Georgia Tech

CSE 6140/ CX 4140:

Computational Science and Engineering ALGORITHMS

Instructor: Anne Benoit

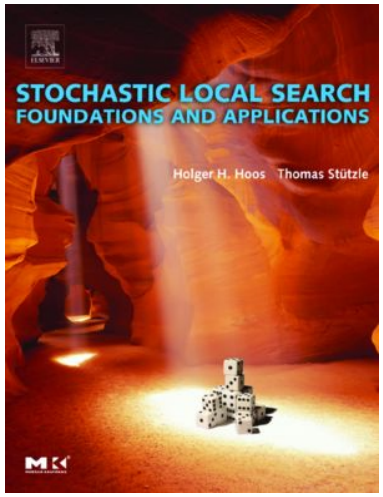
Visiting Associate Professor, CSE

Based on slides by Bistra Dilkina and Holger Hoos

Georgia Tech

CSE 6140/ CX 4140

Empirical Analysis of Algorithms [SLS4]



textbook: STOCHASTIC LOCAL SEARCH FOUNDATIONS AND APPLICATIONS

based on slides by Holger Hoos

Georgia Tech

Las Vegas Algorithms

SLS algorithms are typically *incomplete*: there is no guarantee that an (optimal) solution for a given problem instance will eventually be found.

But: For decision problems, any solution returned is guaranteed to be correct.

Also: The run-time required for finding a solution (in case one is found) is subject to random variation.

$\rightsquigarrow$ These properties define the class of *(generalised) Las Vegas algorithms*, of which SLS algorithms are a subset.

Definition: (Generalised) Las Vegas Algorithm (LVA)

An algorithm A for a problem class Π is a *(generalised) Las Vegas algorithm (LVA)* iff it has the following properties:

(1) If for a given problem instance $\pi \in \Pi$, algorithm A terminates returning a solution s, s is guaranteed to be a correct solution of π.

(2) For any given instance $\pi \in \Pi$, the run-time of A applied to π is a random variable $RT_{A,\pi}$.

Note: This is a slight generalisation of the definition of a Las Vegas algorithm known from theoretical computing science (our definition includes algorithms that are not guaranteed to return a solution).

Application scenarios and evaluation criteria (1)

Evaluation criteria for LVAs depend on the application context:

- **Type 1:** No time limits given, algorithm can be run until a solution is found (off-line computations, non-realtime environments, *e.g.*, configuration of production facility).

  $\rightsquigarrow$ evaluation criterion: expected run-time

- **Type 2:** Hard time limit t_{max} for finding solution; solutions found later are useless (real-time environments with strict deadlines, *e.g.*, dynamic task scheduling or on-line robot control).

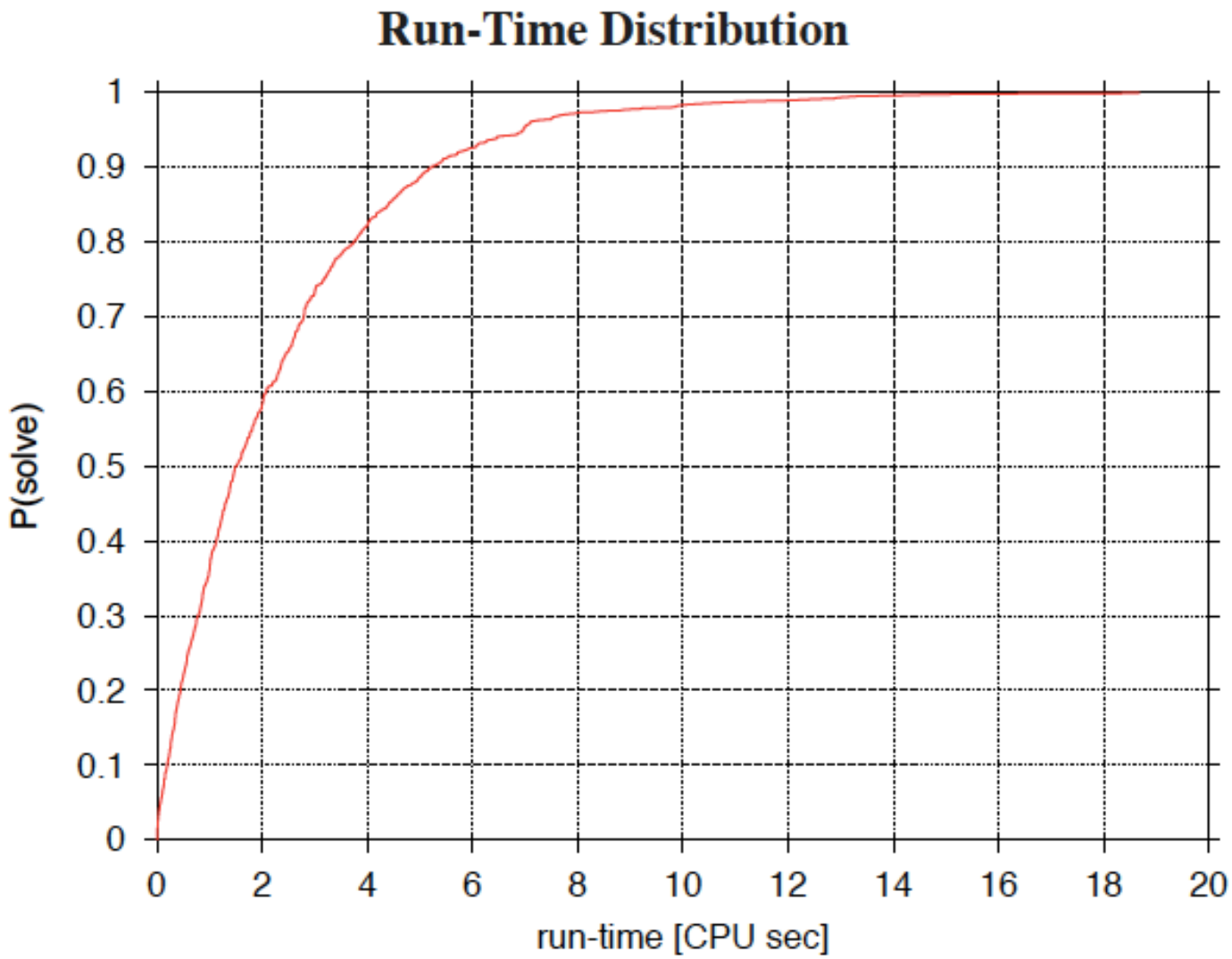
  $\rightsquigarrow$ evaluation criterion: solution probability at time t_{max}

