实验设计

郭健美

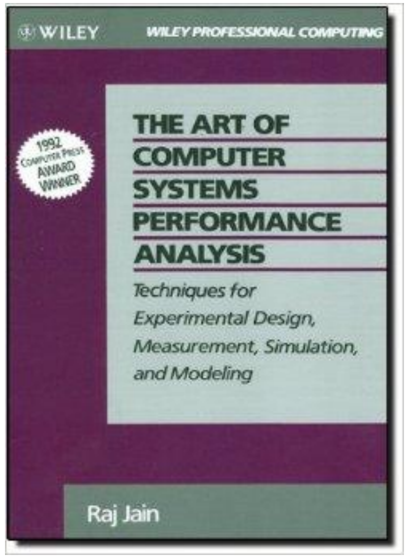
2022年秋

本课内容

- 实验设计 Design of Experiments
- 自动调优 Autotuning

Introduction to Experimental Design

Prof. Raj Jain



https://www.cse.wustl.edu/~jain/cse567-08/ftp/k_16ied.pdf

Experimental Design and Analysis

How to:

- Design a proper set of experiments for measurement or simulation.
- Develop a model that best describes the data obtained.
- Estimate the contribution of each alternative to the performance.
- Isolate the measurement errors.
- Estimate confidence intervals for model parameters.
- Check if the alternatives are significantly different.
- Check if the model is adequate.

Example

Personal workstation design

1. Processor: 68000, Z80, or 8086.
2. Memory size: 512K, 2M, or 8M bytes
3. Number of Disks: One, two, three, or four
4. Workload: Secretarial, managerial, or scientific.
5. User education: High school, college, or post-graduate level.

Five **Factors** at 3x3x4x3x3 **levels**

Terminology

- **Response Variable**: Outcome.

  E.g., throughput, response time
- **Factors**: Variables that affect the response variable.

  E.g., CPU type, memory size, number of disk drives, workload used, and user's educational level.

  Also called predictor variables or predictors.
- **Levels**: The values that a factor can assume, E.g., the CPU type has three levels: 68000, 8080, or Z80.

  # of disk drives has four levels.

  Also called **treatment**.
- **Primary Factors**: The factors whose effects need to be quantified.

  E.g., CPU type, memory size only, and number of disk drives.

Terminology (Cont)

- **Secondary Factors**: Factors whose impact need not be quantified.
  E.g., the workloads.
- **Replication**: Repetition of all or some experiments.
- **Design**: The number of experiments, the factor level and number of replications for each experiment.
  E.g., Full Factorial Design with 5 replications: $3\times 3 \times 4 \times 3 \times 3$ or 324 experiments, each repeated five times.
- **Experimental Unit**: Any entity that is used for experiments.
  E.g., users. Generally, no interest in comparing the units.
- Goal - minimize the impact of variation among the units.

Terminology (Cont)

- **Interaction** $\Rightarrow$ Effect of one factor depends upon the level of the other.

Table 1: Noninteracting Factors

	A_1	A_2
B_1	3	5
B_2	6	8

Table 2: Interacting Factors

	A_1	A_2
B_1	3	5
B_2	6	9

Common Mistakes in Experimentation

- The variation due to experimental error is ignored.
- Important parameters are not controlled.
- Effects of different factors are not isolated
- Simple one-factor-at-a-time designs are used
- Interactions are ignored
- Too many experiments are conducted.

  Better: two phases.

Types of Experimental Designs

- **Simple Designs**: Vary one factor at a time

$$\# \text{ of Experiments} = 1 + \sum_{i=1}^{k} (n_i - 1)$$

  - Not statistically efficient.
  - Wrong conclusions if the factors have interaction.
  - Not recommended.
- **Full Factorial Design**: All combinations.

$$\# \text{ of Experiments} = \prod_{i=1}^{k} n_i$$

  - Can find the effect of all factors.
  - Too much time and money.
  - May try 2^k design first.

