CSE 6140/ CX 4140

Computational Science and Engineering

ALGORITHMS

Coping with NP-completeness - 7

Approximation Algorithms

Empirical analysis

Instructor: Xiuwei Zhang

Assistant Professor

School of Computational Science and Engineering

Based on slides by Prof. Ümit V. Çatalyürek and Bistra Dilkina

Reminder

HW4 (last homework) due Wed. Nov. 18, 11:59pm EST

Project partial report due Friday Nov. 20, 11:59pm EST

Hard deadlines.

Center Selection

- **Theorem**. Let C* be an optimal set of centers. Then $r(C) \leq 2r(C^*)$.

- **Theorem**. Greedy algorithm is a 2-approximation for center selection problem.

- Remark. Greedy algorithm always places centers at sites, but is still within a factor of 2 of best solution that is allowed to place centers anywhere.

  e.g., points in the plane

- **Theorem**. There is no better approximation algorithm (show next).

Center Selection: Hardness of Approximation

- **Theorem**. Unless P = NP, there is no ρ-approximation algorithm for metric k-center problem for any $\rho < 2$.

- **Pf**. We show how we could use a $(2 - \varepsilon)$ approximation algorithm for k-center to solve DOMINATING-SET in poly-time.
  - DOMINATING-SET: Given a graph G, is there a set of vertices U of size at most k such that every other vertex has a neighbor in U
  - Let $[G = (V, E), k]$ be an instance of DOMINATING-SET.
  - Construct instance $[G', k'=k, r=1]$ of k-CENTER with sites V and distances
    - $d(u, v) = 1$ if $(u, v) \in E$
    - $d(u, v) = 2$ if $(u, v) \notin E$
  - Note that G' satisfies the triangle inequality.
  - Claim:
    G has dominating set of size k iff there exists k centers C^* with $r(C^*) = 1$ in G'.

Center Selection: Hardness of Approximation

- **Theorem**. Unless P = NP, there is no ρ-approximation algorithm for metric k-center problem for any ρ < 2.

- **Pf**. We show how we could use a (2 - ε) approximation algorithm for k-center to solve DOMINATING-SET in poly-time.
  - Let [G = (V, E), k] be an instance of DOMINATING-SET.
  - Construct instance [G',k'=k] of k-CENTER with sites V and distances
    - $d(u, v) = 1$ if $(u, v) \in E$
    - $d(u, v) = 2$ if $(u, v) \notin E$
  - If DOMINATING-SET is a yes instance, the optimal solution of k-center is $r(C^*)=1$
  - If there exists an approx algo with ρ < 2, then approx provides $r(C) < 2\, r(C^*)$ $\longrightarrow$ approx returns $r(C) = 1$
  - If DOMINATING-SET is a no instance, the optimal solution of k-center is $r(C^*)=2$, approx provides $r(C) >= r(C^*)=2$

Center Selection: Hardness of Approximation

- Suppose we have a $(2-\epsilon)$-approx. algorithm A for k-CENTER, where $\epsilon > 0$
- $I_1 \rightsquigarrow I_2$ $(G', k' = k, r = 1)$, run A on I_2

- If the solution of A, $r(C)$, is < 2
  - It means $r(C) = 1$ (it is 1 or 2 on I_2)
  - This solution is a valid solution to DOM-SET

  $\Longrightarrow$ I_1 has a solution

- If $r(C) \geq 2$ approx. ratio

  $2 \leq r(C) \leq (2-\epsilon) \cdot r(C^*)$

  $\Longrightarrow r(C^*) \geq \frac{2}{2-\epsilon} > 1$

  $\Longrightarrow$ I_1 has no solution!

$\Longrightarrow$ we can answer DOM-SET in poly-time

$\Longrightarrow$ So unless P=NP, there is no $(2-\epsilon)$-approx. algorithm for k-CENTER

CSE 6140

Empirical Analysis of Algorithms

textbook: STOCHASTIC LOCAL SEARCH
FOUNDATIONS AND APPLICATIONS

based on slides by Holger Hoos

Theoretical *vs.* Empirical Analysis

Ideal: Analytically prove properties of a given algorithm (run-time: worst-case / average-case / distribution, error rates).

Reality: Often only possible under substantial simplifications or not at all.

$\leadsto$ Empirical analysis

Empirical Analysis of Algorithms

Goals

- 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

Empirical Analysis of Algorithms

Issues:

- algorithm implementation (fairness)
- selection of problem instances (benchmarks)
- performance criteria (what is measured?)
- experimental protocol
- data analysis & interpretation

Benchmark Selection

Some criteria for constructing/selecting benchmark sets:

- instance hardness (focus on hard instances)
- instance size (provide range, for scaling studies)
- instance type (provide variety):
  - individual application instances
  - hand-crafted instances (realistic, artificial)
  - ensembles of instances from random distributions (random instance generators)
  - encodings of various other types of problems (e.g., SAT-encodings of graph coloring problems)

CPU Time vs. Elementary Operations

How to measure run-time?

- Measure CPU time (using OS book-keeping & functions)
- Measure elementary operations of algorithm
  (*e.g.*, local search steps, calls of expensive functions)
  and report cost model (CPU time / elementary operation)

Issues:

- accuracy of measurement
- dependence on run-time environment
- fairness of comparison

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.

$\leadsto$ 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).

