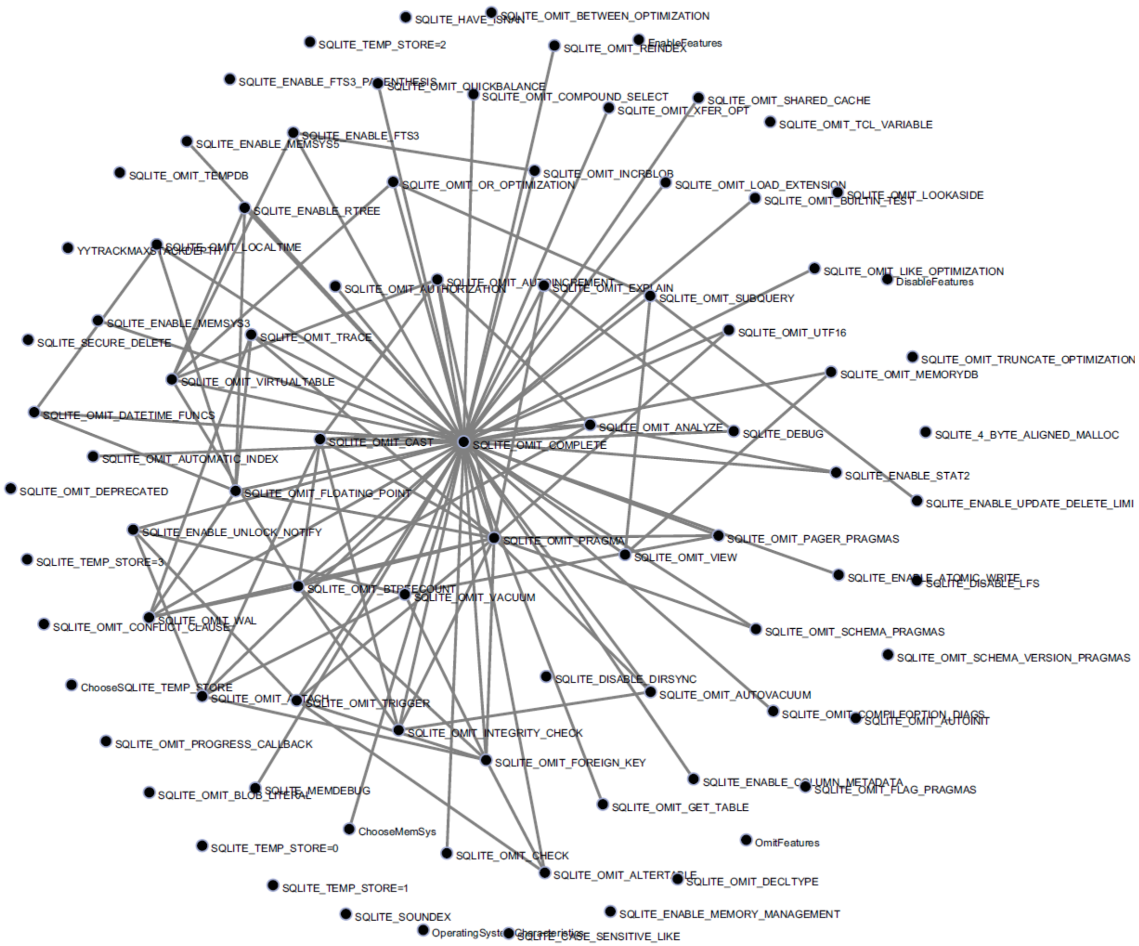
- Follow-up research questions:
  - How do interactions distribute among features?
  - Do all features interact or only few?
  - What order of interactions is most frequent?
  - Are there patterns of interactions?

Distribution of Interactions?



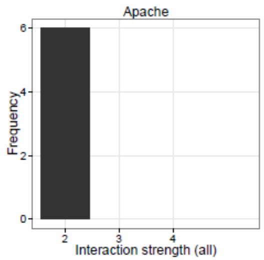
Insight 1: Few features interact with many *(hot-spots) and many features interact with few.*

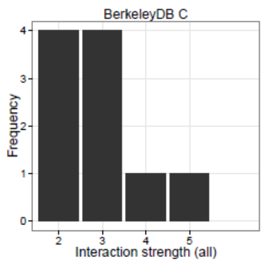
Do all Features Interact or only few?

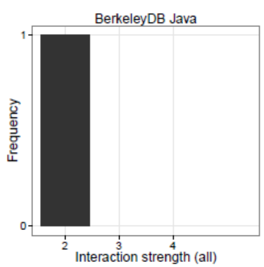


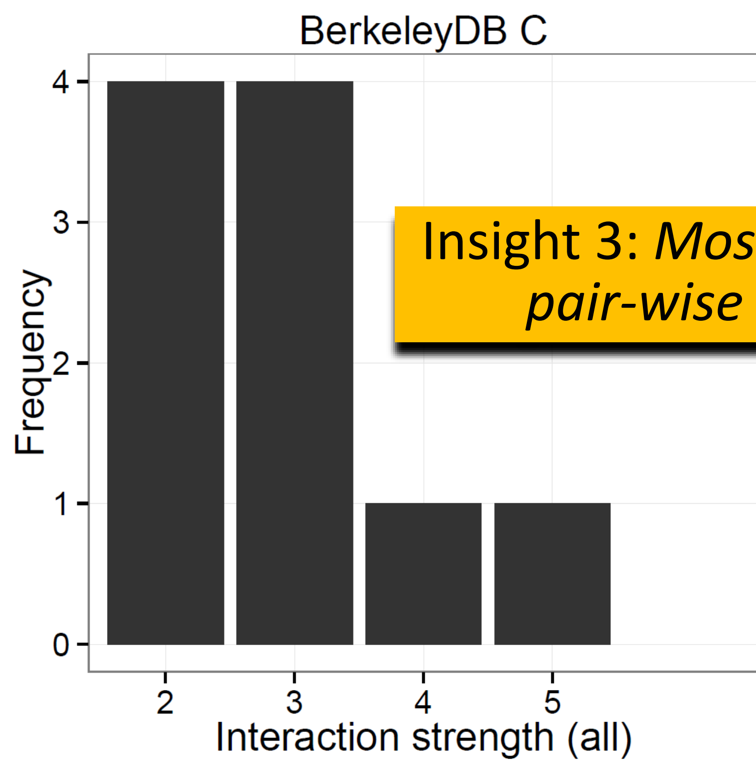
Insight 2: *Many features do not interact!*

How Many Interactions at which Degree?

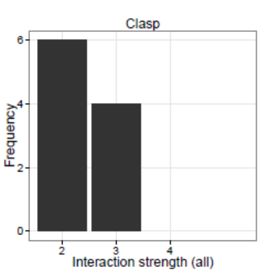


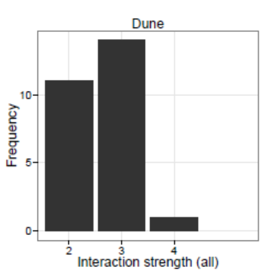


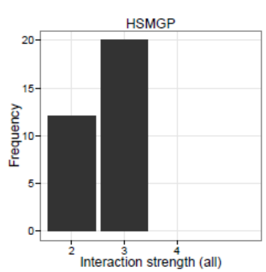


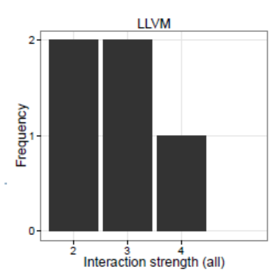


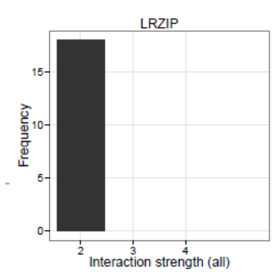
Insight 3: *Most interactions are pair-wise interactions!*

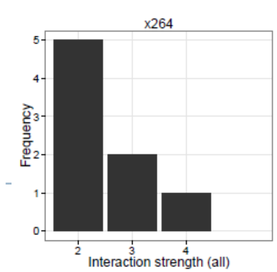




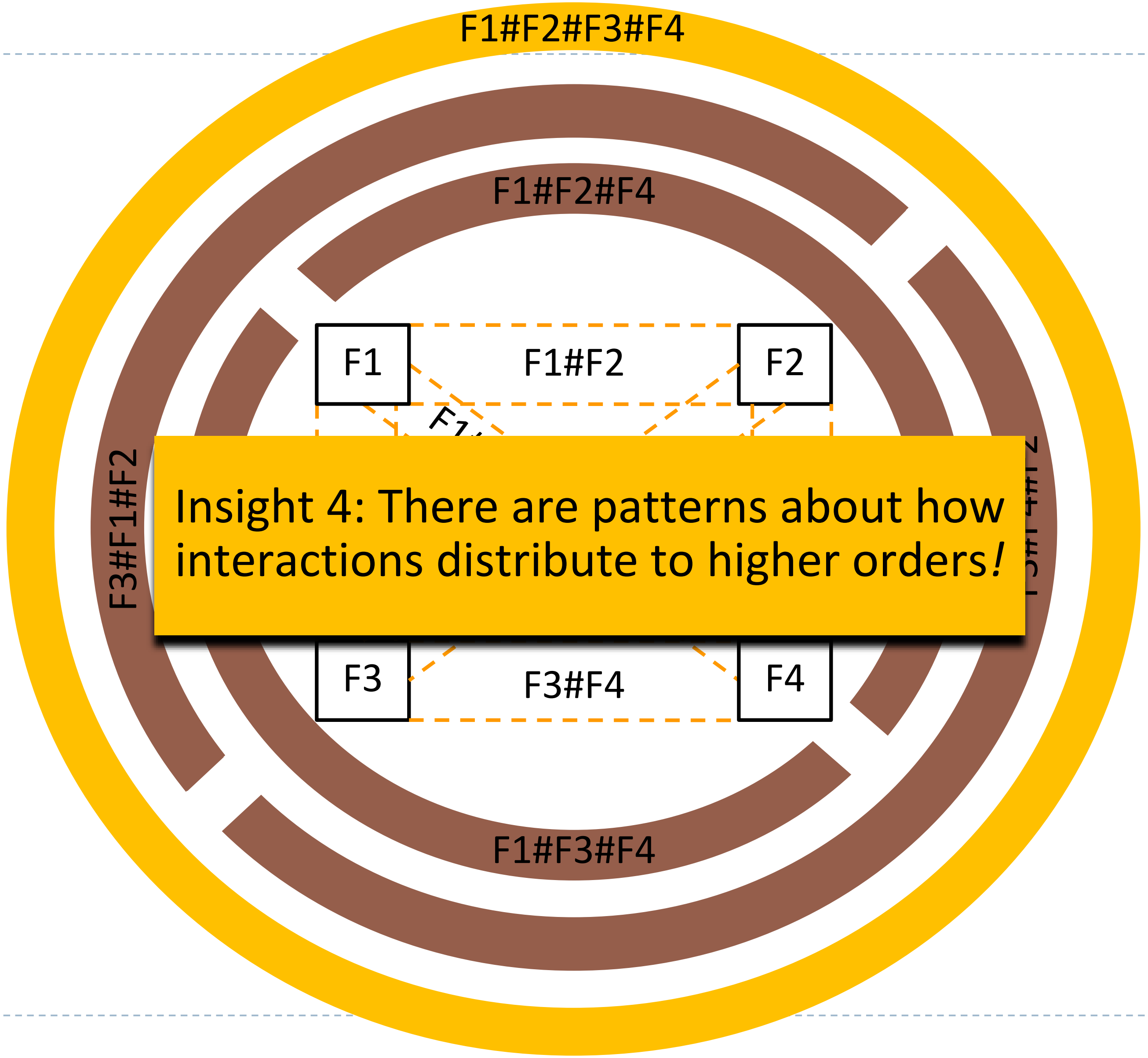








Pattern of Feature Interactions?



