DM811

Heuristics for Combinatorial Optimization

Working tools, Software, Experiments

Marco Chiarandini

Department of Mathematics & Computer Science
University of Southern Denmark

Course Overview

- ✔ Combinatorial Optimization, Methods and Models
- ✔ CH and LS: overview
1. Working Environment and Solver System
2. Methods for the Analysis of Experimental Results
3. Construction Heuristics
4. Local Search: Components, Basic Algorithms
5. Local Search: Neighborhoods and Search Landscape
6. Efficient Local Search: Incremental Updates and Neighborhood Pruning
7. Stochastic Local Search & Metaheuristics
8. Configuration Tools: F-race
9. Very Large Scale Neighborhoods

Examples: GCP, CSP, TSP, SAT, MaxIndSet, SMTWP, Steiner Tree, p-median, set covering

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
- `res/` output: The result, the performance measurements
- `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.
- `log/` other log files produced by the run of the algorithm
- `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

```
comet queens.co -i instance.in -c output.sol -s 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`
  Comet: see example provided `loadDIMACS.co`

```
comet queens.co -i instance.in -c output.sol -solver 2-opt > data.out
```

- 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, 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: (comet visualize)
- 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!

Random Numbers

Carachtersitics of a good pseudo-random generator
(from stochastic simulation)

- long period
- uniform unbiased distribution
- uncorrelated (time series analysis)
- efficient

Suggested: MRG32k3a by L'Ecuyer
http://www.iro.umontreal.ca/~lecuyer/

```
java.lang.Object
  extended by umontreal.iro.lecuyer.rng.RandomStreamBase
      extended by umontreal.iro.lecuyer.rng.MRG32k3a
```

Ideal Random Shuffle

Let's consider a sequence of n elements: $\{e_1, e_2, \ldots e_n\}$.
The ideal random shuffle is a permutation chosen uniformly at random from the set of all possible $n!$ permutations.

- π_1 is uniformly randomly chosen among $\{e_1, e_2, \ldots e_n\}$.
- π_2 is uniformly randomly chosen among $\{e_1, e_2, \ldots e_n\} - \{\pi_1\}$.
- π_3 is uniformly randomly chosen among $\{e_1, e_2, \ldots e_n\} - \{\pi_1, \pi_2\}$
- ...

Joint probability of $(\pi_1, \pi_2 \ldots \pi_n)$ is $\frac{1}{n} \cdot \frac{1}{n-1} \cdot \ldots 1 = \frac{1}{n!}$

```
long int* Random::generate_random_array(const int& size) {
   long int i, j, help;
   long int *v = new long int[size];
   for ( i = 0 ; i < size; i++ )
     v[i] = i;
   for ( i = 0 ; i < size-1 ; i++) {
     j = (long int) ( ranU01( ) * (size - i));
     help = v[i];
     v[i] = v[i+j];
     v[i+j] = help;
   }
   return v; }
```

* What is the perfect (ideal) random shuffle

Let's consider a sequence of n elements: (e1, e2, ...en). Intuitively, the perfect random shuffle will be a permutation chosen uniformly at random from the set of all possible n! permutations. Let (b1, b2, ... bn) be such a random permutation. Of all n! permutations, (n-1)! of them will have e1 at the first position, (n-1)! more will have element e2 at the first position, etc. Therefore, in the perfect random shuffle (b1, b2, ... bn) b1 is uniformly randomly chosen among e1, ... en. The second element b2 of the shuffled sequence is uniformly randomly chosen among e1, ... en - b1. The third element of the shuffle b3 is uniformly randomly chosen among e1, ... en - b1, b2, etc. Therefore, to perform a perfect random shuffle, we need a sequence of numbers (r1, r2, ... rn) where r1 is a sample of a random quantity uniformly distributed within [0..n-1]. r2 is an independent sample of a random quantity uniformly distributed within [0..n-2]. Finally, rn is 0. It easy to see that the joint probability of (r1, r2, ... rn) is 1/n * 1/(n-1) * ... 1 = 1/n! – which is to be expected.

* An imperative implementation: swapping

The imperative implementation of the algorithm is well known. Let's arrange the input sequence (e1, e2, ...en) into an array 'a' so that initially a[i] = ei,

i=1..n. At step 1, we choose a random number r1 uniformly from [0..n-1]. Thus a[1+r1] will be the first element of the permuted sample, b1. We swap a[1+r1] and a[1]. Thus a[1] will contain b1. At step 2, we choose a random number r2 uniformly from [0..n-2]. a[2+r2] will be a uniform sample from e1..en - b1. After we swap a[2] and a[2+r2], a[2] will be b2, and a[3..n] will contain the remaining elements of the original sequence, e1..en - b1,b2. After n-1 steps, we're done. The imperative algorithm in OCaml has been already posted on this thread.

Outline

1. Working Environment
   - Organization
   - Random Numbers
2. Software Tools
   - Constraint-Based Local Search with Comet™
   - Local Solver
3. Experimental Analysis
   - Motivations and Goals
   - Descriptive Statistics
     - Performance Measures
     - Sample Statistics
   - Scenarios of Analysis
     - A. Single-pass heuristics
     - B. Asymptotic heuristics
   - Guidelines for Presenting Data

Software Tools

- Modeling languages
  interpreted languages with a precise syntax and semantics
- Software libraries
  collections of subprograms used to develop software
- Software frameworks
  set of abstract classes and their interactions
  - *frozen spots* (remain unchanged in any instantiation of the framework)
  - *hot spots* (parts where programmers add their own code)

Software Tools

No well established software tool for Local Search:

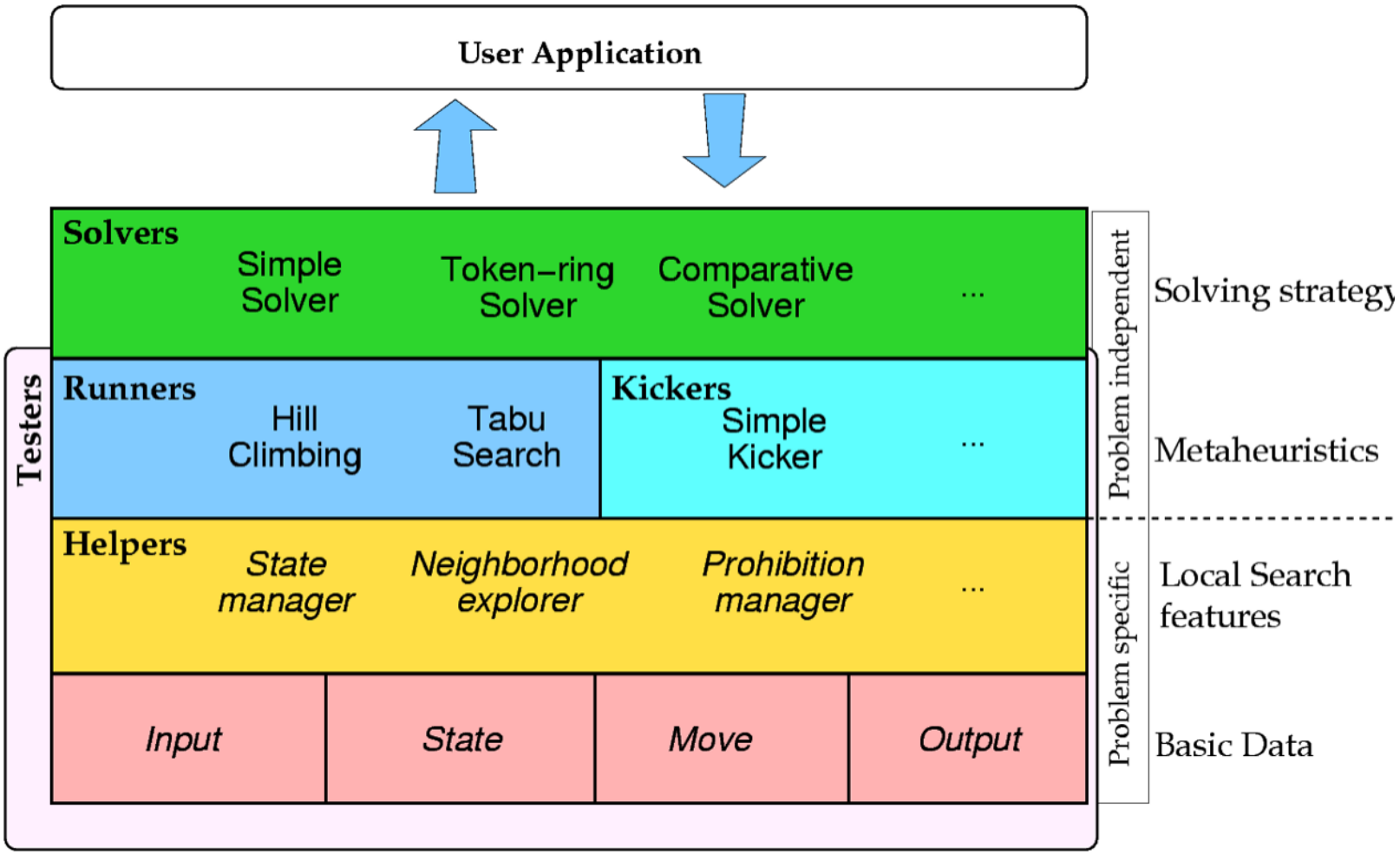
- the apparent simplicity of Local Search induces to build applications from scratch.
- the freedom of problem characteristics that can be tackled
- crucial roles played by delta/incremental updates which are highly problem dependent
- the development of Local Search is in part a craft, beside engineering and science.
- lack of a unified view of Local Search.

Software Tools

EasyLocal++	C++, (Java)	Local Search
ParadisEO	C++	Local Search, Evolutionary Algorithm
OpenTS	Java	Tabu Search
Comet	Language	
LocalSolver	Modelling Language	
Google OR Tools	Libraries	

EasyLocal++	http://tabu.diegm.uniud.it/EasyLocal++/
ParadisEO	http://paradiseo.gforge.inria.fr
OpenTS	http://www.coin-or.org/Ots
Comet	http://dynadec.com/
LocalSolver	http://www.localsolver.com/
Google OR Tools	https://code.google.com/p/or-tools/

A Framework



http://tabu.diegm.uniud.it/EasyLocal++/

Comet is

A programming language

- Syntax inspired by C++
  - Object-oriented
  - Operator overloading
  - Filestreams
- Interpreted or Just-in-Time compiled
- Garbage collection
- High-level features
  - Invariants (one-way-constraints)
    specify what needs to be maintained incrementally without considering how to do so
  - Closures
  - Functional programming-like constructions
    - List comprehension
    - `collect, filter, sum, select, selectMin, selectMax`
  - Sets, dictionaries, etc. are builtin types
  - Events

Comet is

A runtime environment

- With integrated optimization solvers
  - Constraint-Based Local Search
  - Constraint Programming
  - Linear Programming (COIN-OR CLP)
  - Mixed Integer Programming
- 2D graphics library
- Available for many platforms
  - Mac OS X (32 and 64 bit)
  - Windows
  - Linux (32 and 64 bit)
    - Ubuntu
    - SuSE
    - RedHat/Fedora

Comet is

Unfortunately not Open Source

Developed by Pascal Van Hentenryck (Brown University), Laurent Michel (University of Connecticut), now owned by Dynadec.

Not anymore in active development

Constraint Programming is

- Model
  - Variables
    - Domains
  - Objective Function
  - Constraints
- Search
  - Branching
    - Variable selection
    - Value selection
  - Search strategy
    - BFS
    - DFS
    - LDS

Constraint-Based Local Search is

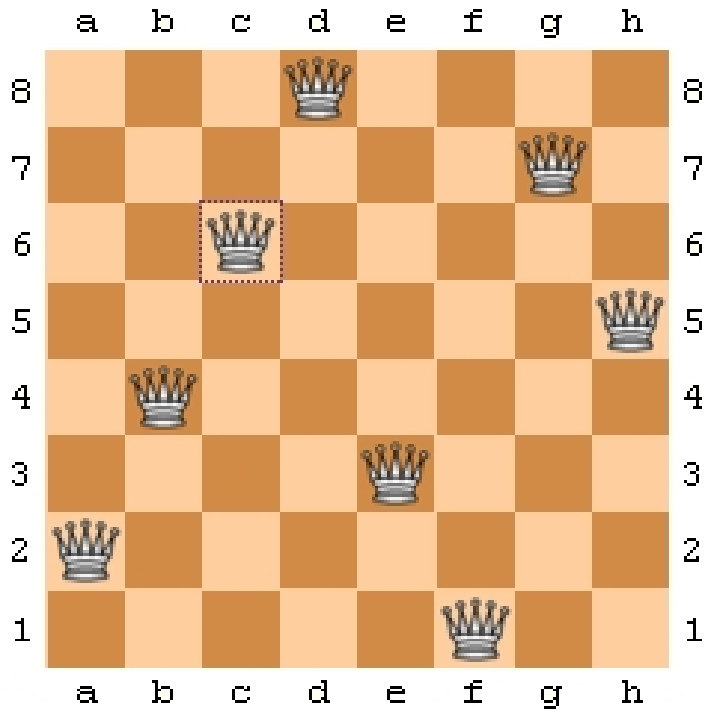
- Model
  - Incremental variables
  - Invariants
  - Differentiable objects
    - Functions
    - Constraints
    - Constraint Systems
- Search
  - Local Search
    - Iterative Improvement
    - Tabu Search
    - Simulated Annealing
    - Guided Local Search

Example

N-Queens problem

Input: A chessboard of size $N \times N$

Task: Find a placement of n queens on the board such that no two queens are on the same row, column, or diagonal.



An LS Example

⇝ Use of differentiable constraints (or objectives)

```
import cotls;
int n = 16;
range Size = 1..n;
UniformDistribution distr(Size);

Solver<LS> m();
var{int} queen[Size](m,Size) := distr.get();
ConstraintSystem<LS> S(m);

S.post(alldifferent(queen));
S.post(alldifferent(all(i in Size) queen[i] + i));
S.post(alldifferent(all(i in Size) queen[i] - i));
m.close();

int it = 0;
while (S.violations() > 0 && it < 50 * n) {
  select(q in Size, v in Size : S.getAssignDelta(queen[q],v) < 0) {
    queen[q] := v;
    cout<<"chng @ "<<it<<": queen["<<q<<"]:="<<v<<" viol: "<<S.violations() <<endl;
  }
  it = it + 1;
}
cout << queen << endl;
```

How to learn more

Comet Tutorial
in the Comet distribution

Constraint-Based Local Search
P. Van Hentenryck, L. Michel
MIT Press, 2005
ISBN-10: 0-262-22077-6

- See: http://www.imada.sdu.dk/~marco/Misc/comet.html
- Ask: http://forums.dynadec.com

Local Search Modelling Language

Enriched mathematical programming formulation:

- Boolean variables (0–1 programming)
- constriants (always satisfied) - decision between soft and hard left to user
- invariants
- objectives (lexicographics ordering)

Example (Bin-packing problem)

Input 3 items x, y, z of height 2,3,4 to pack into 2 piles A, B with B already containing an item of height 5.
Task Minimize height of largest pile