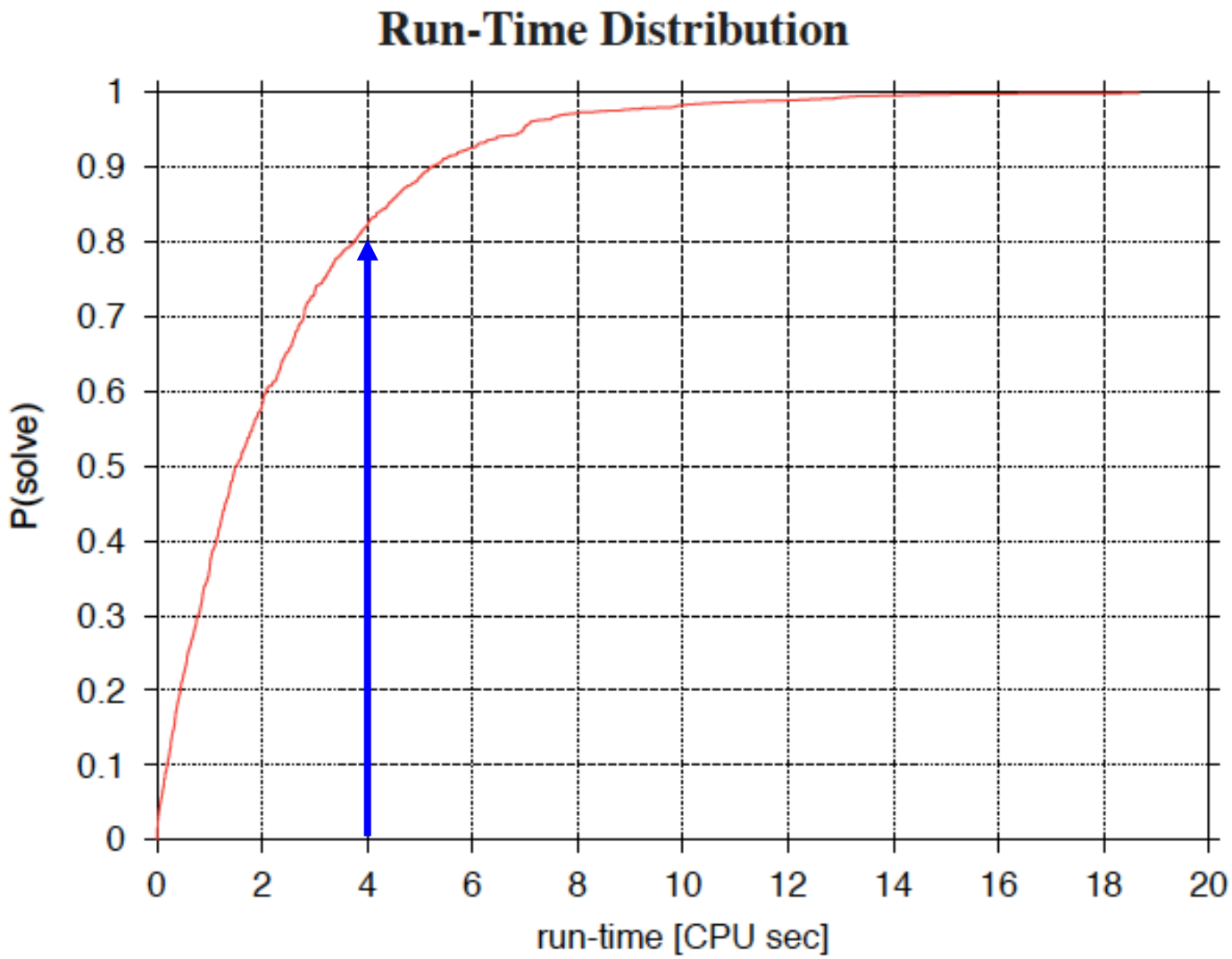
Georgia Tech

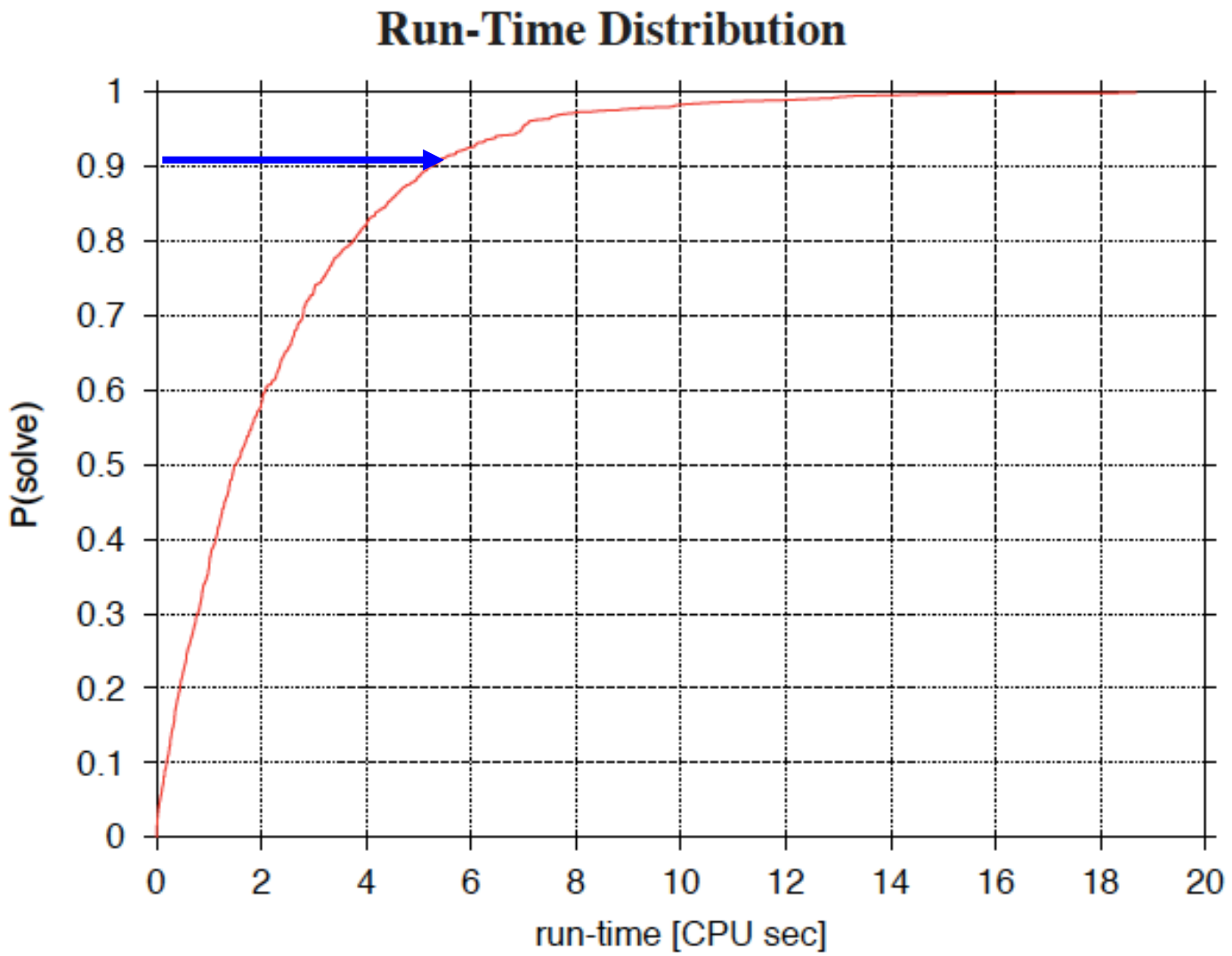
Definition: **Run-Time Distribution (1)**

Given Las Vegas algorithm A for decision problem Π:

- The *success probability* $P_s(RT_{A,\pi} \leq t)$ is the probability that A finds a solution for a soluble instance $\pi \in \Pi$ in time $\leq t$.
- The *run-time distribution (RTD) of A on π* is the probability distribution of the random variable $RT_{A,\pi}$.
- The *run-time distribution function* $rtd : \mathbb{R}^+ \mapsto [0, 1]$, defined as $rtd(t) = P_s(RT_{A,\pi} \leq t)$, completely characterises the RTD of A on π.


Run-Time Distribution
P(solve)
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
2
4
6
8
10
12
14
16
18
20
run-time [CPU sec]


Run-Time Distribution
P(solve)
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
2
4
6
8
10
12
14
16
18
20
run-time [CPU sec]


Run-Time Distribution
P(solve)
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
2
4
6
8
10
12
14
16
18
20
run-time [CPU sec]

