CPSC 590 (AUTUMN 2003)

INTRODUCTION TO EMPIRICAL ALGORITHMICS

Holger H. Hoos

Introduction

Consider the following scenario:

You have just developed a new algorithm A that, given historical weather data, predicts whether it will rain tomorrow.
You believe A is better than any other method for this problem.

Question:

How do you show the superiority of your new algorithm?

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.

$\rightsquigarrow$ Empirical analysis

The Three Pillars of CS:

- **Theory:** abstract models and their properties
  (“eternal thruths”)
- **Engineering:** principled design of artifacts
  (hardware, systems, algorithms, interfaces)
- **(Empirical) Science:** principled study of phenomenae
  (behaviour of hardware, systems, algorithms; interactions)

The “S” in CS – Why CS is a Science

Definition of ”science”:

(according to the Merriam-Webster Unabridged Dictionary)

“3a: knowledge or a system of knowledge covering general truths or the operation of general laws especially as obtained and tested through scientific method”

(Interestingly, this dictionary lists “information science” as well as “informatics”, but not “computer science”.)

Why “Computer Science” is a Misnomer:

CS is not a science of *computers* (in the standard sense of the meaning), but a science of *computing and information*.

CS is concerned with the study of:

- mathematical structures and concepts that model computation and information (theory, software)
- physical manifestations of these models (hardware)
- interaction between these manifestations and humans (HCI)

The Scientific Method

make observations

formulate hypothesis/hypotheses (model)

While not satisfied (and deadline not exceeded) iterate:

1. design experiment to falsify model
2. conduct experiment
3. analyse experimental results
4. revise model based on results

Empirical Analysis of Algorithms

Goals:

- Show that algorithm A improves state-of-the-art.
- Show that algorithm A is better than algorithm B.
- Show that algorithm A has property P.

Issues:

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

Overview

Comparative Empirical Performance Analysis of . . .

- Deterministic Decision Algorithms
- Randomised Algorithms without Error:
  Las Vegas Algorithms
- Randomised Algorithms with One-Sided Error
- Randomised Algorithms with Two-sided Error:
  Monte Carlo Algorithms
- Optimisation Algorithms

Decision Problems

Given: Input data (e.g., graph G and number of colours, k)

Objective: Output "yes" or "no" answer (e.g., to the question "can the vertices in G be coloured with k colours such that no two vertices connected by an edge have the same colour?")

Deterministic Decision Algorithms

Given: Two algorithms A, B for the same decision problem (e.g., graph colouring) that are:

- error-free, i.e., output is always correct
- deterministic, i.e., for given instance (and parameter settings), run-time is constant

Want: Determine whether A is better than B w.r.t. run-time.

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 ($\leadsto$ random instance generators)
  - encodings of various other types of problems (*e.g.*, SAT-encodings of graph colouring problems)