```
xA <- bool(); yA <- bool(); zA <- bool();
xB <- bool(); yB <- bool(); zB <- bool();
constraint booleansum(xA, xB) = 1;
constraint booleansum(yA, yB) = 1;
constraint booleansum(zA, zB) = 1;
heightA <- sum(2xA, 3yA, 4zA);
heightB <- sum(2xB, 3yB, 4zB, 5);
objective <- max(heightA, heightB); minimize objective;
```

Black-Box Local Search Solver

- initial solution: randomized greedy algorithm (constraints satisfied)
- search strategy (standard descent, simulated annealing, random restart via multithreading)
- moves
  specialized for constraints and feasibility
- incremental evaluation machinery
  problem represented as a DAG: variables are roots, objectives leaves, operators induce inner nodes
  bredth-first search in DAG.

Local Solver

Example (Graph Coloring)

```
/* Declares the optimization model. */
function model(){
   x[1..n][1..k] <- bool();
   y[1..k] <- bool();

   // Assign color
   for[i in 1..n]
      constraint sum[l in 1..k](x[i][l]) == 1;

   for[c in 1..m][l in 1..k]
      constraint sum[i in 1..v[c][0]](x[v[c][i]][l]) <= 1;

   y[l in 1..k] <- max[i in 1..n](x[i][l]);

   // Clique constraint
   obj <- sum[l in 1..k](y[l]);
   minimize obj;
}
```

Local Solver

```
/* Parameterizes the solver. */
function param(){
        if(lsTimeLimit == nil)
              lsTimeLimit=600;
  lsTimeBetweenDisplays = 10;
  lsNbThreads = 4;
  lsAnnealingLevel = 5;
}