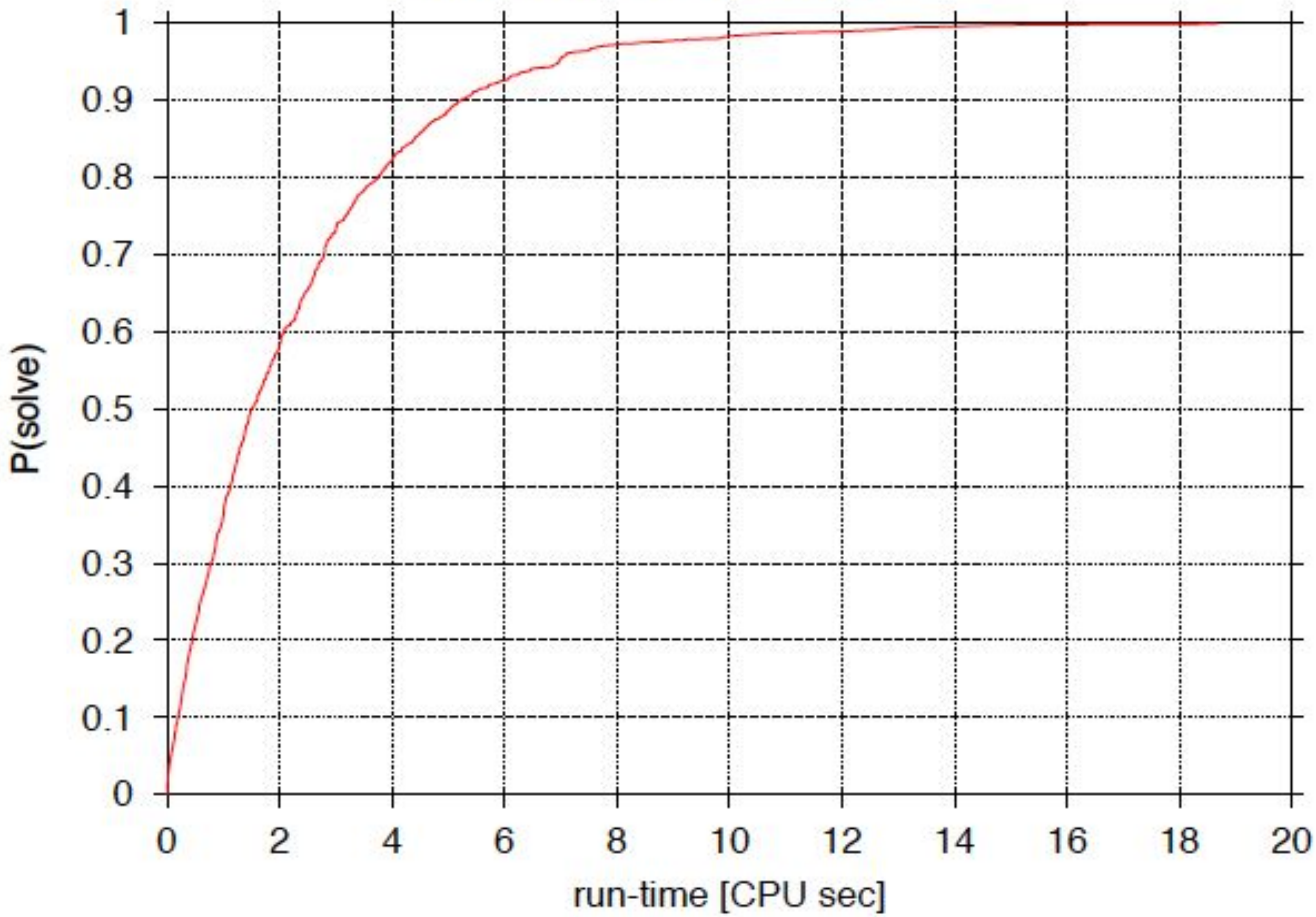
  $\rightsquigarrow$ evaluation criterion: solution probability at time t_{max}