CPU Time *vs.* Elementary Operations

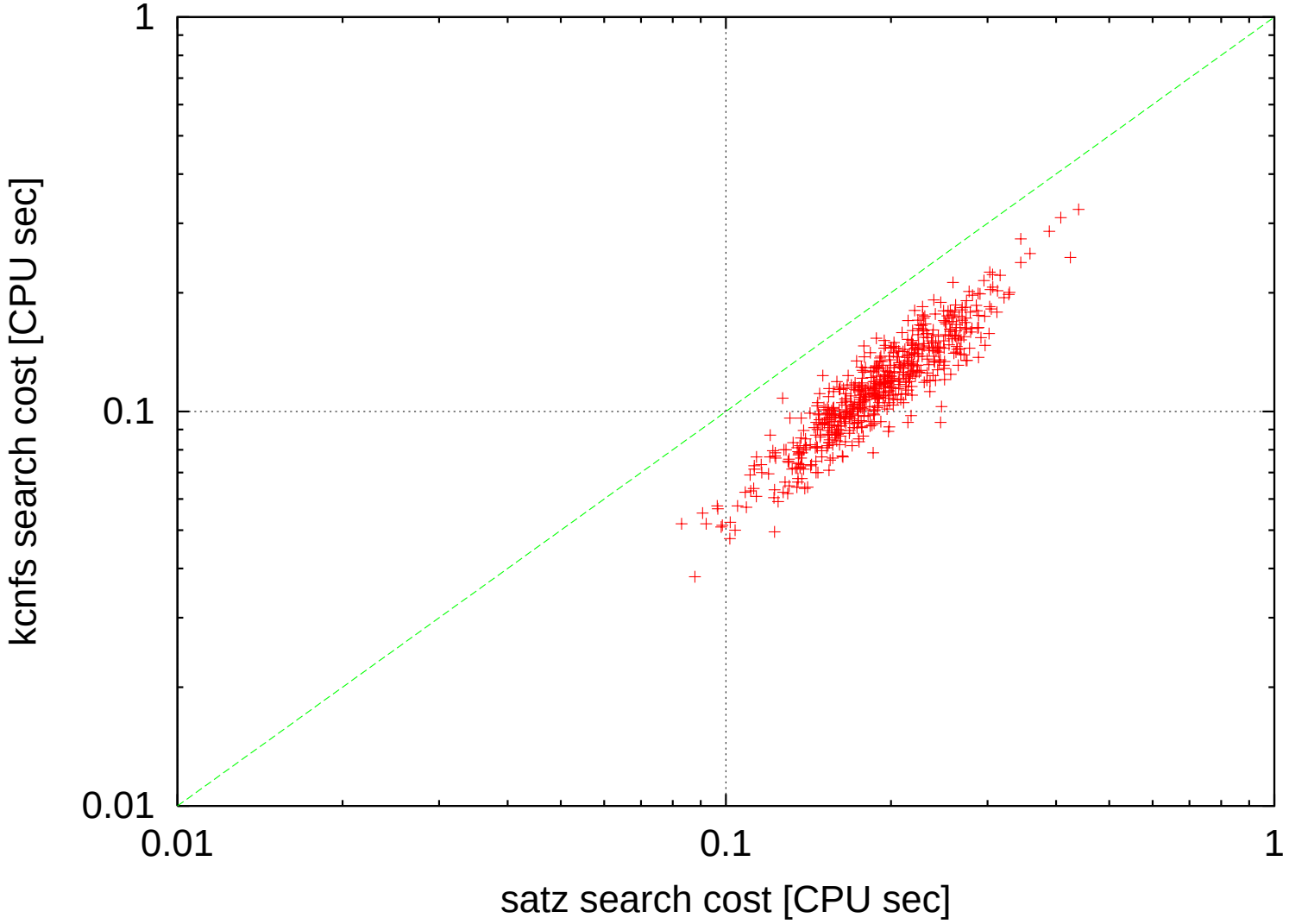
How to measure run-time?

- 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)