/* Writes the solution in a file following the following format:
 * each line contains a vertex number and its subset (1 for S, 0 for V-S) */
function output(){
        println("Write solution into file 'sol.txt'");
        solFile = openWrite("sol.txt");
        for [i in 1..n][l in 1..k]{
      if (getValue(x[i][l]) == true)
        println(solFile, i, " ", l);
   }
}
```

Outline

Contents and Goals

Provide a view of issues in Experimental Algorithmics

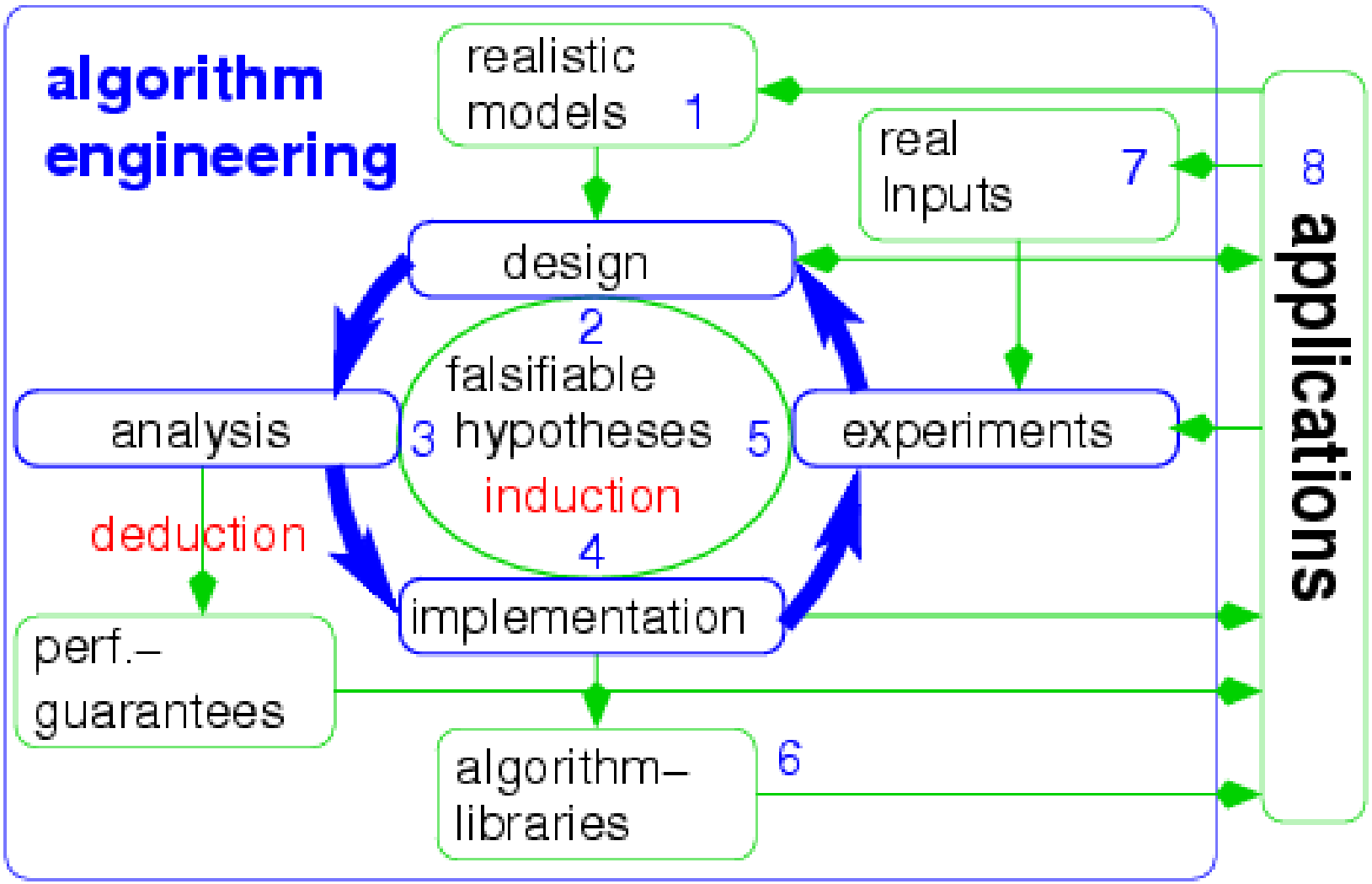
- Exploratory data analysis
- Presenting results in a concise way with graphs and tables
- Organizational issues and Experimental Design

- Basics of inferential statistics
- Sequential statistical testing: race, a methodology for tuning

The goal of Experimental Algorithmics is not only producing a sound analysis but also adding an important tool to the development of a good solver for a given problem.

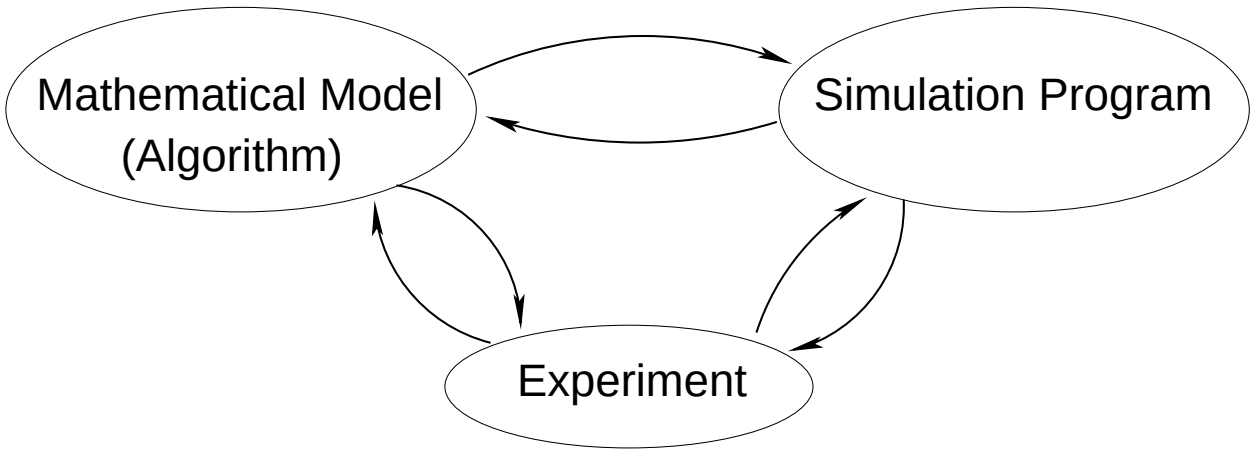
Experimental Algorithmics is an important part in the algorithm production cycle, which is referred to as Algorithm Engineering

The Engineering Cycle



from http://www.algorithm-engineering.de/

Experimental Algorithmics



In empirical studies we consider simulation programs which are the implementation of a mathematical model (the algorithm)

[McGeoch, 1996]

Experimental Algorithmics

Goals

- Defining standard methodologies
- Comparing relative performance of algorithms so as to identify the best ones for a given application
- Characterizing the behavior of algorithms
- Identifying algorithm separators, *i.e.*, families of problem instances for which the performance differ
- Providing new insights in algorithm design

Fairness Principle

Fairness principle: being completely fair is perhaps impossible but try to remove any possible bias

- possibly all algorithms must be implemented with the same style, with the same language and sharing common subprocedures and data structures
- the code must be optimized, e.g., using the best possible data structures
- running times must be comparable, e.g., by running experiments on the same computational environment (or redistributing them randomly)

Definitions

The most typical scenario considered in analysis of search heuristics

Asymptotic heuristics with time/quality limit decided *a priori*

The algorithm $\mathcal{A}^\infty$ is halted when time expires or a solution of a given quality is found.

Deterministic case: $\mathcal{A}^\infty$ on π returns a solution of cost x.

The performance of $\mathcal{A}^\infty$ on π is a scalar $y = x$.

Randomized case: $\mathcal{A}^\infty$ on π returns a solution of cost X, where X is a random variable.

The performance of $\mathcal{A}^\infty$ on π is the univariate $Y = X$.

[This is not the only relevant scenario: to be refined later]

Random Variables and Probability

Statistics deals with random (or stochastic) variables.

A variable is called random if, prior to observation, its outcome cannot be predicted with certainty.

The uncertainty is described by a probability distribution.

Discrete variables

Probability distribution:

$$p_i = P[x = v_i]$$

Cumulative Distribution Function (CDF)

$$F(v) = P[x \leq v] = \sum_i p_i$$

Mean

$$\mu = E[X] = \sum x_i p_i$$

Variance

$$\sigma^2 = E[(X - \mu)^2] = \sum (x_i - \mu)^2 p_i$$

Continuous variables

Probability density function (pdf):

$$f(v) = \frac{dF(v)}{dv}$$

Cumulative Distribution Function (CDF):

$$F(v) = \int_{-\infty}^{v} f(v)dv$$

Mean

$$\mu = E[X] = \int x f(x) dx$$

Variance

$$\sigma^2 = E[(X - \mu)^2] = \int (x - \mu)^2 f(x)\, dx$$

Generalization

For each general problem Π (e.g., TSP, GCP) we denote by C_Π a set (or class) of instances and by $\pi \in C_\Pi$ a single instance.

On a specific instance, the random variable Y that defines the performance measure of an algorithm is described by its probability distribution/density function

$$Pr(Y = y \mid \pi)$$

It is often more interesting to generalize the performance on a class of instances C_Π, that is,

$$Pr(Y = y, C_\Pi) = \sum_{\pi \in \Pi} Pr(Y = y \mid \pi) Pr(\pi)$$

Sampling

In experiments,

1. we sample the population of instances and
2. we sample the performance of the algorithm on each sampled instance

If on an instance π we run the algorithm r times then we have r replicates of the performance measure Y, denoted $Y_1, \ldots, Y_r$, which are independent and identically distributed (i.i.d.), i.e.

$$Pr(y_1, \ldots, y_r|\pi) = \prod_{j=1}^{r} Pr(y_j \mid \pi)$$

$$Pr(y_1, \ldots, y_r) = \sum_{\pi \in C_\Pi} Pr(y_1, \ldots, y_r \mid \pi) Pr(\pi).$$

Instance Selection

In real-life applications a simulation of $p(\pi)$ can be obtained by historical data.

In simulation studies instances may be:

- real world instances
- random variants of real world-instances
- online libraries
- randomly generated instances

They may be grouped in classes according to some features whose impact may be worth studying:

- type (for features that might impact performance)
- size (for scaling studies)
- hardness (focus on hard instances)
- application (e.g., CSP encodings of scheduling problems), ...

Within the class, instances are drawn with uniform probability $p(\pi) = c$

Statistical Methods

The analysis of performance is based on finite-size sampled data.
Statistics provides the methods and the mathematical basis to

- describe, summarizing, the data (descriptive statistics)
- make inference on those data (inferential statistics)

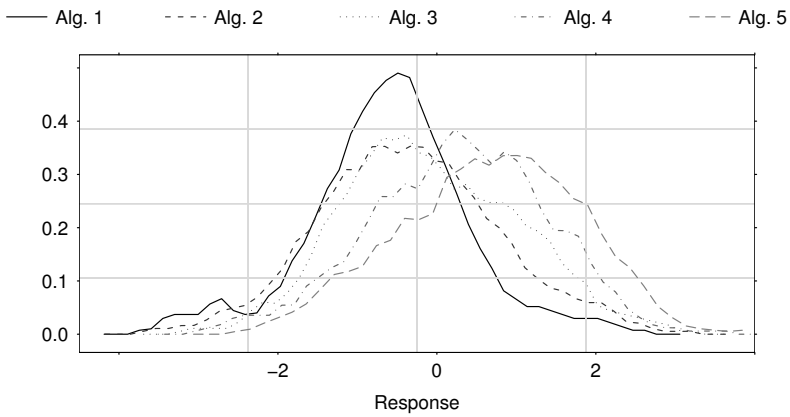
Statistics helps to

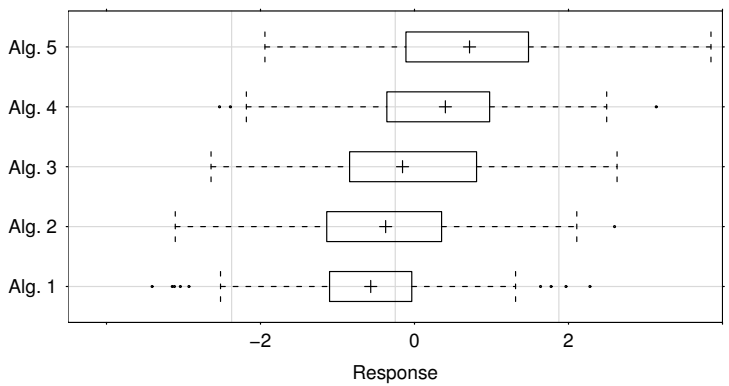
- guarantee reproducibility
- make results reliable