Georgia Tech

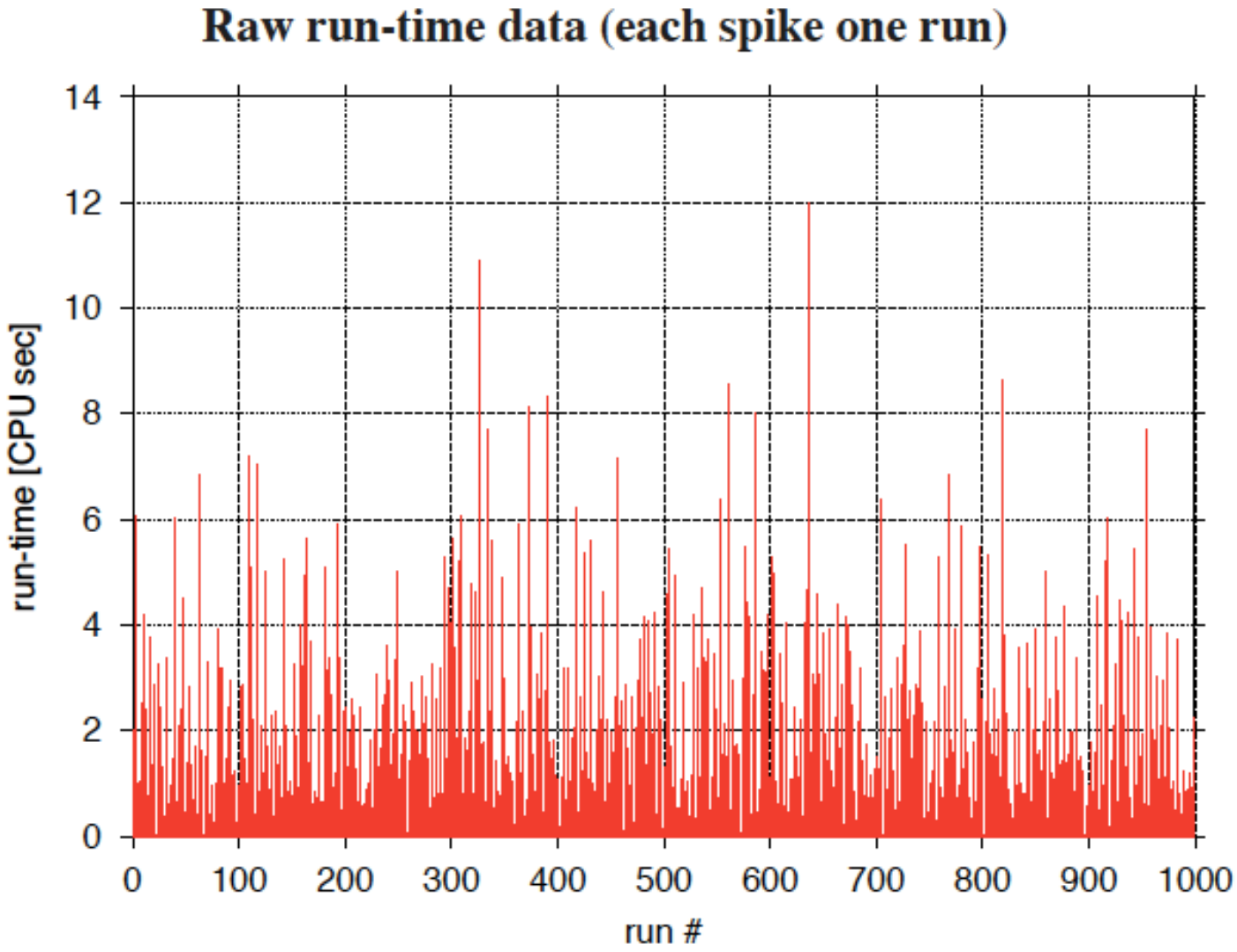
Empirically measuring RTDs

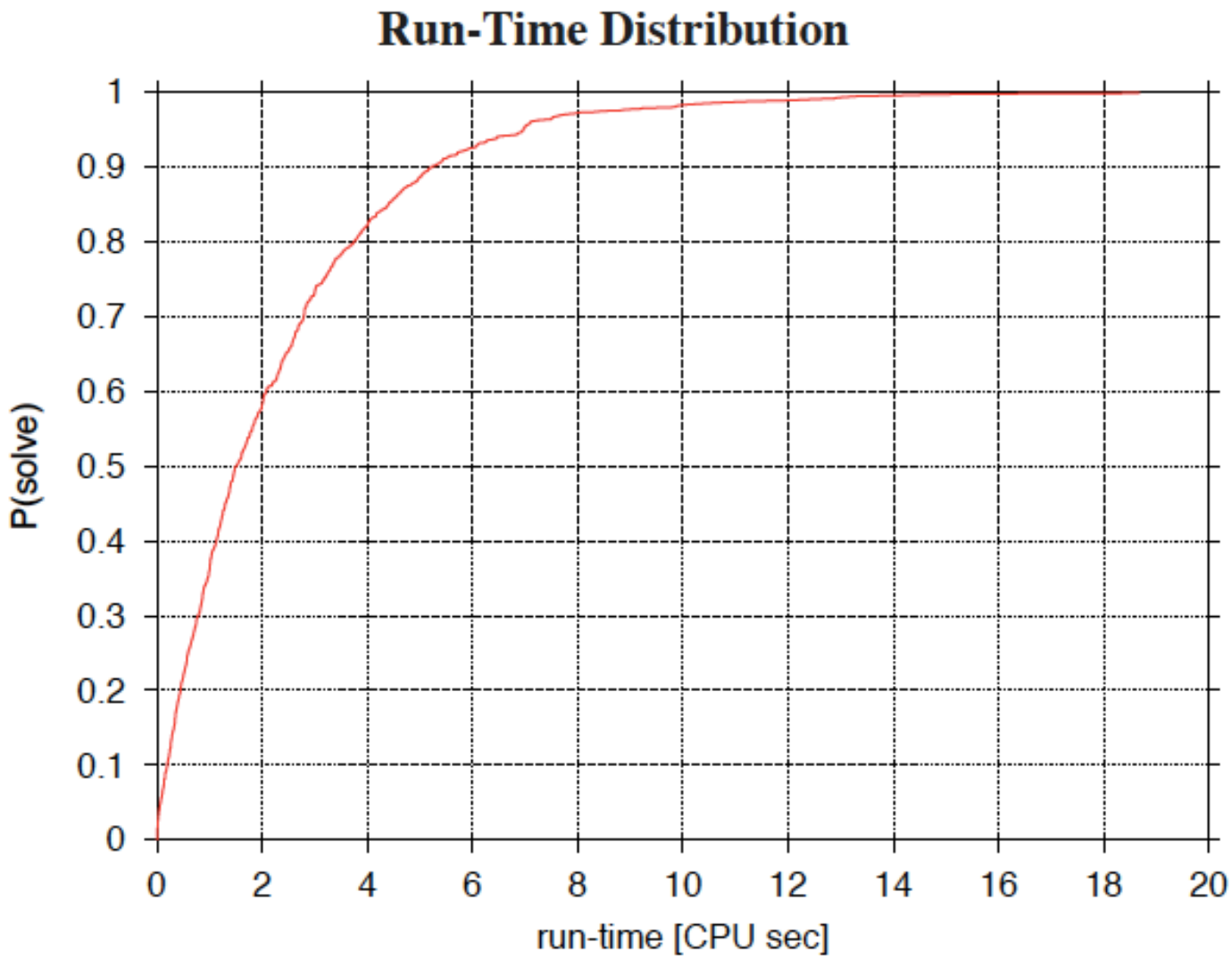
- Except for very simple algorithms, where they can be derived analytically, RTDs are measured empirically.
- Empirical RTDs are approximations of an algorithm's true RTD.
- Empirical RTDs are determined from a number of independent, successful runs of the algorithm on a given problem instance (*samples of theoretical RTD*).
- Higher numbers of runs (larger *sample sizes*) give more accurate approximations of a true RTD.

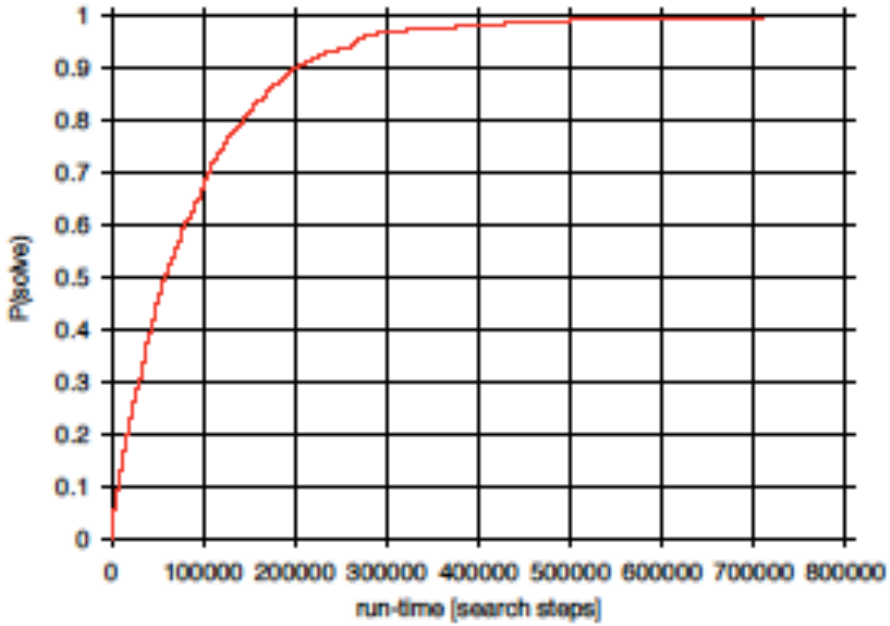
Georgia Tech

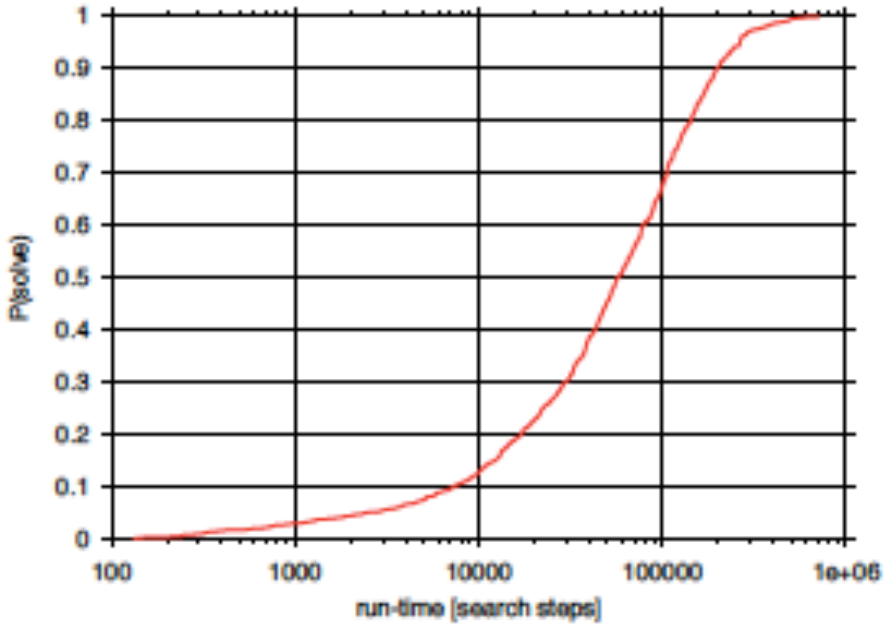
Protocol for obtaining the empirical RTD for an LVA A applied to a given instance π of a decision problem:

- Perform k independent runs of A on π with cutoff time t'. (For most purposes, k should be at least 50–100, and t' should be high enough to obtain at least a large fraction of successful runs.)
- Record number k' of successful runs, and for each run, record its run-time in a list L.
- Sort L according to increasing run-time; let $rt(j)$ denote the run-time from entry j of the sorted list ($j = 1, \ldots, k'$).
- Plot the graph $(rt(j), j/k)$, *i.e.*, the cumulative empirical RTD of A on π.