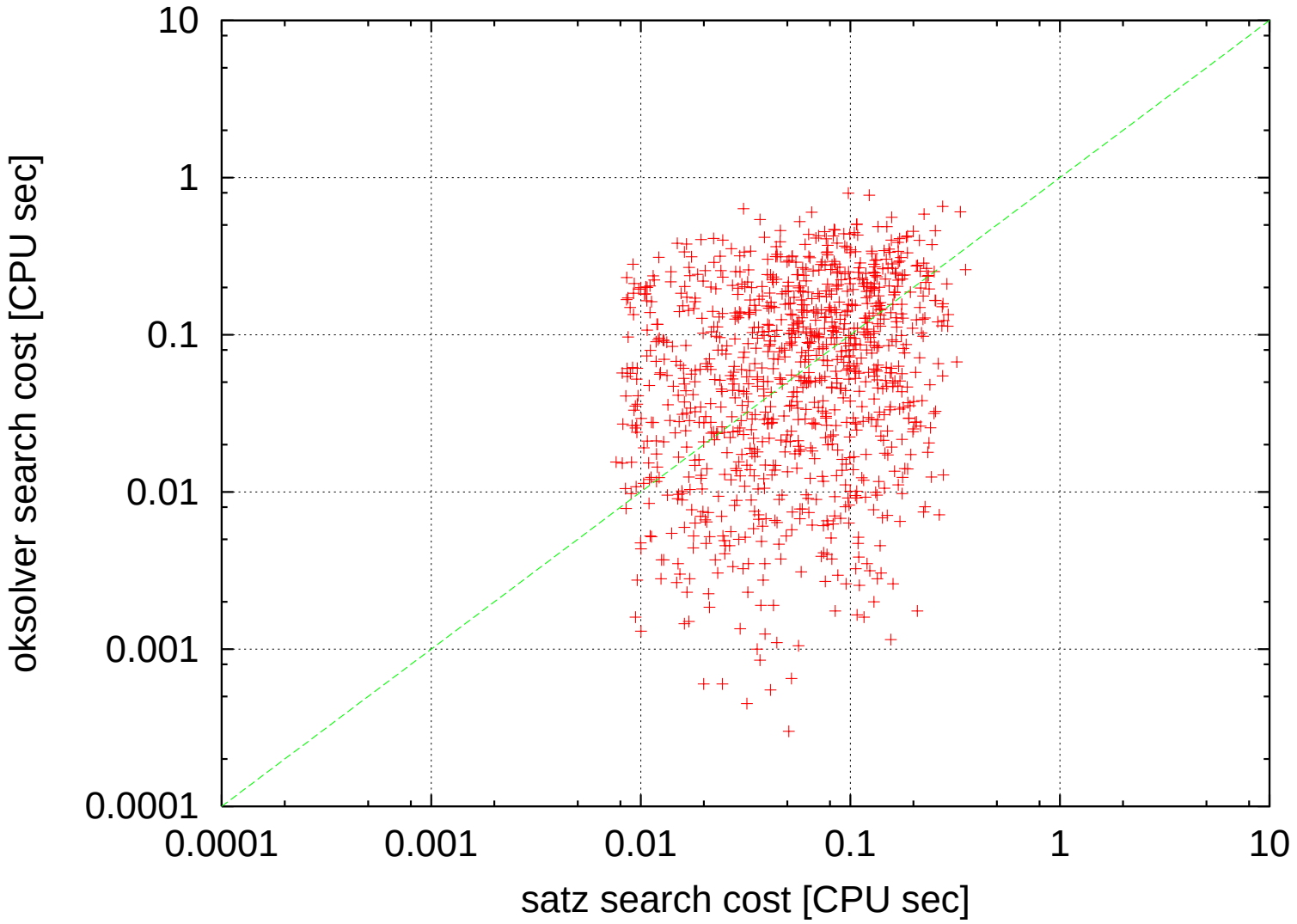
Issues:

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

Correlation of algorithm performance (each point one instance)



Correlation of algorithm performance (each point one instance)



Detecting Performance Differences

Assumption: Test instances drawn from random distribution.

Hypothesis: Median of paired differences is significantly different from 0 (i.e., algorithm A better than B or vice versa)

Test: binomial sign test or Wilcoxon matched pairs signed-rank test

Detecting Performance Correlation

Assumption: Test instances drawn from random distribution.

Hypothesis: There is a significant monotonic relationship between the correlation of A and B