  (are the observed results enough to justify the claims?)
- extract relevant results from large amount of data

In the practical context of heuristic design and implementation (i.e., engineering), statistics helps to take correct design decisions with the least amount of experimentation

Objectives of the Experiments

- **Comparison**:
  bigger/smaller, same/different,
  Algorithm Configuration,
  Component-Based Analysis
  - Standard statistical methods:
    experimental designs, test hypothesis and estimation

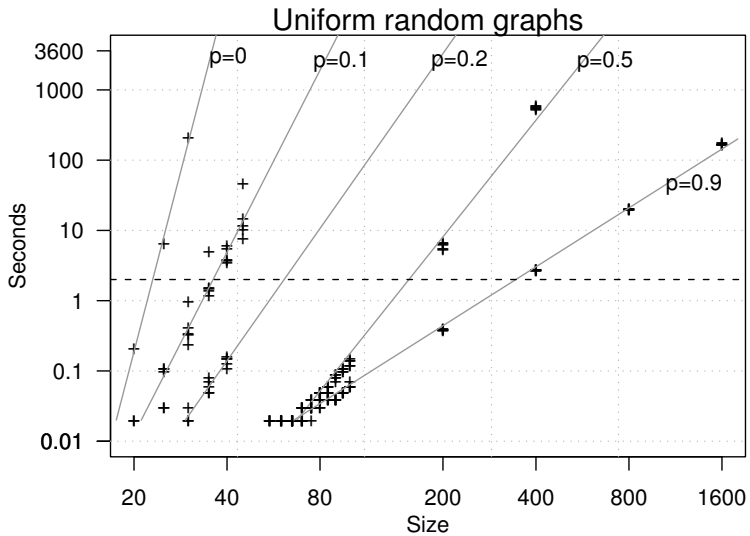




Objectives of the Experiments

- **Comparison**:
  bigger/smaller, same/different,
  Algorithm Configuration,
  Component-Based Analysis
  - Standard statistical methods:
    experimental designs, test hypothesis and estimation

- **Characterization**:
  Interpolation: fitting models to data
  Extrapolation: building models of data, explaining phenomena
  - Standard statistical methods: *linear and non linear regression*
    model fitting



Outline

Measures and Transformations

On a single instance

Design: Several runs on an instance

	Algorithm 1	**Algorithm 2**	...	**Algorithm k**
Instance 1	X_{11}	X_{21}		X_{k1}
⋮	⋮	⋮		⋮
Instance 1	X_{1r}	X_{2r}		X_{kr}

Measures and Transformations

On a single instance

Computational effort indicators

- number of elementary operations/algorithmic iterations
  (e.g., search steps, objective function evaluations, number of visited nodes in the search tree, consistency checks, etc.)
- total CPU time consumed by the process
  (sum of *user* and *system* times returned by `getrusage`)

Solution quality indicators

- value returned by the cost function
- error from optimum/reference value
- (optimality) gap $\frac{UB-LB}{LB+\epsilon}$ (if max $\frac{UB-LB}{UB+\epsilon}$)
  ϵ is an infinitesimal for the case $LB = 0$ but $UB - LB \neq 0$
- ranks

Measures and Transformations

On a class of instances

Design A: One run on various instances

	Algorithm 1	**Algorithm 2**	...	**Algorithm k**
Instance 1	X_{11}	X_{12}		X_{1k}
$\vdots$	$\vdots$	$\vdots$		$\vdots$
Instance b	X_{b1}	X_{b2}		X_{bk}

Design B: Several runs on various instances

	Algorithm 1	**Algorithm 2**	...	**Algorithm k**
Instance 1	$X_{111}, \ldots, X_{11r}$	$X_{121}, \ldots, X_{12r}$		$X_{1k1}, \ldots, X_{1kr}$
Instance 2	$X_{211}, \ldots, X_{21r}$	$X_{221}, \ldots, X_{22r}$		$X_{2k1}, \ldots, X_{2kr}$
$\vdots$	$\vdots$	$\vdots$		$\vdots$
Instance b	$X_{b11}, \ldots, X_{b1r}$	$X_{b21}, \ldots, X_{b2r}$		$X_{bk1}, \ldots, X_{bkr}$

Measures and Transformations

On a class of instances

Computational effort indicators

- no transformation if the interest is in studying scaling
- standardization if a fixed time limit is used
- geometric mean (used for a set of numbers whose values are meant to be multiplied together or are exponential in nature),
- otherwise, better to group homogeneously the instances

Solution quality indicators

Different instances imply different scales $\Rightarrow$ need for an invariant measure

(However, many other measures can be taken both on the algorithms and on the instances [McGeoch, 1996])

Measures and Transformations

On a class of instances (cont.)

Solution quality indicators

- Distance or error from a reference value (assume minimization case):

$$e_1(x,\pi) = \frac{x(\pi) - \bar{x}(\pi)}{\sqrt{\hat{\sigma(\pi)}}} \qquad \text{standard score}$$

$$e_2(x,\pi) = \frac{x(\pi) - x^{opt}(\pi)}{x^{opt}(\pi)} \qquad \text{relative error}$$

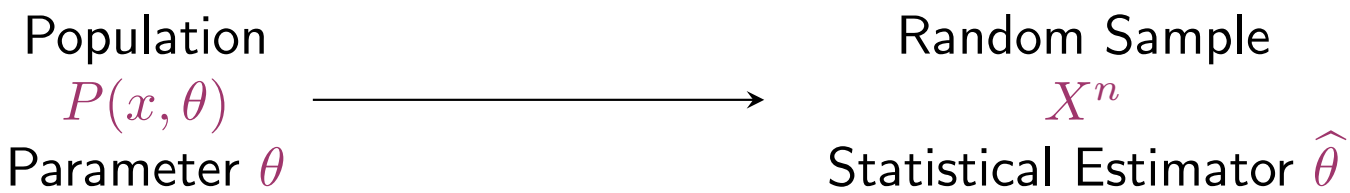
$$e_3(x,\pi) = \frac{x(\pi) - x^{opt}(\pi)}{x^{worst}(\pi) - x^{opt}(\pi)} \qquad \text{invariant [Zemel, 1981]}$$

  - optimal value computed exactly or known by construction
  - surrogate value such bounds or best known values
- Rank (no need for standardization but loss of information)

Outline

Sampling

- We work with samples (instances, solution quality) drawn from populations



Summary Measures

Measures to describe or characterize a population

- Measure of central tendency, location
- Measure of dispersion

One such a quantity is

- a **parameter** if it refers to the population (Greek letters)
- a **statistics** if it is an *estimation* of a population parameter from the sample (Latin letters)

Measures of central tendency

- Arithmetic Average (Sample mean)

$$\bar{X} = \frac{\sum x_i}{n}$$

- *Quantile*: value above or below which lie a fractional part of the data (used in nonparametric statistics)
  - Median

$$\mathcal{M} = x_{(n+1)/2}$$

  - Quartile

$$Q_1 = x_{(n+1)/4} \qquad Q_3 = x_{3(n+1)/4}$$

  - q-quantile

    q of data lies below and $1 - q$ lies above

- Mode

  value of relatively great concentration of data
  (*Unimodal* *vs* *Multimodal* distributions)

Measure of dispersion

- Sample range

$$R = x_{(n)} - x_{(1)}$$

- Sample variance

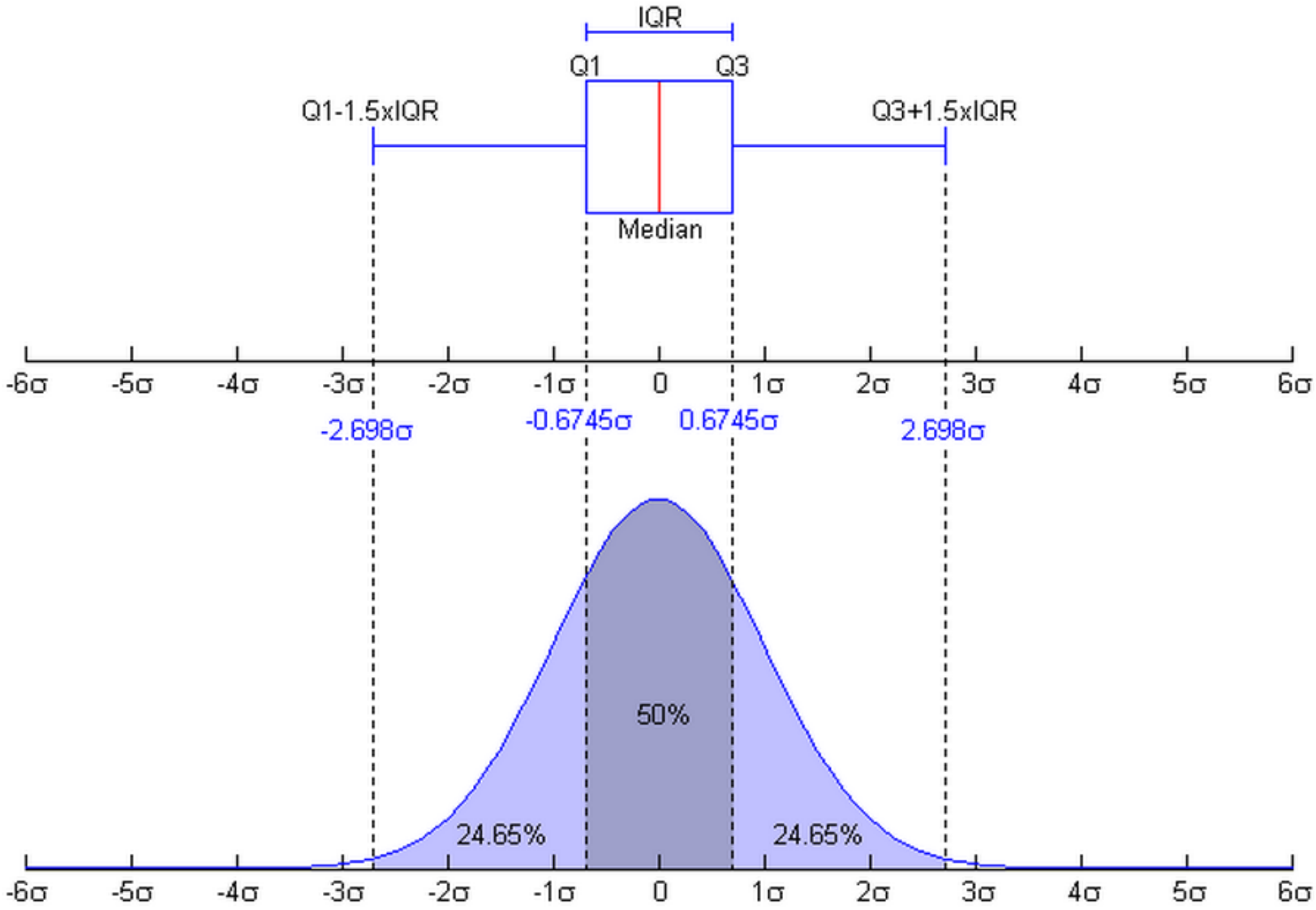
$$s^2 = \frac{1}{n-1} \sum (x_i - \bar{X})^2$$

- Standard deviation

$$s = \sqrt{s^2}$$

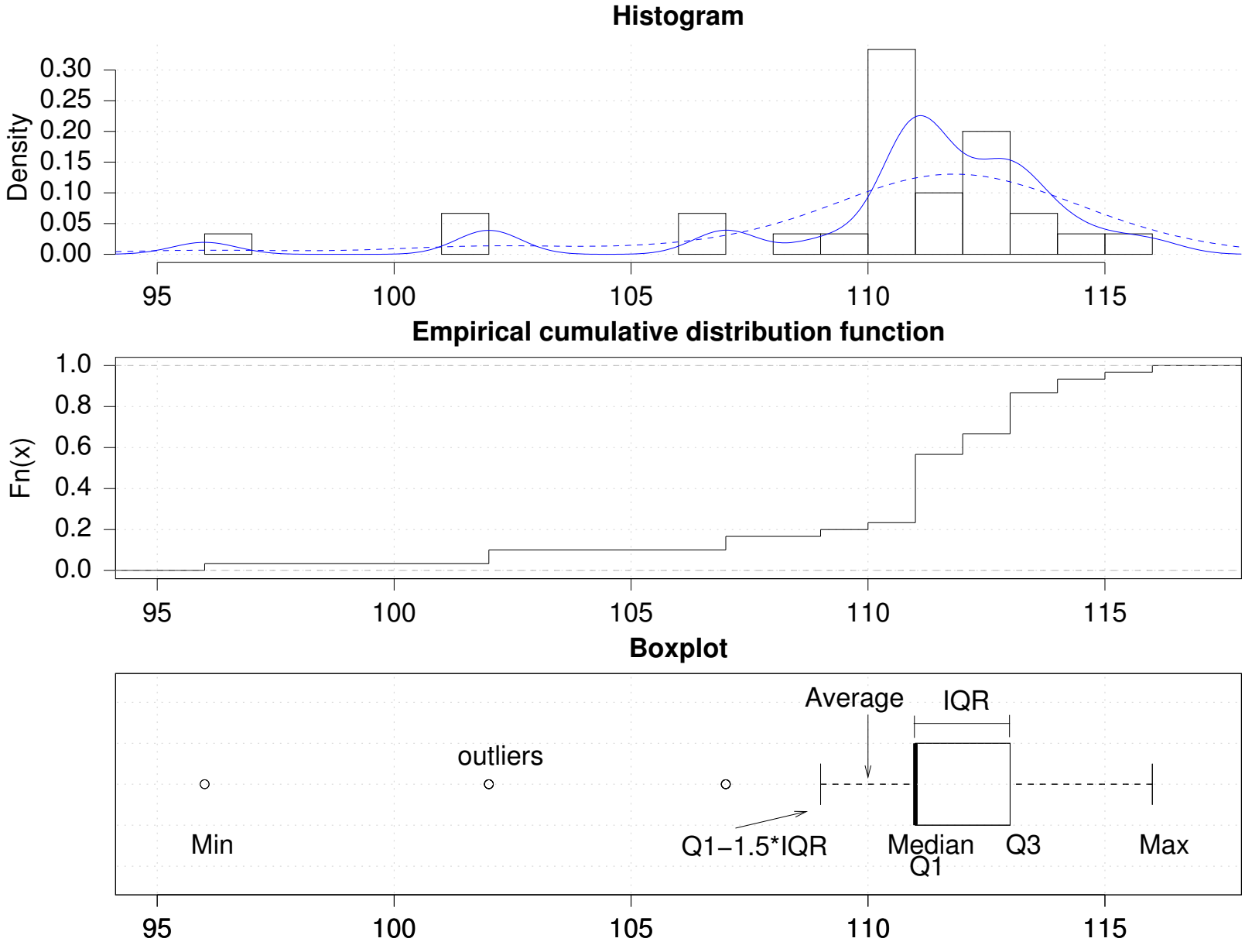
- Inter-quartile range

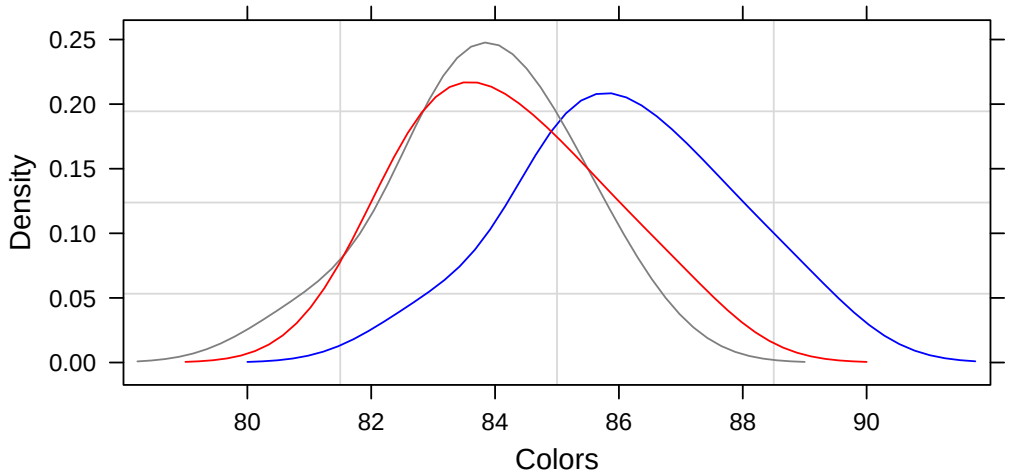
$$IQR = Q_3 - Q_1$$

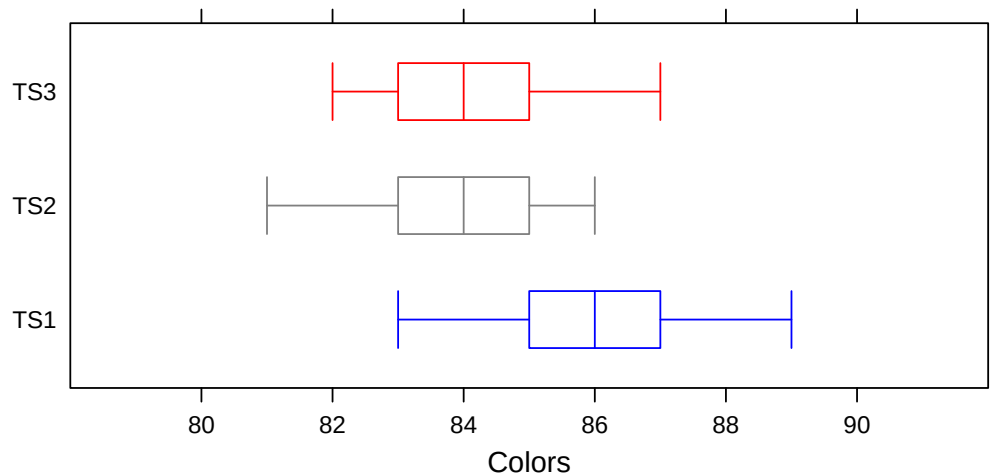


Boxplot and a probability density function (pdf) of a Normal N(0,1) Population. (source: Wikipedia)

[see also: http://informationandvisualization.de/blog/box-plot]


Histogram
Density
0.30
0.25
0.20
0.15
0.10
0.05
0.00
95
100
105
110
115
Empirical cumulative distribution function
Fn(x)
1.0
0.8
0.6
0.4
0.2
0.0
95
100
105
110
115
Boxplot
Average
IQR
outliers
Min
Q1−1.5*IQR
Median
Q1
Q3
Max
95
100
105
110
115


0.25
0.20
0.15
0.10
0.05
0.00
Density
80
82
84
86
88
90
Colors


TS3
TS2
TS1
80
82
84
86
88
90
Colors

In R