Grid Search

Types of Experimental Designs (Cont)

- Fractional Factorial Designs: Less than Full Factorial
  - Save time and expense.
  - Less information.
  - May not get all interactions.
  - Not a problem if negligible interactions

A Sample Fractional Factorial Design

- Workstation Design:
  (3 CPUs)(3 Memory levels)(3 workloads)(3 ed levels)
  = 81 experiments

Experiment Number	CPU	Memory Level	Workload Type	Educational Level
1	68000	512K	Managerial	High School
2	68000	2M	Scientific	Post-graduate
3	68000	8M	Secretarial	College
4	Z80	512K	Scientific	College
5	Z80	2M	Secretarial	High School
6	Z80	8M	Managerial	Post-graduate
7	8086	512K	Secretarial	Post-graduate
8	8086	2M	Managerial	College
9	8086	8M	Scientific	High School

Hyperparameter Optimization

- Grid Search
- Random Search
- Bayesian Optimization
- Gradient-based Optimization
- Evolutionary Optimization
- …

https://en.wikipedia.org/wiki/Hyperparameter_optimization

Configure software to tailor functional behavior

```
x264 --quiet
     --no-progress
     --no-asm
     --rc-lookahead 60
     --ref 9
     -o trailer_480p24.x264
     trailer_2k_480p24.y4m
```

[Guo, et al. Variability-aware performance prediction: A statistical learning approach. ASE 2013.]

Configure software to tailor functional behavior

a command-line tool to encode a video stream

```
x264 --quiet
     --no-progress
     --no-asm
     --rc-lookahead 60
     --ref 9
     -o trailer_480p24.x264
     trailer_2k_480p24.y4m
```

feature1
feature2
feature3
feature4
feature5

configuration1
(variant1)

output stream
input stream

[Guo, et al. Variability-aware performance prediction: A statistical learning approach. ASE 2013.]

Feature-wise measurement?

configuration1

```
x264 --quiet
     --no-progress
     --no-asm
     --rc-lookahead 60
     --ref 9
     -o trailer_480p24.x264
     trailer_2k_480p24.y4m
```

324 seconds

configuration2

```
x264
     --no-progress
     --no-asm
     --rc-lookahead 60
     --ref 9
     -o trailer_480p24.x264
     trailer_2k_480p24.y4m
```

551 seconds

P(quiet)

= 551 – 324

= 227 seconds

[Guo, et al. Variability-aware performance prediction: A statistical learning approach. ASE 2013.]

Feature-wise measurement?

configuration1

```
x264 --quiet
     --no-progress
     --no-asm
     --rc-lookahead 60
     --ref 9
     -o trailer_480p24.x264
     trailer_2k_480p24.y4m
```

324 seconds

configuration2

```
x264
     --no-progress
     --no-asm
     --rc-lookahead 60
     --ref 9
     -o trailer_480p24.x264
     trailer_2k_480p24.y4m
```

551 seconds

P(**quiet**)
= 551 – 324
= **227** seconds

configuration3

```
x264 --quiet
     --no-progress

     --rc-lookahead 60
     --ref 9
     -o trailer_480p24.x264
     trailer_2k_480p24.y4m
```

487 seconds

configuration4

```
x264
     --no-progress

     --rc-lookahead 60
     --ref 9
     -o trailer_480p24.x264
     trailer_2k_480p24.y4m
```

661 seconds

P'(**quiet**)
= 661– 487
= **174** seconds

[Guo, et al. Variability-aware performance prediction: A statistical learning approach. ASE 2013.]

Feature-wise measurement?

configuration1

```
x264 --quiet
     --no-progress
     --no-asm
     --rc-lookahead 60
     --ref 9
     -o trailer_480p24.x264
     trailer_2k_480p24.y4m
```

324 seconds

configuration2

```
x264
     --no-progress
     --no-asm
     --rc-lookahead 60
     --ref 9
     -o trailer_480p24.x264
     trailer_2k_480p24.y4m
```

551 seconds

configuration3

```
x264 --quiet
     --no-progress

     --rc-lookahead 60
     --ref 9
     -o trailer_480p24.x264
     trailer_2k_480p24.y4m
```

487 seconds

configuration4

```
x264
     --no-progress

     --rc-lookahead 60
     --ref 9
     -o trailer_480p24.x264
     trailer_2k_480p24.y4m
```

661 seconds

P(quiet)
= 551 – 324
= **227** seconds

Feature interactions

P'(quiet)
= 661– 487
= **174** seconds

One-factor-at-a-time method

[Guo, et al. Variability-aware performance prediction: A statistical learning approach. ASE 2013.]