How about Designing our own Learning Approach?

Can we automatically find feature interactions

… without domain knowledge

… for black-box systems

…independent of the programming language, configuration technique, and domain

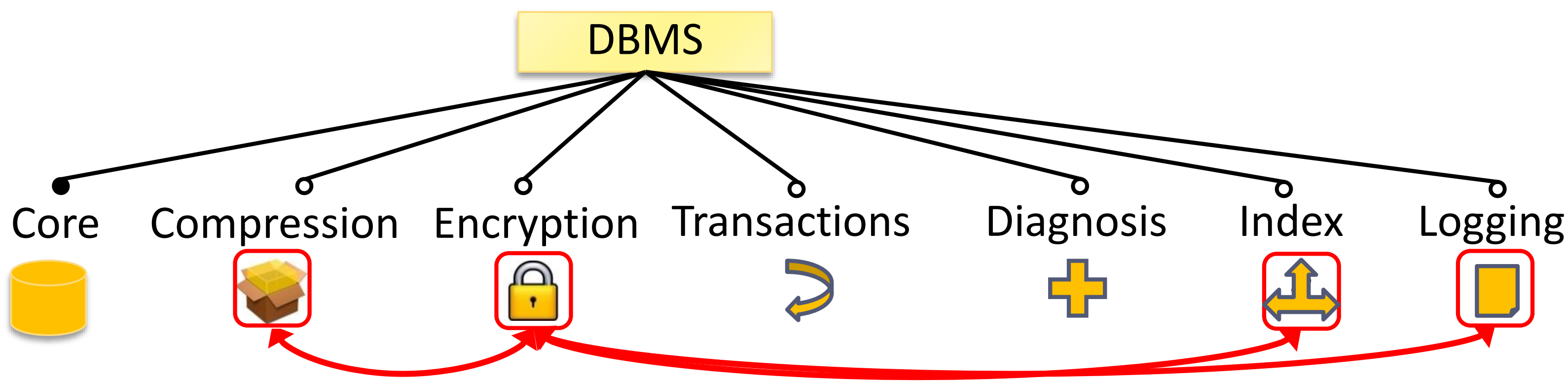
…, to improve our prediction accuracy?

What do we have?

- Insights:
  - Not all features interact
  - Most interactions are pair-wise interactions or of low order
  - Many features interact only with few and few only with many
  - There are patterns about how interactions distribute among higher orders

Idead: Incremental Approach (Insight 2)

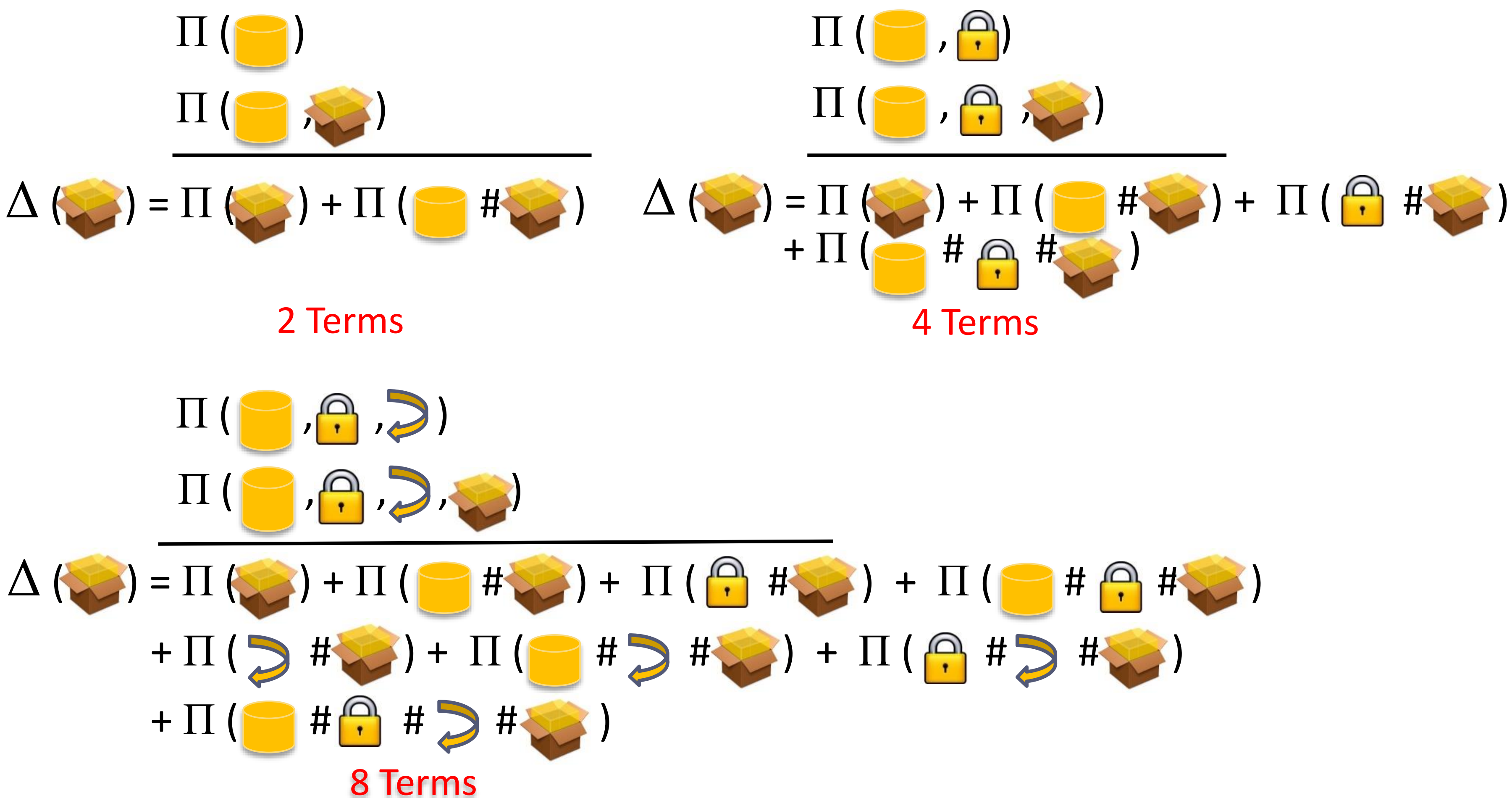
- **Step 1.** Find interacting features
  - Reduce the combinations for which we search for interactions
  - Requires only n+1 additional measurements



- **Step 2.** Find combinations of interacting features that actually cause a feature interaction
  - Using the other insights

Step 1. Find Interacting Features

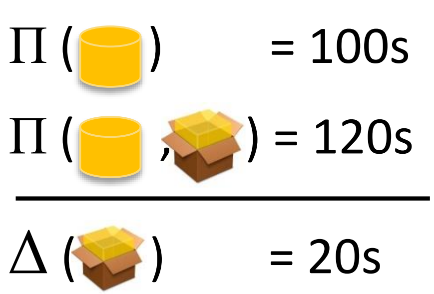
- What is exactly a delta between two measurements?

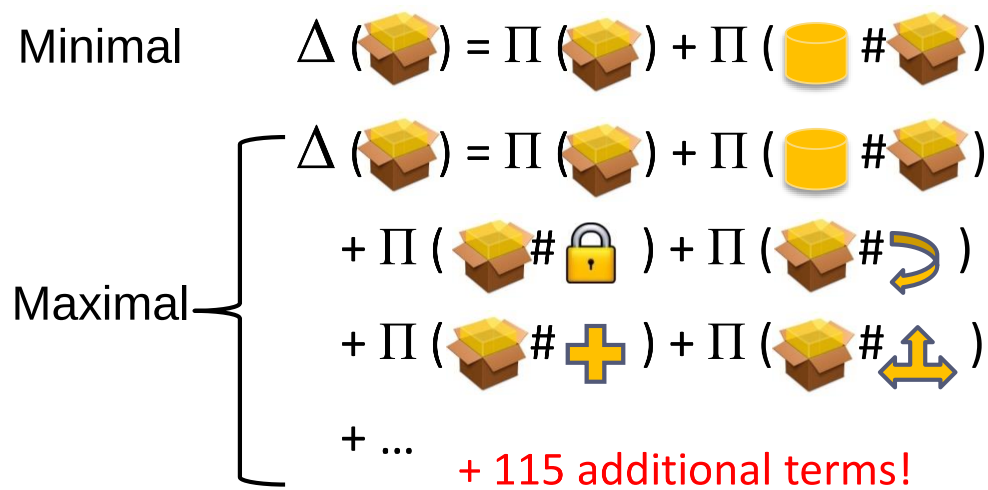


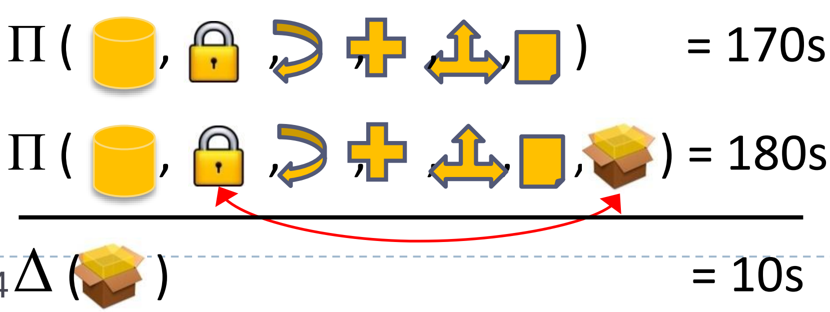
Step 1. Find Interacting Features

- Idea: Compare delta that are most likely to diverge
  - Minimal variant
  - Maximal variant

If minimal Δ ≠ maximal Δ then interacting feature

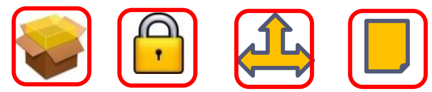






Step 2. Find Actual Feature Interactions

- Which combinations of interacting features to test?



