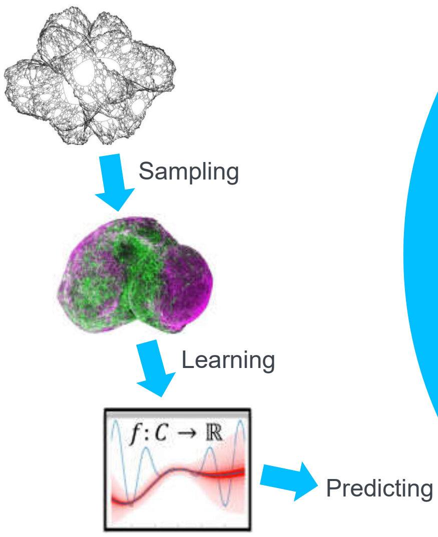
University of Stuttgart
Institute of Software Technology



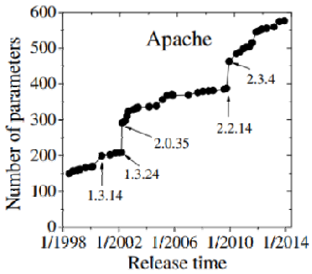
Learning and Predicting the Performance of Configurable Software Systems

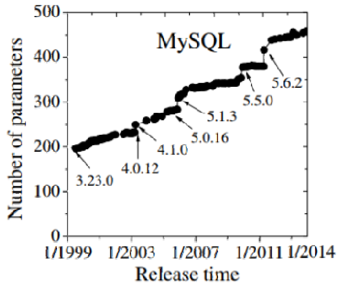
Seminar Advanced Software Engineering, Lion Wagner

10.07.2019

Motivation: More Configurations Options

Hey, You Have Given Me Too Many Knobs! [5]:
Programs are getting increasingly more configuration parameters.

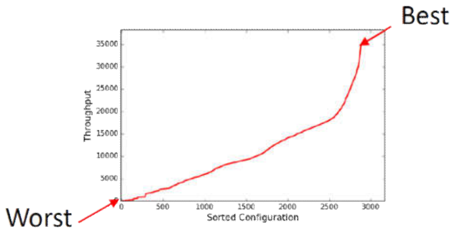


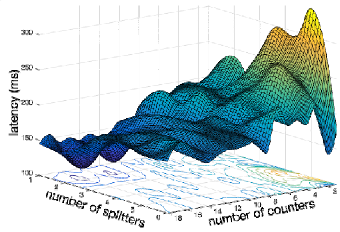


Predicting and Learning the Performance of Configurable Systems, Lion Wagner 10.07.2019 2

Motivation: Configuration matters!

But: Bad configurations can have a big impact on performance:





STORM

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

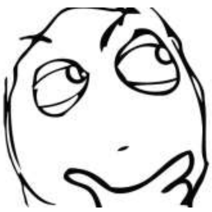
Only by tweaking 2 options out of 200 in Apache Storm - observed **~100%** change in latency

More Configurations → More Possibilities of Performance Influences → Performance prediction is hard.

Predicting and Learning the Performance of Configurable Systems, Lion Wagner 10.07.2019 3

Motivation: Why should we care?

We are looking for:	(Near) Optimal configurations
	Good default configurations
	Implementation problems
	Proof of a good average performance



Can't we just measure every configuration?

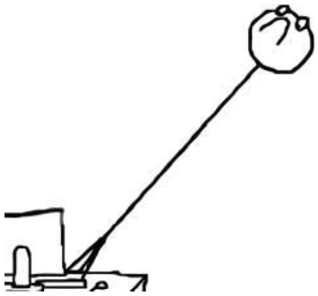
Predicting and Learning the Performance of Configurable Systems, Lion Wagner 10.07.2019 4

The Problem: Examples

of N. Siegmund et al. [1,7]

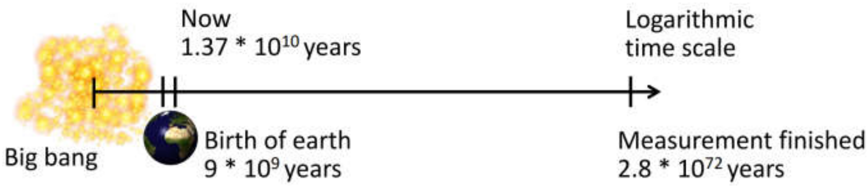
Have a look at Berkley DB (C):

- 19 features
- 2560 configurations
- 426h (~18d) for brute force testing



Let's go **larger** with SQLite!