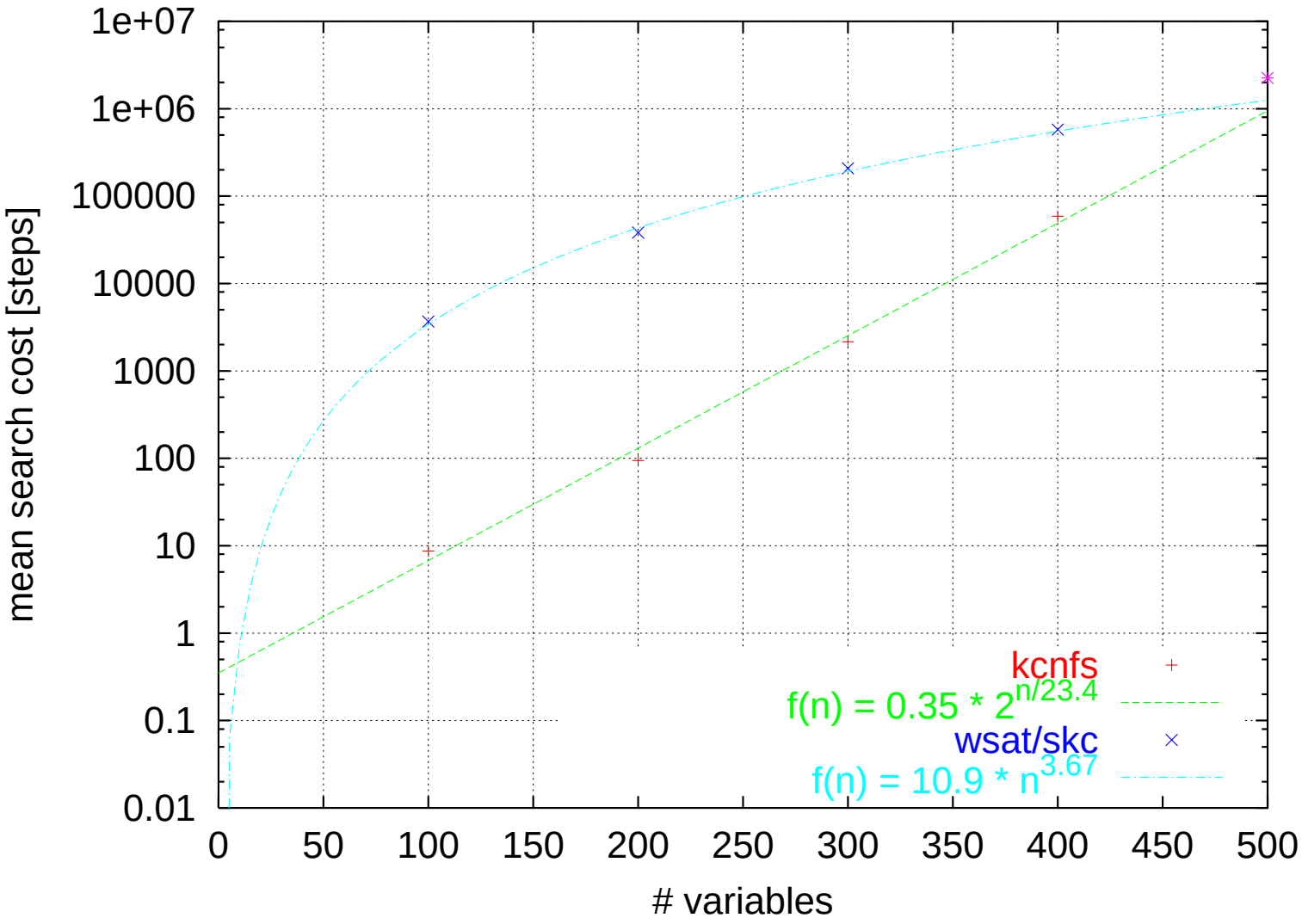
Test: Spearmans rank order test or Kendalls tau test

Scaling Analysis

Analyse scaling of performance with instance size:

- measure performance for various instance sizes
- fit parametric model (e.g., $\alpha \cdot e^{\beta x}$) to data points
- test interpolation / extrapolation

Empirical scaling of algorithm performance



Robustness Analysis

Measure robustness of performance w.r.t. . . .

- algorithm parameter settings
- problem type (*e.g.*, 2-SAT, 3-SAT, ...)
- problem parameters / features (*e.g.*, constrainedness)

Analyse . . .

