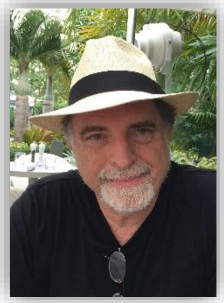
Finding Near-Optimal Configurations in Product Lines by Random Sampling

me

Jeho Oh



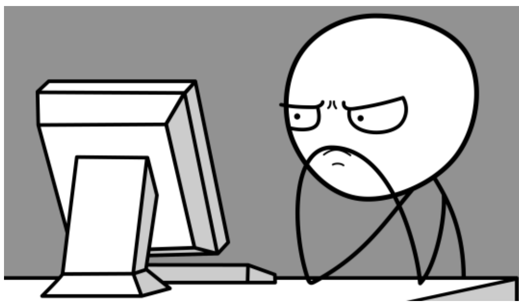
Don Batory

Margaret Myers

Norbert Siegmund

TEXAS
The University of Texas at Austin

Quickly find SPL configurations with near-optimal performance for a given workload

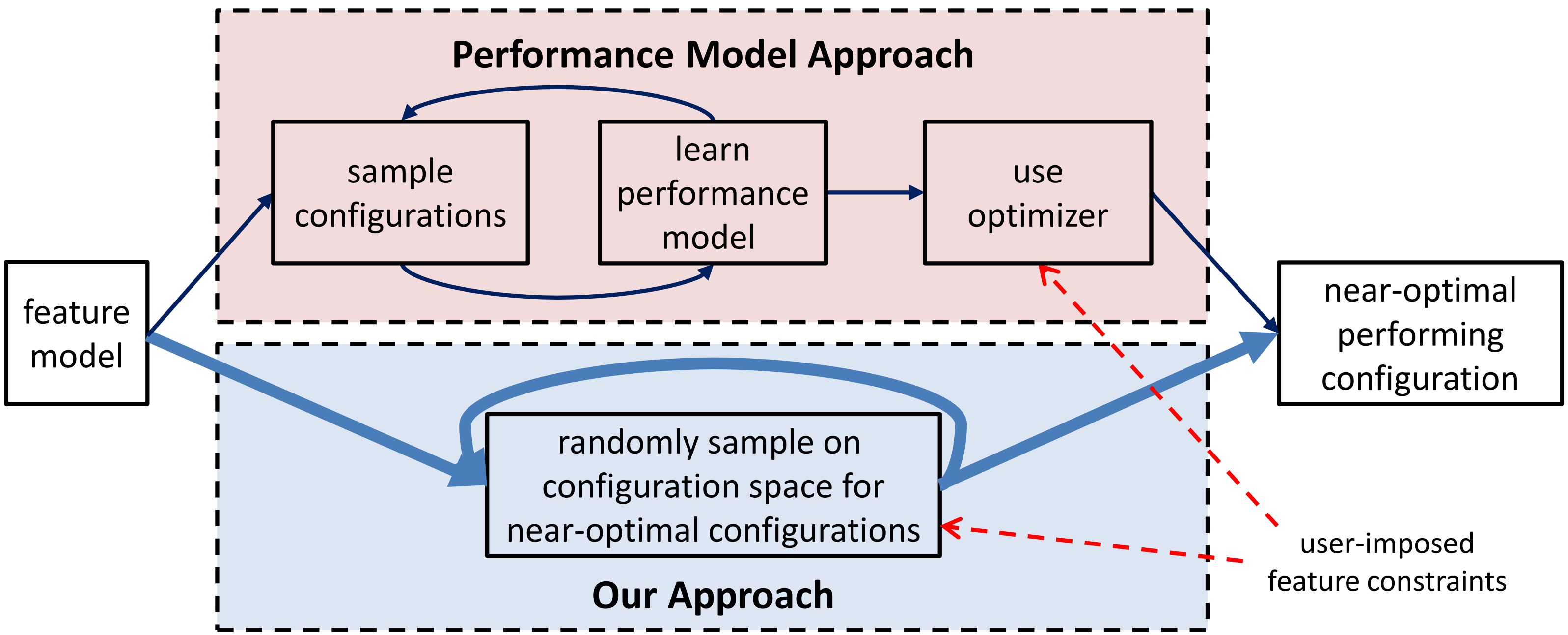


I want a hybrid car with laser headlight, but cheap and light as possible.

Now 273/275 options left to decide…

- Configuration space is often huge: n features $\leq 2^n$ configurations
  (273 optional features: 10^{82} products, one for every atom in universe)
- Searching for the optimal configuration is daunting,
  as benchmarking all configurations is infeasible
- Find a way to get good enough configurations with practical effort

Big Picture


Performance Model Approach
sample configurations
learn performance model
use optimizer
feature model
near-optimal performing configuration
randomly sample on configuration space for near-optimal configurations
Our Approach
user-imposed feature constraints

Contributions

- Allow true random sampling of configurations
- Provide statistical bounds on searching by sampling
- Directly search the space for any given workload

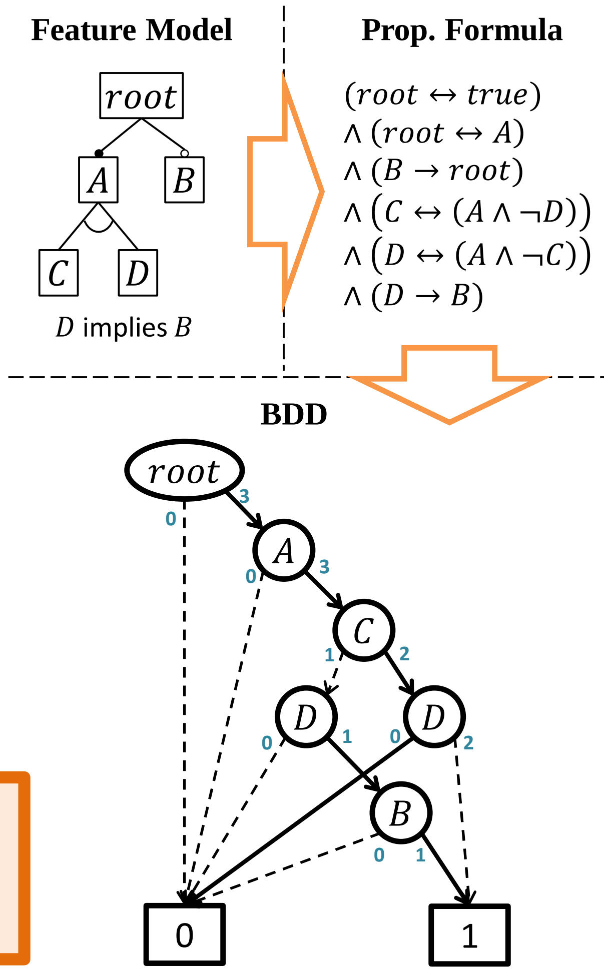
Search by Random Sampling

which we describe next…

Random Sampling with BDD

- To randomly sample configurations from uniform distribution:
  - Identify valid configuration space
  - Select a random number in [1, total # of configs]
  - Return the configuration with matching number
- **Binary Decision Diagram (BDD)**:
  - Compile prop. formula into graph structure
  - Derive all possible solutions (configs)
  - Allows efficient sampling from traversing BDD