- Approach:
  - Measure additional configurations to find interactions
  - Use heuristics based on our insights to determine those additional configurations

Step 2. Pair-wise (PW) and Higher-Order Interactions (HO)

- Heuristic 1: Measure pair-wise combinations first
  - Based on insight 3
- Heuristic 2: If two of the following pair-wise combinations {a#b, b#c, a#c} interact, measure the three-wise interaction {a#b#c}
  - Based on insight 4 (pattern of interactions)
- Heuristic 3: Measure higher-order interactions for identified hot-spot features
  - Based on insight 1

Our Own Approach: Apply Insights for Learning an Accurate Influence Model

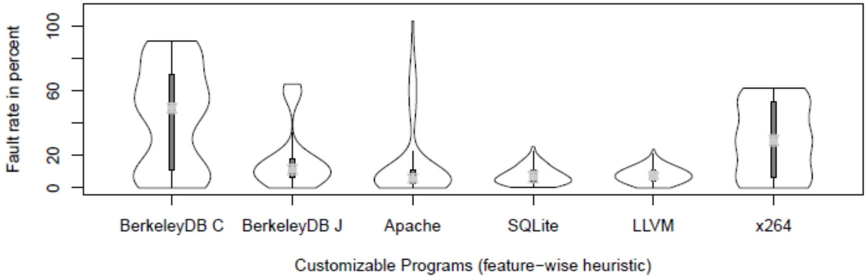
Evaluation

Product Line	Domain	Origin	Languag e	Techn.	Features	Varaints	LOC
Berkeley DB	Database	Industrial	C	C	18	2560	219,811
Berkeley DB	Database	Industrial	Java	C	32	400	42,596
Apache	Web Server	Industrial	C	CF	9	192	230,277
SQLite	Database	Industrial	C	C	39	3,932,160	312,625
LLVM	Compiler	Industrial	C++	CLP	11	1024	47,549
x264	Video Encoder	Industrial	C	CLP	16	1152	45,743

- Setup:
  - Execute standard benchmark
  - Apply heuristics consequtively

Results

Feature-Wise

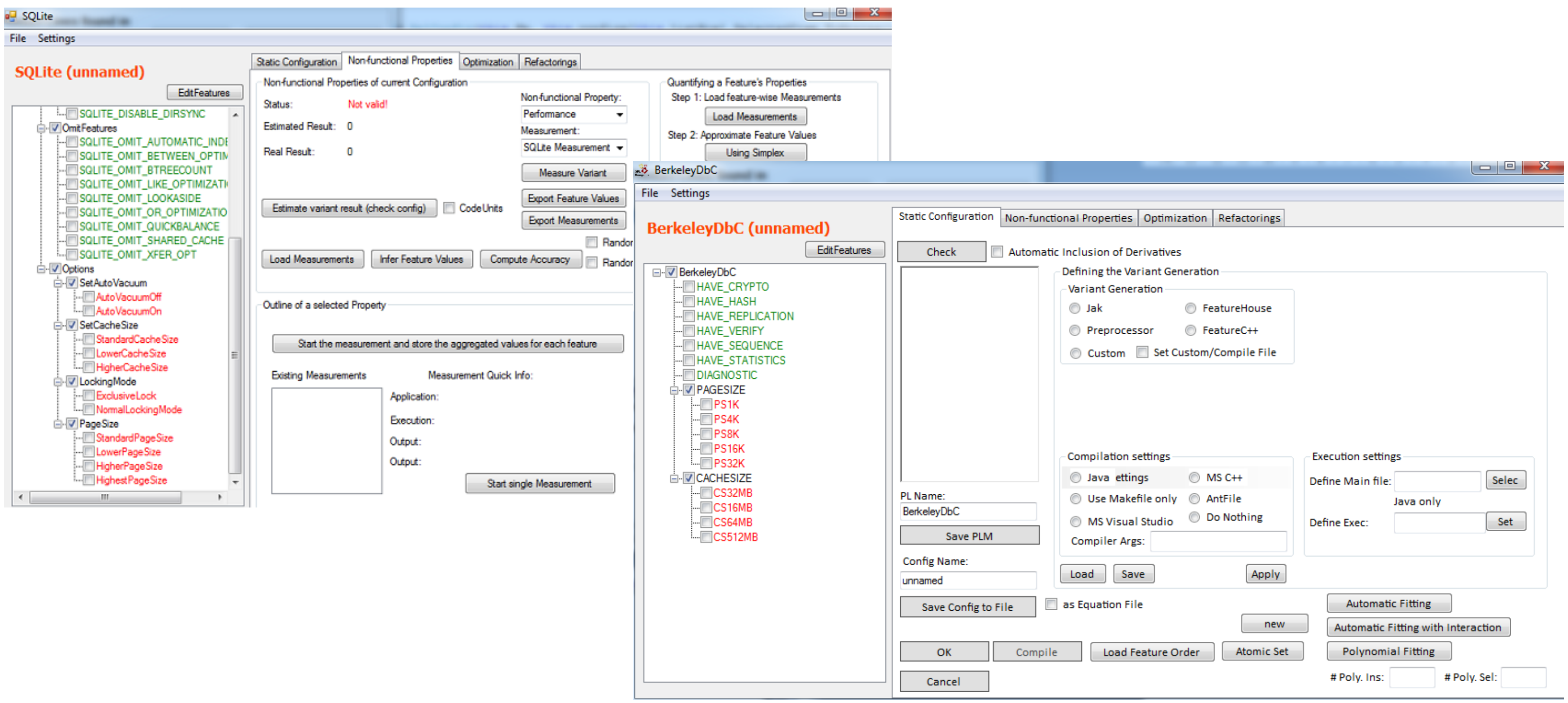


Error rates	
Mean	Median
20.3 %	18.46 %

Average error rate of 4.6% is below measurement uncertainty!

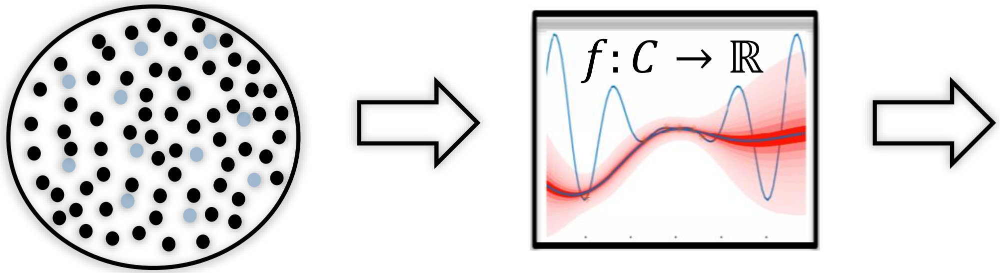
Tool Support: SPL Conqueror

- Sampling + Learning (https://github.com/se-passau/SPLConqueror)



Other Learning Approaches

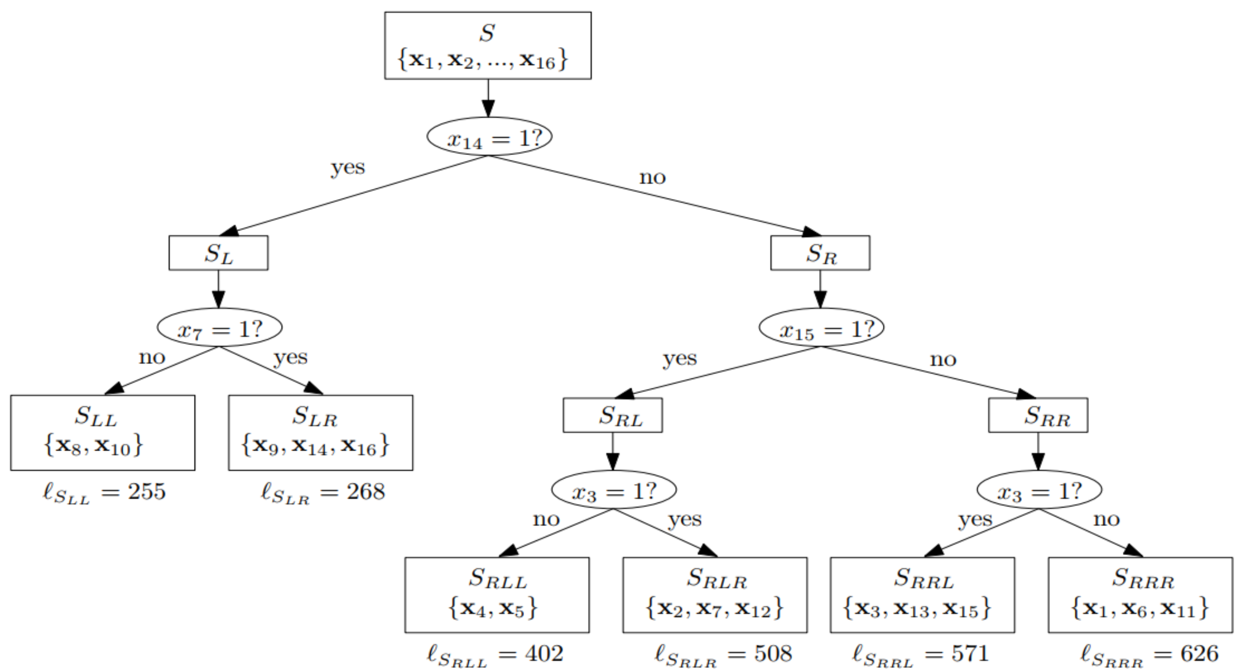
Learning Performance Models



Predict any configuration
Find (near-)optimal configuration
Find influencing options/interactions