Raw run-time data (each spike one run)
run-time [CPU sec]
0
2
4
6
8
10
12
14
0
100
200
300
400
500
600
700
800
900
1000
run #


Run-Time Distribution
P(solve)
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
2
4
6
8
10
12
14
16
18
20
run-time [CPU sec]


P(solve)
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
100000
200000
300000
400000
500000
600000
700000
800000
run-time [search steps]


P(solve)
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
100
1000
10000
100000
1e+06
run-time [search steps]

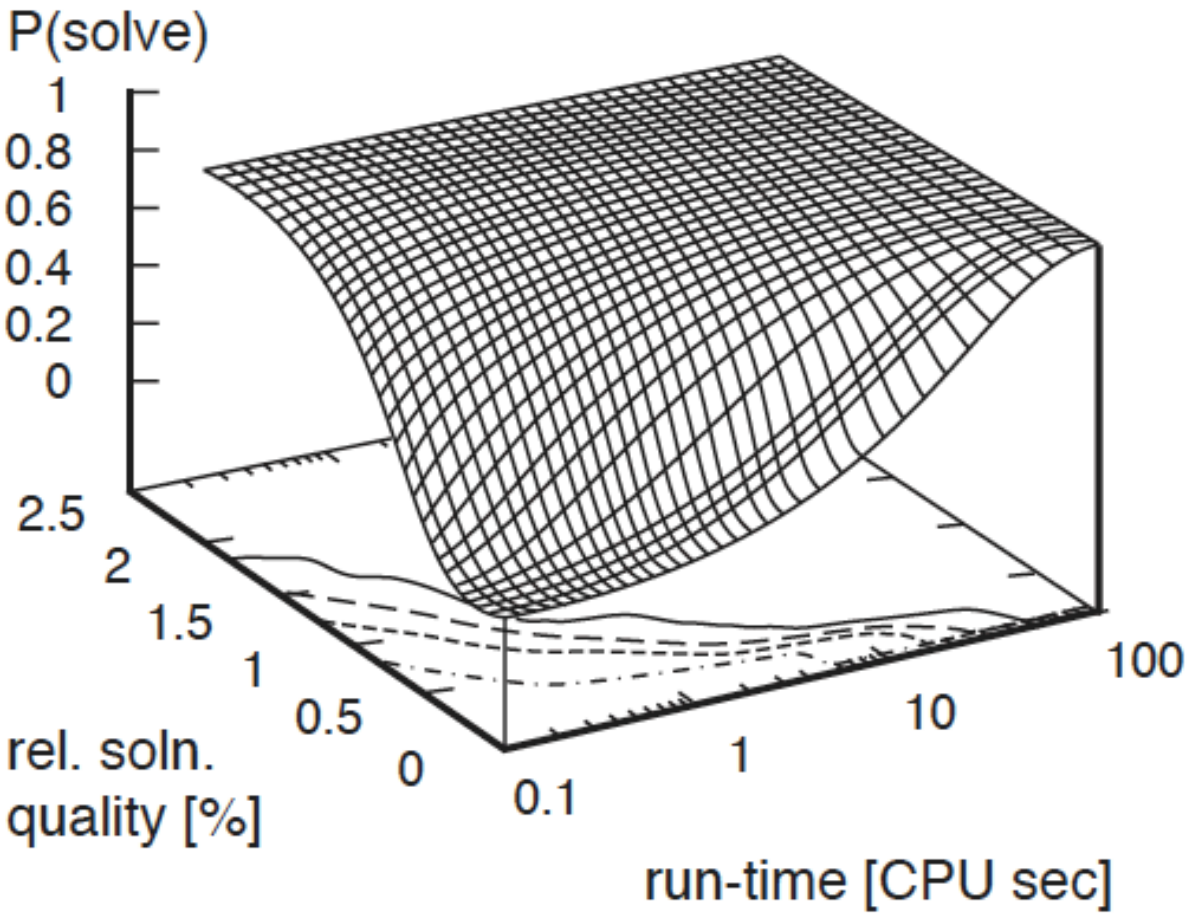
Georgia Tech

Definition: **Run-Time Distribution (2)**

Given OLVA A' for optimisation problem Π':

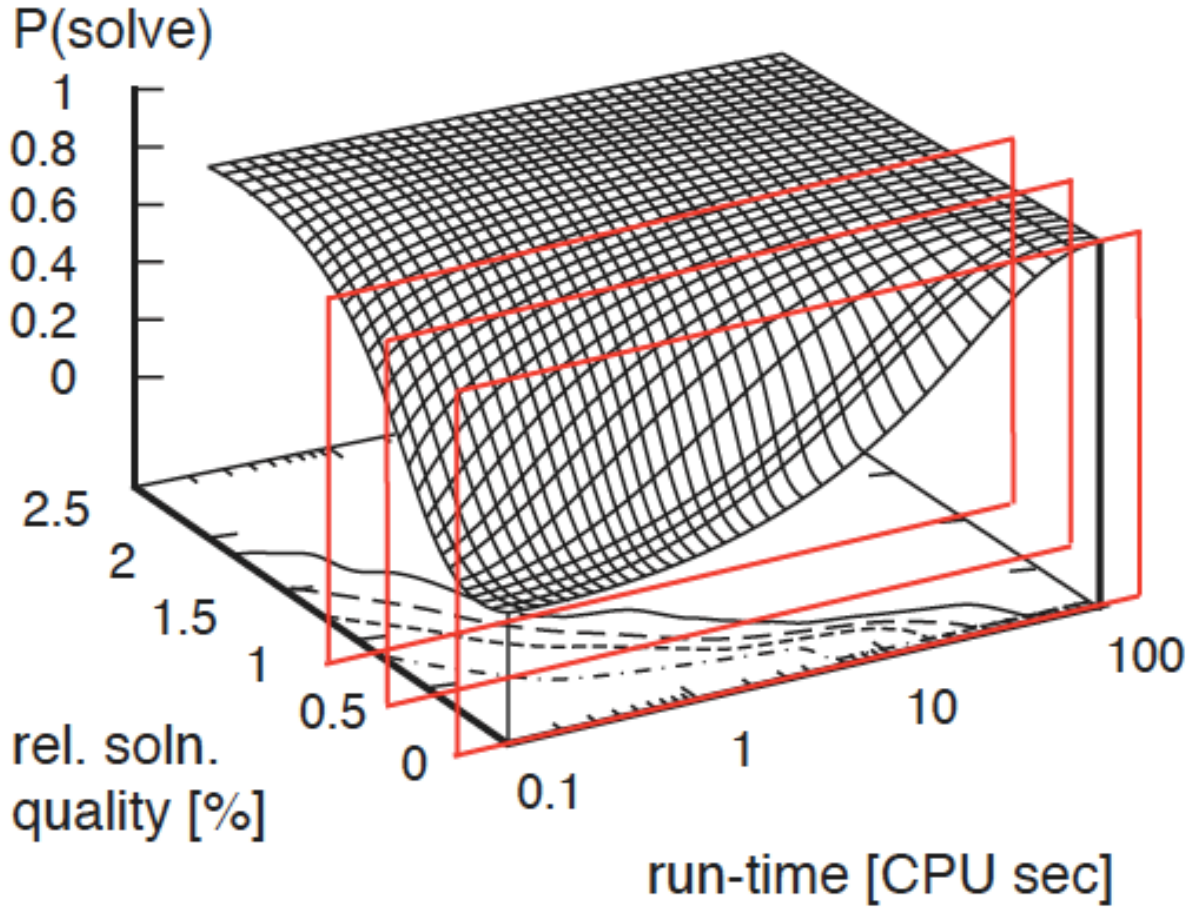
- The *success probability* $P_s(RT_{A',\pi'} \leq t, SQ_{A',\pi'} \leq q)$ is the probability that A' finds a solution for a soluble instance $\pi' \in \Pi'$ of quality $\leq q$ in time $\leq t$.
- The *run-time distribution (RTD) of A' on π'* is the probability distribution of the bivariate random variable $(RT_{A',\pi'}, SQ_{A',\pi'})$.
- The *run-time distribution function rtd* : $\mathbb{R}^+ \times \mathbb{R}^+ \mapsto [0,1]$, defined as $rtd(t,q) = P_s(RT_{A,\pi} \leq t, SQ_{A',\pi'} \leq q)$, completely characterises the RTD of A' on π'.