Randomly sample from space of all valid configurations, not space of all features



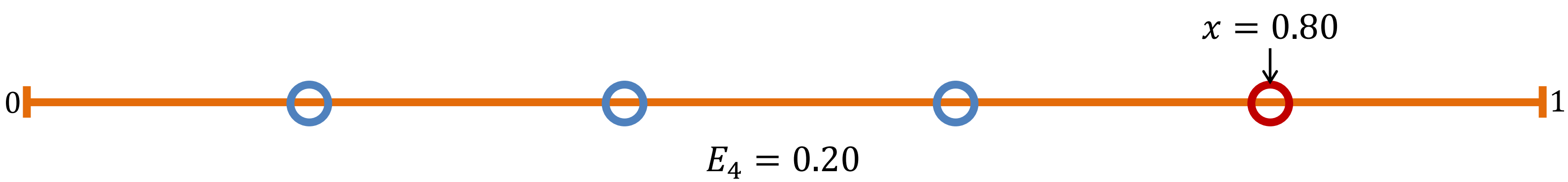
Statistics of Random Sampling (1)

- n random numbers over unit range $[0,1]$, x as the number closest to 1
  Analyze the distance between x and 1, $(1-x)$

- C.D.F. of x over n: $p_n(X \le x) = \int_0^x n \cdot x^{n-1} \cdot dx = x^n$

On average

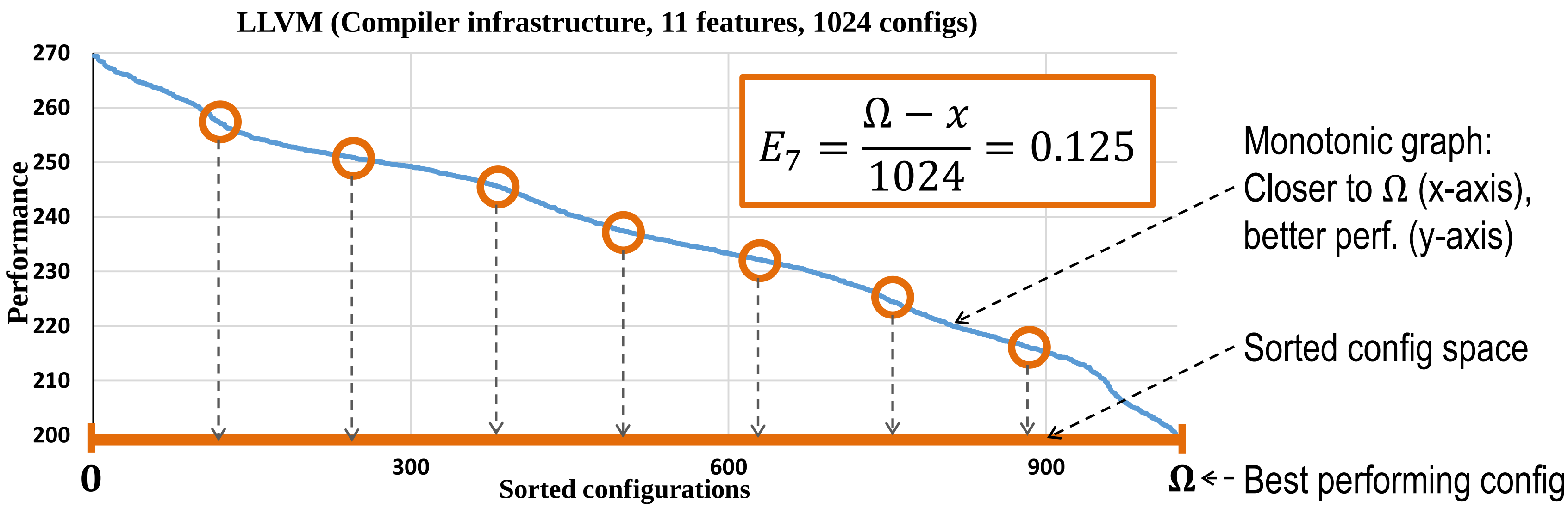
- Average distance from x to 1: $E_n = \int_0^1 (1-x) \cdot n \cdot x^{n-1} \cdot dx = \dfrac{\mathbf{1}}{\boldsymbol{n}\mathbf{+1}}$



Statistical regularity: $E_n = \dfrac{1}{n+1}$ **with bounds** $\sigma_n \approx \dfrac{1}{n+1}$

(Equations for sampling over discrete space on Section 3.3 of the paper)

Statistics of Random Sampling (2)

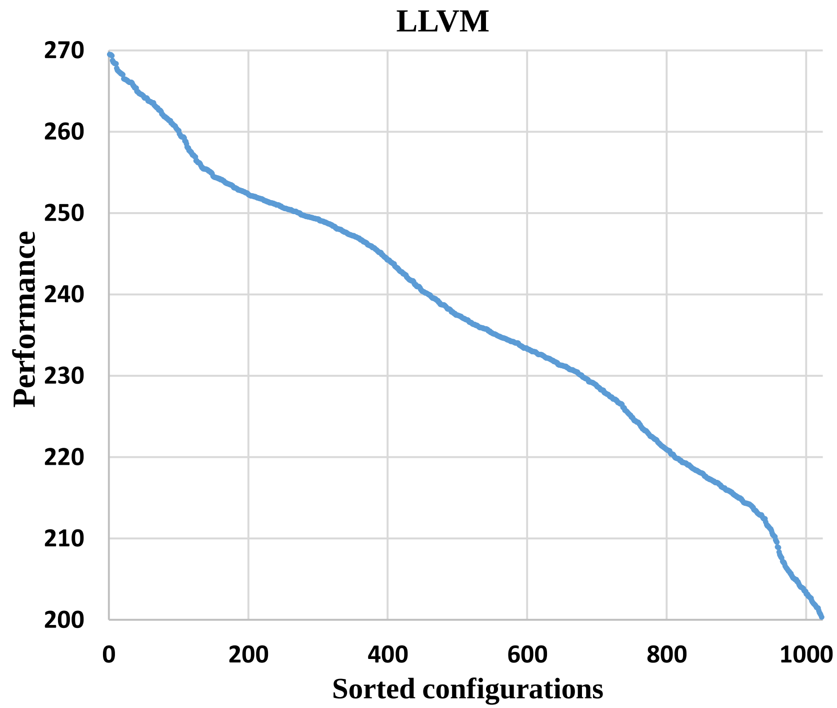


- Correspondences:
  - Selection of numbers in [0,1] ► Selection in [1, total # of configs]
  - Closeness to 1 ► Closeness to the best configuration, Ω