Accurate prediction: Using classification and regression trees (CART)

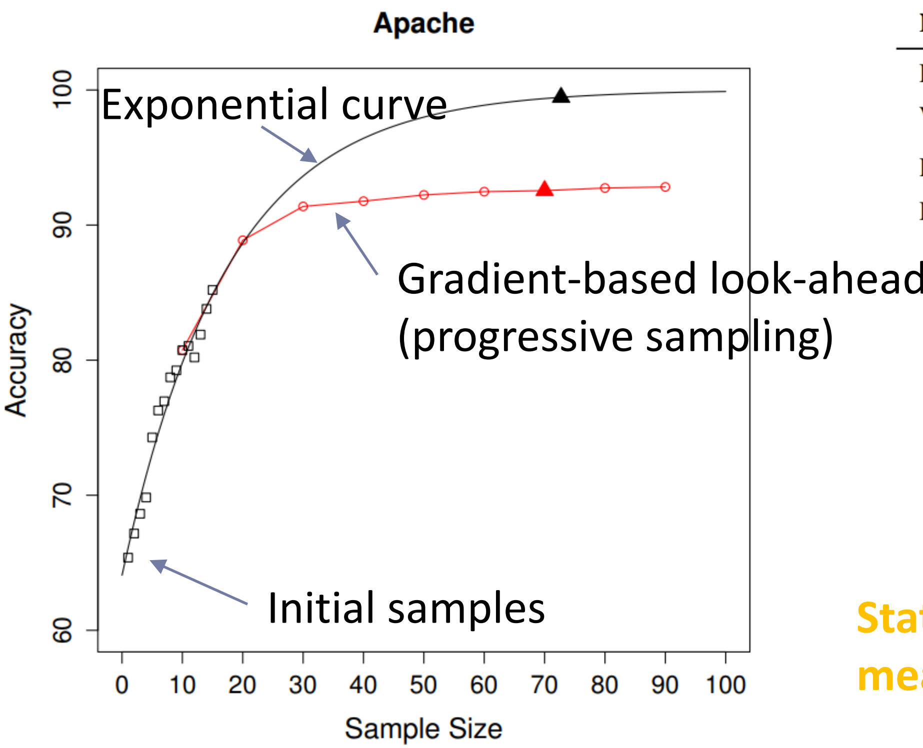
[13] Guo et al. ASE'13:



Learning Performance Models II

Accurate prediction: CART + feature-frequency sampling + early abortion

[23] Sakar et al. ASE’15: Plot #samples with accuracy and fit a function telling when to abort

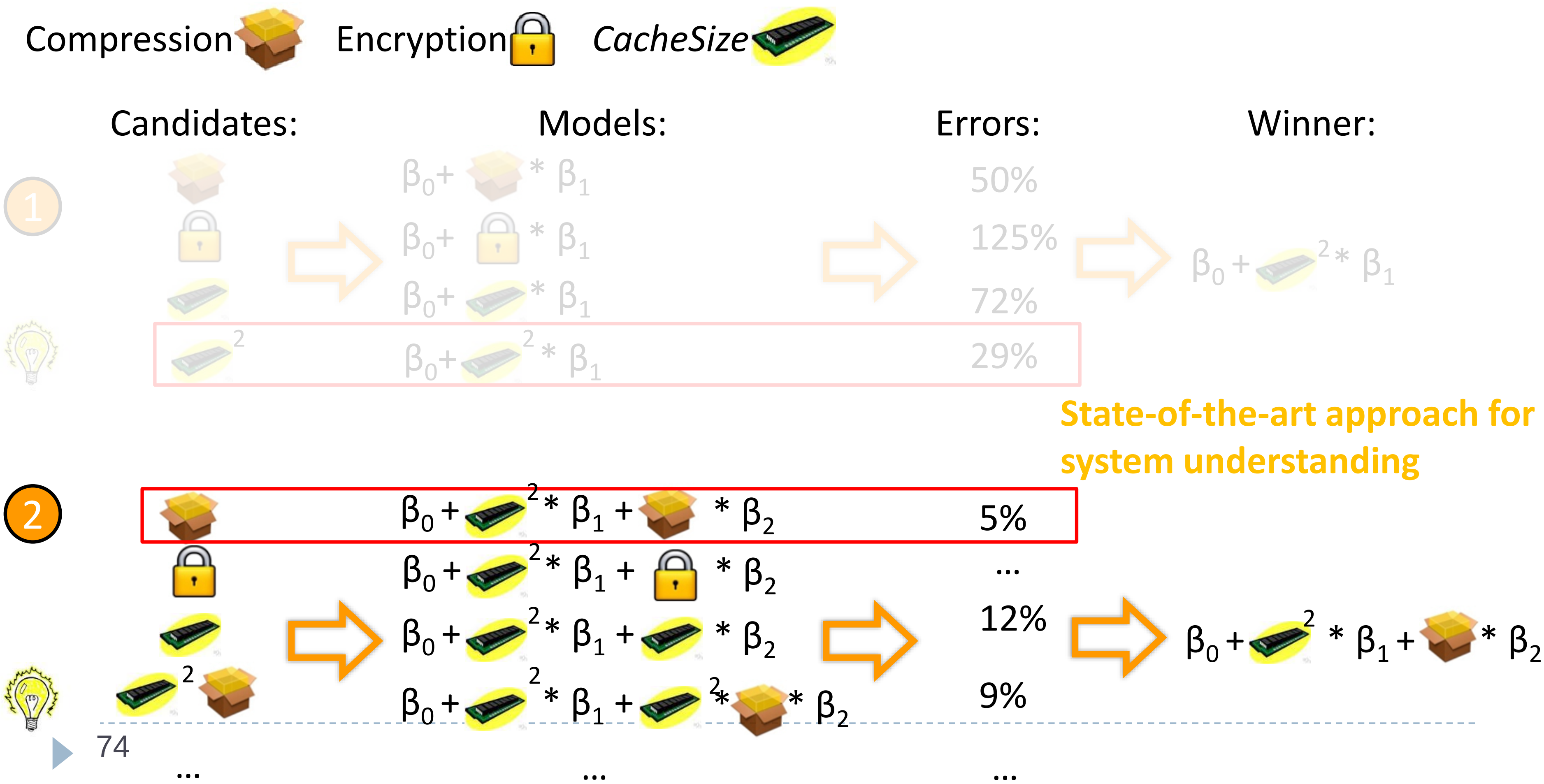


Name	Equation	Optimal Sample Size
Logarithmic	$err(n) = a + b.log(n)$	$n^* = -(R \cdot \lvert S\rvert \cdot b)/2$
Weiss and Tian	$err(n) = a + bn/(n+1)$	$n^* = \sqrt{(-R \cdot \lvert S\rvert \cdot b)/2}$
Power Law	$err(n) = an^b$	$n^* = (\frac{-2}{R \cdot \lvert S\rvert \cdot a \cdot b})^{\frac{1}{b-1}}$
Exponential	$err(n) = ab^n$	$n^* = \log_b\left(\frac{-2}{R \cdot \lvert S\rvert \cdot a \cdot \ln b}\right)$

State-of-the-art approach for accuracy-measurement tradeoff

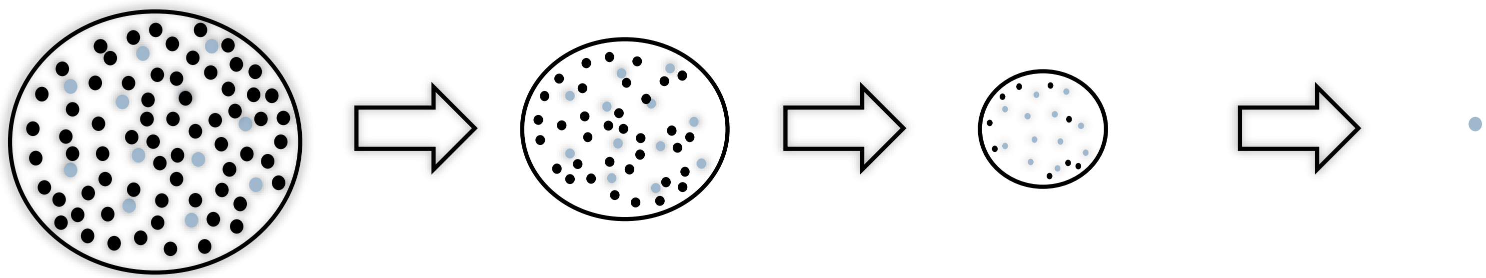
Learning Performance Models III

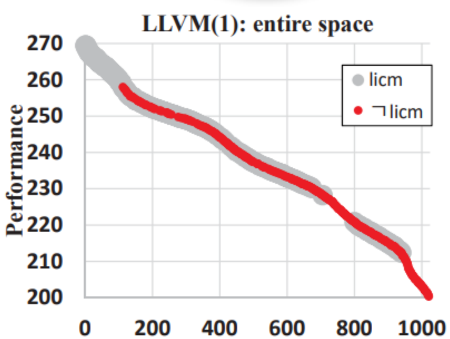
System understanding: [22] Siegmund et al. FSE’15: Find influencing options and interactions via step-wise construction of performance model using multivariate regression

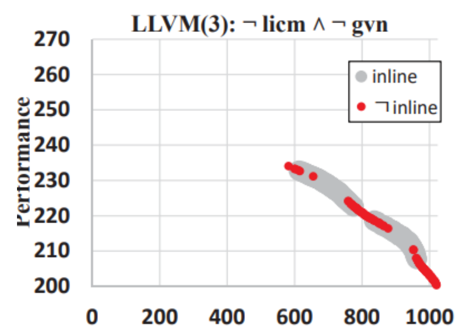


Learning Performance Models IV

Finding near-optimal configurations: [6] Oh et al. FSE’17: True random sampling + select best in sample set + infer good/bad options + shrink configuration space accordingly + repeat