Typical run-time distribution for SLS algorithm applied to hard instance of combinatorial optimisation problem:



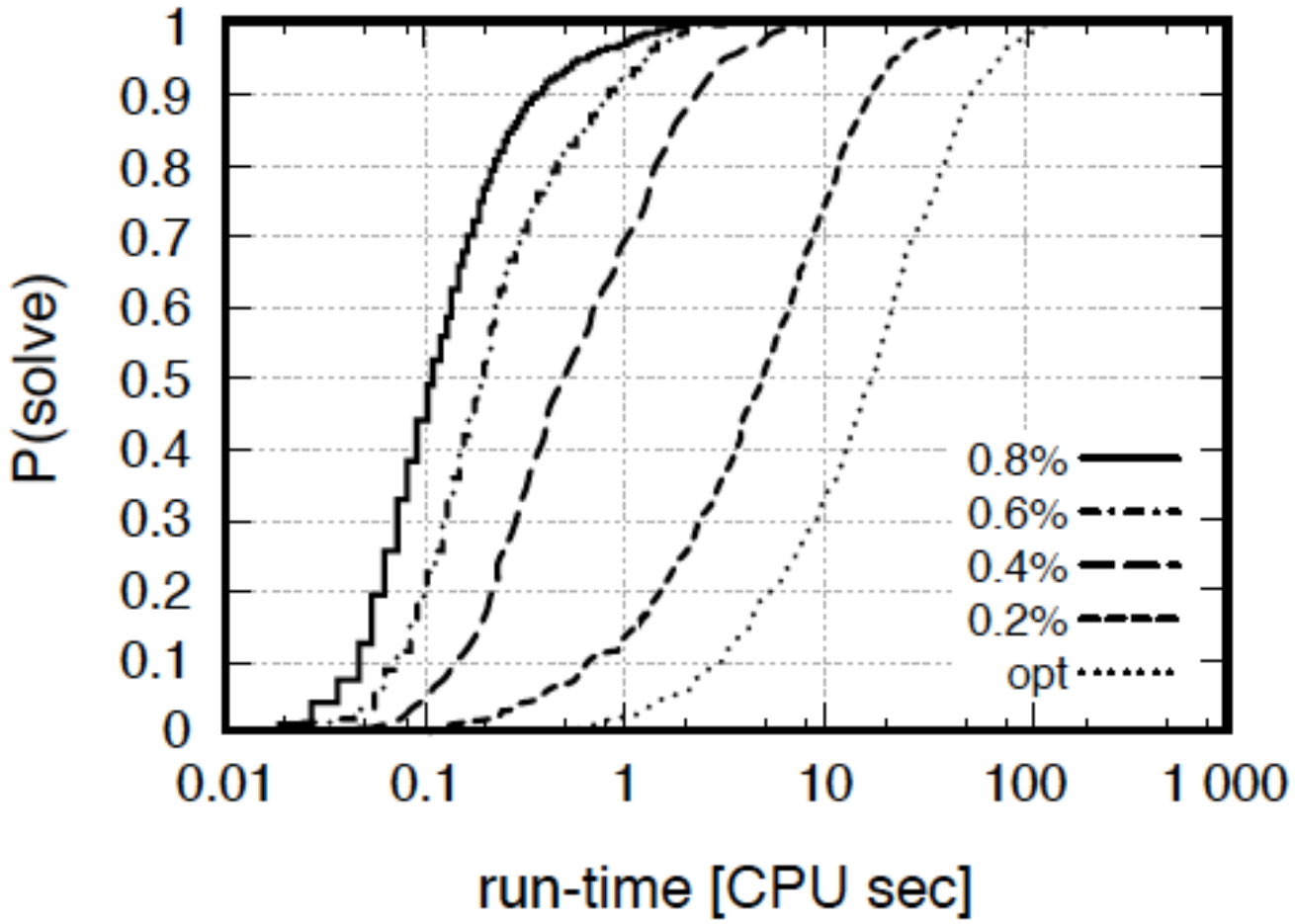
Georgia Tech

Typical run-time distribution for SLS algorithm applied to hard instance of combinatorial optimisation problem:



Georgia Tech

Qualified RTDs for various solution qualities:



Georgia Tech

Qualified run-time distributions (QRTDs)

- A *qualified run-time distribution (QRTD) of an OLVA A' applied to a given problem instance π' for solution quality q'* is a marginal distribution of the bivariate RTD $rtd(t, q)$ defined by:

$$qrtd_{q'}(t) := rtd(t, q') = P_s(RT_{A',\pi'} \leq t, SQ_{A',\pi'} \leq q').$$

Qualified run-time distributions (QRTDs)

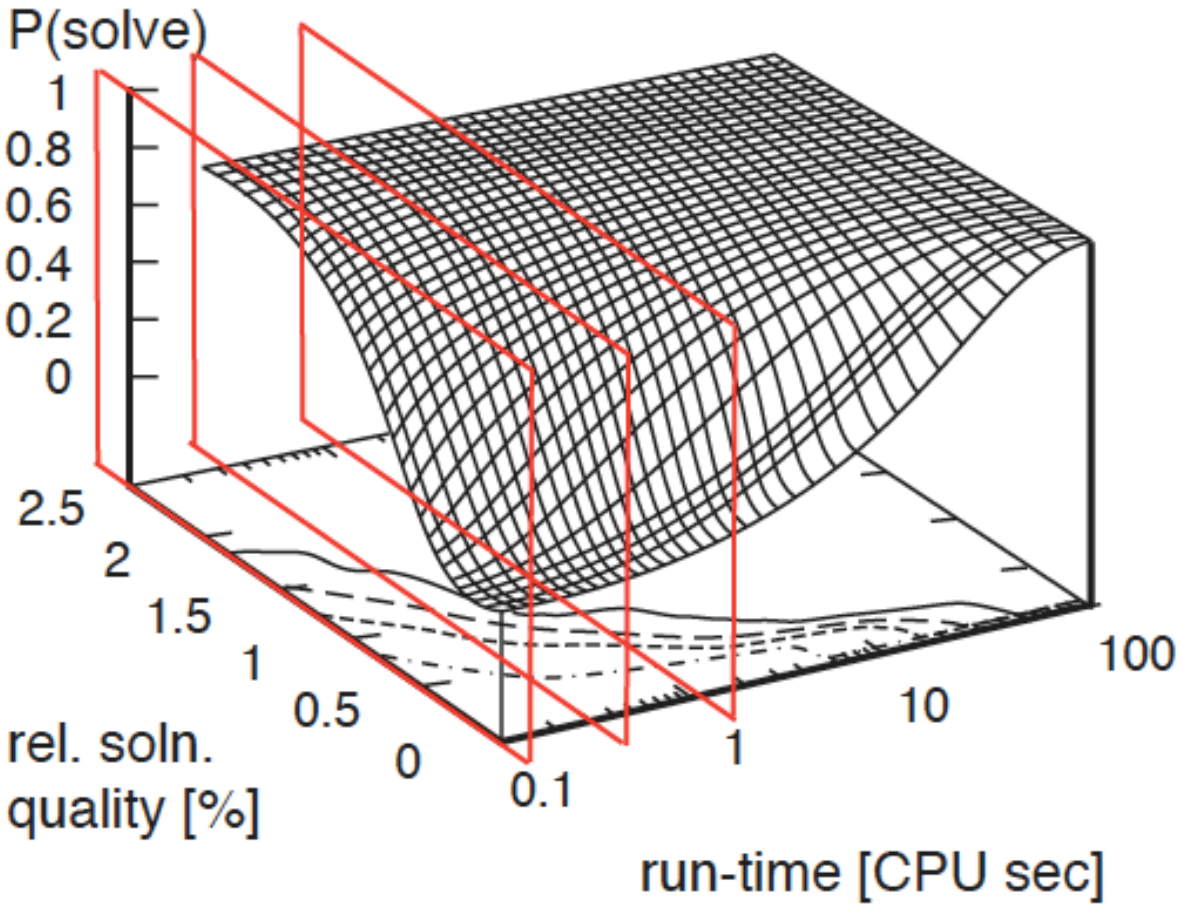
- A *qualified run-time distribution (QRTD) of an OLVA A' applied to a given problem instance π' for solution quality q'* is a marginal distribution of the bivariate RTD $rtd(t, q)$ defined by:

$$qrtd_{q'}(t) := rtd(t, q') = P_s(RT_{A',\pi'} \leq t, SQ_{A',\pi'} \leq q').$$

- QRTDs correspond to cross-sections of the two-dimensional bivariate RTD graph.
- QRTDs characterise the ability of a given SLS algorithm for a combinatorial optimisation problem to solve the associated decision problems.