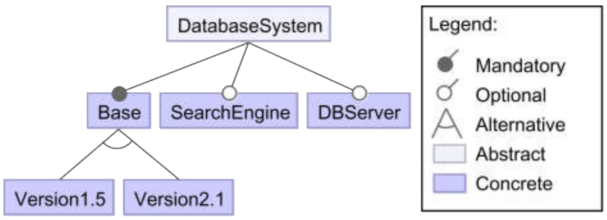
- 77 features
- $3 \cdot 10^{77}$ configurations
- Assumption: 1 measurement = 5min (compilation + benchmark)



Predicting and Learning the Performance of Configurable Systems, Lion Wagner 10.07.2019 5

The Problem: Exponential Configuration Space

Presentation of valid configurations as Feature Model



Version1.5 ⇒ ¬DBServer

⇕

Valid = Base ∧ (Version1.5 ⊕ Version2.1) ∧ (Version1.5 ⇒ ¬DBServer)

Example:

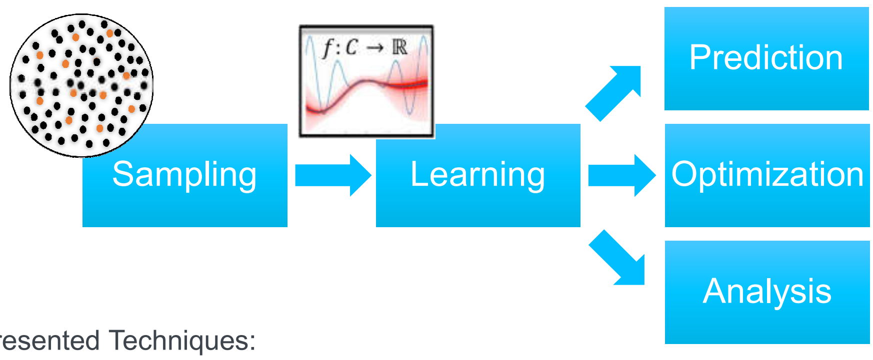
Default = {Base = 1, Version2.1 = 1, SearchEngine = 0, DBServer = 0}

⇒ all valid configurations (Configuration Space) ∈ $O(2^{\#options})$

Predicting and Learning the Performance of Configurable Systems, Lion Wagner 10.07.2019 6

Performance Learning and Prediction

The General Approach



Presented Techniques:

- ~~Brute Force~~
- Automated Feature Interaction Detection
- Projective Sampling

Automated Feature Interaction Detection

Goal: Assign a performance influence value ∏ to each feature (interaction).

A simplified example:

System S w/o features
∏(S) = 100

System S with a
∏(Sxa) = 110

System S with b
∏(Sxb) = 95

System S with a and b
∏(Sxaxb) = 103

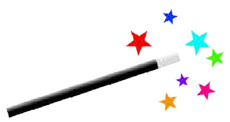
∏(Sxa) = ∏(S) + ∏(a)
⇒ ∏(a) = 10

∏(b) = -5

∏(Sxaxb) = ∏(S) + ∏(a) + ∏(b) + ∏(a#b)
⇒ ∏(a#b) = -2

Automated Feature Interaction Detection Cont'd

Step 1: Find interacting features



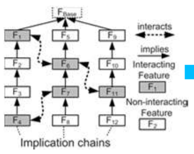
Step 2: Measure feature interaction impact

Pair-Wise Heuristic

Measure Performance Impact

Higher-Order Heuristic

Hot-Spot Heuristic



$O(n^2)$ Pairs

Automated Feature Interaction Detection Cont'd

Step 1: Find interacting features

for each feature f do…

∏(min(f)×f)
∏(max(f)×f)

≈ equal?

yes: ∏(f)=∏(min(f)×f)-∏(min(f))

no: Mark f as interactive

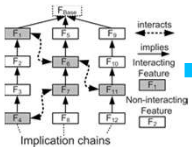
Step 2: Measure feature interaction impact