On average, sample with the best performance has top $\frac{100}{n+1}$% performance among all configurations

Statistical Recursive Searching (SRS)

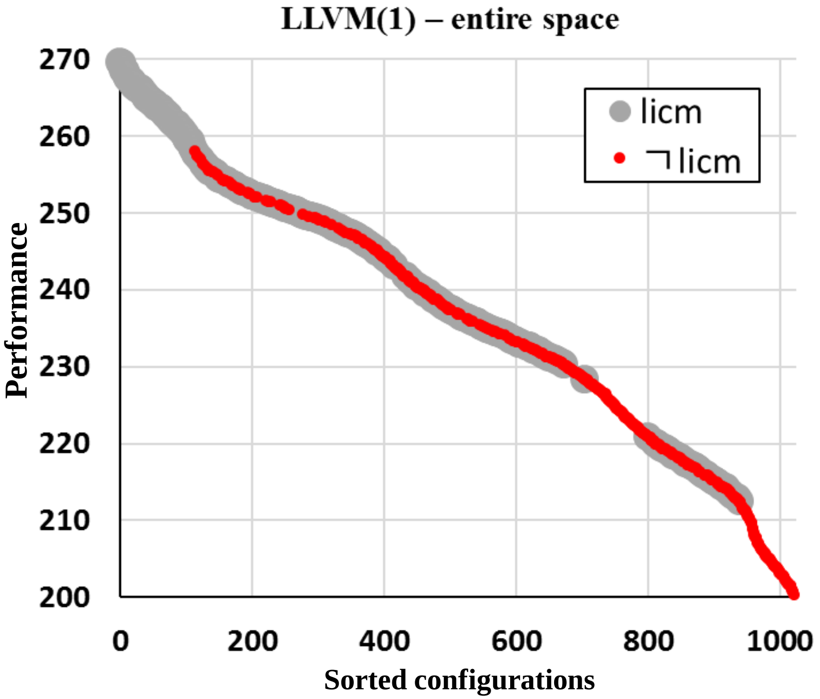
- Can search better than 99 samples for 1% by random sampling
- Feature selection:
  - Makes some configs perform better
  - Constrict config space
- Search within smaller and better config space by feature selection
- Find most influential features by:
  - Performance difference
  - Welch's t-Test



Use samples to recursively constrict the search space

Statistical Recursive Searching (SRS)

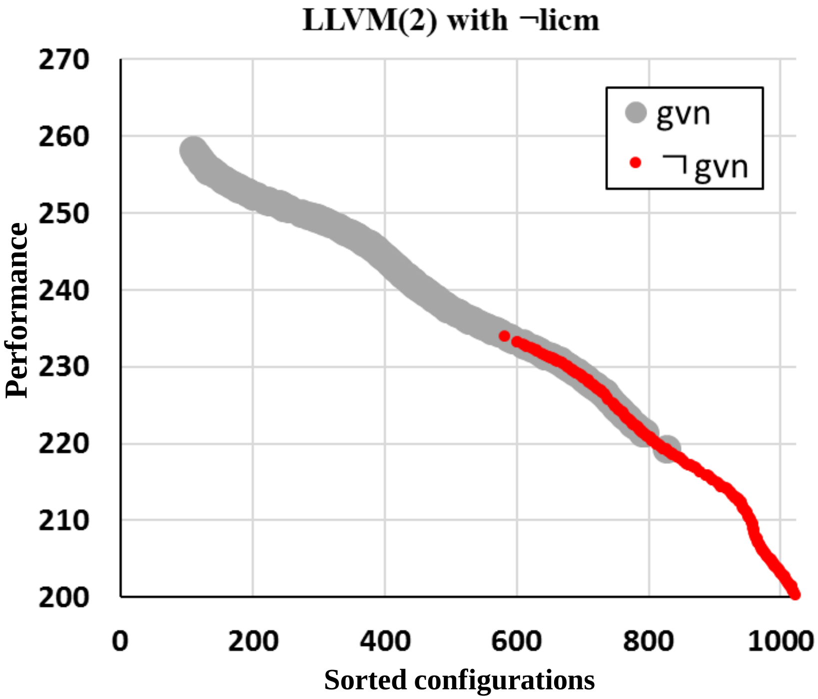
- Can search better than 99 samples for 1% by random sampling
- Feature selection:
  - Makes some configs perform better
  - Constrict config space
- Search within smaller and better config space by feature selection
- Find most influential features by:
  - Performance difference
  - Welch's t-Test



Use samples to recursively constrict the search space

Statistical Recursive Searching (SRS)

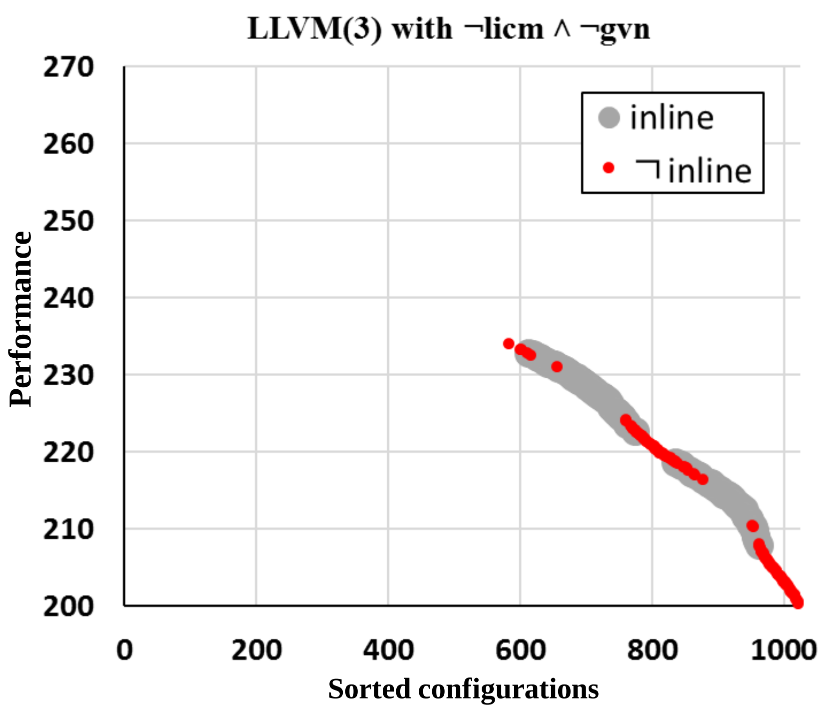
- Can search better than 99 samples for 1% by random sampling
- Feature selection:
  - Makes some configs perform better
  - Constrict config space
- Search within smaller and better config space by feature selection
- Find most influential features by:
  - Performance difference
  - Welch's t-Test