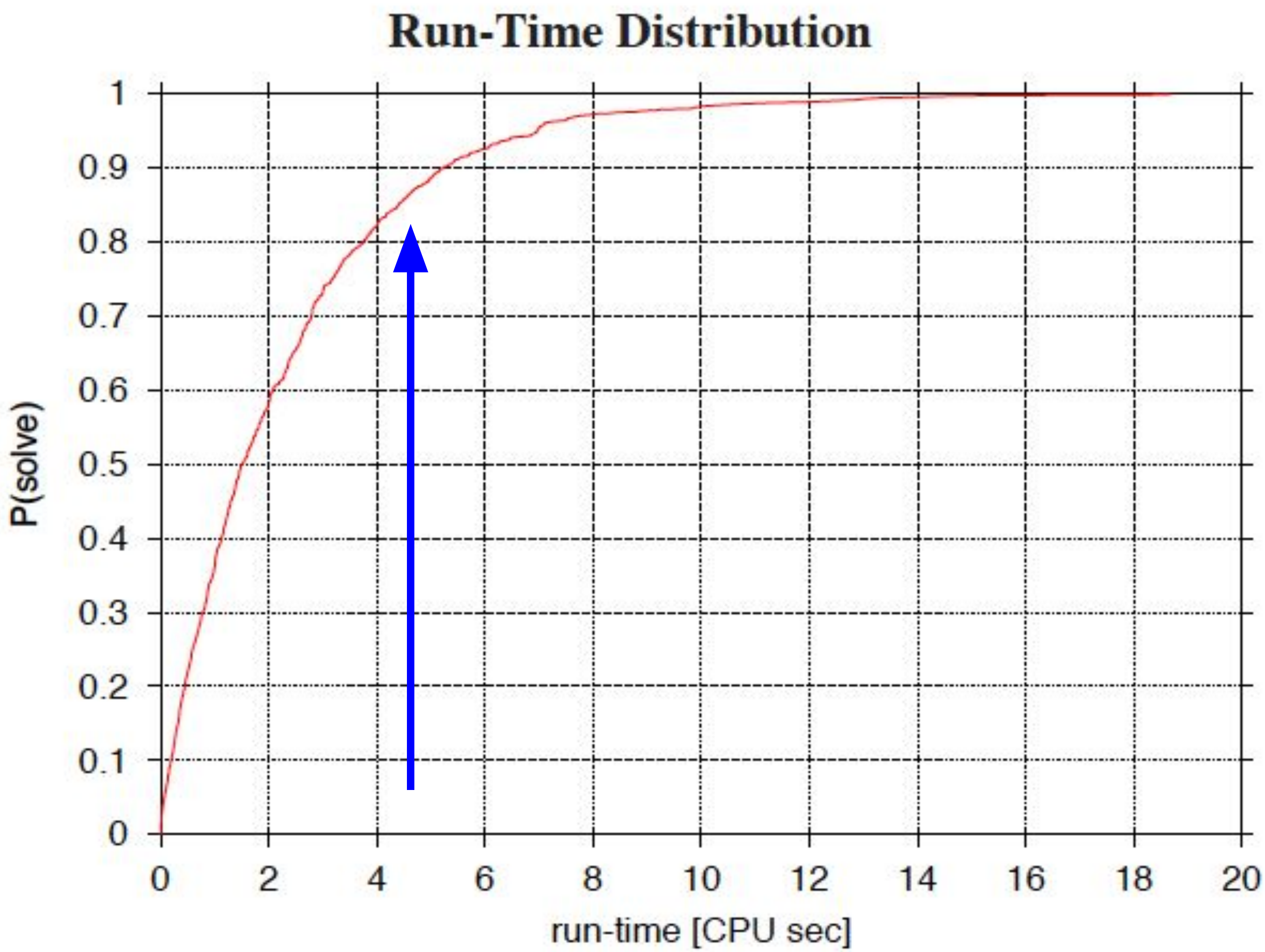
Definition: **Run-Time Distribution (1)**

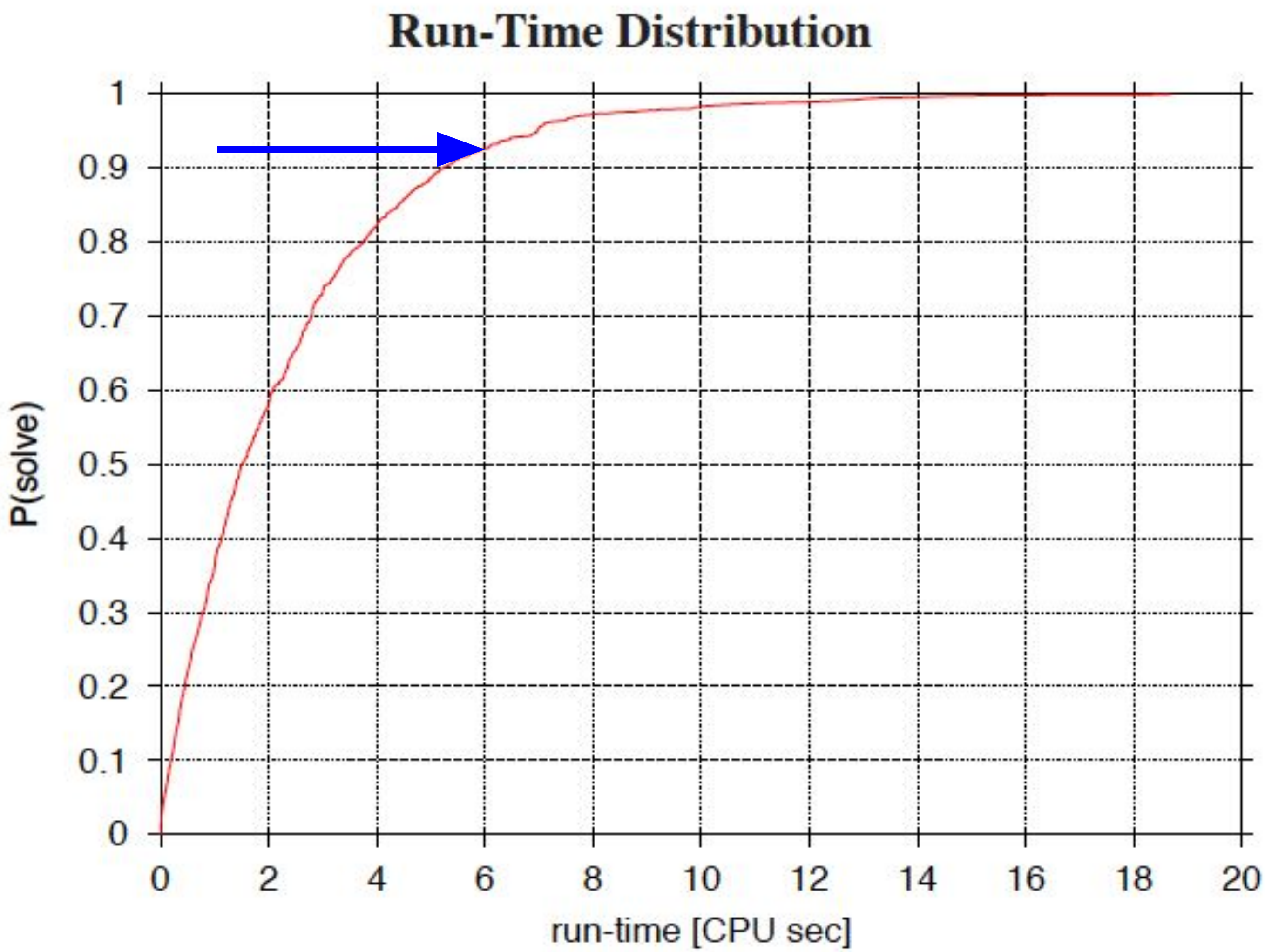
Given Las Vegas algorithm A for decision problem Π:

- 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




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

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.