- performance variation
- correlation with parameter values

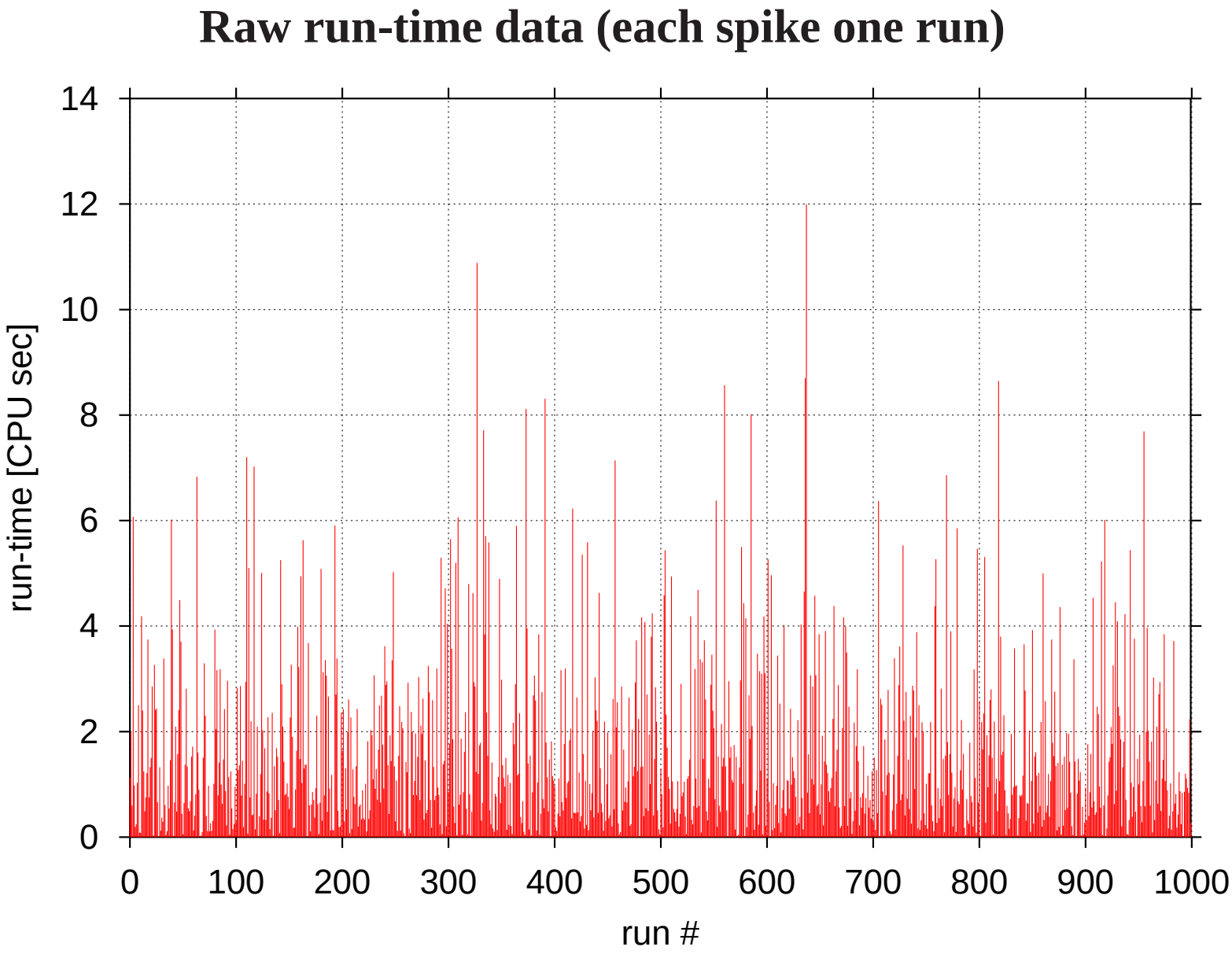
Randomised Algorithms without Error

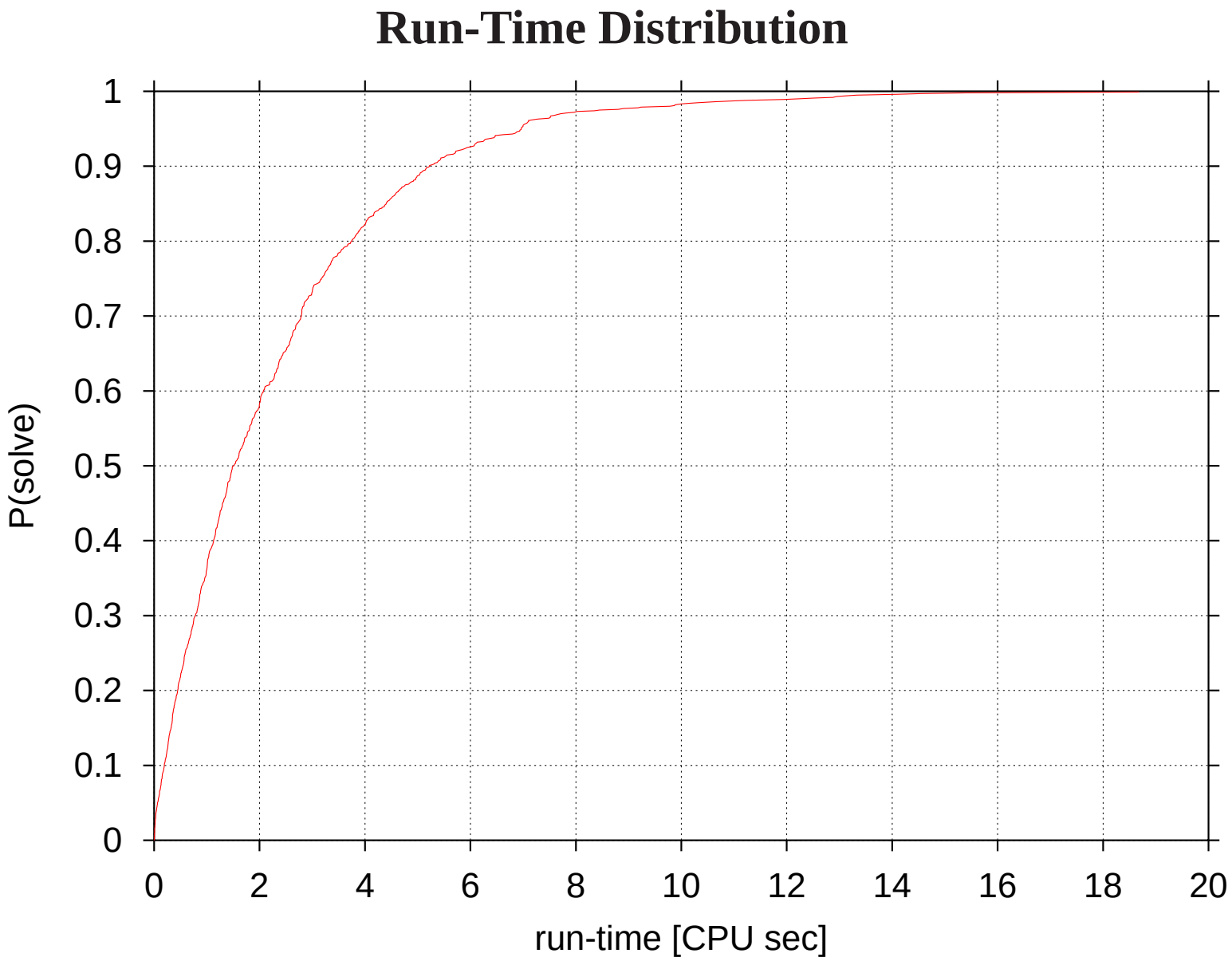
Las Vegas Algorithms (LVAs):

- decision algorithms whose output is always correct