Use samples to recursively constrict the search space

Statistical Recursive Searching (SRS)

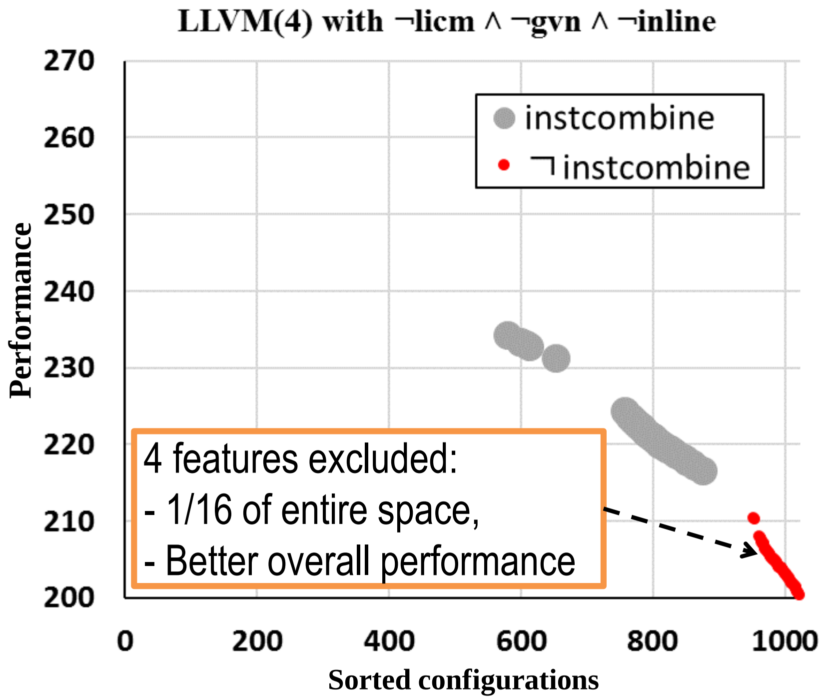
- Can search better than 99 samples for 1% by random sampling
- Feature selection:
  - Makes some configs perform better
  - Constrict config space
- Search within smaller and better config space by feature selection
- Find most influential features by:
  - Performance difference
  - Welch's t-Test



Use samples to recursively constrict the search space

Statistical Recursive Searching (SRS)

- Can search better than 99 samples for 1% by random sampling
- Feature selection:
  - Makes some configs perform better
  - Constrict config space
- Search within smaller and better config space by feature selection
- Find most influential features by:
  - Performance difference
  - Welch's t-Test



Use samples to recursively constrict the search space

ICSE 2012

Predicting Performance via Automated Feature-Interaction Detection

Norbert Siegmund,* Sergiy S. Kolesnikov,† Christian Kästner,‡ Sven Apel,†
Don Batory,§ Marko Rosenmüller,* and Gunter Saake*
* University of Magdeburg, Germany
† University of Passau, Germany
‡ Philipps University Marburg, Germany
§ University of Texas at Austin, USA

***Abstract*—Customizable programs and program families provide user-selectable features to allow users to tailor a program to an application scenario. Knowing in advance which feature selection yields the best performance is difficult because a direct measurement of all possible feature combinations is infeasible. Our work aims at predicting program performance based on selected features. However, when features interact, accurate predictions are challenging. An interaction occurs when a particular feature combination has an unexpected influence on performance. We present a method that automatically detects performance-relevant feature interactions to improve prediction accuracy. To this end, we propose three heuristics to reduce the number of measurements required to detect interactions. Our evaluation consists of six real-world case studies from varying domains (e.g., databases, encoding libraries, and web servers) using different configuration techniques (e.g., configuration files and preprocessor flags). Results show an average prediction accuracy of 95 %.**

features, called a *configuration*, that yields a valid program. However, finding *the best* configuration efficiently is a hard task. There can be hundreds of features resulting in myriads of configurations: 33 optional and independent features yields a configuration for each human on the planet, and 320 optional features yields more configurations than there are estimated atoms in the universe. To find the configuration with the best performance for a specific workload requires an intelligent search; brute-force is infeasible.

We aim at *predicting* a configuration's non-functional properties for a specific workload based on the user-selected features [3][4]. That is, we aggregate the influence of each selected feature on a non-functional property to compute the properties of a specific configuration. Here, we concentrate on performance predictions only. Unfortunately, the accuracy