```
> x<-runif(10,0,1)
 mean(x), median(x), quantile(x), quantile(x,0.25)
 range(x), var(x), sd(x), IQR(x)
> fivenum(x)
#(minimum, lower−hinge, median, upper−hinge, maximum)
[1] 0.18672 0.26682 0.28927 0.69359 0.92343
> summary(x)
> aggregate(x,list(factors),median)
> boxplot(x)
```

Scenarios

A. Single-pass heuristics

B. Asymptotic heuristics:
   Two approaches:

   1. Univariate

      1.a Time as an external parameter decided *a priori*
      1.b Solution quality as an external parameter decided *a priori*

   2. Cost dependent on running time:

Outline

Scenario A

Single-pass heuristics

Deterministic case: $\mathcal{A}^{\dashv}$ on class C_{Π} returns a solution of cost x with computational effort t (*e.g.*, running time).

The performance of $\mathcal{A}^{\dashv}$ on class C_{Π} is the vector $\vec{y} = (x, t)$.

Randomized case: $\mathcal{A}^{\dashv}$ on class C_{Π} returns a solution of cost X with computational effort T, where X and T are random variables.

The performance of $\mathcal{A}^{\dashv}$ on class C_{Π} is the bivariate $\vec{Y} = (X, T)$.

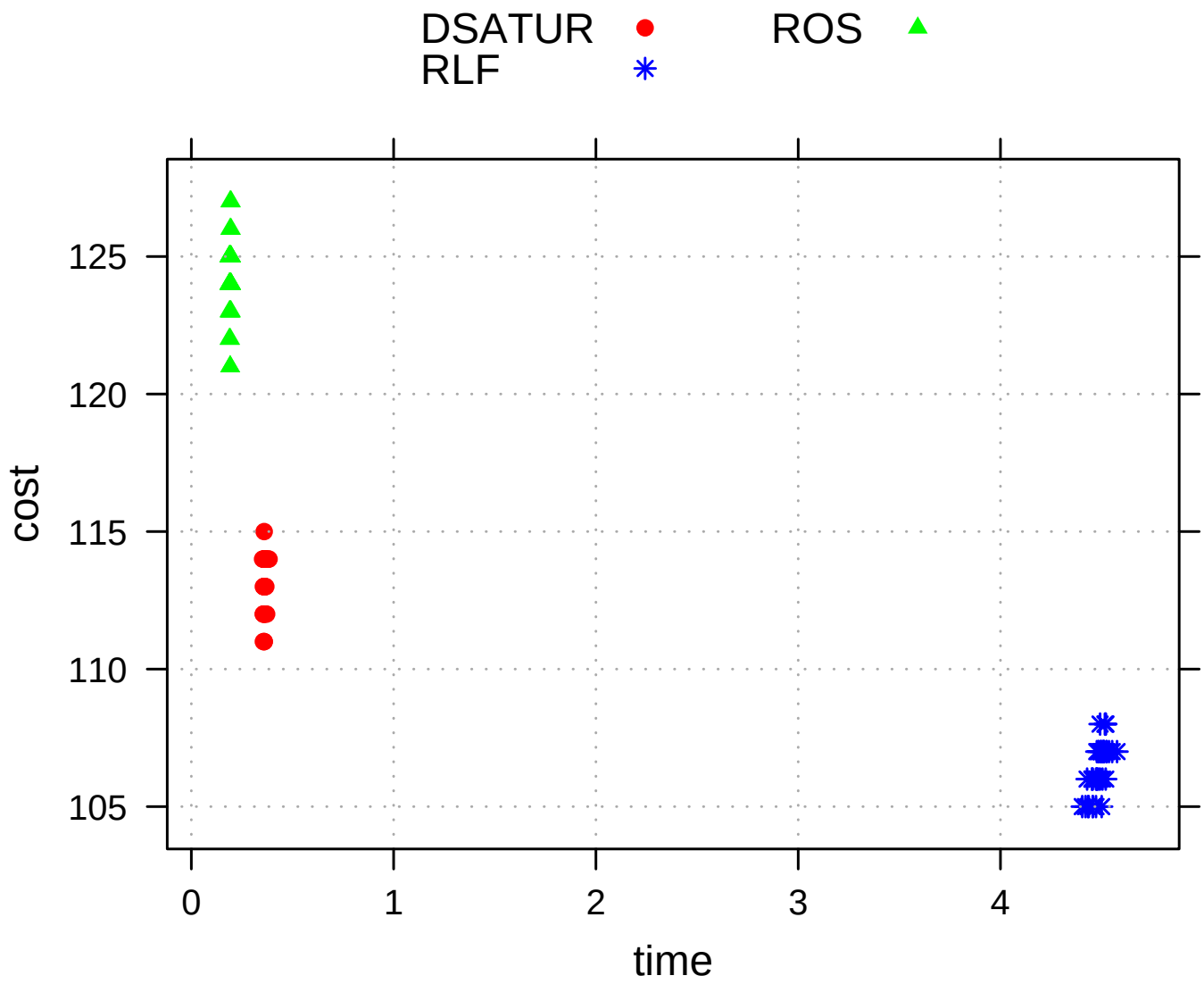
Example

Scenario:

- ▷ 3 heuristics $\mathcal{A}_1^{\dashv}$, $\mathcal{A}_2^{\dashv}$, $\mathcal{A}_3^{\dashv}$ on class C_{Π}.
- ▷ homogeneous instances or need for data transformation.
- ▷ 1 or r runs per instance
- ▶ Interest: inspecting solution cost and running time to observe and compare the level of approximation and the speed.

Tools:

- Scatter plots of solution-cost and run-time


DSATUR
ROS
RLF
cost
125
120
115
110
105
0
1
2
3
4
time

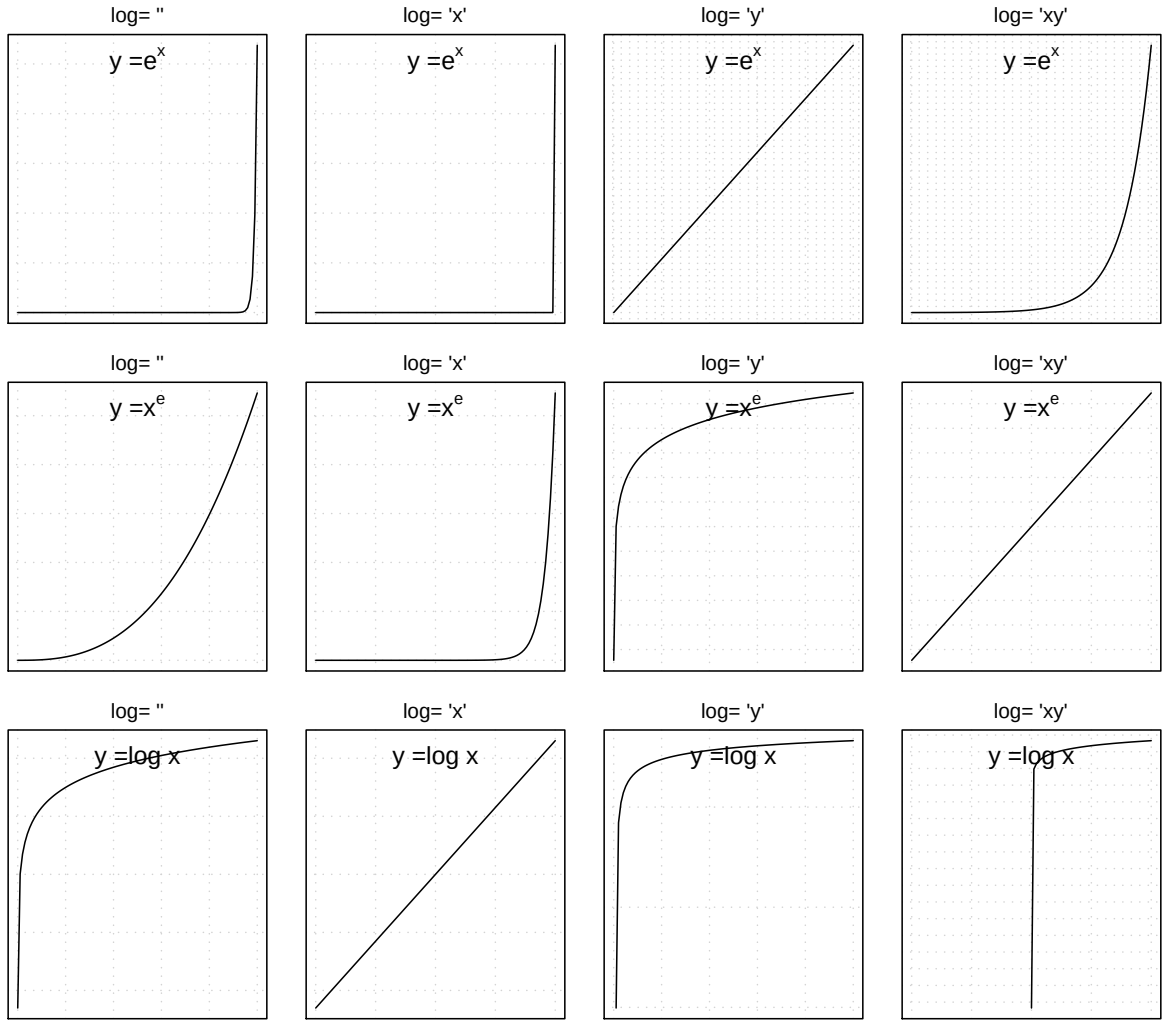
Multi-Criteria Decision Making

Needed some definitions on dominance relations

In Pareto sense, for points in $\mathbf{R}^2$

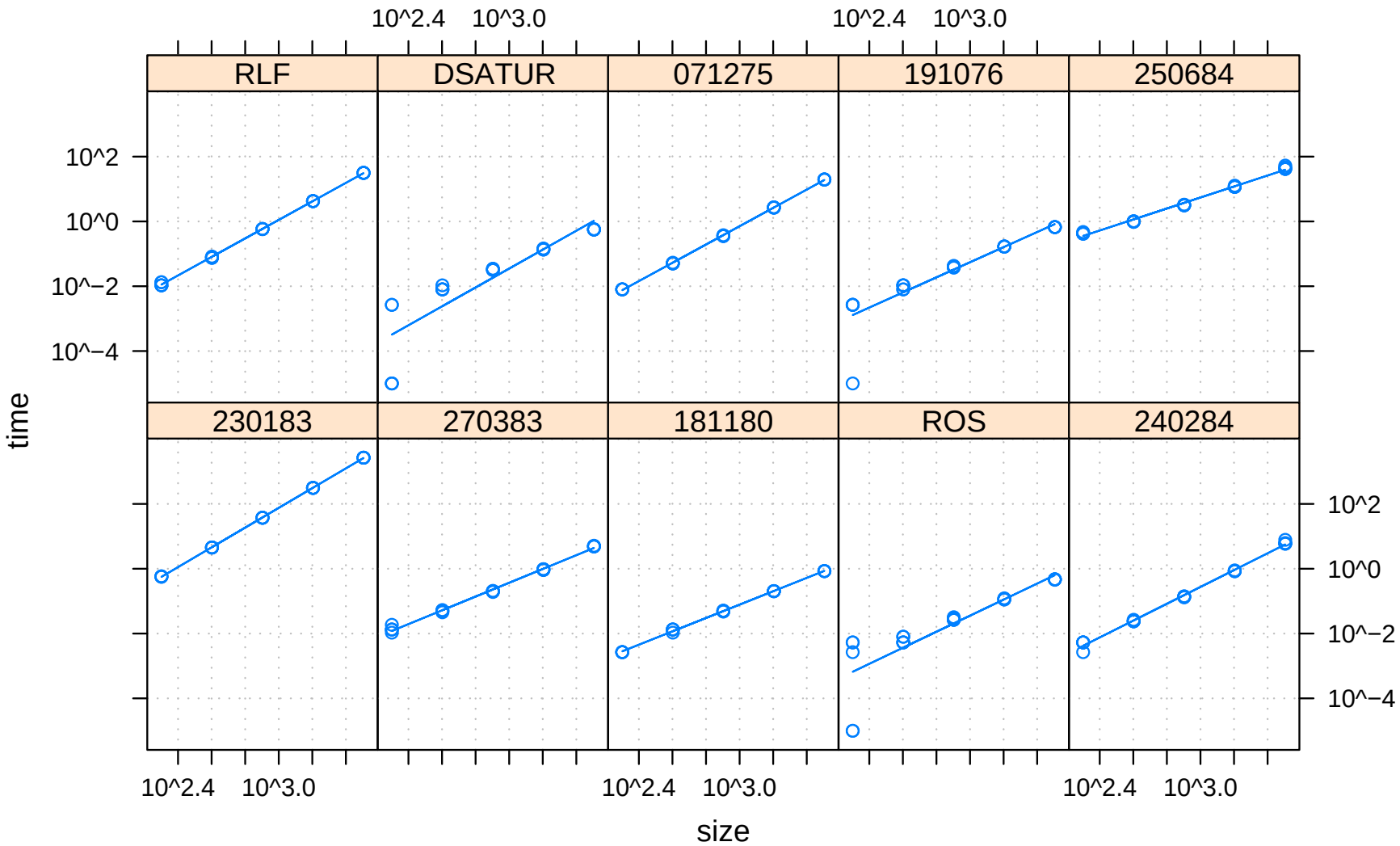
$\vec{x}^1 \preceq \vec{x}^2$	weakly dominates	$x_i^1 \leq x_i^2$ for all $i = 1, \ldots, n$
$\vec{x}^1 \parallel \vec{x}^2$	incomparable	neither $\vec{x}^1 \preceq \vec{x}^2$ nor $\vec{x}^2 \preceq \vec{x}^1$

Scaling Analysis

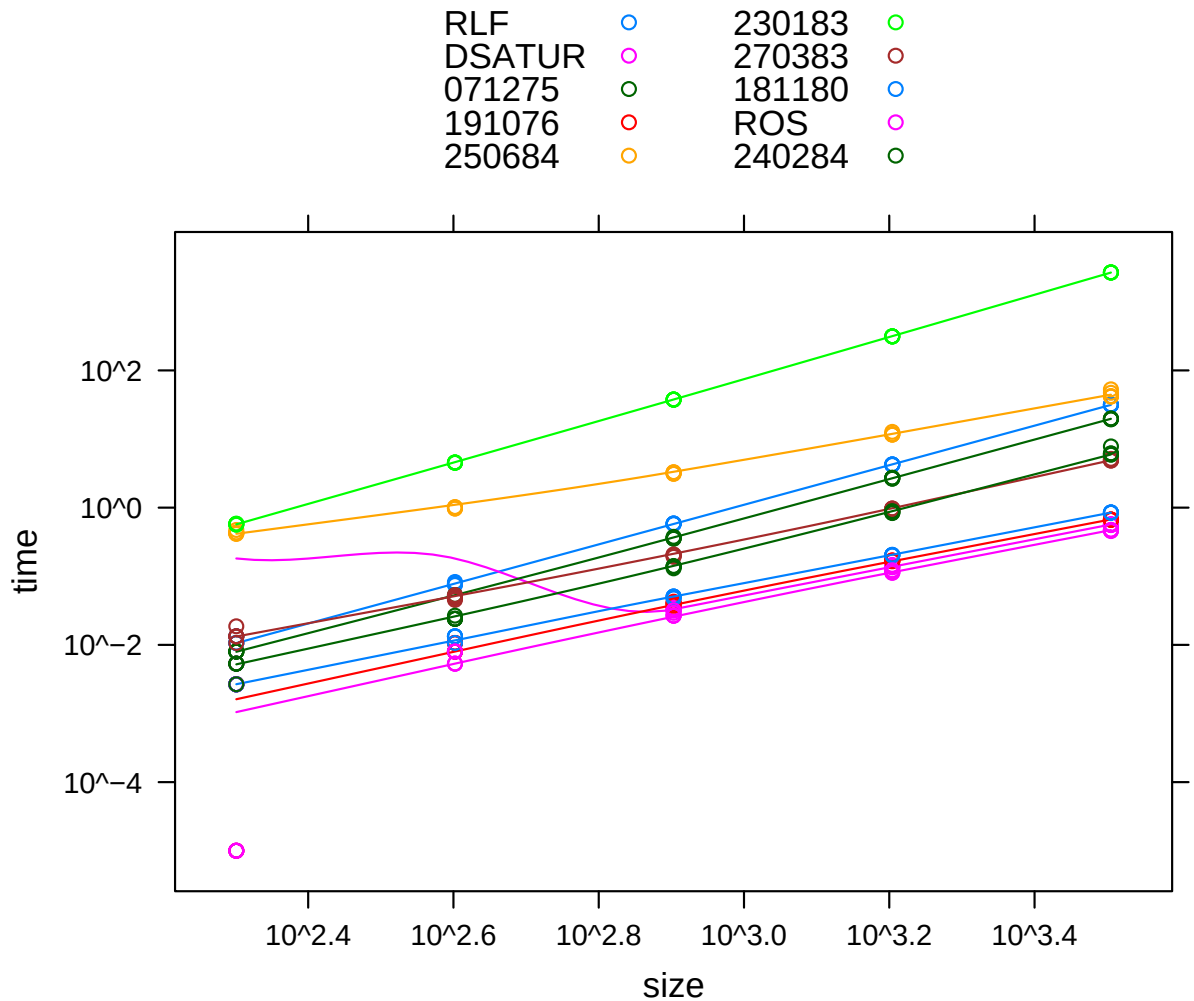


Linear regression in log-log plots $\Rightarrow$ polynomial growth

Linear regression in log-log plots $\Rightarrow$ polynomial growth



Comparative visualization



Outline

Scenarios

A. Single-pass heuristics

B. Asymptotic heuristics:
   Two approaches:

   1. Univariate

      1.a Time as an external parameter decided *a priori*
      1.b Solution quality as an external parameter decided *a priori*

   2. Cost dependent on running time:

Scenario B

Asymptotic heuristics

There are two approaches:

1.a. Time as an external parameter decided *a priori*.
The algorithm is halted when time expires.

Deterministic case: $\mathcal{A}^\infty$ on class C_Π returns a solution of cost x.

The performance of $\mathcal{A}^\infty$ on class C_Π is the scalar $y = x$.

Randomized case: $\mathcal{A}^\infty$ on class C_Π returns a solution of cost X, where X is a random variable.

The performance of $\mathcal{A}^\infty$ on class C_Π is the univariate $Y = X$.

Example

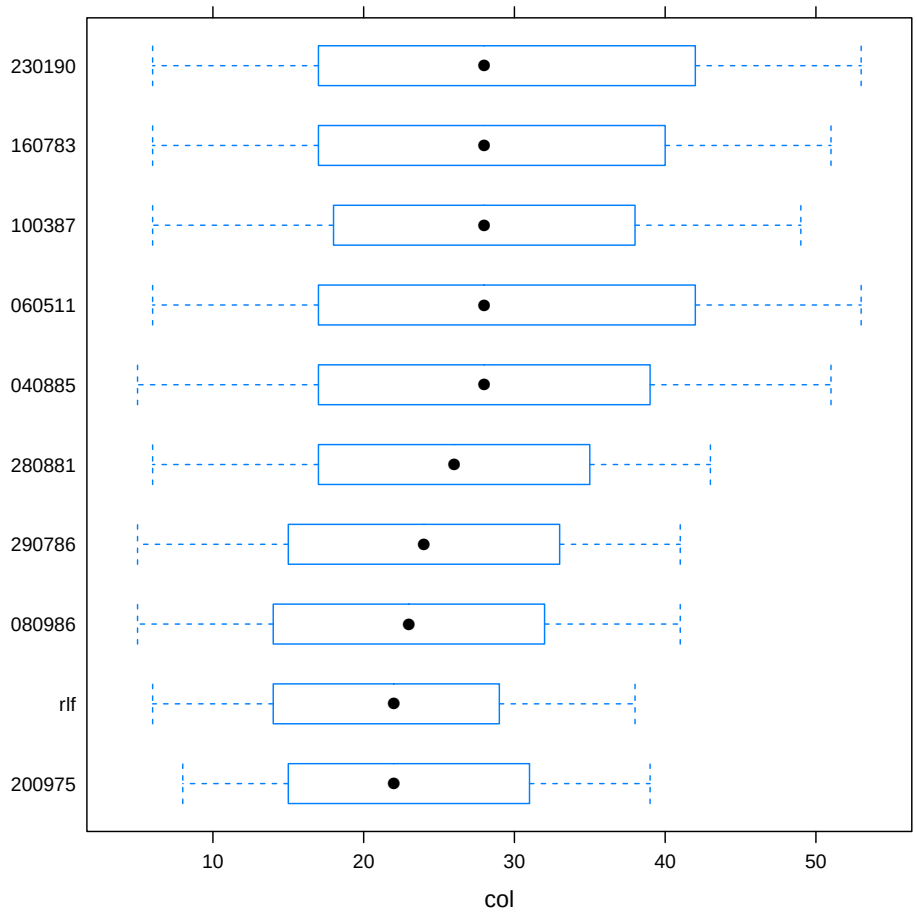
Scenario:

- ▷ 3 heuristics $\mathcal{A}_1^\infty$, $\mathcal{A}_2^\infty$, $\mathcal{A}_3^\infty$ on class C_Π.
  (Or 3 heuristics $\mathcal{A}_1^\infty$, $\mathcal{A}_2^\infty$, $\mathcal{A}_3^\infty$ on class C_Π without interest in computation time because negligible or comparable)
- ▷ homogeneous instances (no data transformation) or heterogeneous (data transformation)
- ▷ 1 or r runs per instance
- ▷ a priori time limit imposed
- ▶ Interest: inspecting solution cost

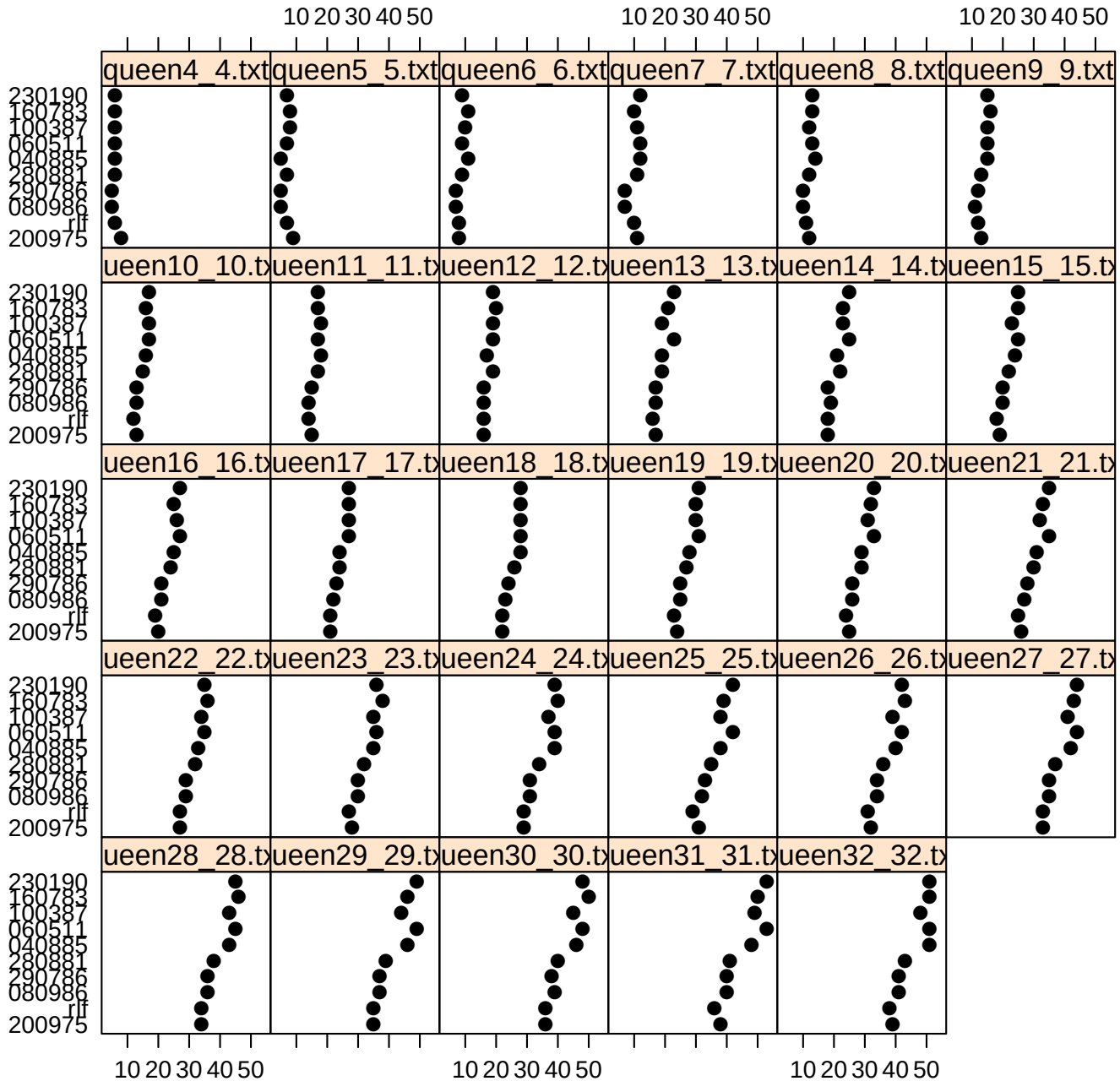
Tools:

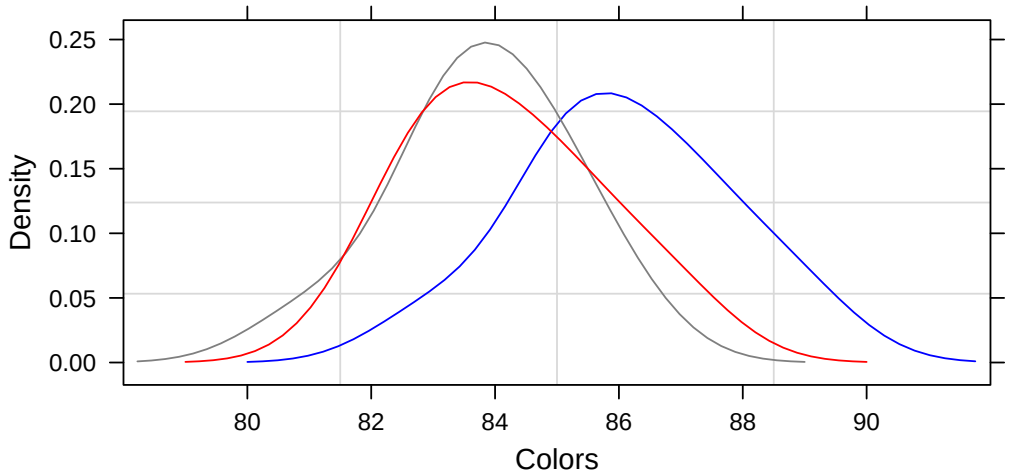
- Histograms (summary measures: mean or median or mode?)
- Boxplots
- Empirical cumulative distribution functions (ECDFs)

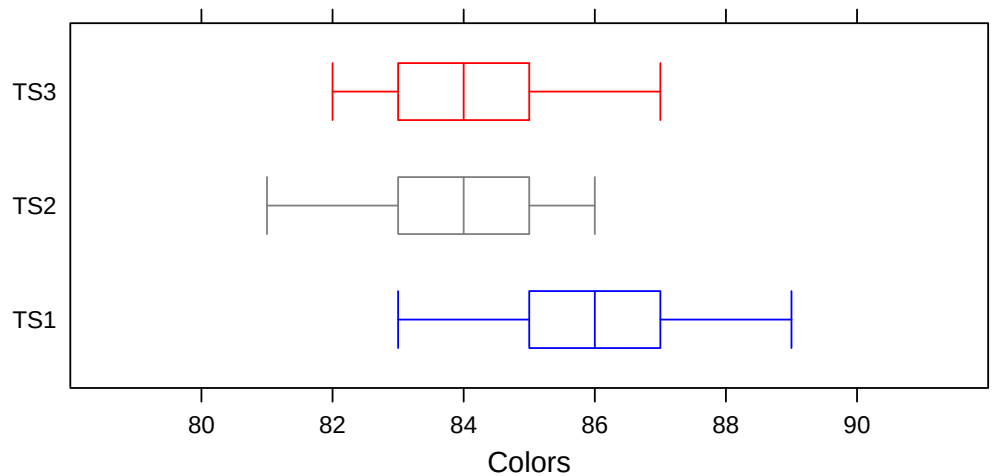
```
## load the data
> load("results.rda")
> levels(DATA$instance)
 [1] "queen4_4.txt" "queen5_5.txt" "queen6_6.txt" "queen7_7.txt"
 [5] "queen8_8.txt" "queen9_9.txt" "queen10_10.txt" "queen11_11.txt"
 [9] "queen12_12.txt" "queen13_13.txt" "queen14_14.txt" "queen15_15.txt"
[13] "queen16_16.txt" "queen17_17.txt" "queen18_18.txt" "queen19_19.txt"
[17] "queen20_20.txt" "queen21_21.txt" "queen22_22.txt" "queen23_23.txt"
[21] "queen24_24.txt" "queen25_25.txt" "queen26_26.txt" "queen27_27.txt"
[25] "queen28_28.txt" "queen29_29.txt" "queen30_30.txt" "queen31_31.txt"
[29] "queen32_32.txt"
> bwplot(reorder(alg, col, median)~col,data=DATA)
```