Pair-Wise Heuristic

Measure Performance Impact

Higher-Order Heuristic

Hot-Spot Heuristic



O(n^2) Pairs

Automated Feature Interaction Detection

Advantages/Disadvantages

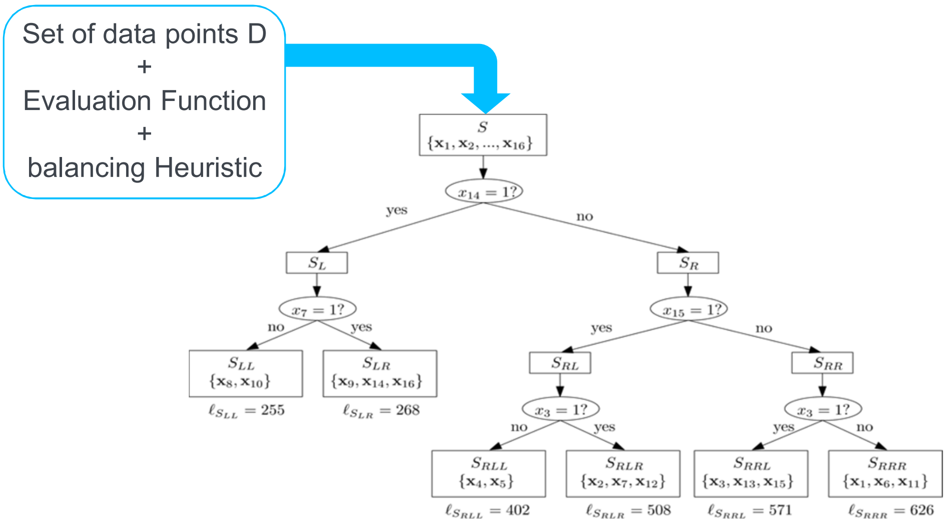
- \+ Sample size ∈ O(n^2)
- \+ Knowledge is explainable
- \+ Works well on small programs

- − Sample size automatically determined
- − (Complicated)

- ○ Average accuracy of 95%

Further details in: N. Siegmund et al.: Predicting performance via automated feature interaction detection. [1]

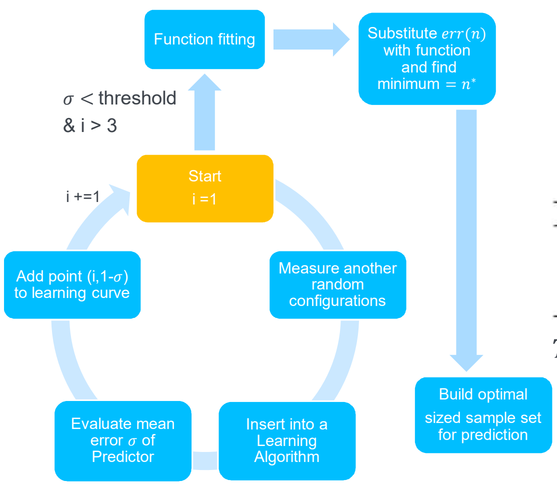
Classification and Regression Trees (CART)

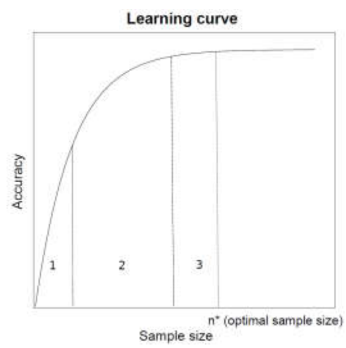


- Configurations are classified by thier performance values

Projective Sampling

- Tires to find an optimal, cost efficient sample size with curve fitting
- Iterative process