Conf.	Features																Perf. (s)
c_i	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}	x_{13}	x_{14}	x_{15}	x_{16}	p_i
c_1	1	1	0	1	1	1	1	0	1	0	0	1	1	0	0	1	651
c_2	1	1	1	1	1	1	0	1	1	1	0	0	1	0	1	0	536
c_3	1	1	1	1	0	0	0	0	1	1	0	0	1	0	0	1	581
c_4	1	0	0	0	0	0	1	0	1	1	0	0	1	0	1	0	381
c_5	1	1	0	1	0	0	0	1	1	1	0	0	1	0	1	0	424
c_6	1	1	0	0	1	0	1	1	1	1	0	0	1	0	0	1	615
c_7	1	0	1	0	1	1	1	0	1	1	0	0	1	0	1	0	477
c_8	1	0	1	0	0	0	0	1	1	0	0	1	1	1	0	0	263
c_9	1	0	0	0	0	0	1	1	1	0	0	1	1	1	0	0	272
c_{10}	1	1	1	1	0	0	0	1	1	0	0	1	1	1	0	0	247
c_{11}	1	0	0	0	0	0	0	0	1	0	1	0	1	0	0	1	612
c_{12}	1	0	1	1	1	0	0	0	1	0	0	1	1	0	1	0	510
c_{13}	1	1	1	1	0	1	1	0	1	0	1	0	1	0	0	1	555
c_{14}	1	1	0	0	1	0	1	1	1	0	0	1	1	1	0	0	264
c_{15}	1	0	1	0	0	1	1	1	1	0	0	1	1	0	0	1	576
c_{16}	1	0	1	0	1	0	1	1	1	0	1	0	1	1	0	0	268

[Guo, et al. Variability-aware performance prediction: A statistical learning approach. ASE 2013.]

A non-linear regression problem

Predictors

Response

feature1: “quiet”

Boolean variables

Numeric variable

A small random sample

Conf.	Features																Perf. (s)
c_i	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}	x_{13}	x_{14}	x_{15}	x_{16}	p_i
c_1	1	1	0	1	1	1	1	0	1	0	0	1	1	0	0	1	651
c_2	1	1	1	1	1	1	0	1	1	1	0	0	1	0	1	0	536
c_3	1	1	1	1	0	0	0	0	1	1	0	0	1	0	0	1	581
c_4	1	0	0	0	0	0	1	0	1	1	0	0	1	0	1	0	381
c_5	1	1	0	1	0	0	0	1	1	1	0	0	1	0	1	0	424
c_6	1	1	0	0	1	0	1	1	1	1	0	0	1	0	0	1	615
c_7	1	0	1	0	1	1	1	0	1	1	0	0	1	0	1	0	477
c_8	1	0	1	0	0	0	0	1	1	0	0	1	1	1	0	0	263
c_9	1	0	0	0	0	0	1	1	1	0	0	1	1	1	0	0	272
c_{10}	1	1	1	1	0	0	0	1	1	0	0	1	1	1	0	0	247
c_{11}	1	0	0	0	0	0	0	0	1	0	1	0	1	0	0	1	612
c_{12}	1	0	1	1	1	0	0	0	1	0	0	1	1	0	1	0	510
c_{13}	1	1	1	1	0	1	1	0	1	0	1	0	1	0	0	1	555
c_{14}	1	1	0	0	1	0	1	1	1	0	0	1	1	1	0	0	264
c_{15}	1	0	1	0	0	1	1	1	1	0	0	1	1	0	0	1	576
c_{16}	1	0	1	0	1	0	1	1	1	0	1	0	1	1	0	0	268
c	1	1	0	1	1	0	0	0	0	1	0	0	1	0	1	0	?

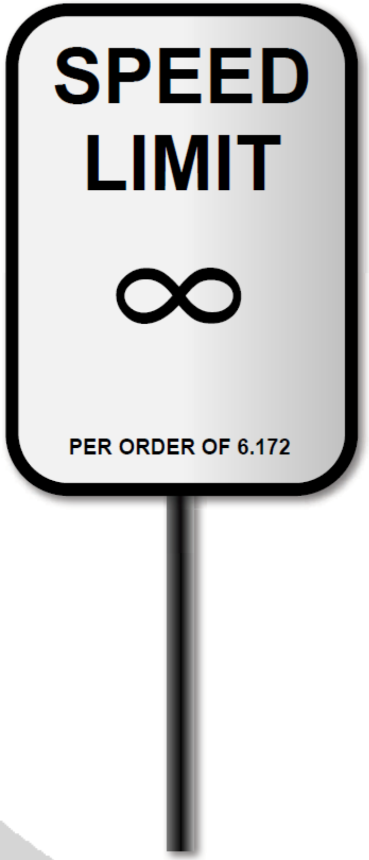
Random Search

[Guo, et al. Variability-aware performance prediction: A statistical learning approach. ASE 2013.]