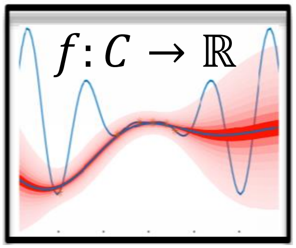




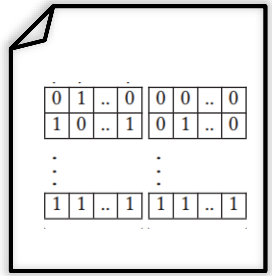
State-of-the-art approach for finding the near-optimal configuration with minimal #measurements

Finding the "Best" Configuration

Optimization Overview



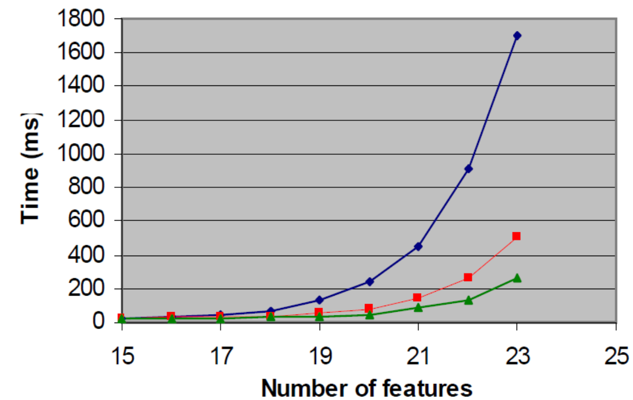
Surrogate model



Single-objective optimization
Multi-/Many-objective optimization
Partial configuration support

[33] Benavides et al. CAiSE'05 : Translating to constraint satisfaction problem

[16] Siegmund et al. SQJ'12: Similar as [33] + qualitative constraints



Problem: Exponential solving time (NP-hard); proved in:

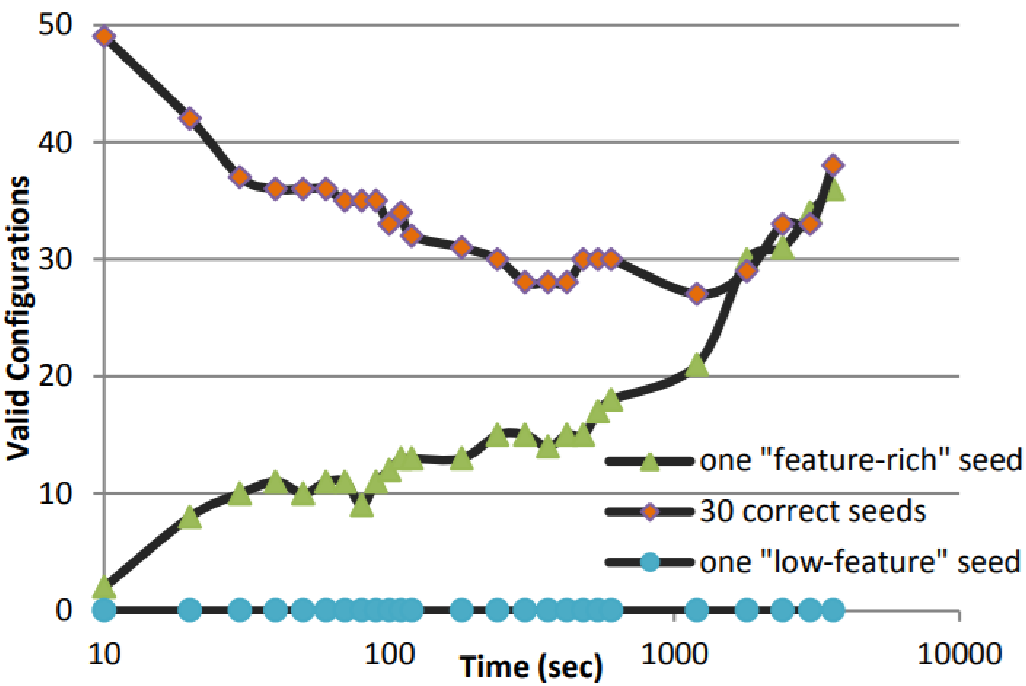
[24] White et al. JSS'09: Translating to knapsack problem via filtered cartesian flattening

Solution: Non-exact method, such as meta-heuristics, with main focus on how to handle constraints

Meta-Heuristic Based Optimization

Fix invalid configurations: [26] Guo et al. JSS'11: Genetic algorithm + search in invalid space + repair operation to return in valid configuration space

Encode constraints as additional objectives: [31,32] Sayyad et al. ICSE'13,ASE'13: Genetic algorithm (NSGA-II + IBEA) + improving fitness by reducing unsatisfied constraints



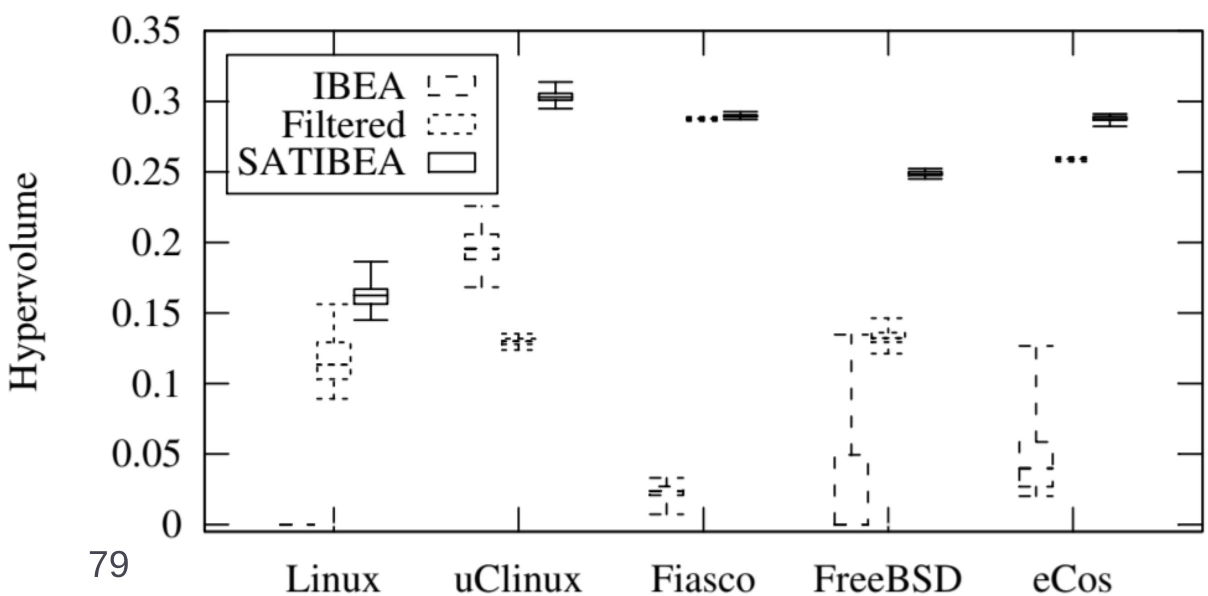
Scalability problems (30mins for 30 valid solutions based on 1 initial valid solution)

(see my other lecture on Search-Based Software Engineering)

Meta-Heuristic Based Optimization

Consider only valid configurations: [5] Henard et al. ICSE’15: “random” SAT-based sampling + constraint-aware mutation + configuration replacement + IBEA

Feature model	Version	Features *(mandatory)*	Constraints
Linux [25]	2.6.28.6	6,888 *(58)*	343,944
uClinux [26]	20100825	1,850 *(7)*	2,468
Fiasco [26]	2011081207	1,638 *(49)*	5,228
FreeBSD [25]	8.0.0	1,396 *(3)*	62,183
eCos [25], [27]	3.0	1,244 *(0)*	3,146



Improved scalability
More valid solutions

And many more…

Optimizing Selection of Competing Features via Feedback-Directed Evolutionary Algorithms

Tian Huat Tan† Yinxing Xue* Manman Chen* Jun Sun† Yang Liu‡ Jin Song Dong*
†Singapore University of Technolog
*National University of Sing
‡Nanyang Technological Uni

[39] Tan et al. ISSTA'15

SIP: Optimal Product Selection from Feature Models Using Many-Objective Evolutionary Optimization

ROBERT M. HIERONS, MIQING LI, and XIAOHUI LIU, Brunel University London, UK
SERGIO SEGURA, University of Seville, Spain
WEI ZHENG, Northwestern Polytechnical University, China

[40] Hierons et al. TOSEM'16

Combining Evolutionary Algorithms with Constraint Solving for Configuration Optimization

[41] Kai Shi ICSME'17

Comparison of Exact and Approximate Multi-Objective Optimization for Software Product Lines

Rafael Olaechea, Derek Rayside, Jianmei Guo, Krzysztof Czarnecki
University of Waterloo
Waterloo, Ontario
{rolaechea, gjm,kczarnec}@gsd.uwaterloo.ca, {drayside}@uwaterloo.ca

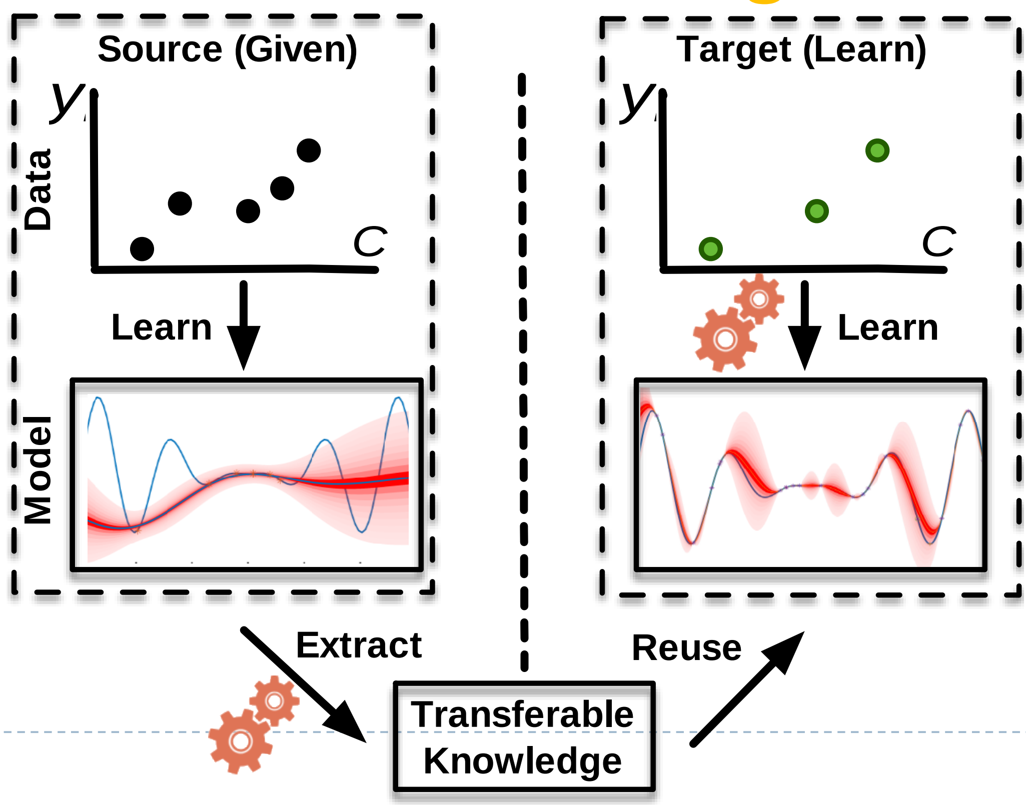
[42] Olaechea et al. SPLC'14

Vision: Transfer Learning I

So far, one performance model for one scenario/workload/hardware:

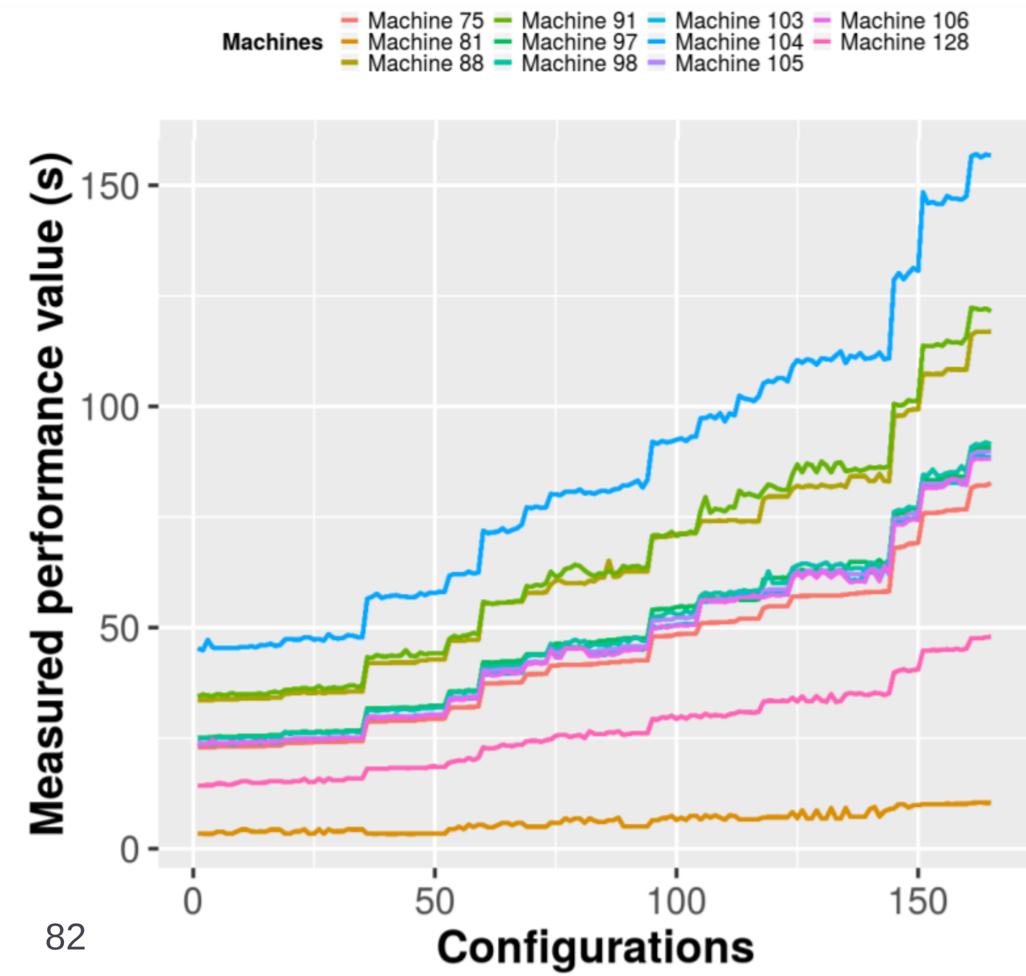
Environment change - >
New performance model

Transfer Learning



Transfer Learning II

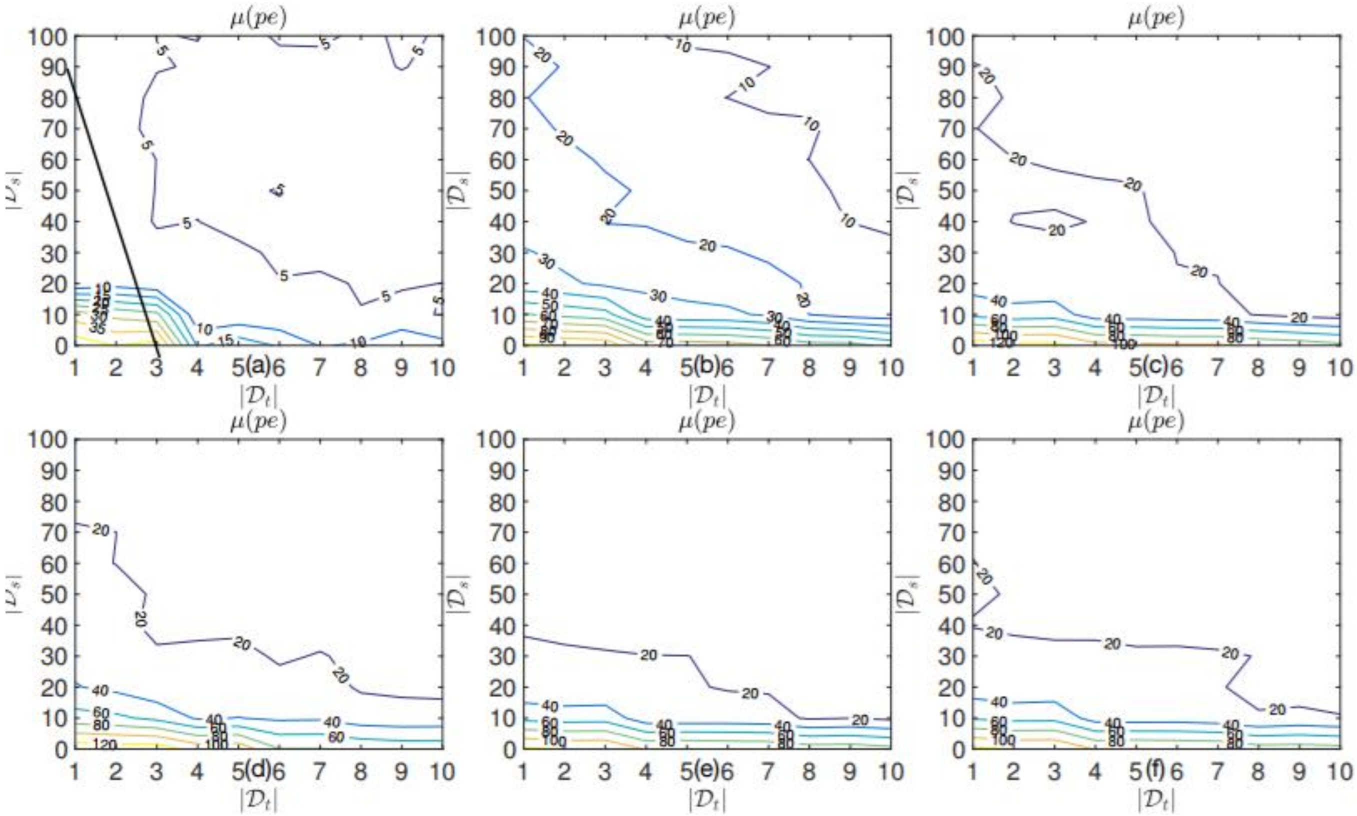
Handle hardware changes: [43] Valov et al. ICPE'17: Adapt a learned performance model to a changed hardware using a linear function



Handles only very simple changes
Linearity is too limited

Transfer Learning III

Handle arbitrary changes: [44] Jamshidi et al. SEAMS'17: Using a kernel function + Gaussian Process (GP) Model to handle version, workload, and hardware changes



GP is not scalable

General transferability shown, but what knowledge exactly can be transferred?

Transfer Learning IV

Handle arbitrary changes: [45] Jamshidi et al. ASE'17: Empirical analysis about transferable knowledge of environmental changes (hardware, software version, workload)

TABLE II: Results indicate that there exist several forms of knowledge that can be transfered across environments and can be used in transfer learning.