```
> bwplot(reorder(alg, col, median)~col|instance,data=DATA,as.table=TRUE)
```

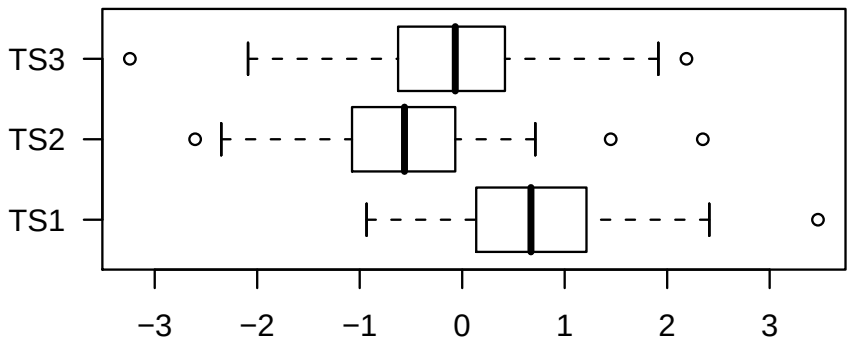



0.25
0.20
0.15
0.10
0.05
0.00
Density
80
82
84
86
88
90
Colors

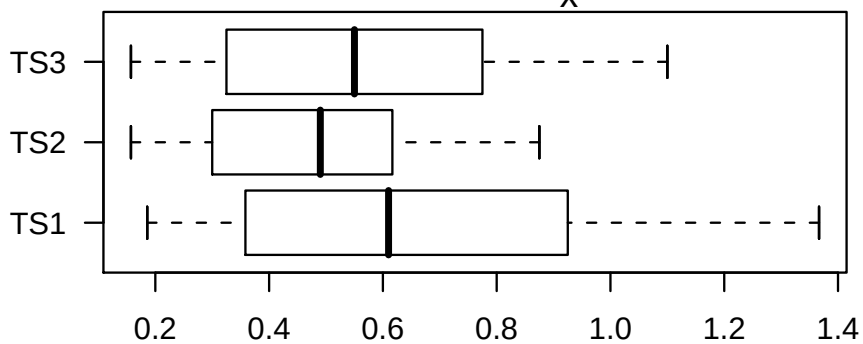

TS3
TS2
TS1
80
82
84
86
88
90
Colors

On a class of instances

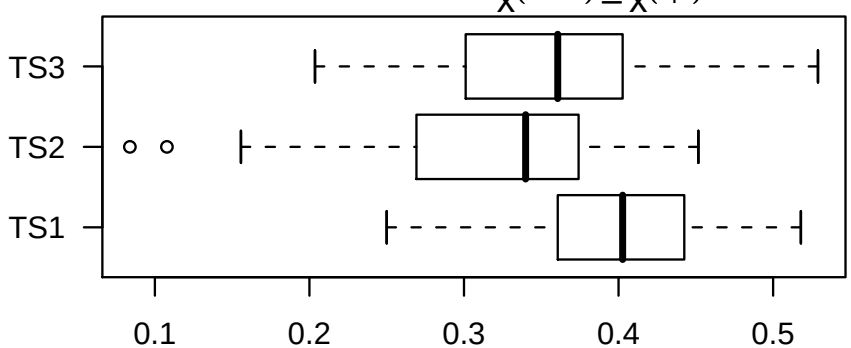
Standard error: $\frac{x - \bar{x}}{\sqrt{\sigma}}$



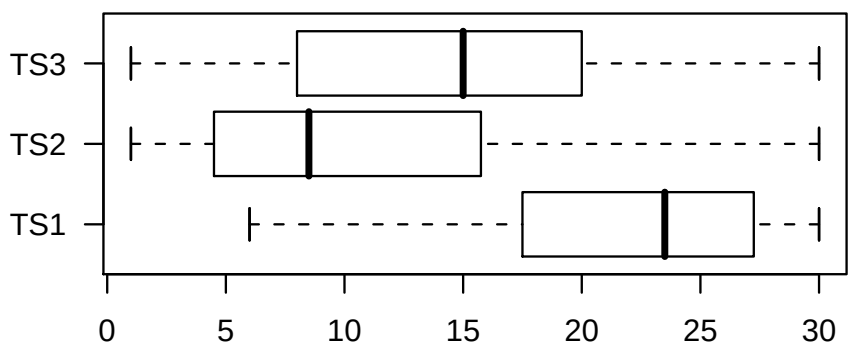
Relative error: $\frac{x - x^{(opt)}}{x^{(opt)}}$



Invariant error: $\frac{x - x^{(opt)}}{x^{(worst)} - x^{(opt)}}$

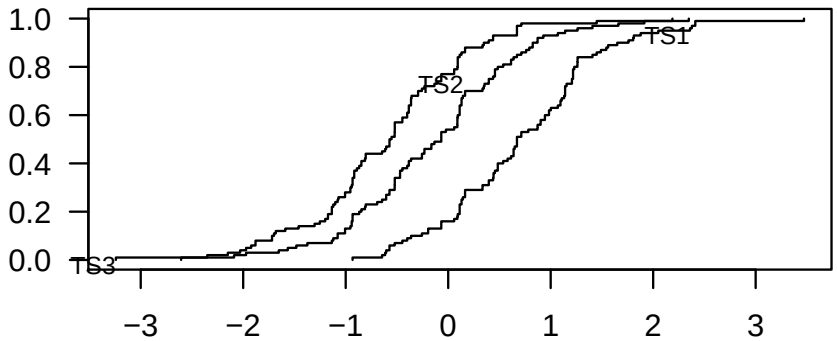


Ranks

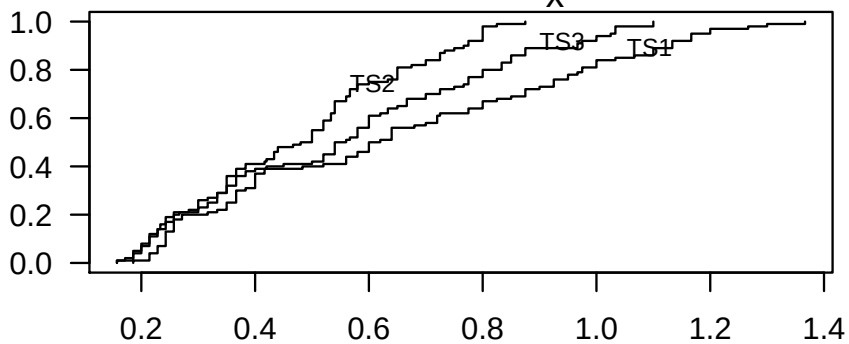


On a class of instances

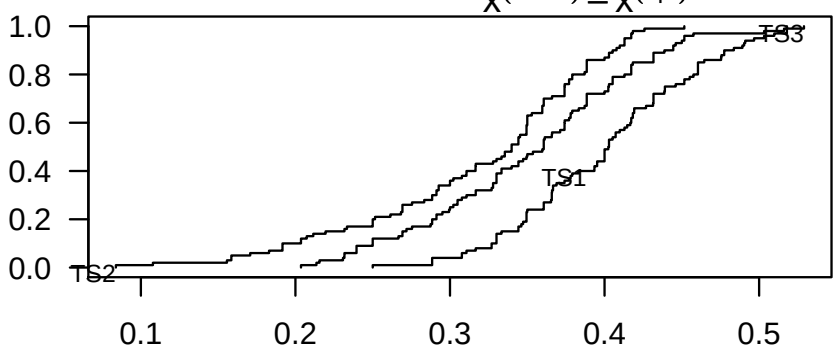
Standard error: $\frac{x - \bar{x}}{\sqrt{\sigma}}$



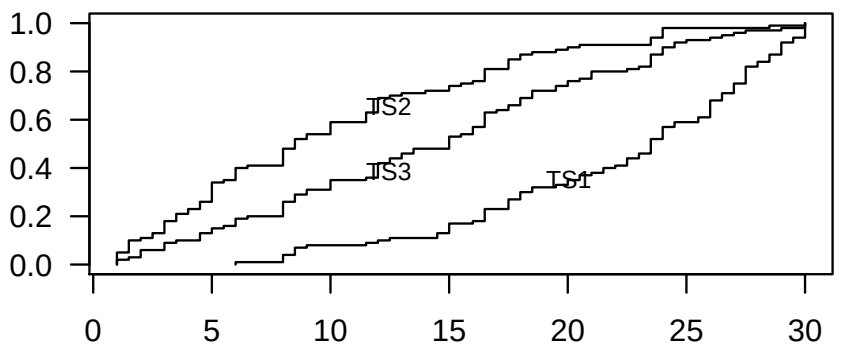
Relative error: $\frac{x - x^{(opt)}}{x^{(opt)}}$



Invariant error: $\frac{x - x^{(opt)}}{x^{(worst)} - x^{(opt)}}$



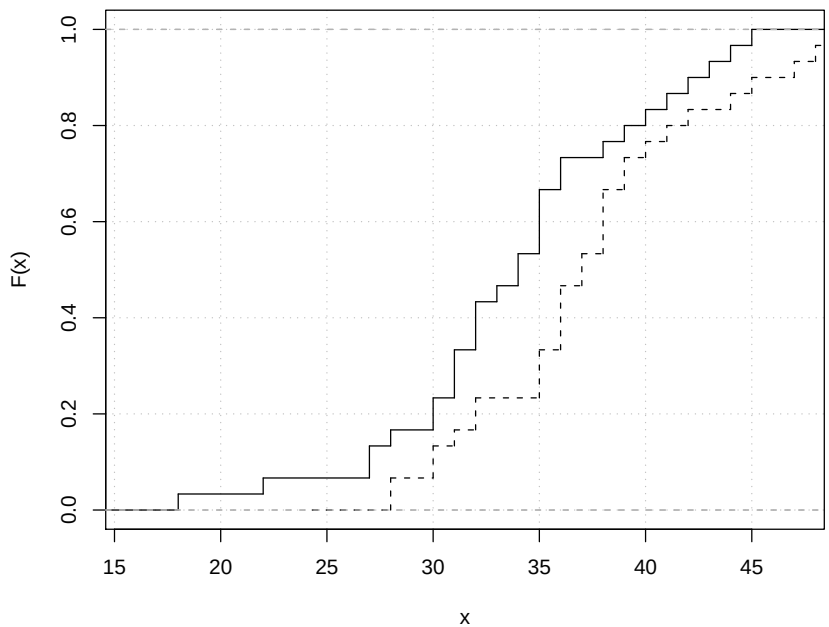
Ranks

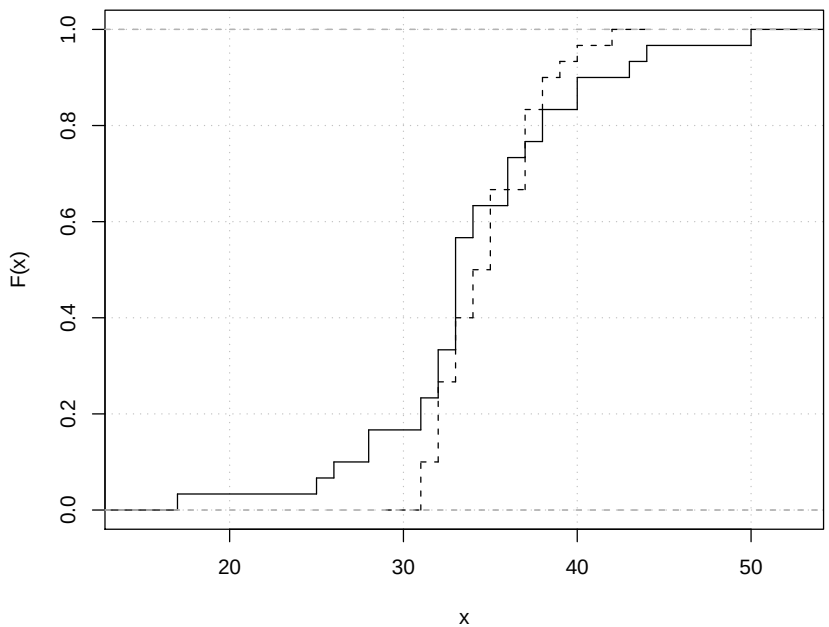


Stochastic Dominance

Definition: Algorithm $\mathcal{A}_1$ probabilistically dominates algorithm $\mathcal{A}_2$ on a problem instance, iff its CDF is always "below" that of $\mathcal{A}_2$, *i.e.*:

$$F_1(x) \leq F_2(x), \qquad \forall x \in X$$





R code behind the previous plots

We load the data and plot the comparative boxplot for each instance.