EVALUATION

Evaluation Method

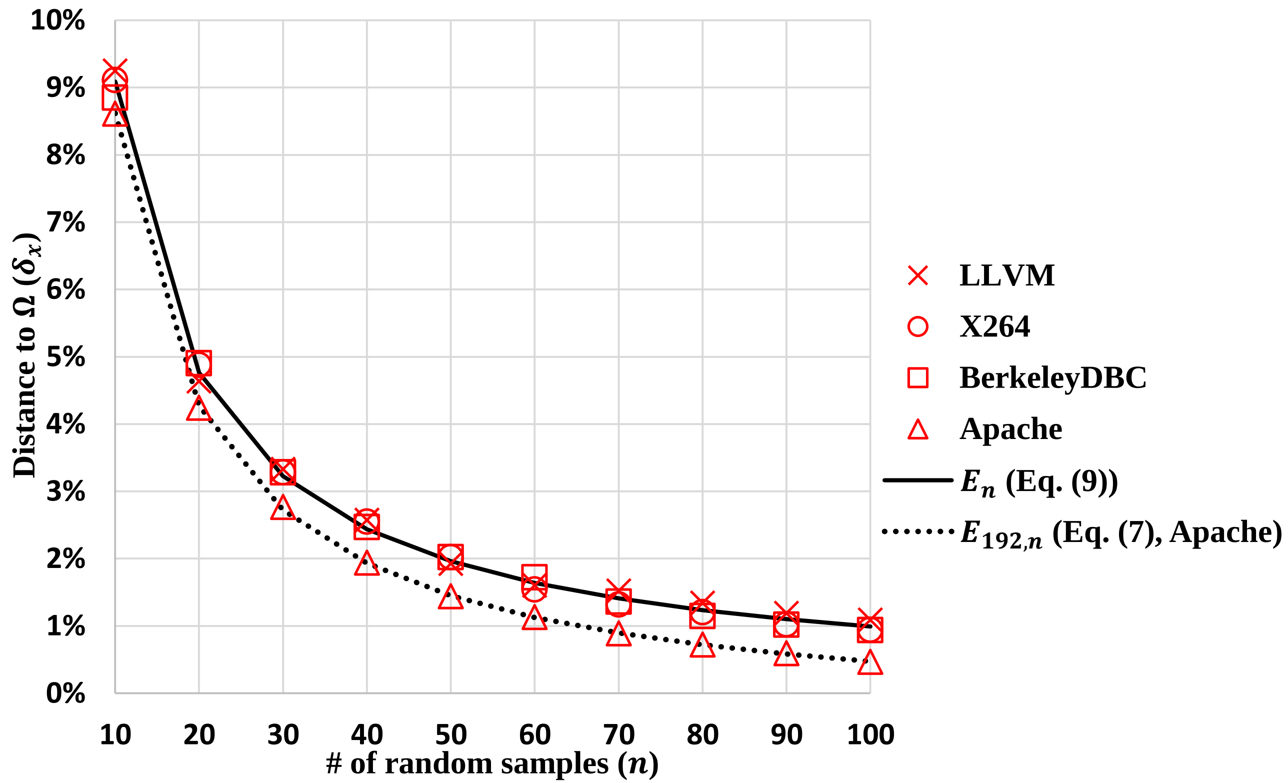
- Use ground truth data from Siegmund et al. (http://fosd.de/SPLConqueror)

SPL	Type	# features	# configs	Performance
LLVM	Compiler infrastructure	11	1024	Test suite compilation time
BerkeleyDBC	Database system	18	2560	Benchmark response time
X264	Video encoder	16	1152	Video encoding time
Apache	Web server	9	192	Maximum server load

- Measured accuracy of search:
  - $\boldsymbol{\delta_x}$: % of configurations better than the best config found so far
  - $\boldsymbol{\delta_y}$: % performance difference to Ω
- Averaged from 100 searches per different conditions
- Full result is available in Section 5 of the paper

δ_x: Theory vs. Actual

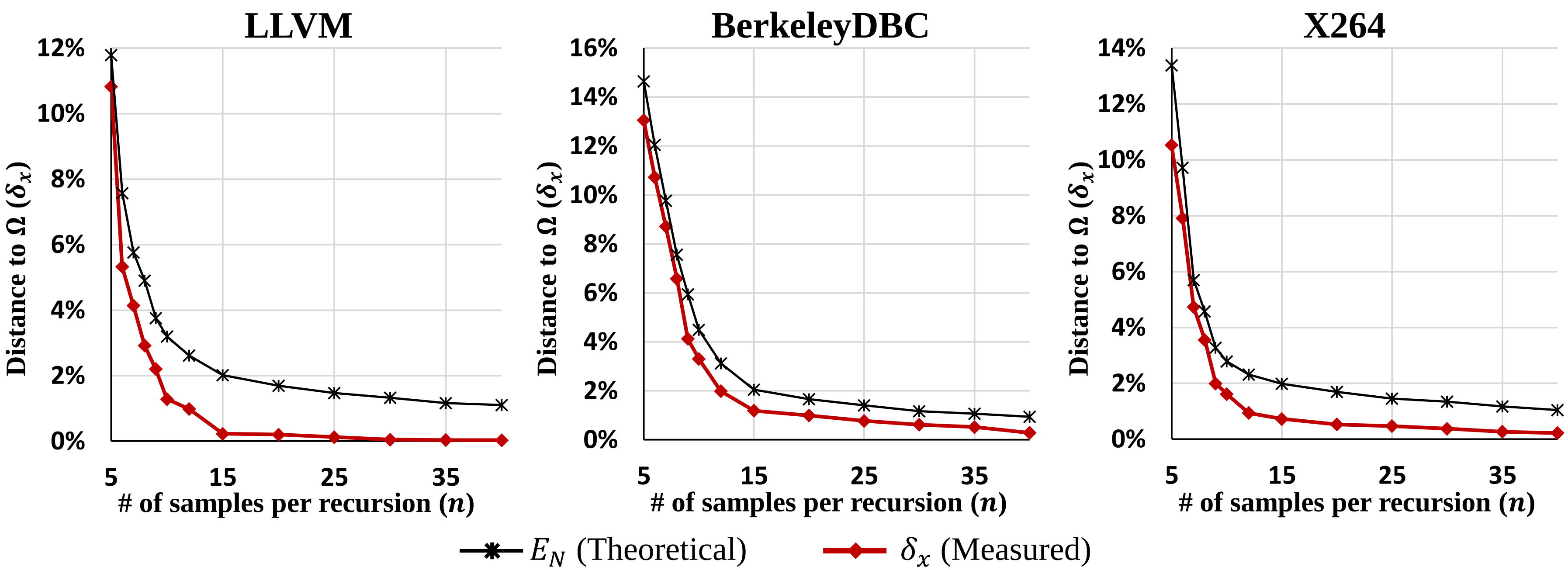
- δ_x from randomly sampling different # of samples
  vs. theoretical value derived using same # of samples



Theory matches observations

δ_x : SRS vs. Random Sampling (non-recursive)