		RQ1					RQ2					RQ3				RQ4			
		H1.1	H1.2	H1.3	H1.4		H2.1				H2.2	H3.1			H3.2	H4.1			H4.2
$ec_6 : [h_1, w_1 \to w_2, v_1 \to v_2]$	L	-0.10	**0.72**	-0.05	0.35	0.04	5	6	1	3	**0.68**	**7**	**21**	7	0.31	**0.50**	**0.45**	**1**	**0.96**
$ec_7 : [h_1 \to h_2, w_1 \to w_4, v_2 \to v_1]$	VL	-0.10	6.95	0.14	0.41	0.15	6	4	2	2	**0.88**	**21**	**7**	7	-0.44	**0.47**	**0.50**	**1**	**0.97**
x264— Workload (#pictures/size): $w_1 : 8/2, w_2 : 32/11, w_3 : 128/44$; Version: $v_1 : r2389, v_2 : r2744, v_3 : r2744$																			
$ec_1 : [h_2 \to h_1, w_3, v_3]$	SM	**0.97**	1.00	**0.99**	**0.97**	**0.92**	9	10	8	0	0.86	21	33	18	1.00	0.49	0.49	1	1
$ec_2 : [h_2 \to h_1, w_1, v_3]$	S	**0.96**	0.02	**0.96**	**0.76**	**0.79**	9	9	8	0	0.94	36	27	24	1.00	0.49	0.49	1	1
$ec_3 : [h_1, w_1 \to w_2, v_3]$	M	0.65	**0.06**	0.63	0.53	0.58	9	11	8	1	**0.89**	27	33	22	**0.96**	0.49	0.49	1	1
$ec_4 : [h_1, w_1 \to w_3, v_3]$	M	0.67	**0.06**	0.64	0.53	0.56	9	10	7	1	**0.88**	27	33	20	**0.96**	0.49	0.49	1	1
$ec_3 : [h_2, w_1 \to w_2, v_1]$	S	**0.96**	1.27	**0.83**	0.40	0.35	2	3	1	0	1	9	9	7	0.99	N/A	N/A	N/A	N/A
$ec_4 : [h_2, w_3 \to w_4, v_1]$	M	0.50	**1.24**	0.43	0.17	0.43	1	1	0	0	1	4	2	2	**1.00**	N/A	N/A	N/A	N/A
$ec_5 : [h_1, w_1, v_1 \to v_2]$	M	**0.95**	1.00	0.79	0.24	0.29	2	4	1	0	1	12	11	7	0.99	N/A	N/A	N/A	N/A
$ec_6 : [h_1, w_2 \to w_1, v_1 \to v_2]$	L	0.51	**2.80**	0.44	0.25	0.30	3	4	1	1	0.31	7	11	6	**0.96**	N/A	N/A	N/A	N/A
$ec_7 : [h_2 \to h_1, w_2 \to w_1, v_1 \to v_2]$	VL	0.53	4.91	0.53	0.42	0.47	3	5	2	1	0.31	7	13	6	**0.97**	N/A	N/A	N/A	N/A
$ec_8 : [h_1, w_3 \to w_4, v_1]$	L	0.68	**1.70**	0.56	0.00	**0.91**	14	13	9	1	**0.88**	57	67	36	0.34	0.11	0.14	0.05	0.67
$ec_9 : [h_1, w_3 \to w_5, v_1]$	VL	0.06	3.68	0.20	0.00	0.64	16	10	9	0	**0.90**	51	58	35	-0.52	0.11	0.21	0.06	-0.41
$ec_{10} : [h_1, w_4 \to w_5, v_1]$	L	0.70	4.85	0.76	0.00	**0.75**	12	12	11	0	**0.95**	58	57	43	0.29	0.14	0.20	0.64	-0.14
$ec_{11} : [h_1, w_6 \to w_7, v_1]$	S	0.82	5.79	0.77	0.25	**0.88**	36	30	28	2	**0.89**	109	164	102	**0.96**	N/A	N/A	N/A	N/A
$ec_{12} : [h_1, w_6 \to w_8, v_1]$	S	**1.00**	0.52	**0.92**	**0.80**	**0.97**	38	30	22	6	0.94	51	53	43	0.99	N/A	N/A	N/A	N/A
$ec_{13} : [h_1, w_8 \to w_7, v_1]$	S	**1.00**	0.32	**0.92**	0.53	**0.99**	30	33	26	1	0.98	53	89	51	1.00	N/A	N/A	N/A	N/A
$ec_{14} : [h_1, w_9 \to w_{10}, v_1]$	L	0.24	4.85	0.56	0.44	0.77	22	21	18	3	0.69	**237**	**226**	**94**	**0.86**	N/A	N/A	N/A	N/A

ES: Expected severity of change (Sec. III-B): **S**: small change; **SM**: small medium change; **M**: medium change; **L**: large change; **VL**: very large change.
SaC workload descriptions: srad: random matrix generator; pfilter: particle filtering; hotspot: heat transfer differential equations; k-means: clustering; nw: optimal matching; nbody: simulation of dynamic systems; cg: conjugate gradient; gc: garbage collector. Hardware descriptions (ID: Type/CPUs/Clock (GHz)/RAM (GiB)/Disk):
h1: NUC/4/1.30/15/SSD; **h2**: NUC/2/2.13/7/SCSI; **h3**:Station/2/2.8/3/SCSI; **h4**: Amazon/1/2.4/1/SSD; **h5**: Amazon/1/2.4/0.5/SSD; **h6**: Azure/1/2.4/3/SCSI
Metrics: **M1**: Pearson correlation; **M2**: Kullback-Leibler (KL) divergence; **M3**: Spearman correlation; **M4/M5**: Perc. of top/bottom conf.; **M6/M7**: Number of influential options; **M8/M9**: Number of options agree/disagree; **M10**: Correlation btw importance of options; **M11/M12**: Number of interactions; **M13**: Number of interactions agree on effects;

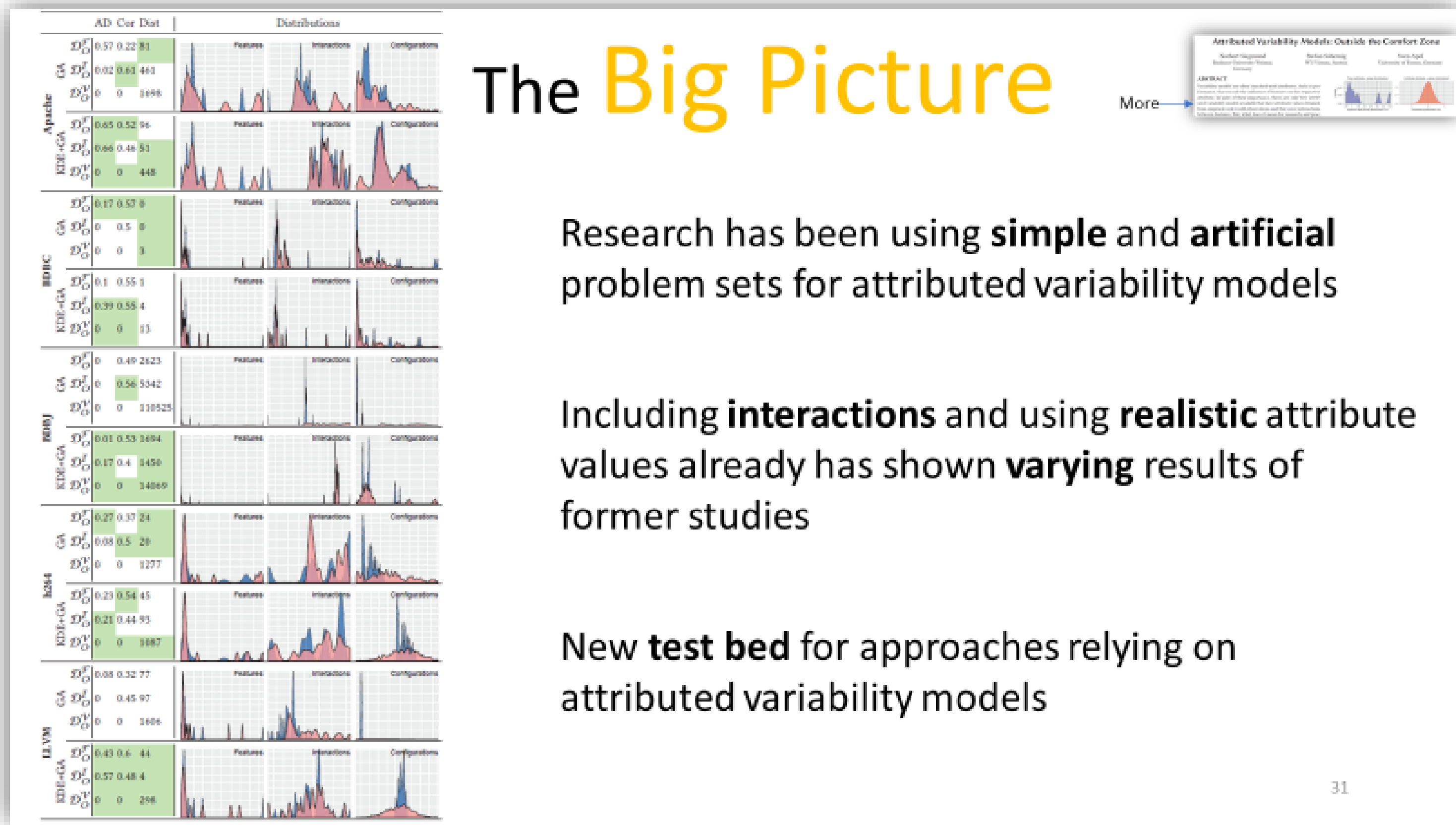
Insight 1. **Performance distributions can be transferred:** Potential for learning a non-linear transfer function.

Insight 2. **Configuration ranks can be transferred:** Good configurations stay good for changing hardware.

Insight 3. **Influential options and interactions can be transferred:** Relevant options in one environment stay relevant in other environments.

Vision: Reproducibility in SBSE

Reproducibility & realistic settings: [17] Siegmund et al. FSE'17: Replication study of [31,32] showed partially changed outcome when having a realistic optimization setting



Thor, the accompanying tool

https://github.com/se-passau/thor-avm

Summary

- Non-functional properties are important when deriving a new variant from a product line
- Qualitative and quantitative properties
- Problem of the huge measurement effort for quantitative properties

- Idea: Sampling a few configurations, measure them, build an influence model, and use the influence model to find the best configuration or predict unseen configurations

Outlook

- Big Picture
  - Product lines

Literature

- [1] Boehm et al., Characteristics of Software Quality. Elsevier, 1978.
- [2a] Siegmund et al. Scalable Prediction of Non-functional Properties in Software Product Lines. In Proceedings of International Software Product Lines Conference (SPLC), pages 160–169. IEEE, 2011.
- [2b] Siegmund et al. Predicting Performance via Automated Feature-Interaction Detection. In Proceedings of International Conference on Software Engineering (ICSE), 2012.
- [3] Patrik Berander et al, Software quality attributes and trade-offs, Blekinge Institute of Technology, 2005 (http://www.uio.no/studier/emner/matnat/ifi/INF5180/v10/undervisningsmateriale/reading-materials/p10/Software_quality_attributes.pdf)
- [4] Stanley S. Stevens. On the theory of scales of measurement. Sciences, 103(2684):677–680, 1946.
- [5] Batory et al., Feature interactions, products, and composition. In Proceedings of the International Conference on Generative Programming (GPCE), pages 13–22. ACM, 2011.
- [6] Martin Glinz. On non-functional requirements. In International Conference on Requirements Engineering, pages 21–26. IEEE, 2007.
- [7] McCall et al., Factors in software quality. Volume I. Concepts and definitions of software quality. Technical Report, General Electric Co Sunnyvale California, 1977.

References I

[1] Tianyin Xu, Long Jin, Xuepeng Fan, Yuanyuan Zhou, Shankar Pasupathy, and Rukma Talwadker. Hey, you have given me too many knobs!: Understanding and dealing with over-designed configuration in system software. In Foundations of Software Engineering (FSE), 2015.

