DM841

Discrete Optimization

Lecture 9
Working Environment

Marco Chiarandini

Department of Mathematics & Computer Science
University of Southern Denmark

Outline

Outline

Building a Working Environment

What will you need during the project? How will you organize it? How will you make things work together?

- `src/` code that implements the algorithm (likely, several versions)
- `bin/` place where to put your executables
- `data/` input: Instances for the solver, parameters to guide the solver
- `scripts/` code that runs batches of experiments or parses files
- `res/` output: The result, the performance measurements
- `log/` other log files produced by the run of the algorithm
- `r/` analysis tools: statistics, data analysis, visualization
- `doc/` or `tex/` journal/report: A record of your experiments and findings, together with description of the algorithms.
- `Makefile` compiles the sources in `src` and puts the executables in `bin`.
- `README` explains how to compile, test and run the program. Eventually, it explains differences among versions.

$\rightsquigarrow$ organize everything like if you had to reproduce the same results in a few years from now.

Example

Input controls on command line

```
xyz --main::instance ins1.txt --main::output-file log.txt --main::seed 12 > data.log
```

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 a single 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`
  EasyLocal: boost libraries
- use identifying labels in naming file outputs
  Example:
  `c0010.i0002.t0001.s02010.log`

Example

- You will need:
  multiple runs, multiple instances, multiple classes and multiple algorithms.
  Arrange this outside of your program: ➨ unix scripts (eg, bash one line program, perl, python, php)

- Parse outputfiles:
  Example

  ```
  grep #stat | cut -f 2 -d " "
  ```

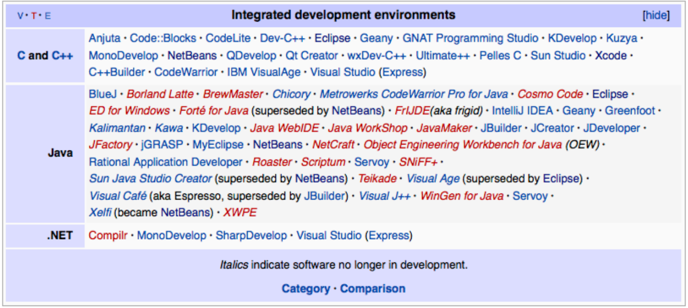
  See http://www.gnu.org/software/coreutils/manual/ for shell tools.

- 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
  ...
  ```

Text Editor

- vim (Vi IMproved) http://www.moolenaar.net/habits.html
- emacs
- Integrated development environment. Choose your favourite: http://en.wikipedia.org/wiki/Integrated_development_environment



V · T · E	Integrated development environments [hide]
C and C++	Anjuta · Code::Blocks · CodeLite · Dev-C++ · Eclipse · Geany · GNAT Programming Studio · KDevelop · Kuzya · MonoDevelop · NetBeans · QDevelop · Qt Creator · wxDev-C++ · Ultimate++ · Pelles C · Sun Studio · Xcode · C++Builder · CodeWarrior · IBM VisualAge · Visual Studio (Express)
Java	BlueJ · *Borland Latte* · *BrewMaster* · *Chicory* · *Metrowerks CodeWarrior Pro for Java* · *Cosmo Code* · Eclipse · *ED for Windows* · *Forté for Java* (superseded by NetBeans) · *FrIJDE(aka frigid)* · IntelliJ IDEA · Geany · Greenfoot · *Kalimantan* · *Kawa* · KDevelop · *Java WebIDE* · *Java WorkShop* · *JavaMaker* · JBuilder · JCreator · JDeveloper · *JFactory* · jGRASP · MyEclipse · NetBeans · *NetCraft* · *Object Engineering Workbench for Java (OEW)* · Rational Application Developer · *Roaster* · *Scriptum* · Servoy · *SNiFF+* · *Sun Java Studio Creator* (superseded by NetBeans) · *Teikade* · *Visual Age* (superseded by Eclipse) · *Visual Café* (aka Espresso, superseded by JBuilder) · *Visual J++* · *WinGen for Java* · Servoy · *Xelfi* (became NetBeans) · *XWPE*
.NET	Compilr · MonoDevelop · SharpDevelop · Visual Studio (Express)

Italics indicate software no longer in development.

Category · **Comparison**

Graphics

Visualization helps understanding

- Problem visualization (graphviz, igraph)
- Algorithm animation
- Results visualization: recommended R (more on this later)

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)
    `http://visualvm.java.net/`

Software Development

Extreme Programming & Scrum

Planning

Release planning creates the schedule • Make frequent small releases • The project is divided into iterations • Publish early, revise often

Designing

Simplicity • No functionality is added early • Refactor: eliminate unused functionality and redundancy

Coding

Code must be written to agreed standards • Code the unit test first • All production code is pair programmed • Leave optimization till last • No overtime • Pair programming

Testing

All code must have unit tests • All code must pass all unit tests before it can be released • When a bug is found tests are created

Development of Heuristics

- Implement
- experiment
- fail
- think
- try again!