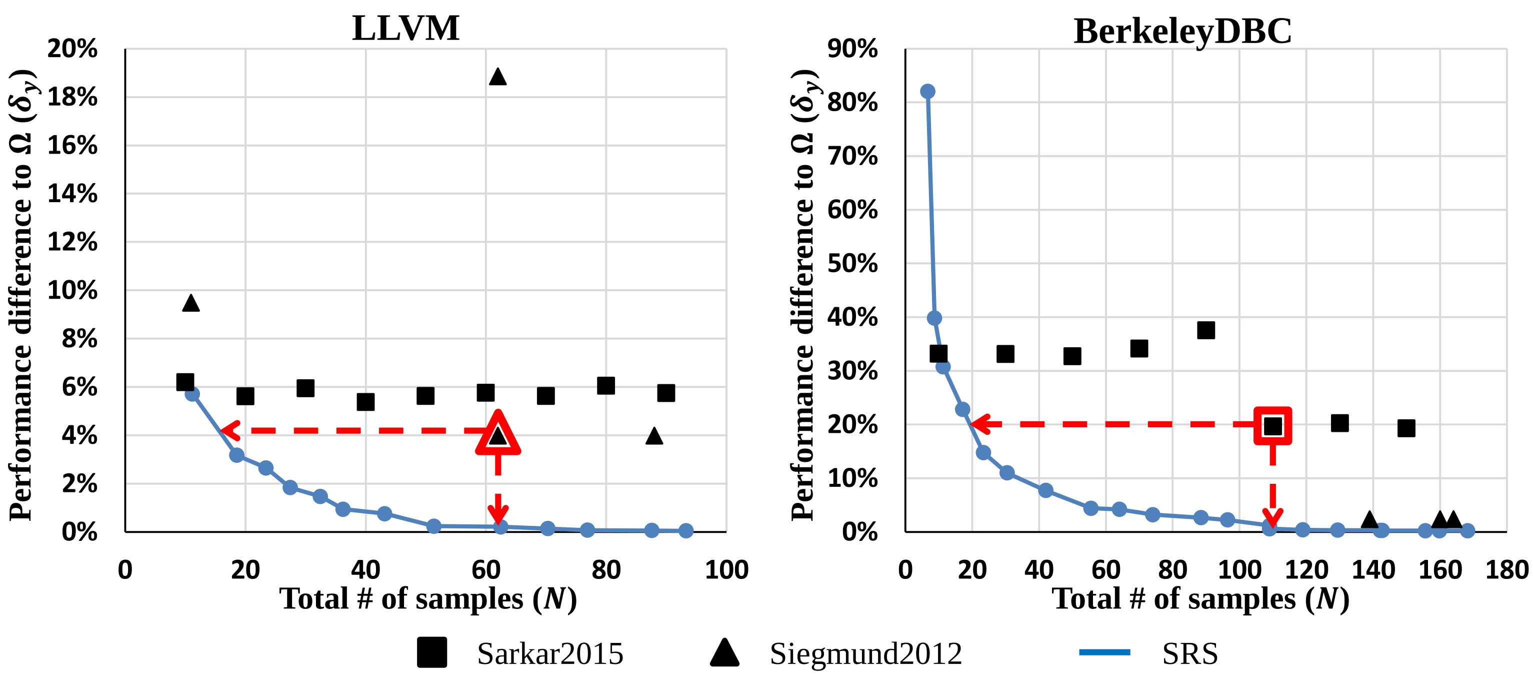
- δ_x of SRS over different # of samples per recursion vs. theoretical δ_x of random sampling with same total # of samples (N)



SRS is more efficient than random sampling alone

δ_y: SRS vs. Performance Models (1)

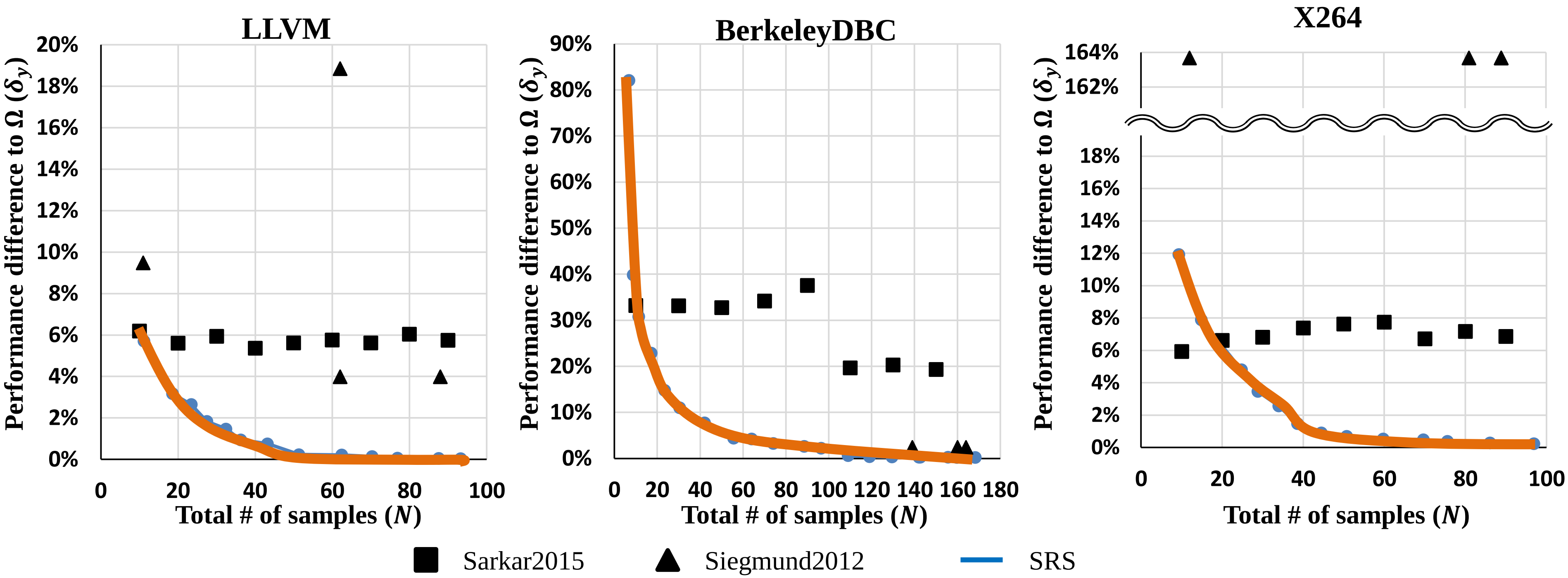
- δ_y over total # of samples in SRS vs.
  δ_y of perf. models assuming an ideal optimizer (always finds best predicted config)



SRS needs many fewer samples for same accuracy, and yields much better accuracy for same number of samples

δ_y: SRS vs. Performance Models (2)

- δ_y over total # of samples in SRS vs.
  δ_y of performance models assuming an ideal optimizer