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.

Example for runtime plot

t'=20s, k=10

runtime

run1: 10

run2: fail

run3: 5

run4: 4

run5: 12

run6: 14

run7: fail

run8: 15

run9: 8

run10: 11

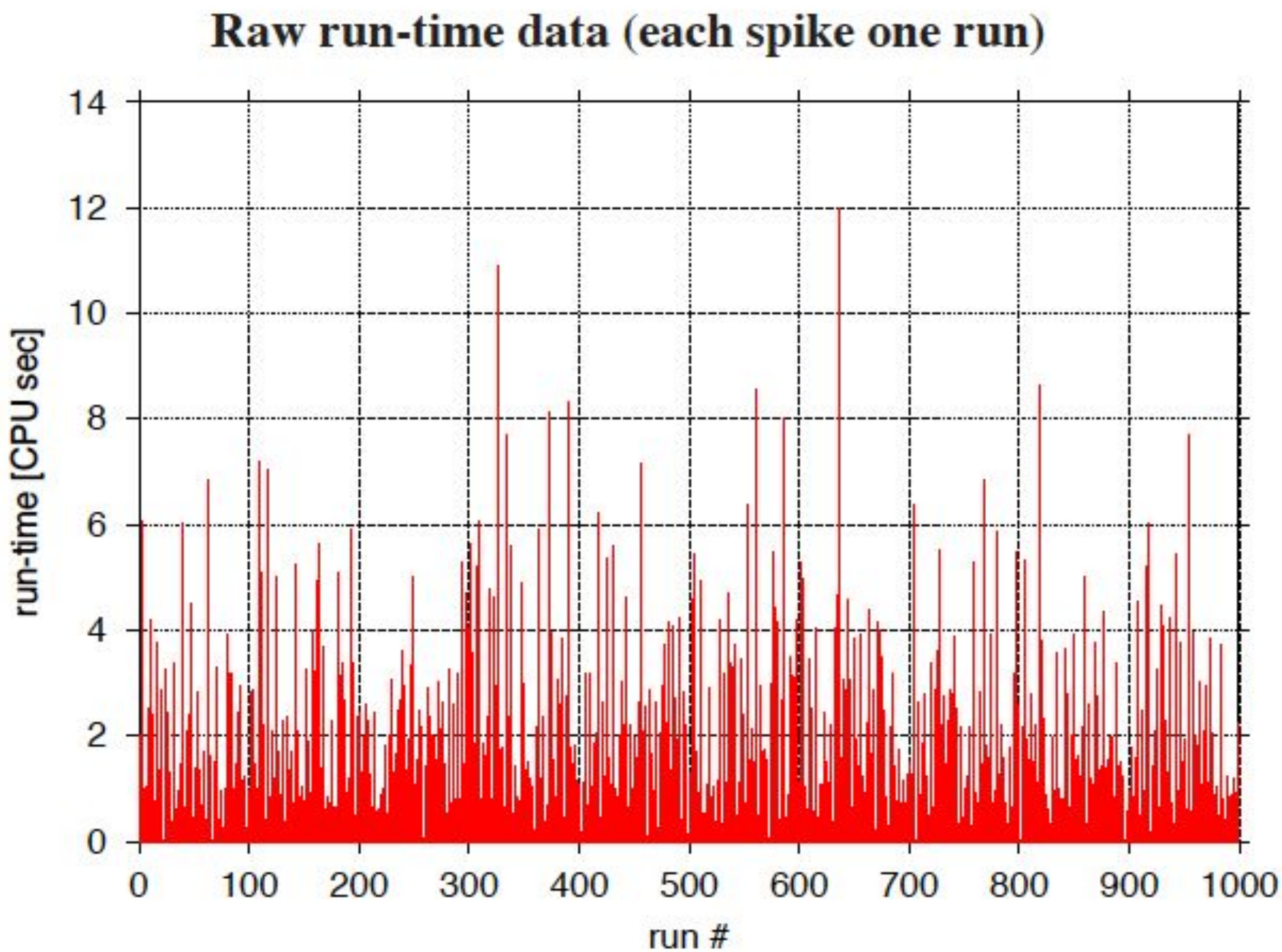
k'=8

Sorted runtime:

rt = {4, 5, 8, 10, 11, 12, 14, 15}

plot:

(4, 0.1), (5, 0.2), (8, 0.3),
(10, 0.4), (11, 0.5), (12, 0.6),
(14, 0.7), (15, 0.8)


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]