```
> load("TS.class-G.dataR")
> G[1:5,]
  alg inst run sol time.last.imp tot.iter parz.iter exit.iter exit.time opt
1 TS1 G-1000-0.5-30-1.1.col 1 59 9.900619 5955 442 5955 10.02463 30
2 TS1 G-1000-0.5-30-1.1.col 2 64 9.736608 3880 130 3958 10.00062 30
3 TS1 G-1000-0.5-30-1.1.col 3 64 9.908618 4877 49 4877 10.03263 30
4 TS1 G-1000-0.5-30-1.1.col 4 68 9.948622 6996 409 6996 10.07663 30
5 TS1 G-1000-0.5-30-1.1.col 5 63 9.912620 3986 52 3986 10.04063 30
>
> library(lattice)
> bwplot(alg ~ sol | inst,data=G)
```

If we want to make an aggregate analysis we have the following choices:

- maintain the raw data,
- transform data in standard error,
- transform the data in relative error,
- transform the data in an invariant error,
- transform the data in ranks.

Maintain the raw data

```
> par(mfrow=c(3,2),las=1,font.main=1,mar=c(2,3,3,1))
> #original data
> boxplot(sol~alg,data=G,horizontal=TRUE,main="Original data")
```

Transform data in standard error

```
> #standard error
> T1 <- split(G$sol,list(G$inst))
> T2 <- lapply(T1,scale,center=TRUE,scale=TRUE)
> T3 <- unsplit(T2,list(G$inst))
> T4 <- split(T3,list(G$alg))
> T5 <- stack(T4)
> boxplot(values~ind,data=T5,horizontal=TRUE,main=expression(paste("Standard error: ",
      frac(x-bar(x),sqrt(sigma)))))
> library(latticeExtra)
> ecdfplot(~values,group=ind,data=T5,main=expression(paste("Standard error:
",frac(x-bar(x),sqrt(sigma)))))

> #standard error
> G$scale <- 0
> split(G$scale, G$inst) <- lapply(split(G$sol, G$inst), scale,center=TRUE,scale=TRUE)
```

Transform the data in relative error

```
> #relative error
> G$err2 <- (G$sol-G$opt)/G$opt
> boxplot(err2~alg,data=G,horizontal=TRUE,main=expression(paste("Relative error: ",
      frac(x-x^(opt),x^(opt)))))
> ecdfplot(G$err2,group=G$alg,main=expression(paste("Relative error: ",frac(x-x^(opt),
      x^(opt)))))
```

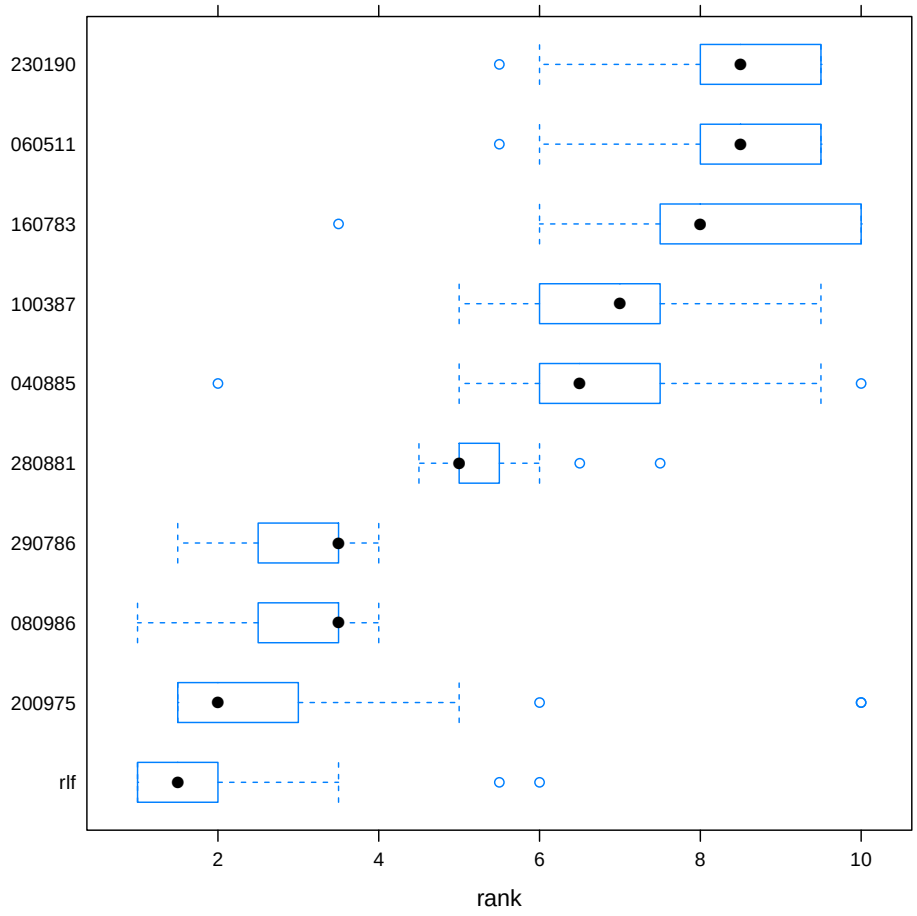
Transform the data in an invariant error
We use as surrogate of x^{worst} the median solution returned by the simplest algorithm for the graph coloring, that is, the ROS heuristic.

```
> #error 3
> load("ROS.class-G.dataR")
> F1 <- aggregate(F$sol,list(inst=F$inst),median)
> F2 <- split(F1$x,list(F1$inst))
> G$ref <- sapply(G$inst,function(x) F2[[x]])
> G$err3 <- (G$sol-G$opt)/(G$ref-G$opt)
> boxplot(err3~alg,data=G,horizontal=TRUE,main=expression(paste("Invariant error: ",
    frac(x-x^(opt),x^(worst)-x^(opt)))))
> ecdfplot(G$err3,group=G$alg,main=expression(paste("Invariant error: ",frac(x-x^(opt)
    ,x^(worst)-x^(opt)))))
```

Transform the data in ranks

```
> #rank
> G$rank <- G$sol
> split(G$rank, G$inst) <- lapply(split(G$sol, D$inst), rank)
> bwplot(rank~reorder(alg,rank,median),data=G,horizontal=TRUE,main="Ranks")
> ecdfplot(rank,group=alg,data=G,main="Ranks")
```