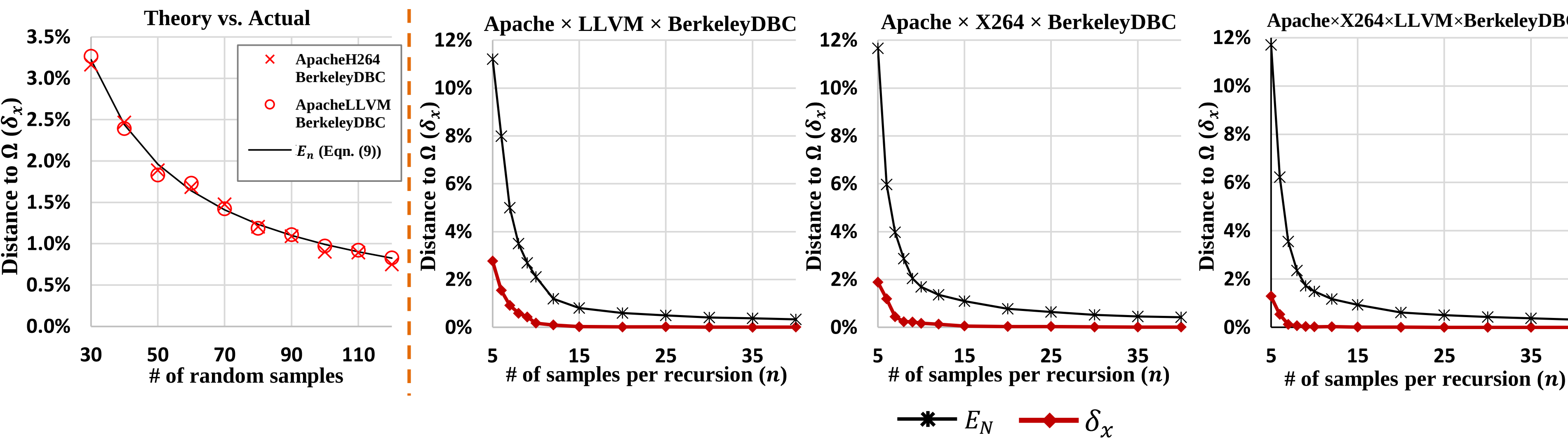


In SRS, more samples yields better accuracy

δ_x: Scalability of Searching

- Combine SPLs to simulate larger configuration spaces
- Measure δ_x for SRS and non-recursive searching

Combined Systems	# of Features	# of Confgs.
Apache × LLVM × BerkeleyDBC	38	503,316,480
Apache × X264 × BerkeleyDBC	51	566,231,040
LLVM × Apache × X264 × BerkeleyDBC	62	579,820,584,960



Accuracy is independent of the size of the configuration space

Conclusions and Contributions

1. True random sampling of configuration spaces

2. Guaranteed tight statistical bounds on finding good configurations

3. Can recursively search through configuration space for more efficient searching

4. Scalable search method - accuracy independent of the configuration space size

Thank You !

Finding Near-Optimal Configurations in Product Lines by Random Sampling

Jeho Oh, Don Batory, Margaret Myers, Norbert Siegmund

Supplemental Slides from Now On

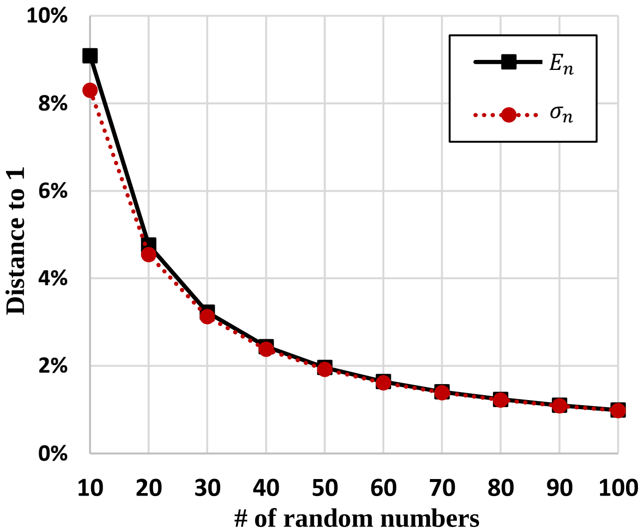
Statistics of Random Sampling

- n random numbers over unit range $[0,1]$, x is the number closest to 1
  Analyze the distance between x and 1, $(1-x)$

- C.D.F. of x over n:
  $p_n(X \leq x) = \int_0^x n \cdot x^{n-1} \cdot dx = x^n$

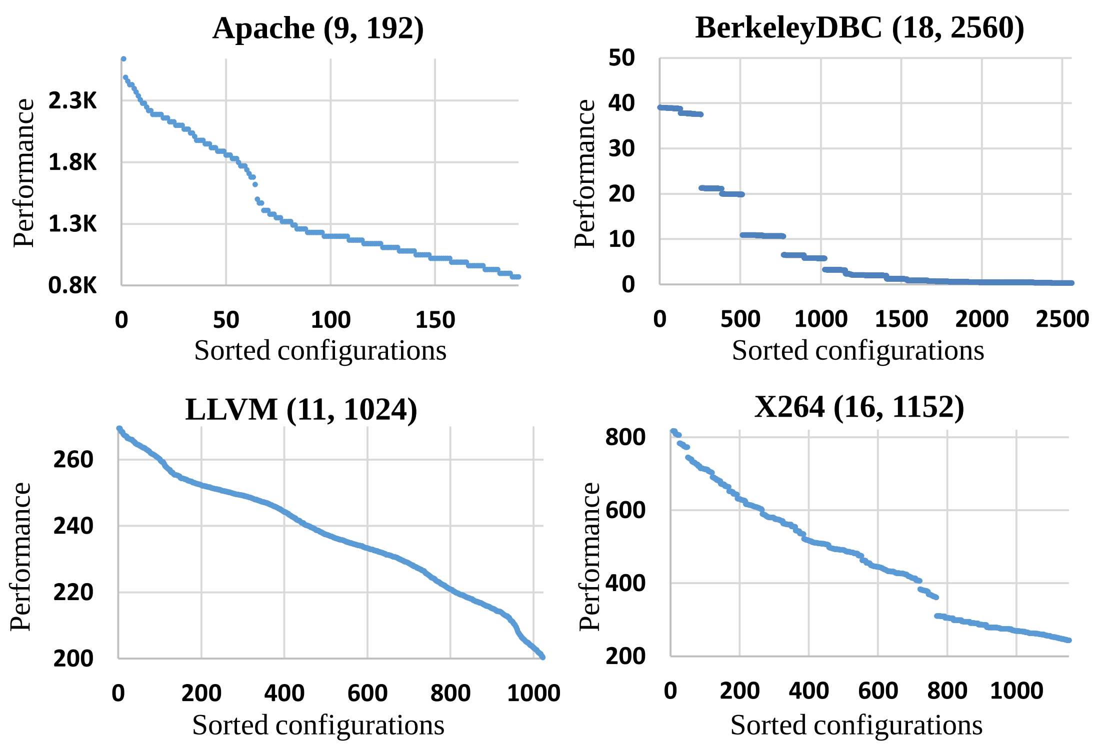
- Average distance from x to 1:
  $E_n = \int_0^1 (1-x) \cdot n \cdot x^{n-1} \cdot dx = \frac{\mathbf{1}}{\boldsymbol{n}\mathbf{+1}}$