本课内容

- 实验设计 Design of Experiments
- 自动调优 Autotuning

6.172
Performance
Engineering of
Software
Systems



LECTURE 18
Domain Specific Languages and Autotuning

Saman Amarasinghe

https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-172-performance-engineering-of-software-systems-fall-2018/lecture-slides/MIT6_172F18_lec18.pdf

OpenTuner

- Performance Engineering is all about finding the right:
  - block size in matrix multiply (voodoo parameters)
  - strategy in the dynamic memory allocation project
  - flags in calling GCC to optimize the program
  - schedule in Halide & TVM
  - schedule in GraphIt

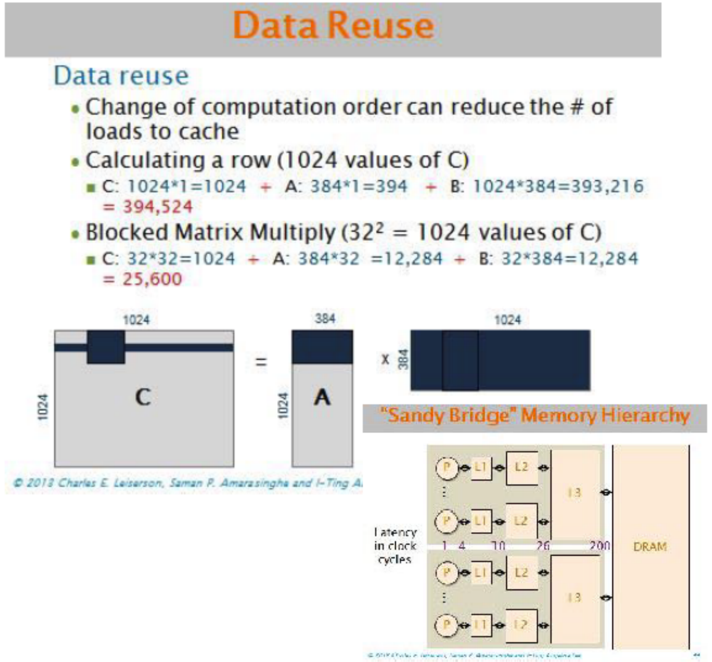
How to find the right value

1. Model-Based
2. Heuristic-Based
3. Exhaustive Search
4. Autotuned (OpenTuner)

1. Model Based Solutions

Come-up with a comprehensive model

- In this case, a model for the memory system and data reuse



Pros:

- Can explain exactly why we chose a given tile size
- "Optimal"

Cons:

- Hard to build models
- Cannot model everything
- Our model may miss an important component

2. Heuristic Based Solutions

"A rule of thumb" that works most of the time

- In this case, small two-to-the-power tile sizes works most of the time
- Hard-code them (eg: S = 8)

Pros

- Simple and easy to do
- Works most of the time

Cons

- Simplistic
- However, always suboptimal performance
- In some cases may be really bad

3. Exhaustive Search

Empirically evaluate all the possible values

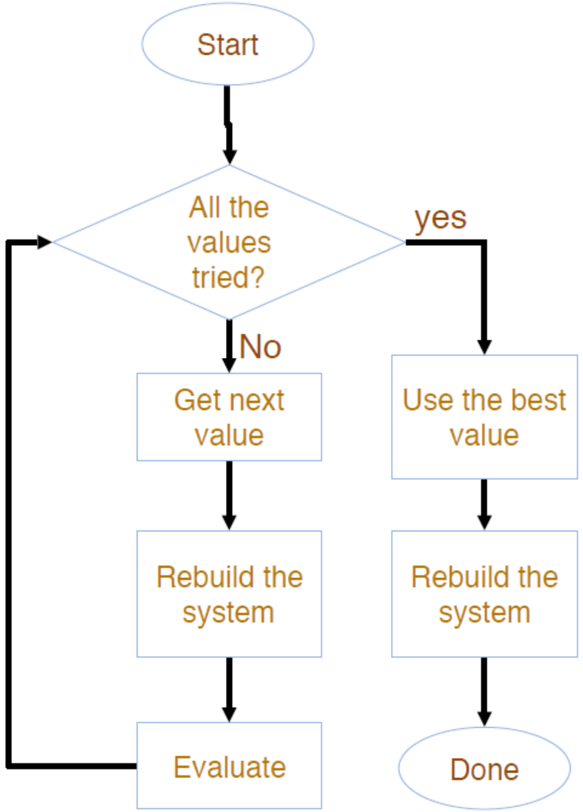
- All possible integers for S

Pros:

- Will find the “optimal” value

Cons:

- Only for the inputs evaluated
- Can take a looooong time!
  - Prune the search space
    - Only integers that are powers-of-2 from vector register size to the cache size?