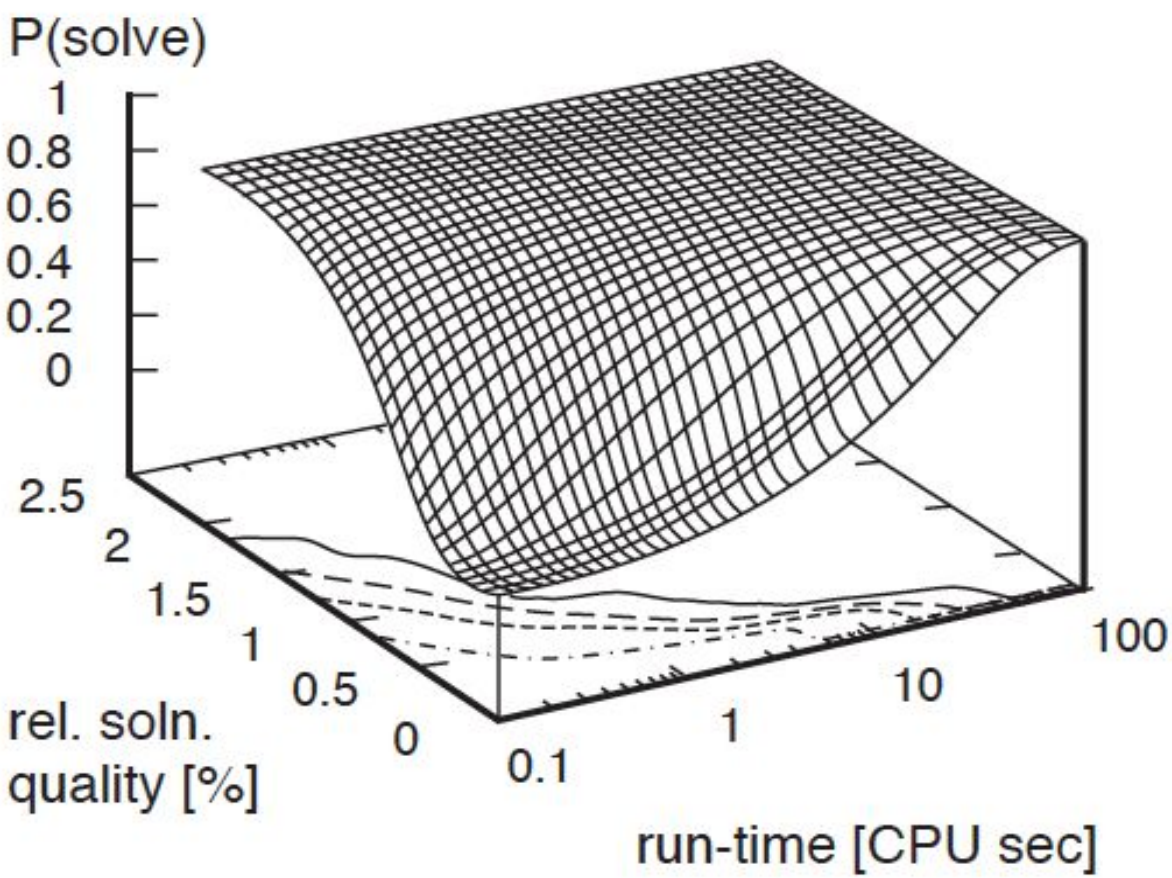
Optimization

Definition: Run-Time Distribution (2)

Given OLVA A' for optimisation problem Π':

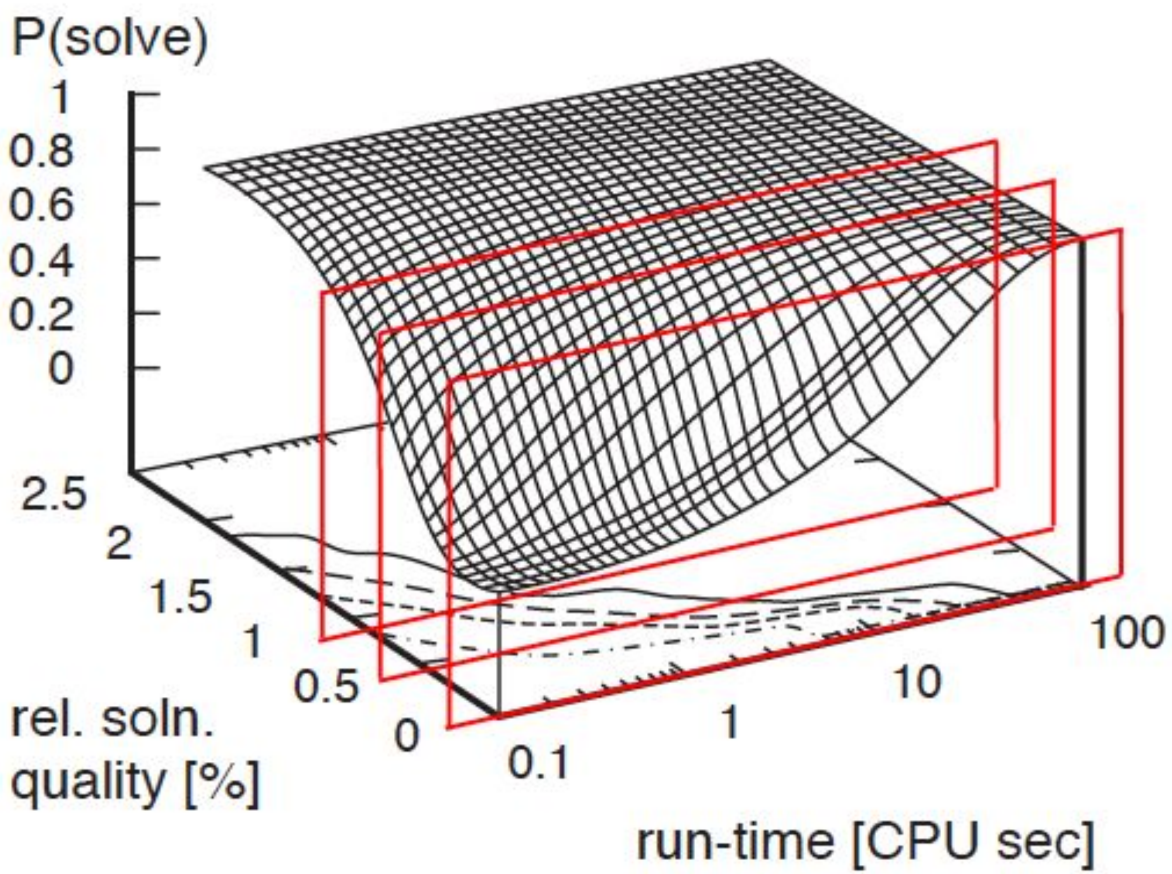
- The *success probability* $P_s(RT_{A',\pi'} \leq t, SQ_{A',\pi'} \leq q)$ is the probability that A' finds a solution for a soluble instance $\pi' \in \Pi'$ of quality $\leq q$ in time $\leq t$.
- The *run-time distribution (RTD) of A' on π'* is the probability distribution of the bivariate random 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:

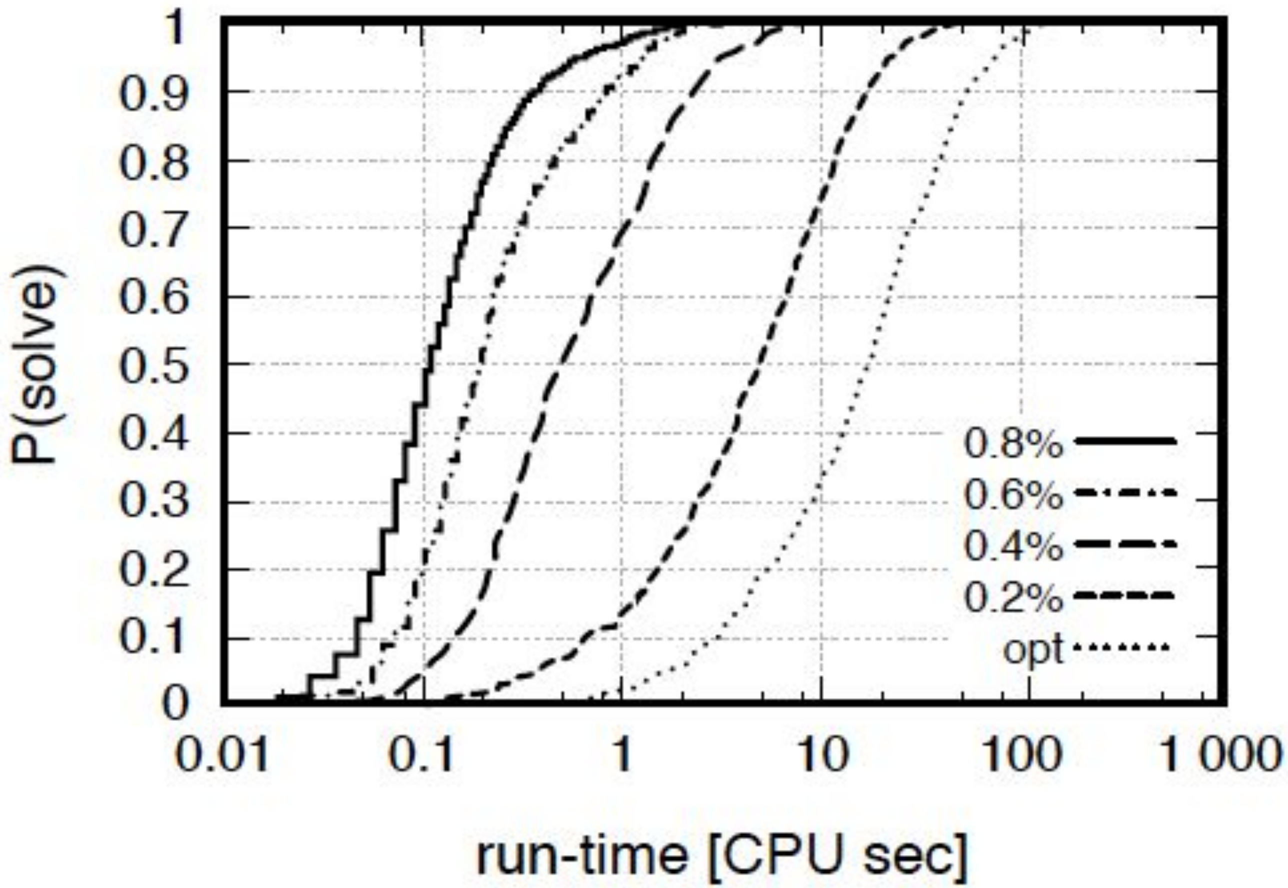


Solution quality: Relative error (Alg - OPT)/OPT

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



Qualified RTDs for various solution qualities:



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)