- Standard deviation:
  $\bar{E}_n = \int_0^1 (1-x)^2 \cdot n \cdot x^{n-1} \cdot dx = \frac{1}{(n+1)(n+2)}$
  $\sigma_n = \sqrt{\bar{E}_n - E_n^2} = \sqrt{\frac{\mathbf{1}}{(\boldsymbol{n}\mathbf{+1})(\boldsymbol{n}\mathbf{+2})} - \frac{\mathbf{1}}{(\boldsymbol{n}\mathbf{+1})^2}}$



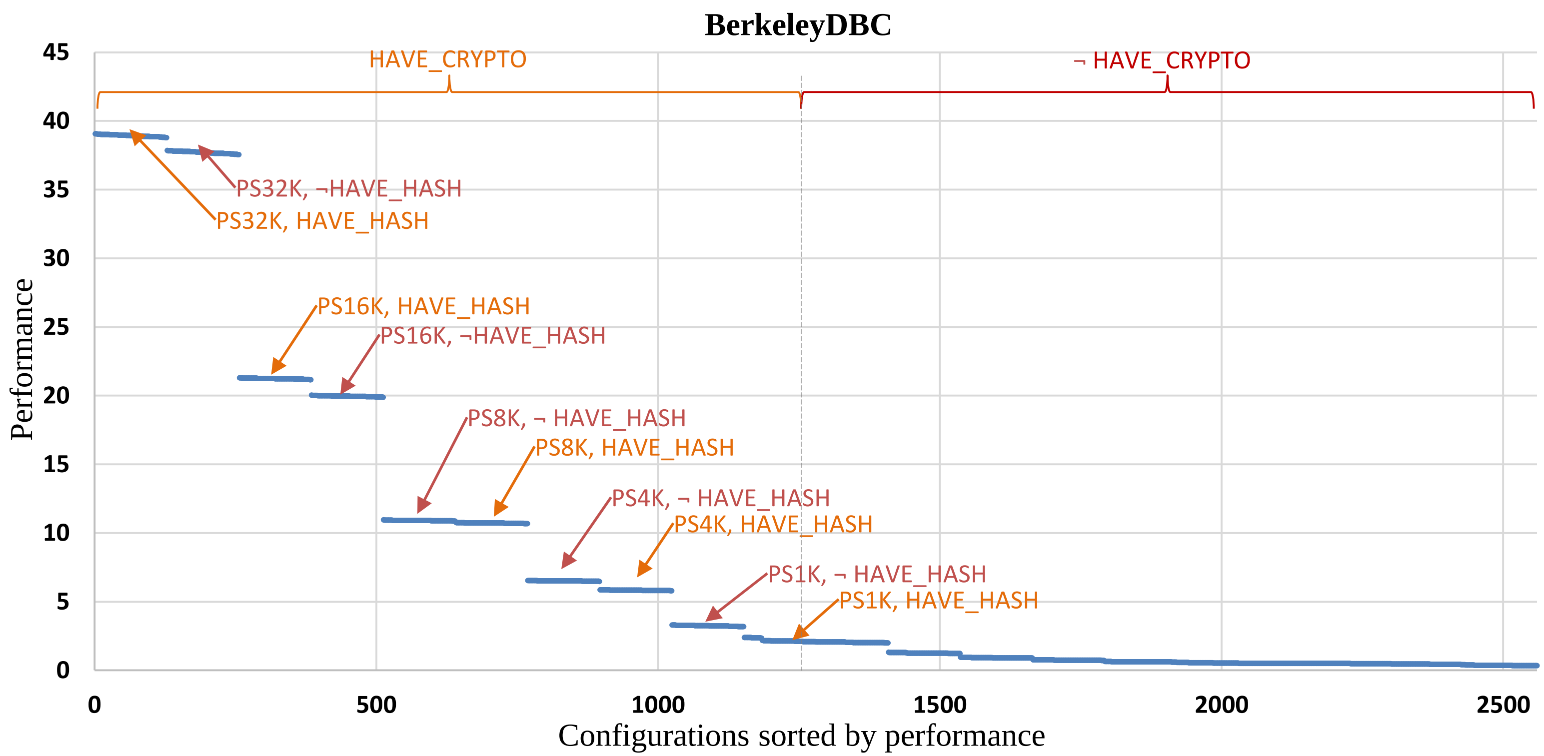
PCS Graphs of Real Systems

- All configuration measurements sorted by performance (descending order)



*** System Name (# of features, # of configs)**

Statistical Recursive Searching (SRS)



Use samples to recursively reduce the config space to focus

Finding Features to Recurse

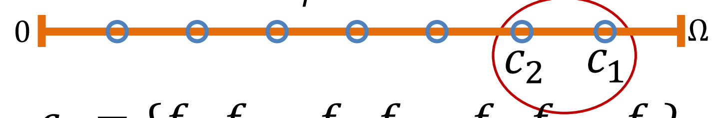
1. Random sample from config space
2. From best 2 configs, get common decisions d
3. For each common decision d, measure:

   δ_d = Avg. perf. of samples with d

   $\delta_{\neg d}$ = Avg. perf. of samples without d

   $\Delta_d = \delta_d - \delta_{\neg d}$
4. If a Δ_d indicates performance improvement, perform Welch's T-test to evaluate its certainty
5. Use features certain to improve performance to constrain the configuration space to recurse

$\phi = FM$



$c_2 = \{f_1, f_2, \neg f_3, f_4, \neg f_5, f_7, \neg f_8\}$

$c_1 = \{f_1, f_2, f_3, \neg f_4, f_5, f_7, \neg f_8\}$

$d = \{f_1, f_2, f_7, \neg f_8\}$

Common Decisions	f_1	f_2	f_7	$\neg f_8$
Δ_d improves perf.	Yes	No	No	Yes
Welch's T-test	Pass	-	-	Fail

Decisions to recurse: f_1

$\phi = FM \wedge f_1$

The University of Texas at Austin