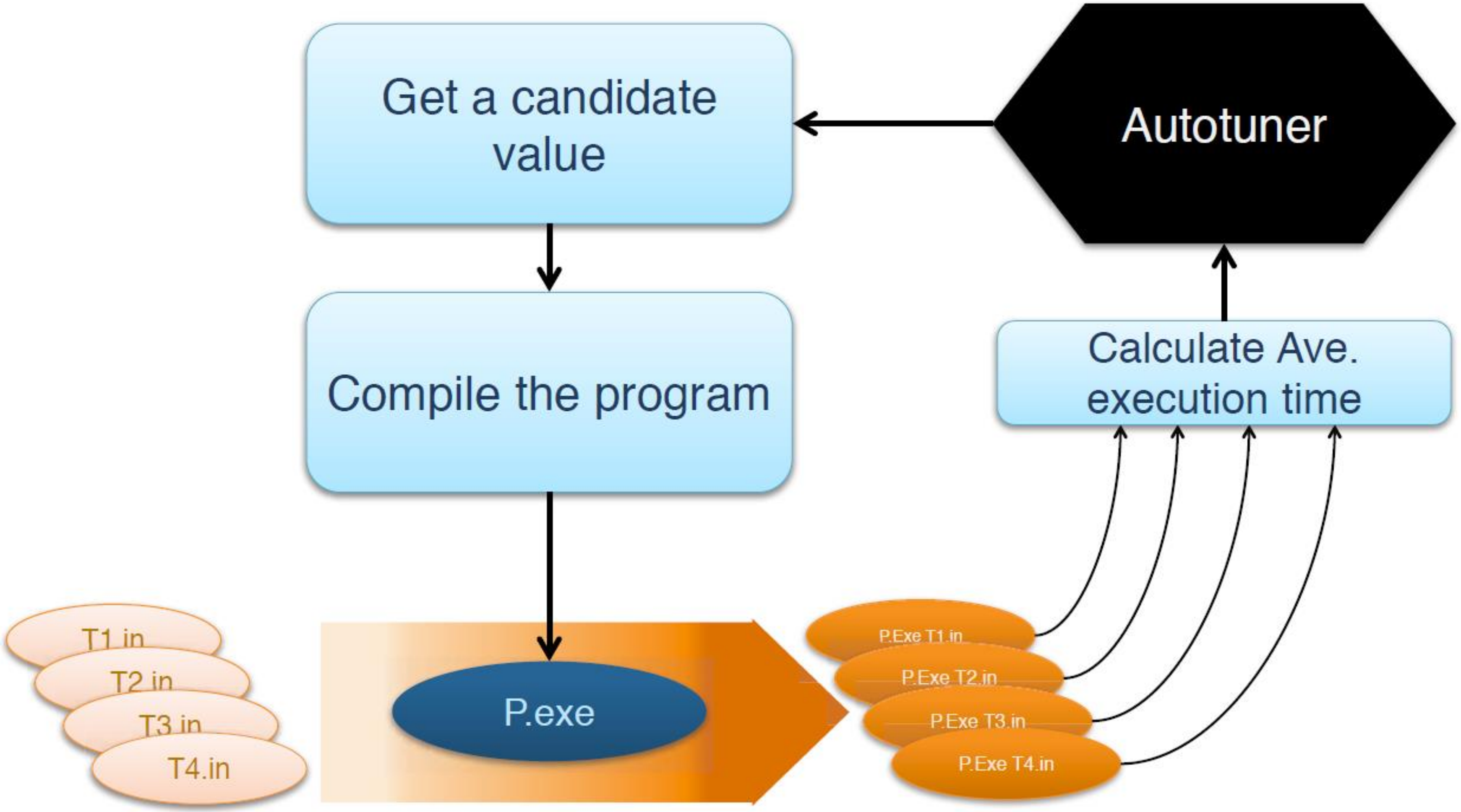


67

4. Autotuning Based Solutions

① Define a space of acceptable values
② Choose a value at random from that space
③ Evaluate the performance given that value
④ If satisfied with the performance or time limit exceeded, then finish
⑤ Choose a new value from the feedback
⑥ Goto 3

Autotuning A Program



Summary

- Design of experiments and autotuning help **find the optimal or near-optimal configuration efficiently** for performance optimization.