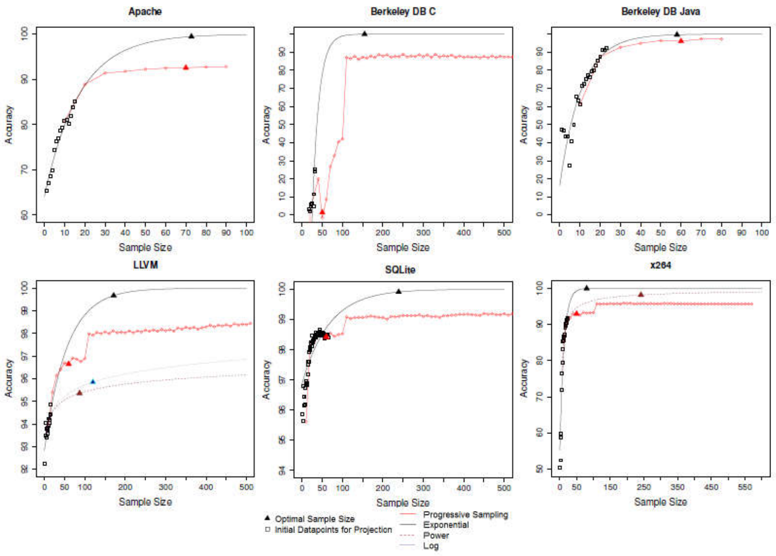




Name	Equation
Logarithmic	$err(n) = a + b.log(n)$
Weiss and Tian	$err(n) = a + bn/(n+1)$
Power Law	$err(n) = an^b$
Exponential	$err(n) = ab^n$

$$TotalCost(n) = 2n + err(n) \cdot |S| \cdot R$$

Projective Sampling: Learning Curve Examples



Progressive Sampling

Advantages/Disadvantages

+ also considers sampling cost
+ up to 99% accuracy

− can use a significantly larger sample then other approaches
− initial sample generation has high influence on accuracy

Further details in: A. Sarkar et al.: Cost-Efficient Sampling for Performance Prediction of Configurable Systems. [3]

Other Sampling Techniques

Random Sampling, J. Gou et al. [2]

Chose random sample (no fixed size) | Whole sample is measured | Put Data into Learning Algorithm

Clustering based Sampling, V. Nair, et al. [4]

Cluster Configurations (e.g. by distance) | Measure one Configuration of each leaf cluster | Put Data into Learning Algorithm

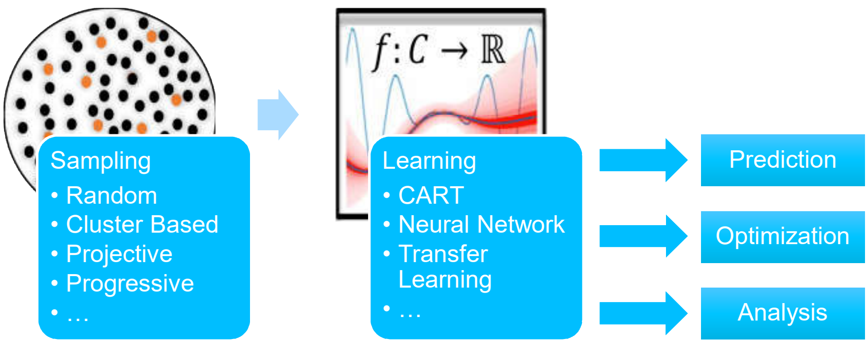
Progressive Sampling, A. Sarkar et al. [3]

Insert configurations stepwise into a Learning Process | Let this sample grow until the learning rate stagnates or the cost function reaches a minimum.

Predicting and Learning the Performance of Configurable Systems, Lion Wagner 10.07.2019 17

Conclusion

- Main Problem: Reduction/Filtering of an exponential Configuration Space
  - Solved by using good sampling techniques
- Learning with typical machine learning strategies



- A good balance between the sample size and accuracy is key
- There is no perfect solution

Predicting and Learning the Performance of Configurable Systems, Lion Wagner 10.07.2019 18

References

[1] N. Siegmund et al.: Predicting performance via automated feature interaction detection. *In Proc. of ICSE*, pages 167-177. IEEE, 2012.

[2] J. Guo et al.: Variability aware performance prediction: A statistical learning approach. *Automated Software Engineering (ASE), 2013 IEEE/ACM 28th International Conference*, pages 301-311. IEEE Press, 2013

[3] A. Sarkar et al.: Cost-Efficient Sampling for Performance Prediction of Configurable Systems. *30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 342-352, 2015

[4] V. Nair, et al.: Faster discovery of faster system configurations with spectral learning. *CoRR* , 2017

[5] T. Xu et al.: Hey, You Have Given Me Too Many Knobs! *ESEC/FSE 2015 Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering,* pages 307-319

References

[6] N. Siegmund et al.: Challenges and Insights from Optimizing Configurable Software Systems.*VAMOS 2019*, 2019.

[7] C. Kästner et al.: Software Product Line Engineering, Bauhaus-University Weimar, 2018

Figures and pictures taken either from Creative Commons or the papers above.