DM811
HEURISTICS AND LOCAL SEARCH ALGORITHMS FOR COMBINATORIAL OPTIMZATION

Lecture 14

Experimental Analysis

Marco Chiarandini

slides partly based on
McGeoch's lectures
at the summer school in Lipari, 2008

Outline

1. Developing an Experimental Environment

2. Program Optimization

Outline

1. Developing an Experimental Environment

2. Program Optimization

Building an experimental environment

You will need these files for your project:

- The code that implements the algorithm. (Several versions.)
- The input:
  Instances for the algorithm, parameters to guide the algorithm, instructions for reporting.
- The output:
  The result, the performance measurements, perhaps animation data.
- The journal:
  A record of your experiments and findings.
- Analysis tools:
  statistics, data analysis, visualization, report.

How will you organize them? How will you make them work together?

Example

Input and reporting controls on command line

```
mssh -i instance.in -o output.sol -l run.log > data.out
```

Output on stdout self-describing

```
#stat instance.in 30 90
seed: 9897868
Parameter1: 30
Parameter2: A
Read instance. Time: 0.016001
begin try 1
best 0 col 22 time 0.004000 iter 0 par_iter 0
best 3 col 21 time 0.004000 iter 0 par_iter 0
best 1 col 21 time 0.004000 iter 0 par_iter 0
best 0 col 21 time 0.004000 iter 1 par_iter 1
best 6 col 20 time 0.004000 iter 3 par_iter 1
best 4 col 20 time 0.004000 iter 4 par_iter 2
best 2 col 20 time 0.004000 iter 6 par_iter 4
exit iter 7 time 1.000062
end try 1
```

Example

If one program that implements many heuristics

- re-compile for new versions but take old versions with a journal in archive.
- use command line parameters to choose among the heuristics
- C: `getopt`, `getopt_long`, opag (option parser generator)
  Java: package `org.apache.commons.cli`

  ```
  mssh -i instance.in -o output.sol -l run.log --solver 2-opt > data.out
  ```
- use identifying labels in naming file outputs

Example

- So far: one run per instance. Multiple runs, multiple instances and multiple algorithms ➨ unix script (eg, bash one line program, perl, php)
- Data analysis: Select line identifier from output file, combine, send to grasp scripts.
  Example

  ```
  grep #stat | cut -f 2 -d " "
  ```
- Data in form of matrix or data frame goes directly into R imported by `read.table()`, untouched by human hands

  ```
  alg instance     run sol time
  ROS le450_15a.col 3 21 0.00267
  ROS le450_15b.col 3 21 0
  ROS le450_15d.col 3 31 0.00267
  RLF le450_15a.col 3 17 0.00533
  RLF le450_15b.col 3 16 0.008
  ...
  ```
- Visualization: Select animation commands from output file, send to animation tool.

Outline

1. Developing an Experimental Environment

2. Program Optimization

Program Profiling

- Check the correctness of your solutions many times
- Plot the development of
  - best visited solution quality
  - current solution quality

  over time and compare with other features of the algorithm.

Code Optimization

- Profile time consumption per program components
  - under Linux: `gprof`
    1. add flag `-pg` in compilation
    2. run the program
    3. `gprof gmon.out > a.txt`
  - Java VM profilers (plugin for eclipse)

– Can't control / isolate components of interest.

– All profilers will affect runtime.

– Library function calls not shown.

– Timing is not so accurate (based on interval counts), especially for quick functions. Function times rarely add up to whole.

– Doesn't work with multithreaded, multicore programs.

Where do speedups come from?

Where can maximum speedup be achieved?
How much speedup should you expect?

Code Tuning

- Caution: proceed carefully! Let the optimizing compiler do its work!
- Expression Rules: Recode for smaller instruction counts.
- Loop and procedure rules: Recode to avoid loop or procedure call overhead.
- Hidden costs of high-level languages
- String comparisons in C: proportional to length of the string, not constant
- Object construction / de-allocation: very expensive
- Matrix access: row-major order $\neq$ column-major order
- Exploit algebraic identities

Where Speedups Come From?

McGeoch reports conventional wisdom, based on studies in the literature.

- Concurrency is tricky: bad -7x to good 500x
- Classic algorithms: to 1trillion and beyond
- Data-aware: up to 100x
- Memory-aware: up to 20x
- Algorithm tricks: up to 200x
- Code tuning: up to 10x
- Change platforms: up to 10x

Relevant Literature

Bentley, **Writing Efficient Programs; Programming Pearls** (Chapter 8 Code Tuning)

Kernighan and Pike, **The Practice of Programming** (Chapter 7 Performance).
Shirazi, **Java Performance Tuning**, O'Reilly

McCluskey, **Thirty ways to improve the performance of your Java program.** Manuscript and website: `www.glenmcci.com/jperf`

Randal E. Bryant e David R. O'Hallaron: **Computer Systems: A Programmer's Perspective**, Prentice Hall, 2003, (Chapter 5)