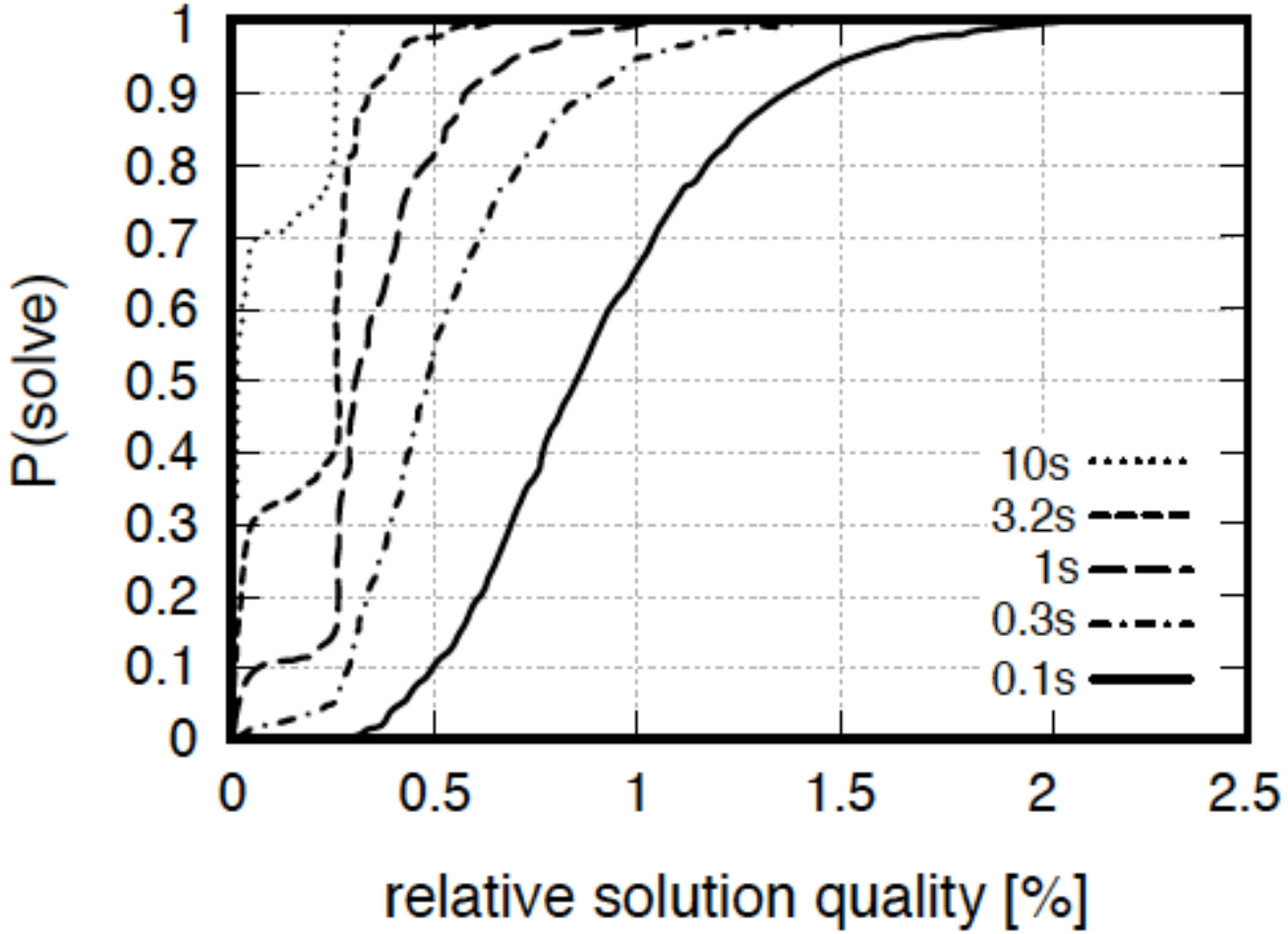
Note: Solution qualities q are often expressed as *relative solution qualities* $q/q^* - 1$, where q^* = optimal solution quality for given problem instance.

Typical solution quality distributions for SLS algorithm applied to hard instance of combinatorial optimisation problem:



Georgia Tech

Solution quality distributions for various run-times:



Georgia Tech

Solution quality distributions (SQDs)

- A *solution quality distribution (SQD) of an OLVA A' applied to a given problem instance π' for run-time t'* is a marginal distribution of the bivariate RTD $rtd(t, q)$ defined by:

$$sqd_{t'}(q) := rtd(t', q) = P_s(RT_{A',\pi'} \leq t', SQ_{A',\pi'} \leq q).$$

- SQDs correspond to cross-sections of the two-dimensional bivariate RTD graph.

- SQDs characterise the solution qualities achieved by a given SLS algorithm for a combinatorial optimisation problem within a given run-time bound (useful for type 2 application scenarios).

Georgia Tech

Protocol for obtaining the empirical RTD for an OLVA A' applied to a given instance π' of an optimisation problem:

- Perform k independent runs of A' on π' with cutoff time t'.
- During each run, whenever the incumbent solution is improved, record the quality of the improved incumbent solution and the time at which the improvement was achieved in a *solution quality trace*.
- Let $sq(t', j)$ denote the best solution quality encountered in run j up to time t'. The cumulative empirical RTD of A' on π' is defined by $\widehat{P}_s(RT \leq t', SQ \leq q') := \#\{j \mid sq(t', j) \leq q'\}/k$.

Note: Qualified RTDs, SQDs and SQT curves can be easily derived from the same solution quality traces.

Georgia Tech

Measuring run-times (1):

- CPU time measurements are based on a specific *implementation* and *run-time environment* (machine, operating system) of the given algorithm.

- To ensure reproducibility and comparability of empirical results, CPU times should be measured in a way that is as independent as possible from machine load.

  When reporting CPU times, the run-time environment should be specified (at least CPU type, model, speed and cache size; amount of RAM; OS type and version); ideally, the implementation of the algorithm should be made available.

Georgia Tech

RTD-based Analysis of LVA Behaviour

Run-time distributions (and related concepts) provide an excellent basis for

- analysis and characterisation of LVA behaviour;
- comparative performance analyses of two or more LVAs;
- investigations of the effects of parameters, problem instance features, *etc.* on the behaviour of an LVA.

RTD-based empirical analysis in combination with proper statistical techniques (hypothesis tests) is a state-of-the-art approach in empirical algorithmics.

Georgia Tech

Probabilistic Domination

Definition: Algorithm A *probabilistically dominates* algorithm B on problem instance π, iff

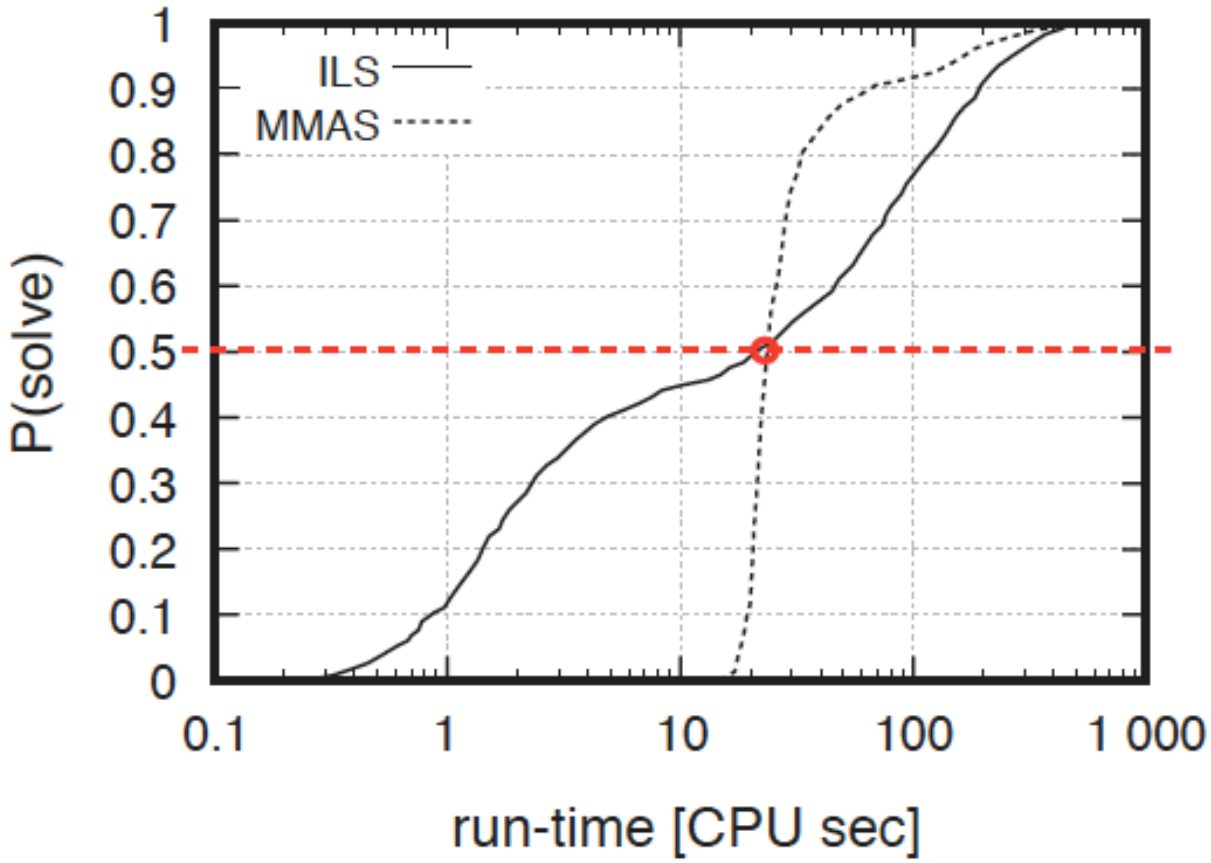
$$\forall t : P(RT_{A,\pi} \leq t) \geq P(RT_{B,\pi} \leq t) \quad (1)$$

$$\exists t : P(RT_{A,\pi} \leq t) > P(RT_{B,\pi} \leq t) \quad (2)$$

Graphical criterion: RTD of A is "above" that of B

Georgia Tech

Example of crossing RTDs for two SLS algorithms for the TSP applied to a standard benchmark instance (1000 runs/RTD):



Georgia Tech

Comparative analysis for instance ensembles (1)

Goal: Compare performance of Las Vegas algorithms A and B on a given ensemble of instances.