```
> ## Let's make the ranks of the colors
> T1 <- split(DATA["col"], DATA["instance"])
> T2 <- lapply(T1, rank, na.last = "keep")
> T3 <- unsplit(T2, DATA["instance"])
> DATA$rank <- T3
>
> ## we plot the ranks for an aggregate analysis
> ## reoder sort the factor algorithm by median values
> bwplot(reorder(alg, rank, median) ~ rank, data = DATA)
```



Scenarios

A. Single-pass heuristics

B. Asymptotic heuristics:
   Two approaches:

   1. Univariate

      1.a Time as an external parameter decided *a priori*
      1.b Solution quality as an external parameter decided *a priori*

   2. Cost dependent on running time:

Scenario B

Asymptotic heuristics

There are two approaches:

1.b. Solution quality as an external parameter decided *a priori*. The algorithm is halted when quality is reached.

Deterministic case: $\mathcal{A}^\infty$ on class C_Π finds a solution in running time t.

The performance of $\mathcal{A}^\infty$ on class C_Π is the scalar $y = t$.

Randomized case: $\mathcal{A}^\infty$ on class C_Π finds a solution in running time T, where T is a random variable.

The performance of $\mathcal{A}^\infty$ on class C_Π is the univariate $Y = T$.

Dealing with Censored Data

Asymptotic heuristics, Approach 1.b

- ▷ Heuristic $\mathcal{A}^{\dashv}$ stopped before completion or $\mathcal{A}^{\infty}$ truncated (always the case)
- ▶ Interest: determining whether a prefixed goal (optimal/feasible) has been reached

The computational effort to attain the goal can be specified by a cumulative distribution function $F(t) = P(T < t)$ with T in $[0, \infty)$.

If in a run i we stop the algorithm at time L_i then we have a Type I right censoring, that is, we know either

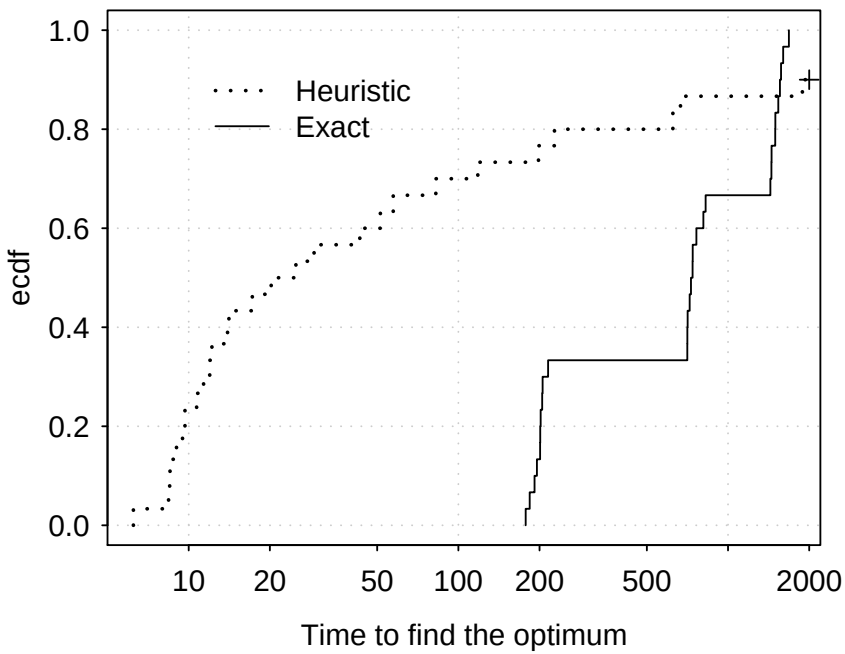
- T_i if $T_i \leq L_i$
- or $T_i \geq L_i$.

Hence, for each run i we need to record $\min(T_i, L_i)$ and the indicator variable for observed optimal/feasible solution attainment, $\delta_i = I(T_i \leq L_i)$.

Example

Asymptotic heuristics, Approach 1.b: Example

- An exact *vs* an heuristic algorithm for the *2-edge-connectivity augmentation problem*.
- Interest: time to find the optimum on different instances.



Uncensored:

$$F(t) = \frac{\#\text{ runs} < t}{n}$$

Censored:

$$F(t) = \frac{\#\text{ runs} < t}{n}$$

Scenarios

A. Single-pass heuristics

B. Asymptotic heuristics:
   Two approaches:

   1. Univariate

      1.a Time as an external parameter decided *a priori*
      1.b Solution quality as an external parameter decided *a priori*

   2. Cost dependent on running time:

Scenario B

Asymptotic heuristics

There are two approaches:

2. Cost dependent on running time:

Deterministic case: $\mathcal{A}^\infty$ on π returns a current best solution x at each observation in $t_1, \ldots, t_k$.

The performance of $\mathcal{A}^\infty$ on π is the profile indicated by the vector $\vec{y} = \{x(t_1), \ldots, x(t_k)\}$.

Randomized case: $\mathcal{A}^\infty$ on π produces a monotone stochastic process in solution cost $X(\tau)$ with any element dependent on the predecessors.

The performance of $\mathcal{A}^\infty$ on π is the multivariate $\vec{Y} = (X(t_1), X(t_2), \ldots, X(t_k))$.

Example

Scenario:

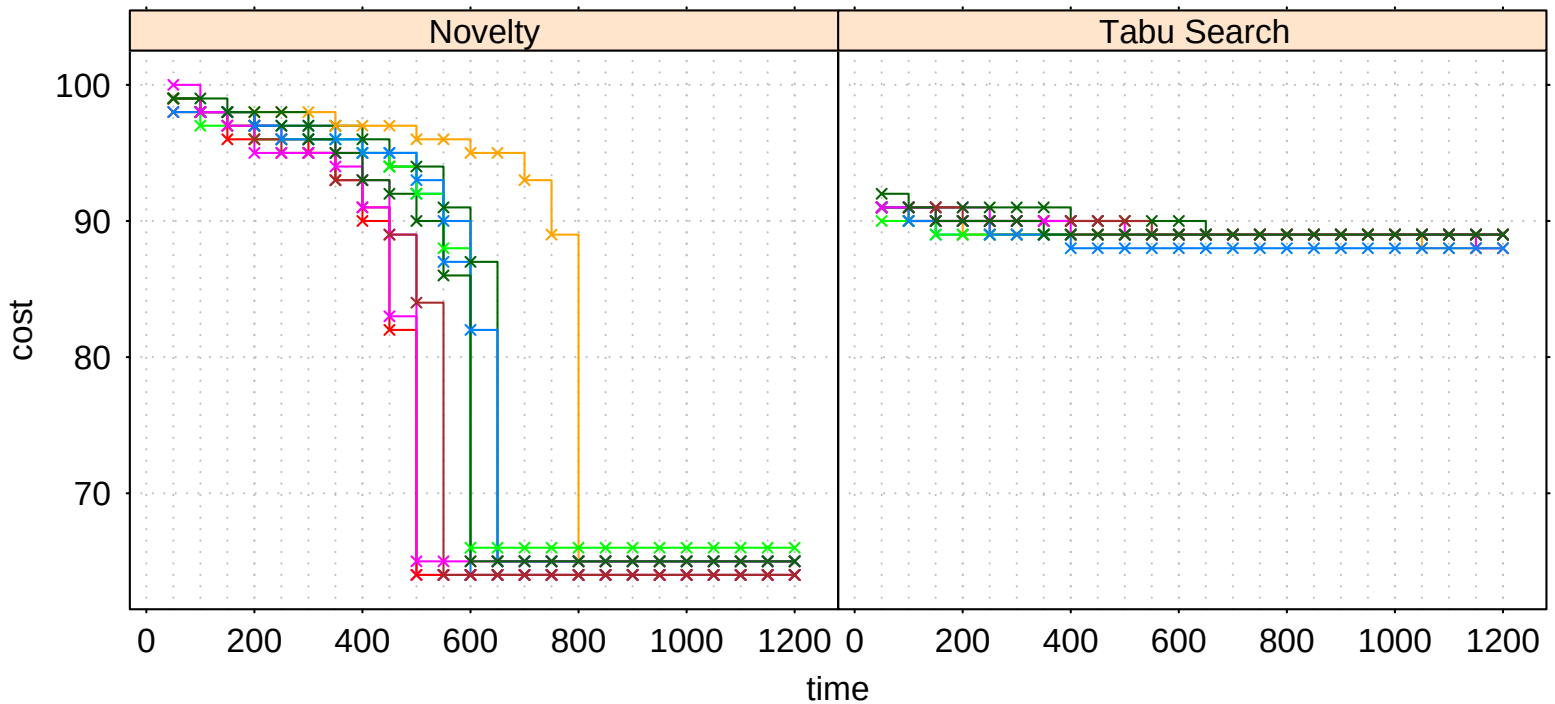
- ▷ 3 heuristics $\mathcal{A}_1^\infty$, $\mathcal{A}_2^\infty$, $\mathcal{A}_3^\infty$ on instance π.
- ▷ single instance hence no data transformation.
- ▷ r runs
- ▶ Interest: inspecting solution cost over running time to determine whether the comparison varies over time intervals

Tools:

- Quality profiles

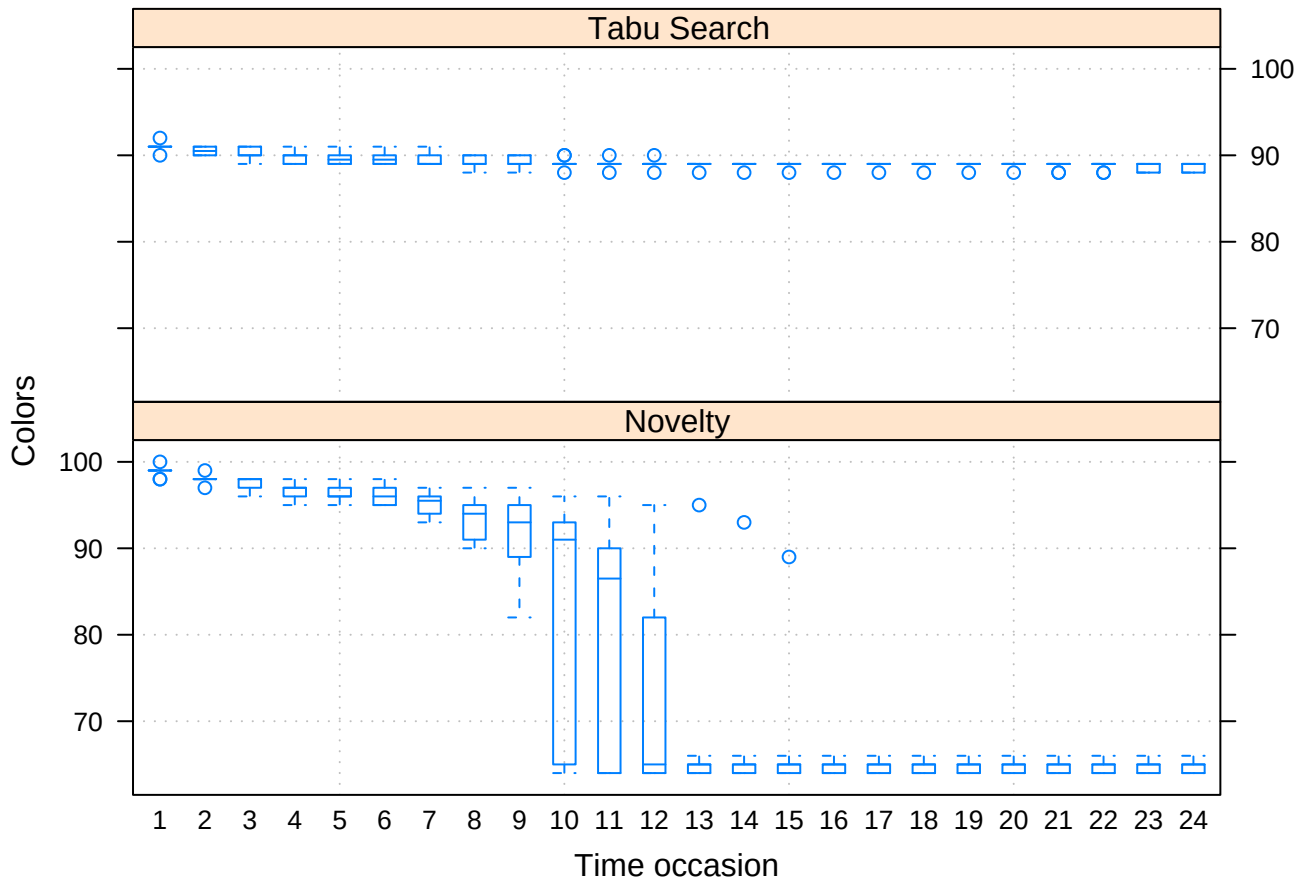
The performance is described by multivariate random variables of the kind $\vec{Y} = \{Y(t_1), Y(t_2), \ldots, Y(l_k)\}$.

Sampled data are of the form $\vec{Y_i} = \{Y_i(t_1), Y_i(t_2), \ldots, Y_i(t_k)\}$, $i = 1, \ldots, 10$ (10 runs per algorithm on one instance)



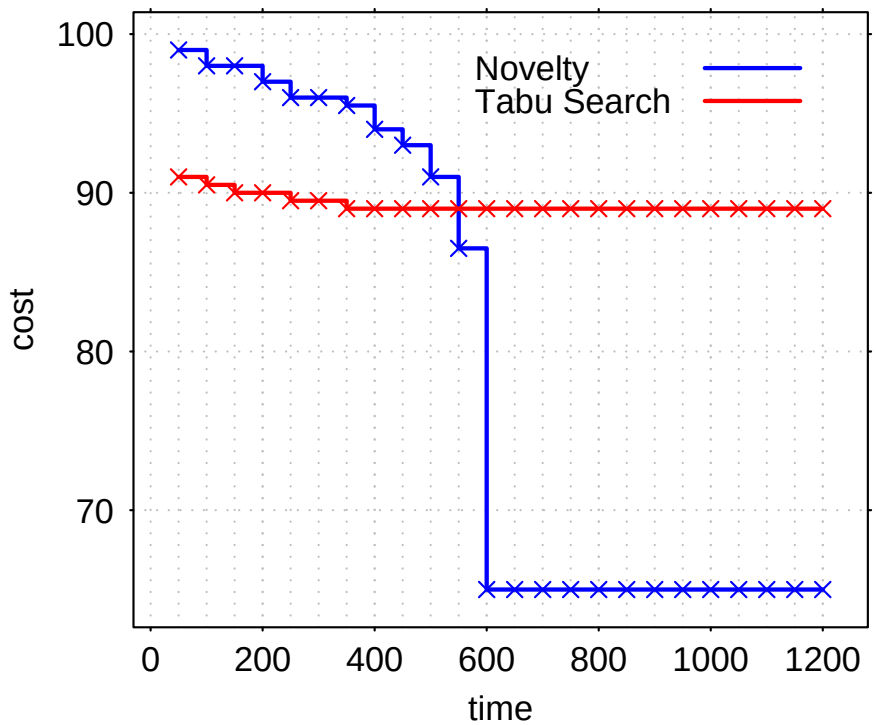
The performance is described by multivariate random variables of the kind $\vec{Y} = \{Y(t_1), Y(t_2), \ldots, Y(l_k)\}$.

Sampled data are of the form $\vec{Y_i} = \{Y_i(t_1), Y_i(t_2), \ldots, Y_i(t_k)\}$, $i = 1, \ldots, 10$ (10 runs per algorithm on one instance)



The performance is described by multivariate random variables of the kind $\vec{Y} = \{Y(t_1), Y(t_2), \ldots, Y(l_k)\}$.

Sampled data are of the form $\vec{Y_i} = \{Y_i(t_1), Y_i(t_2), \ldots, Y_i(t_k)\}$, $i = 1, \ldots, 10$ (10 runs per algorithm on one instance)



The median behavior of the two algorithms

Summary

Visualize your data for your analysis and for communication to others

Explore your data:

- make plots: histograms, boxplots, empirical cumulative distribution functions, correlation/scatter plots
- look at the numerical data and interpret them in practical terms: computation times, distance from optimum
- look for patterns

All the above both at a single instance level and at an aggregate level.

Making Plots

http://algo2.iti.uni-karlsruhe.de/sanders/courses/bergen/bergenPresenting.pdf

[Sanders, 2002]

- Should the experimental setup from the exploratory phase be redesigned to increase conciseness or accuracy?
- What parameters should be varied? What variables should be measured?
- How are parameters chosen that cannot be varied?
- Can tables be converted into curves, bar charts, scatter plots or any other useful graphics?
- Should tables be added in an appendix?
- Should a 3D-plot be replaced by collections of 2D-curves?
- Can we reduce the number of curves to be displayed?
- How many figures are needed?
- Should the x-axis be transformed to magnify interesting subranges?

- Should the x-axis have a logarithmic scale? If so, do the x-values used for measuring have the same basis as the tick marks?
- Is the range of x-values adequate?
- Do we have measurements for the right x-values, i.e., nowhere too dense or too sparse?
- Should the y-axis be transformed to make the interesting part of the data more visible?
- Should the y-axis have a logarithmic scale?
- Is it misleading to start the y-range at the smallest measured value? (if not too much space wasted start from 0)
- Clip the range of y-values to exclude useless parts of curves?
- Can we use banking to 45^o?
- Are all curves sufficiently well separated?
- Can noise be reduced using more accurate measurements?
- Are error bars needed? If so, what should they indicate? Remember that measurement errors are usually not random variables.

- Connect points belonging to the same curve.
- Only use splines for connecting points if interpolation is sensible.
- Do not connect points belonging to unrelated problem instances.
- Use different point and line styles for different curves.
- Use the same styles for corresponding curves in different graphs.
- Place labels defining point and line styles in the right order and without concealing the curves.
- Give axis units
- Captions should make figures self contained.
- Give enough information to make experiments reproducible.
- Golden ratio rule: make the graph wider than higher [Tufte 1983].
- Rule of 7: show at most 7 curves (omit those clearly irrelevant).
- Avoid: explaining axes, connecting unrelated points by lines, cryptic abbreviations, microscopic lettering, pie charts

References

Benoist T., Estellon B., Gardi F., Megel R., and Nouioua K. (2011). **LocalSolver 1.x: a black-box local-search solver for 0-1 programming**. *4OR*, 9(3), pp. 299–316.

Birattari M., Stützle T., Paquete L., and Varrentrapp K. (2002). **A racing algorithm for configuring metaheuristics**. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2002)*, edited by L. et al., pp. 11–18. Morgan Kaufmann Publishers, New York.

Chiarandini M. (2009). **Experimental analysis of optimization heuristics using R**. Lecture notes available at `http://www.imada.sdu.dk/~marco/Teaching/Files/Rnotes.pdf`.

Sanders P. (2002). **Presenting data from experiments in algorithmics**. In *Experimental Algorithmics – From Algorithm Design to Robust and Efficient Software,*, vol. 2547 of **LNCS**, pp. 181–196. Springer.

Van Hentenryck P. and Michel L. (2005). **Constraint-Based Local Search**. The MIT Press, Cambridge, USA.