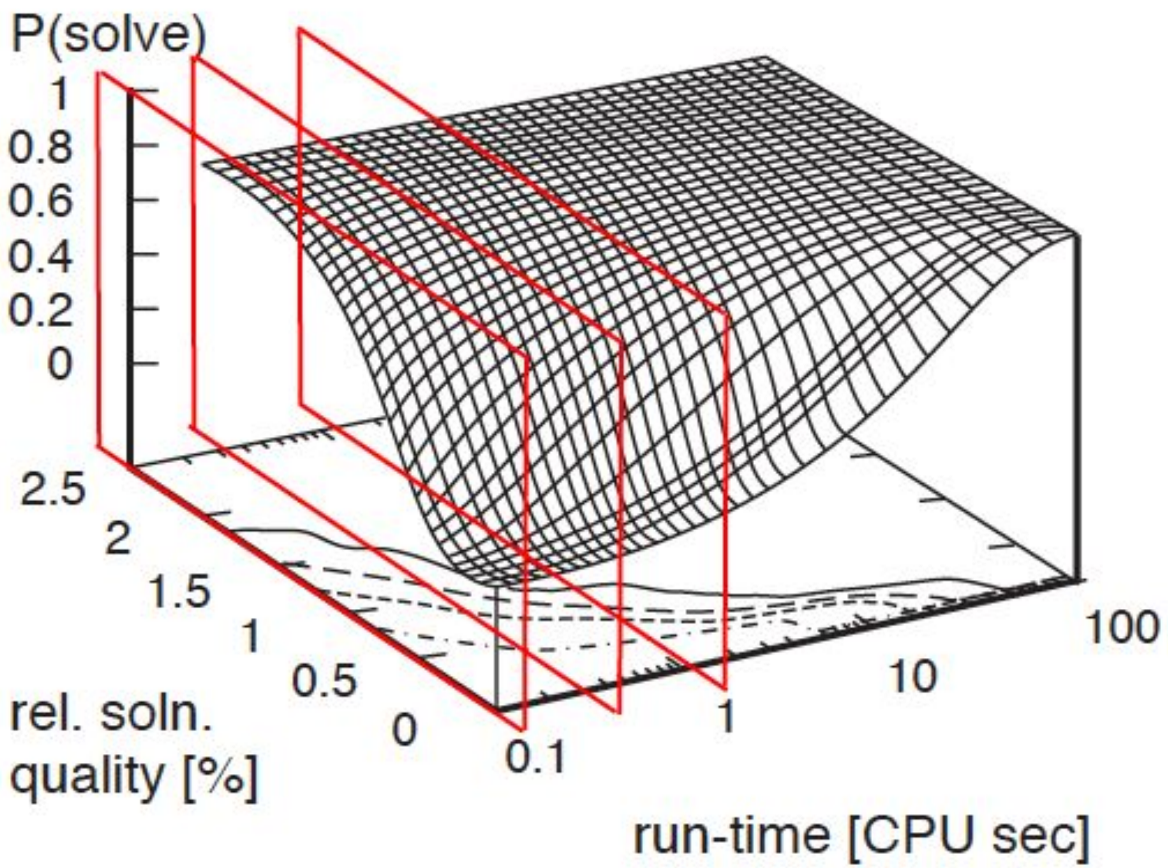
- A *qualified run-time distribution (QRTD) of an OLVA A' applied to a given problem instance π' for solution quality q'* is a marginal distribution of the bivariate RTD $rtd(t, q)$ defined by:

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

- 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.

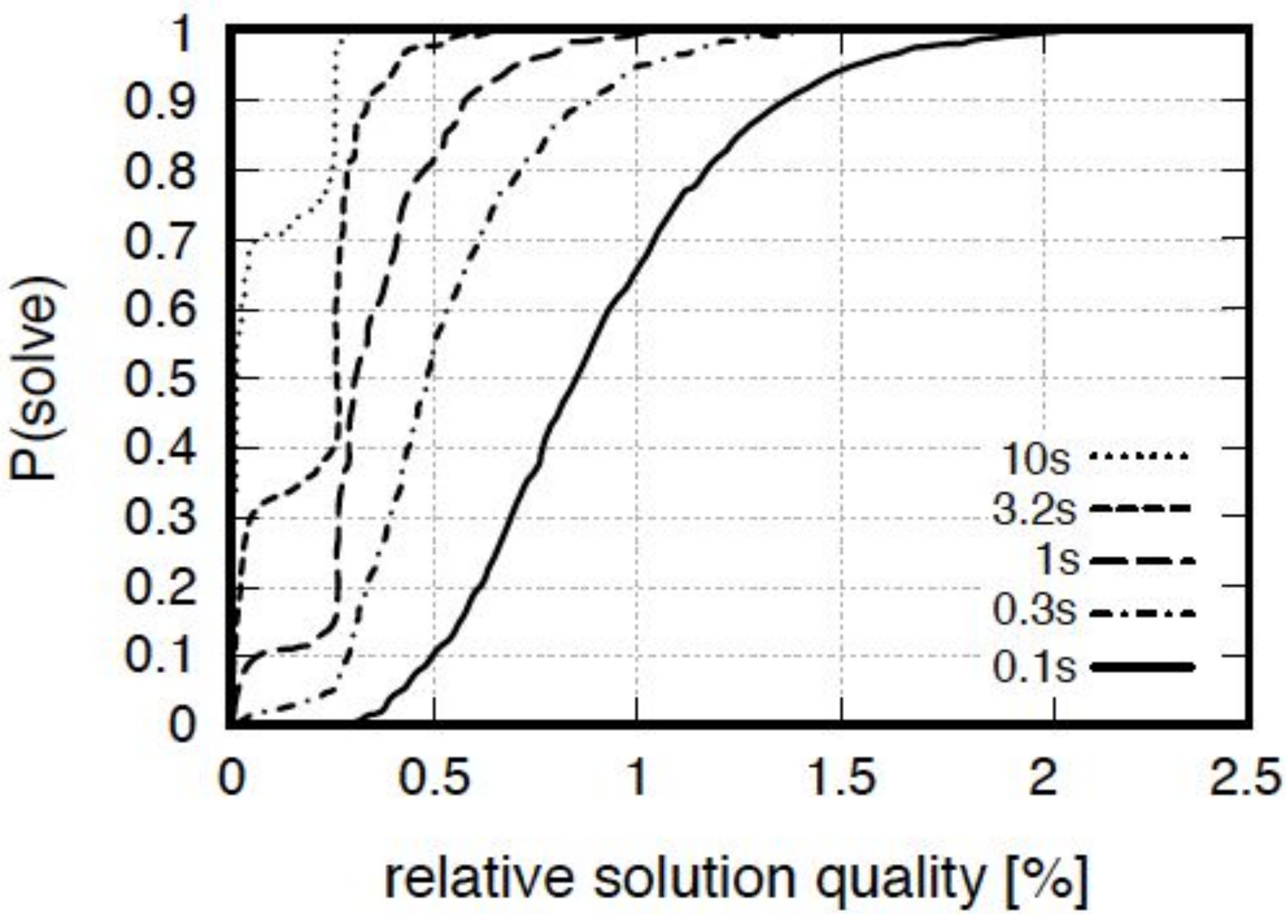
Note: Solution qualities q are often expressed as *relative solution qualities* $q/q^* - 1$, where q^* = optimal solution quality for given problem instance.