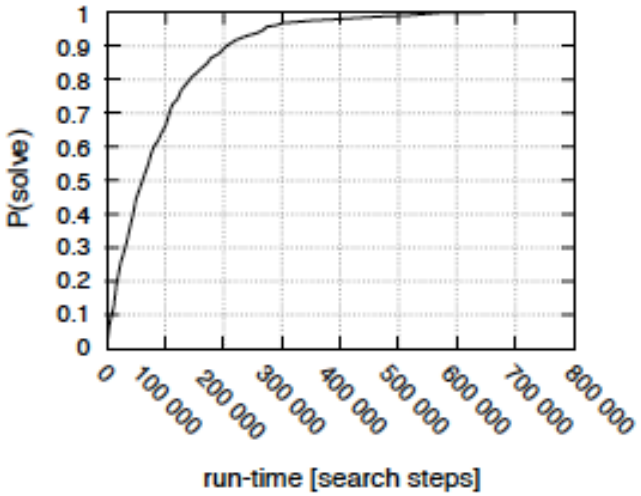
- Use instance-based analysis to partition given ensemble into three subsets:
  - instances on which A probabilistically dominates B;
  - instances on which B probabilistically dominates A;
  - instances on which there is no probabilistic domination between A and B (crossing RTDs).

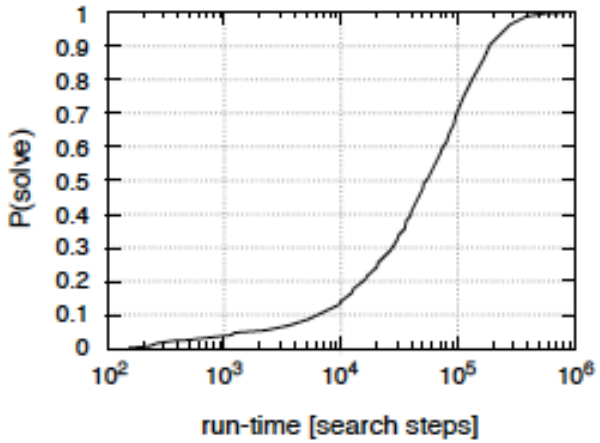
  The size of these subsets gives a rather detailed picture of the algorithms' relative performance on the given ensemble.

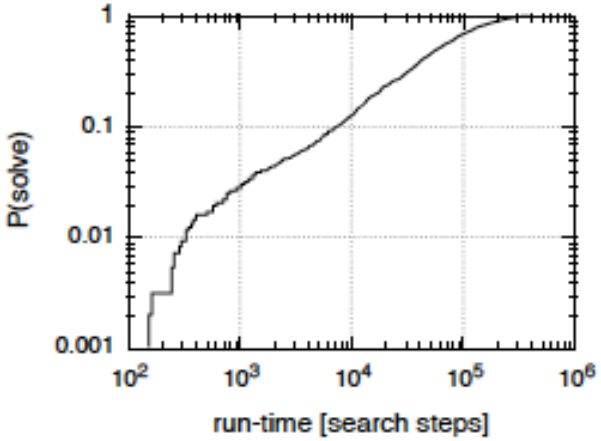
RTD plots are useful for the *qualitative analysis* of LVA behaviour:

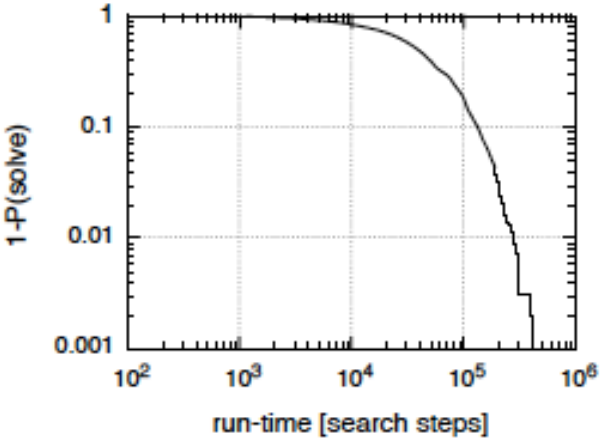
- *Semi-log plots* give a better view of the distribution over its entire range.
- Uniform performance differences characterised by a constant factor correspond to shifts along horizontal axis.
- *Log-log plots* of an RTD or its associated *failure rate decay function*, $1 - rtd(t)$, are often useful for examining behaviour for very short or very long runs.

Various graphical representations of a typical RTD:









Georgia Tech

A few general guidelines:

- Design your experiments carefully.
- Look at your data (all of it, from different angles).
- Be prepared for surprises (good and bad).
- Don't discard results (unless there is a *really* obvious reason).
- Report negative observations.
- If it looks too good to be true . . . it probably isn't true.
- Be sceptical – don't blindly trust anyone (not even yourself).
- Be a scientist – ask "why?".
- Be an explorer – and boldly go where no one has gone before!

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

Measure of dispersion

- Sample range

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

- Sample variance

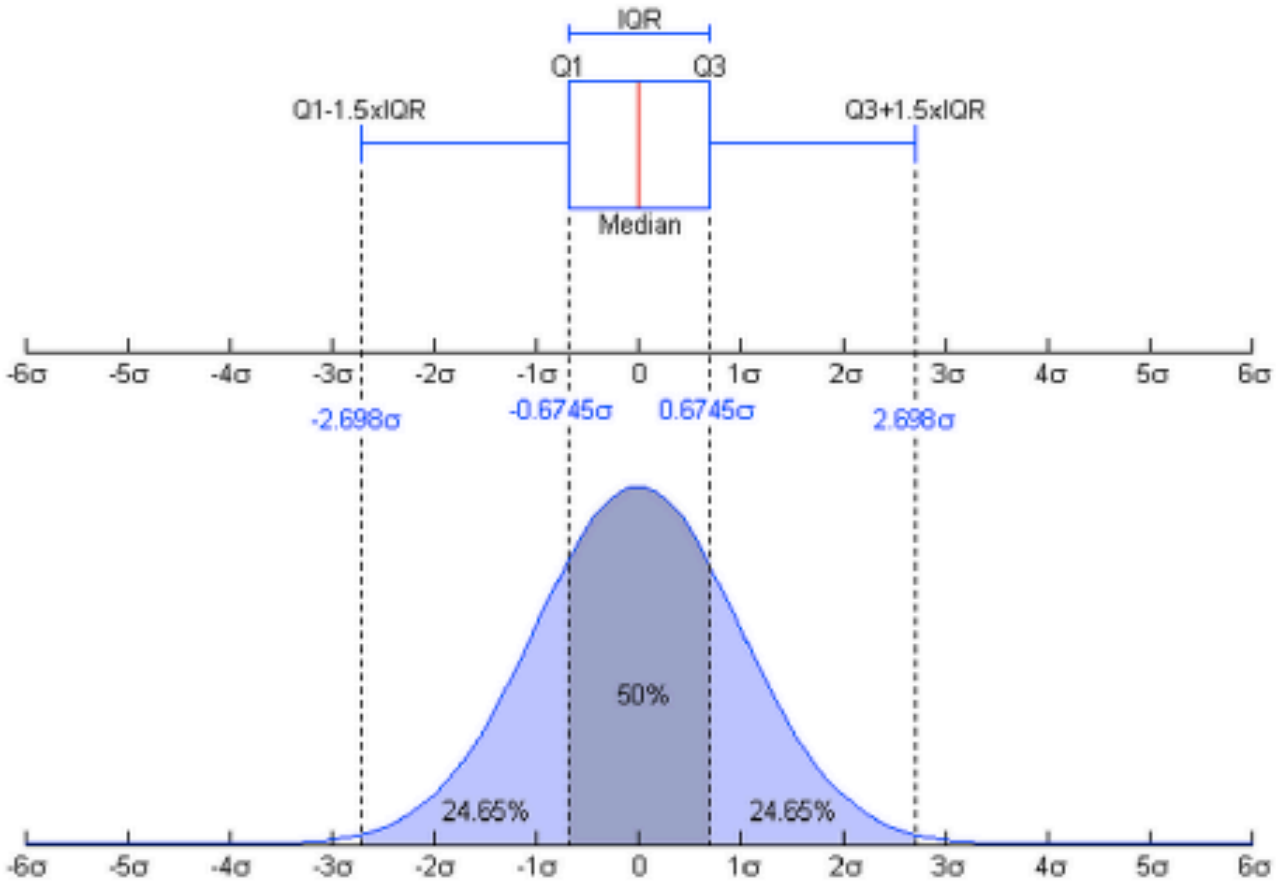
$$s^2 = \frac{1}{n-1}\sum(x_i - \bar{X})^2$$

