Finding Near-Optimal Configurations in Product Lines by Random Sampling

Jeho Oh
Don Batory
Margaret Myers
Norbert Siegmund
Quickly find SPL configurations with near-optimal performance for a given workload

- 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

I want a hybrid car with laser headlight, but cheap and light as possible. Now 273/275 options left to decide...
Big Picture

Performance Model Approach

- sample configurations
- learn performance model
- use optimizer

Our Approach

- feature model
- randomly sample on configuration space for near-optimal configurations
- near-optimal performing configuration
- 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 $n$: $p_n(X \leq x) = \int_0^x n \cdot x^{n-1} \cdot dx = x^n$

- Average distance from $x$ to $1$: $E_n = \int_0^1 (1 - x) \cdot n \cdot x^{n-1} \cdot dx = \frac{1}{n+1}$
  \[ x = 0.80 \]
  
  $E_4 = 0.20$

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

(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 \([ 1, \text{total # of configs} ]\)
  - Closeness to 1 ▶ Closeness to the best configuration, \(\Omega\)

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

\[ E_7 = \frac{\Omega - x}{1024} = 0.125 \]

Monotonic graph:
- Closer to \(\Omega\) (x-axis), better perf. (y-axis)
- Sorted config space

LLVM (Compiler infrastructure, 11 features, 1024 configs)

\(\Omega \leftarrow \text{Best performing config}\)
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
Predicting Performance via Automated Feature-Interaction Detection

Norbert Siegmund,* Sergey S. Kolesnikov,† Christian Kästner,‡ Sven Apel,§
Don Babory,† 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 Method

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

<table>
<thead>
<tr>
<th>SPL</th>
<th>Type</th>
<th># features</th>
<th># configs</th>
<th>Performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>LLVM</td>
<td>Compiler infrastructure</td>
<td>11</td>
<td>1024</td>
<td>Test suite compilation time</td>
</tr>
<tr>
<td>BerkeleyDBC</td>
<td>Database system</td>
<td>18</td>
<td>2560</td>
<td>Benchmark response time</td>
</tr>
<tr>
<td>X264</td>
<td>Video encoder</td>
<td>16</td>
<td>1152</td>
<td>Video encoding time</td>
</tr>
<tr>
<td>Apache</td>
<td>Web server</td>
<td>9</td>
<td>192</td>
<td>Maximum server load</td>
</tr>
</tbody>
</table>

- Measured accuracy of search:
  - $\delta_x : %$ of configurations better than the best config found so far
  - $\delta_y : %$ performance difference to $\Omega$

- Averaged from 100 searches per different conditions

- Full result is available in Section 5 of the paper
$\delta_x$: Theory vs. Actual

- $\delta_x$ from randomly sampling different # of samples vs. theoretical value derived using same # of samples

Theory matches observations
\( \delta_x : \text{SRS vs. Random Sampling} \)  
(\text{non-recursive})

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

**SRS is more efficient than random sampling alone**
\( \delta_y : SRS \text{ vs. Performance Models (1)} \)

- \( \delta_y \) over total \# of samples in SRS vs. \( \delta_y \) 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
\( \delta_y : \text{SRS vs. Performance Models (2)} \)

- \( \delta_y \) over total \# of samples in SRS vs. \( \delta_y \) of performance models assuming an ideal optimizer

**In SRS, more samples yields better accuracy**
δₓ: Scalability of Searching

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

### Combined Systems

<table>
<thead>
<tr>
<th>Combined Systems</th>
<th># of Features</th>
<th># of Configs.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache × LLVM × BerkeleyDBC</td>
<td>38</td>
<td>503,316,480</td>
</tr>
<tr>
<td>Apache × X264 × BerkeleyDBC</td>
<td>51</td>
<td>566,231,040</td>
</tr>
<tr>
<td>LLVM × Apache × X264 × BerkeleyDBC</td>
<td>62</td>
<td>579,820,584,960</td>
</tr>
</tbody>
</table>

Distance to Ω(δₓ): # of samples per recursion (n)

- **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:
  \[ E_n = \int_0^1 (1 - x) \cdot n \cdot x^{n-1} \cdot dx = \frac{1}{n+1} \]

- Standard deviation:
  \[ \bar{E}_n = \int_0^1 (1 - x)^2 \cdot n \cdot x^{n-1} \cdot dx = \frac{1}{(n+1)(n+2)} \]
  \[ \sigma_n = \sqrt{E_n - E_n^2} = \sqrt{\frac{1}{(n+1)(n+2)} - \frac{1}{(n+1)^2}} \]
PCS Graphs of Real Systems

- All configuration measurements sorted by performance (descending order)

* System Name (# of features, # of configs)
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:
   $\delta_d = \text{Avg. perf. of samples with } d$
   $\delta_{\neg d} = \text{Avg. perf. of samples without } d$
   $\Delta_d = \delta_d - \delta_{\neg d}$
4. If a $\Delta_d$ 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