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



Georgia Tech

Solution quality distributions for various run-times:



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).

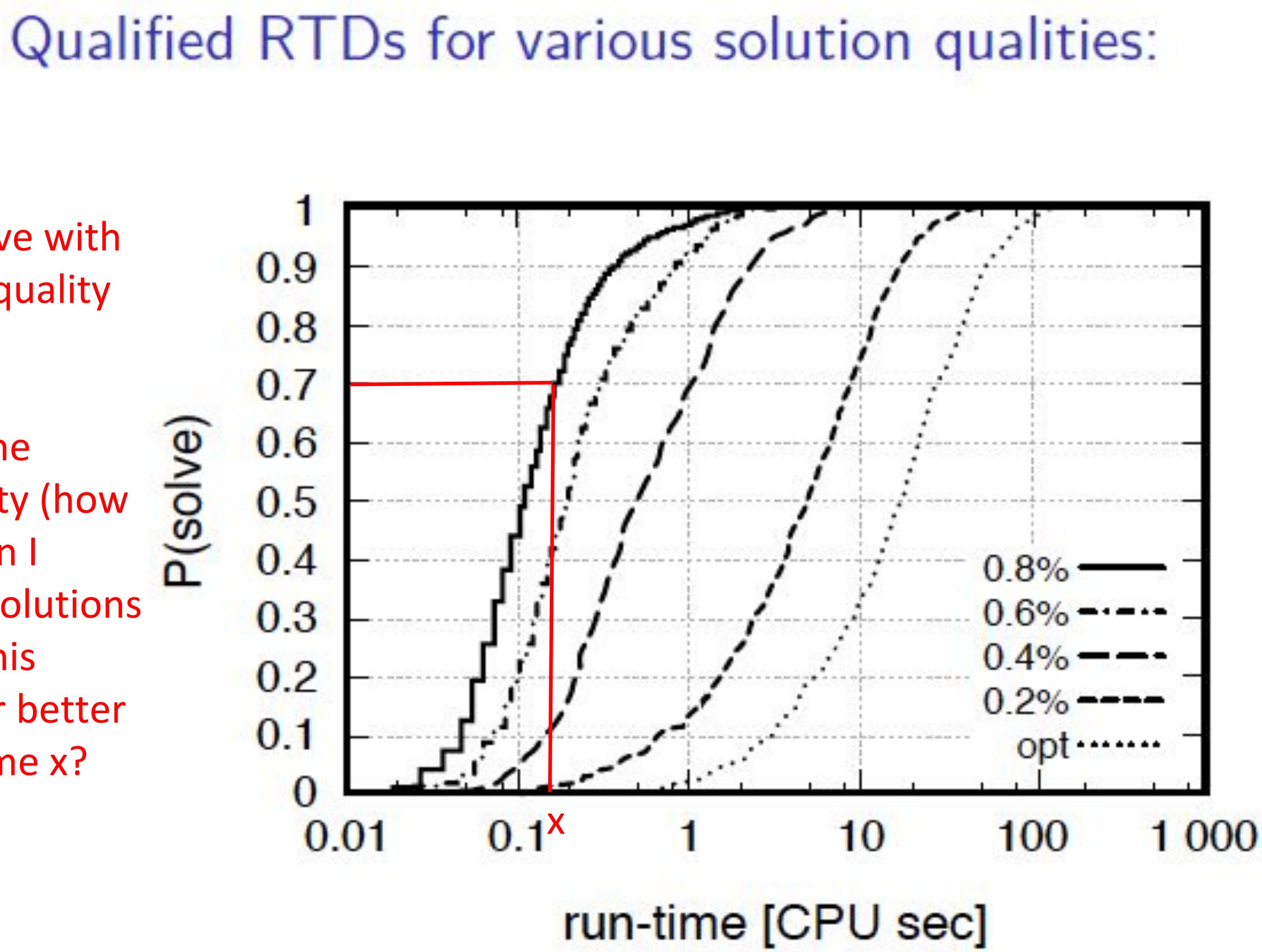
Protocol for obtaining the empirical RTD for an OLVA A' applied to a given instance π' of an optimisation problem:

- 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.

Qualified RunTime Distribution

Solution quality: Relative error (Alg - OPT)/OPT



For a curve with solution quality 0.8%:

What's the probability (how often) can I achieve solutions with of this quality or better within time x?