- randomised, i.e., for given instance (and parameter settings), run-time is random variable

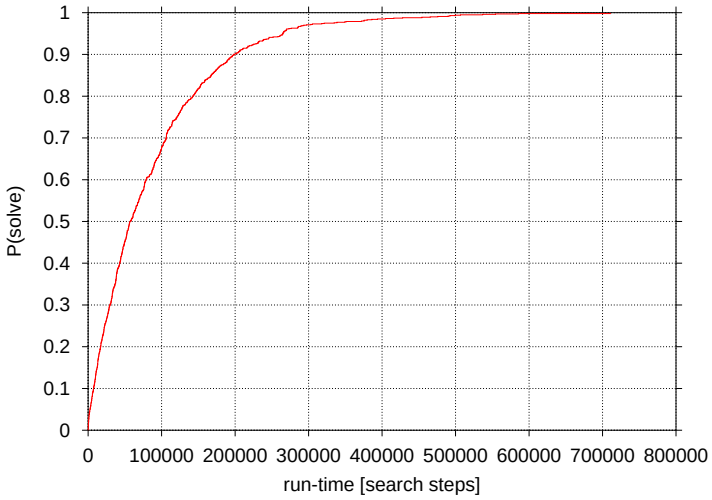
Given: Two algorithms Las Vegas Algorithms A, B
for the same decision problem (e.g., graph colouring)