[2] Dana Van Aken, Andrew Pavlo, Geoffrey J Gordon, and Bohan Zhang. Automatic database management system tuning through large-scale machine learning. In International Conference on Management of Data (ICMD). ACM, 2017.

[3] Pooyan Jamshidi and Giuliano Casale. An uncertainty-aware approach to optimal configuration of stream processing systems. In Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS), 2016.

[4] Herodotos Herodotou, Harold Lim, Gang Luo, Nedyalko Borisov, Liang Dong, Fatma Bilgen Cetin, and Shivnath Babu. Starfish: A self-tuning system for big data analytics. In Conference on Innovative Data Systems Research (CIDSR), 2011.

[5] Christopher Henard, Mike Papadakis, Mark Harman, and Yves Le Traon. Combining Multi-Objective Search and Constraint Solving for Configuring Large Software Product Lines. In International Conference on Software Engineering (ICSE), 2015.

[6] Jeho Oh, Don S. Batory, Margaret Myers, and Norbert Siegmund. Finding near-optimal Configurations in Product Lines by Random Sampling. In Foundations of Software Engineering (ESEC/FSE), 2017.

[7] S. Chakraborty, D. Fremont, K. S. Meel, S. A. Seshia, and M. Vardi. Distribution-aware Sampling and Weighted Model Counting for SAT. In Conference on Artificial Intelligence (AAAI), 2014.

[8] V. Gogate and R. Dechter. A new Algorithm for Sampling CSP Solutions Uniformly at Random. In International Conference on Principles and Practice of Constraint Programming (CP), 2006.

[9] Rina Dechter, Kalev Kask, Eyal Bin, and Roy Emek. Generating Random Solutions for Constraint Satisfaction Problems. In National Conference on Artificial Intelligence (AAAI), 2002.

[10]F. Medeiros, C. Kästner, M. Ribeiro, R. Gheyi, and S. Apel. A Comparison of 10 Sampling Algorithms for Configurable Systems. In International Conference on Software Engineering (ICSE), 2016.

[11] C. Henard, M. Papadakis, G. Perrouin, J. Klein, P. Heymans, and Y. Traon. Bypassing the Combinatorial Explosion: Using Similarity to Generate and Prioritize t-wise Test Configurations for Software Product Lines. In IEEE Transactions on Software Engineering (TSE), 2014.

[12] Paul Temple, Mathieu Acher, Jean-Marc Jézéquel, Léo Noel-Baron, and José Galindo. Learning-Based Performance Specialization of Configurable Systems. In Research Report IRISA (TR), 2017.

[13] Jianmei Guo, Krzysztof Czarnecki, Sven Apel, Norbert Siegmund, and Andrzej Wasowski. Variability-aware performance prediction: A statistical learning approach. (ASE), 2013

References II

[14] Vivek Nair, Tim Menzies, Norbert Siegmund, and Sven Apel. Using Bad Learners to find Good Configurations. In Foundations of Software Engineering (FSE), 2017.

[15] Yi Zhang, Jianmei Guo, Eric Blais, and Krzysztof Czarnecki. Performance Prediction of Configurable Software Systems by Fourier Learning. In International Conference on Automated Software Engineering (ASE), 2015.

[16] Norbert Siegmund, Marko Rosenmüller, Martin Kuhlemann, Christian Kästner, Sven Apel, and Gunter Saake. SPL Conqueror: Toward optimization of non-functional properties in software product lines. In Software Quality Journal (SQJ), 2012.

[17] Norbert Siegmund, Stefan Sobernig, and Sven Apel. Attributed Variability Models: Outside the Comfort Zone. In Foundations of Software Engineering (FSE), 2017.

[18] M. B. Cohen, M. B. 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 (TSE), 2008.

[19] M. F. Johansen, Ø. Haugen, and F. Fleurey. An Algorithm for Generating t-wise Covering Arrays from Large Feature Models. In International Software Product Line Conference (SPLC), 2012.

[20]Norbert Siegmund, Marko Rosenmüller, Christian Kästner, Paolo G. Giarrusso, Sven Apel, and Sergiy Kolesnikov. Scalable Prediction of Non-functional Properties in Software Product Lines. In International Software Product Line Conference (SPLC), 2011.

[21] Norbert Siegmund, Sergiy Kolesnikov, Christian Kästner, Sven Apel, Don Batory, Marko Rosenmüller, and Gunter Saake. Predicting Performance via Automated Feature-Interaction Detection. In International Conference on Software Engineering (ICSE), 2012.

[22] Norbert Siegmund, Alexander Grebhahn, Sven Apel, and Christian Kästner. Performance-Influence Models for Highly Configurable Systems. In Foundations of Software Engineering (FSE), 2015.

[23] Atrisha Sarkar, Jianmei Guo, Norbert Siegmund, Sven Apel, and Krzysztof Czarnecki. Cost-Efficient Sampling for Performance Prediction of Configurable Systems (T). In (ASE), 2015.

[24] Jules White, Brian Dougherty, and Douglas Schmidt. Selecting Highly Optimal Architectural Feature Sets with Filtered Cartesian Flattening. In Journal of Systems and Software (JSS), 2009.

[25] Norbert Siegmund, Marko Rosenmüller, Christian Kästner, Paolo Giarrusso, Sven Apel, and Sergiy Kolesnikov. Scalable Prediction of Non-functional Properties in Software Product Lines: Footprint and Memory Consumption. Information & Software Technology (IST), 2013.

[26] Jianmei Guo, Jules White, Guangxin Wang, Jian Li, and Yinglin Wang. A Genetic Algorithm for Optimized Feature Selection with Resource Constraints in Software Product Lines. Journal of Systems and Software (JSS), 2011.

References III

[27] Marcela Zuluaga, , Andreas Krause, and Markus Püschel. e-PAL: An Active Learning Approach to the Multi-Objective Optimization Problem. In Journal of Machine Learning Research (JML), 2016.

[28] Marcela Zuluaga, Guillaume Sergent, Andreas Krause, and Markus Püschel. Active Learning for Multi-Objective Optimization. In International Conference on Machine Learning (ICML), 2013.

[29] Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Sequential model-based Optimization for General Algorithm Configuration. In International Conference on Learning and Intelligent Optimization (LION), 2005.

[30] F. Wu, W. Weimer, M. Harman, Y. Jia, and J. Krinke. Deep parameter optimisation. In Conference on Genetic and Evolutionary Computation (GECCO), 2015

[31] Abdel Sayyad, Tim Menzies, and Hany Ammar. On the Value of User Preferences in Search-based Software Engineering: A Case Study in Software Product Lines. In International Conference on Software Engineering (ICSE), 2013.

[32] Abdel Sayyad, Joseph Ingram, Tim Menzies, and Hany Ammar. Scalable Product Line Configuration: A Straw to Break the Camel's Back. In International Conference on Automated Software Engineering (ASE), 2013.

[33] David Benavides, Pablo Trinidad Martín-Arroyo, and Antonio Cortés. Automated Reasoning on Feature Models. In Conference on Advanced Information Systems Engineering (CAiSE), 2005.

[39] Tian Tan, Yinxing Xue, Manman Chen, Jun Sun, Yang Liu, and Jin Song Dong. Optimizing Selection of Competing Features via Feedback-directed Evolutionary Algorithms. In International Symposium on Software Testing and Analysis (ISSTA), 2015.

[40] Robert Hierons , Miqing Li, Xiaohui Liu, Sergio Segura, and Wie Zheng. SIP: Optimal Product Selection from Feature Models Using Many-Objective Evolutionary Optimization. ACM Transactions on Software Engineering and Methodology (TOSEM), 2016.

[41] Kai Shi. Combining Evolutionary Algorithms with Constraint Solving for Configuration Optimization. In International Conference on Software Maintenance and Evolution (ICSME), 2017.

[42] Rafael Olaechea, Derek Rayside, Jianmei Guo, and Krzysztof Czarnecki. Comparison of Exact and Approximate multi-objective Optimization for Software Product Lines. In International Software Product Line Conference (SPLC), 2014.

[43] Pavel Valov, Jean-Christophe Petkovich, Jianmei Guo, Sebastian Fischmeister, and Krzysztof Czarnecki. Transferring Performance Prediction Models Across Different Hardware Platforms. In International Conference on Performance Engineering (ICPE), 2017.

References IV

[44] Pooyan Jamshidi, Miguel Velez, Christian Kästner, Norbert Siegmund, and Prasad Kawthekar. Transfer Learning for Improving Model Predictions in Highly Configurable Software. In (SEAMS), 2017.

[45] Pooyan Jamshidi, Norbert Siegmund, Miguel Velez, Christian Kästner, Akshay Patel, and Yuvraj Agarwal. Transfer learning for performance modeling of configurable systems: an exploratory analysis. In (ASE), 2017.

[46] A. Filieri, H. Hoffmann, and M. Maggio. Automated multi-objective Control for self-adaptive Software Design. In International Symposium on Foundations of Software Engineering (FSE), 2015.

[47] T. Osogami and S. Kato. Optimizing System Configurations Quickly by Guessing at the Performance. In International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS), 2007.

[48] W. Zheng, R. Bianchini, and T. Nguyen. Automatic Configuration of Internet Services. ACM SIGOPS Operating Systems Review(OSR), 2007.

[49] B. Xi, Z. Liu, M. Raghavachari, C. H. Xia, and L. Zhang. A smart Hill-Climbing Algorithm for Application Server Configuration. In International Conference on World Wide Web (WWW), 2004.

[50] Norbert Siegmund, Alexander von Rhein, and Sven Apel. Family-based Performance Measurement. (GPCE), 2013.

[51] Vivek Nair, Tim Menzies, Norbert Siegmund, and Sven Apel. Faster Discovery of Faster System Configurations with Spectral Learning. CoRR abs/1701.08106, arXiv, 2017.

[52] Sven Apel, Sergiy S. Kolesnikov, Norbert Siegmund, Christian Kästner, and Brady Garvin. Exploring feature interactions in the wild: The new feature-interaction challenge. International Workshop on Feature-Oriented Software Development (FOSD), 2013.

[53] Sergiy S. Kolesnikov, Norbert Siegmund, Christian Kästner, and Sven Apel. On the Relation of External and Internal Feature Interactions: A Case Study. CoRR abs/1712.07440 arXiv, 2017.