- Standard deviation

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

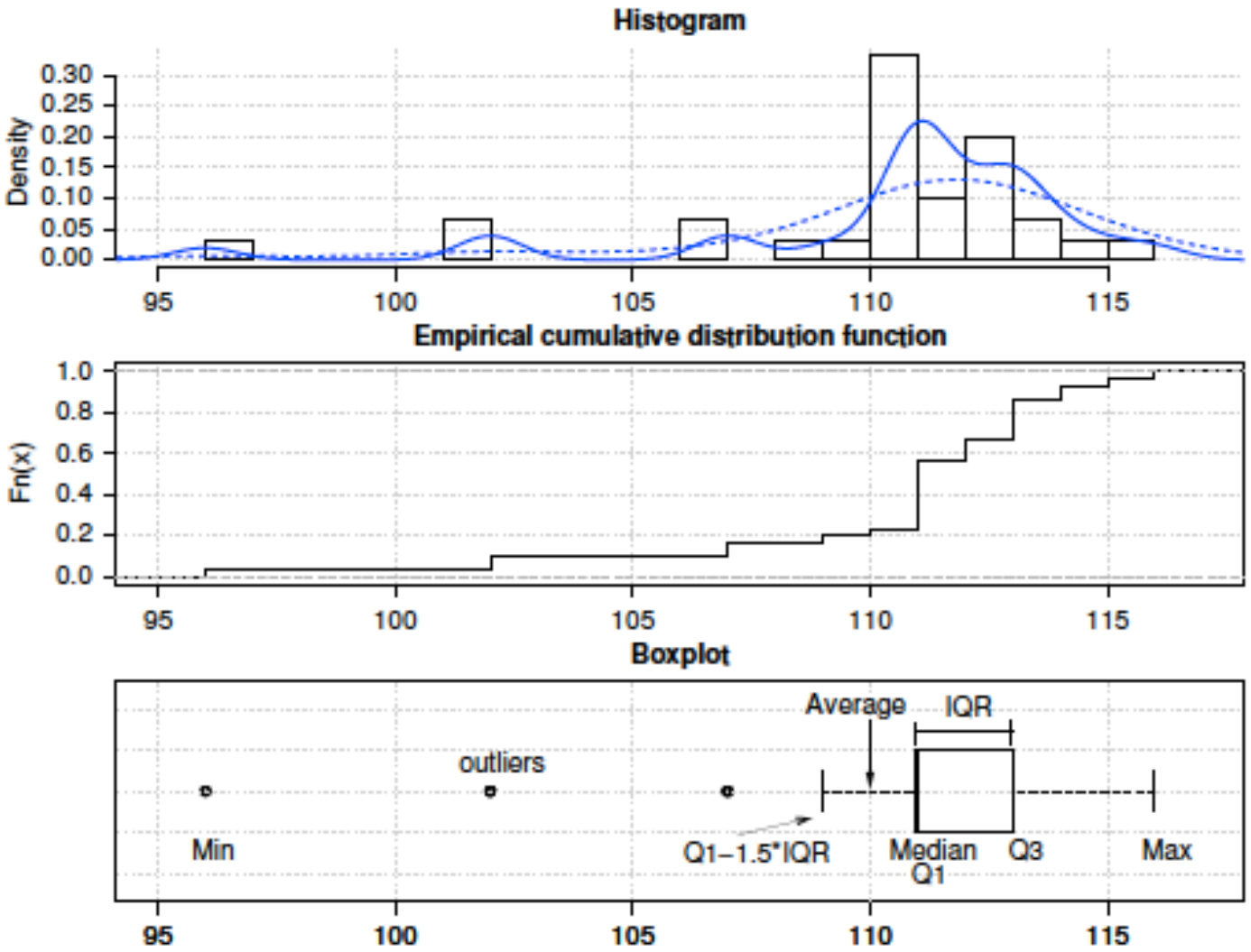
- Inter-quartile range

$$IQR = Q_3 - Q_1$$



Boxplot and a probability density function (pdf) of a Normal N(0,1s2) 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

Georgia Tech

CSE 6140/ CX 4140

Approximation Algorithms

[KT11, CLRS35, BRV8.1]

Dealing with NP-complete problems

Georgia Tech

Branch & bound, Backtracking	Sacrifice running time: create an algorithm with running time exponential in the input size (but which might do well on the inputs you use)
Parameterized algorithms	Sacrifice running time: allow the running time to have an exponential factor, but ensure that the exponential dependence is not on the entire input size but just on some parameter that is hopefully small
Approximation	Sacrifice quality of the solution: quickly find a solution that is *provably not very bad*
Local search	Quickly find a solution for which you cannot give any quality guarantee (but which might often be good in practice on real problem instances)
Restriction	By restricting the structure of the input (e.g., to planar graphs, 2SAT), faster algorithms are usually possible.
Randomization	Use randomness to get a faster average running time, and allow the algorithm to fail to find optimum with some small probability.

Performance bounds

- Approximation algorithms guarantee performance bounds, i.e., a bound on the worst deviation from the optimal quality, with a polynomial-time algorithm.
- **Ratio bound** (approximation factor)
  - Assume solution costs are positive
  - Given input X of size n, OPT(X) is optimum, A(X) is solution quality produced by algorithm A:

$$\max\left(\frac{A(X)}{OPT(X)}, \frac{OPT(X)}{A(X)}\right) \leq \rho(n)$$

  - For minimization:

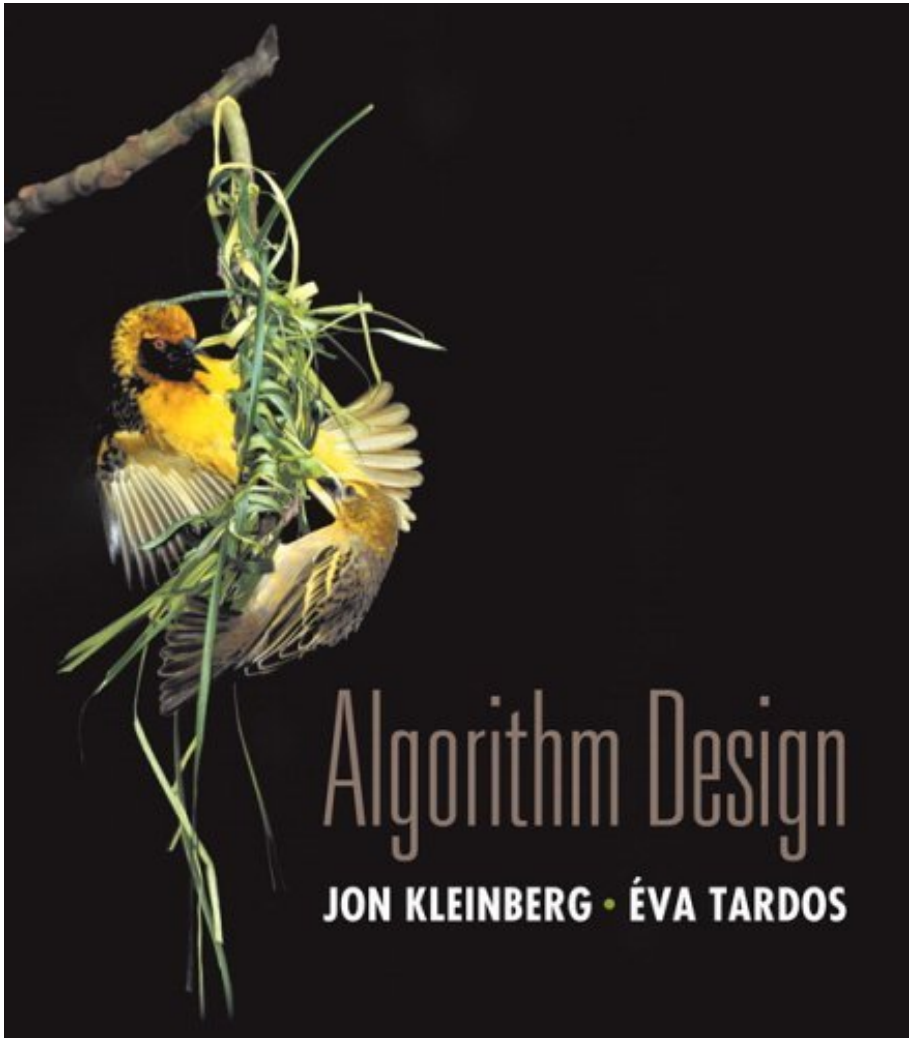
$$OPT(X) \leq A(X) \leq \rho(n) OPT(X)$$

  - In general, $\rho(n) \geq 1$, and if it does not depend on n, we have a **constant factor approximation**

Categories of approximability

- **Fully approximable**: we can get arbitrarily close to optimum, in polynomial time ($\rho(n)$=1+ε)
- **Partly approximable**: there are poly-time algorithms that achieve approximation bound for some range of $\rho(n)$, but unless P=NP, this range does not reach all the way down to 1
- **Inapproximable**: there is no poly-time approximation algorithm with however large ratio bound, unless P=NP

KT 11.1 Load Balancing



Load Balancing

Input. m identical machines; n jobs, job j has processing time t_j.

- Job j must run contiguously on one machine.
- A machine can process at most one job at a time.

Def. Let J(i) be the subset of jobs assigned to machine i. The load of machine i is $L_i = \sum_{j \in J(i)} t_j$.

Def. The makespan is the maximum load on any machine $L = \max_i L_i$.

Load balancing. Assign each job to a machine to minimize makespan.

Load Balancing: List Scheduling

List-scheduling algorithm.

- Consider n jobs in some fixed order.
- Assign job j to machine whose load is smallest so far.

```
List-Scheduling(m, n, t1,t2,...,tn) {
    for i = 1 to m {
        Li ← 0          ← load on machine i
        J(i) ← φ        ← jobs assigned to machine i
    }

    for j = 1 to n {
        i = argmin_k Lk          ← machine i has smallest load
        J(i) ← J(i) ∪ {j}        ← assign job j to machine i
        Li ← Li + tj             ← update load of machine i
    }
    return J(1), …, J(m)
}
```

Implementation. O(n log m) using a priority queue.