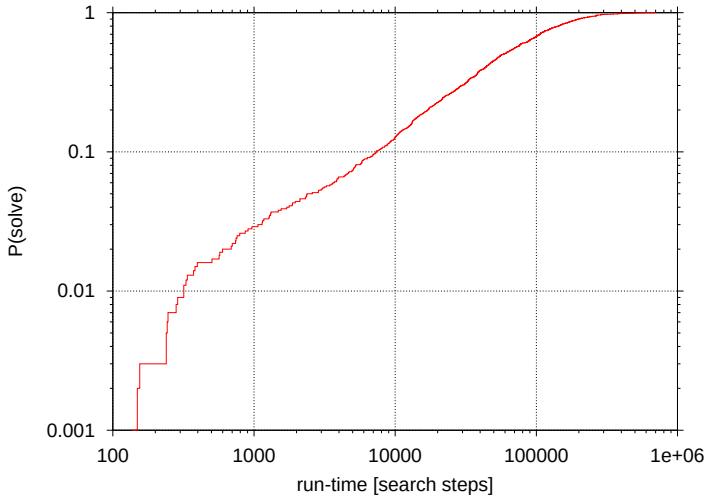
Want: Determine whether A is better than B w.r.t. run-time.

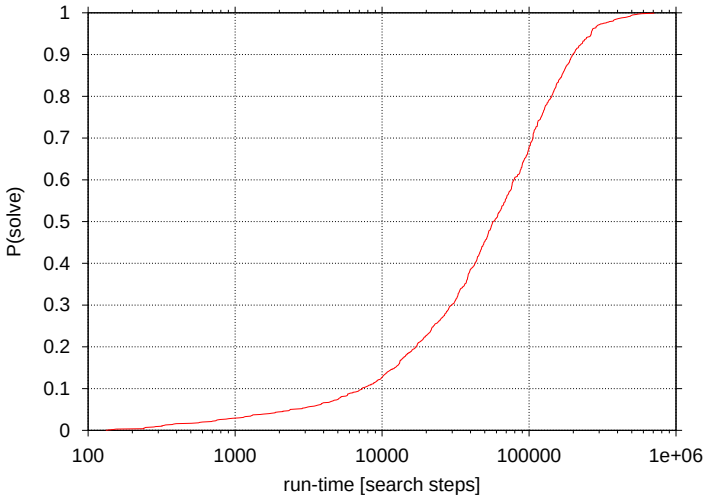

Raw run-time data (each spike one run)
run-time [CPU sec]
0
2
4
6
8
10
12
14
0
100
200
300
400
500
600
700
800
900
1000
run #

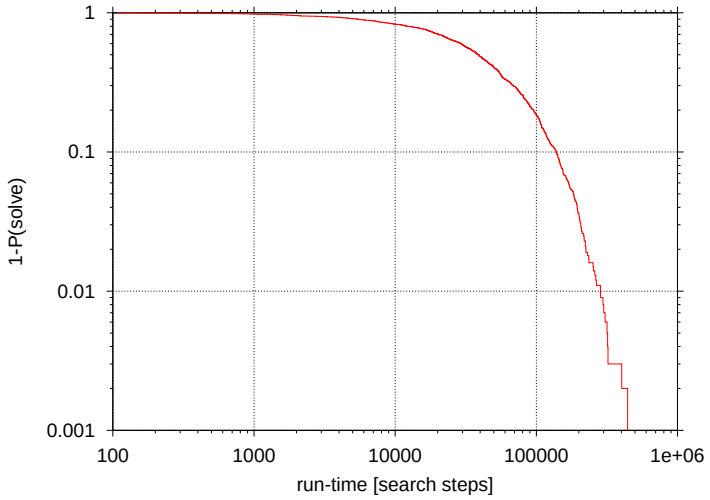

Run-Time Distribution
P(solve)
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
2
4
6
8
10
12
14
16
18
20
run-time [CPU sec]

RTD Graphs


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


P(solve)
run-time [search steps]
0.001
0.01
0.1
1
100
1000
10000
100000
1e+06


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


1-P(solve)
run-time [search steps]
0.001
0.01
0.1
1
100
1000
10000
100000
1e+06

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

Comparative performance analysis on single problem instance:

- measure RTDs
- check for probabilistic domination (crossing RTDs)
- use statistical tests to assess significance of performance differences (e.g., Mann-Whitney U-test)

Significance Performance Differences

Given: RTDs for algorithms A, B on the same problem instance

Hypothesis: There is a significant difference in the median of the RTDs (i.e., median performance of algorithm A is better than that of B or vice versa)

Test: Mann-Whitney U-Test

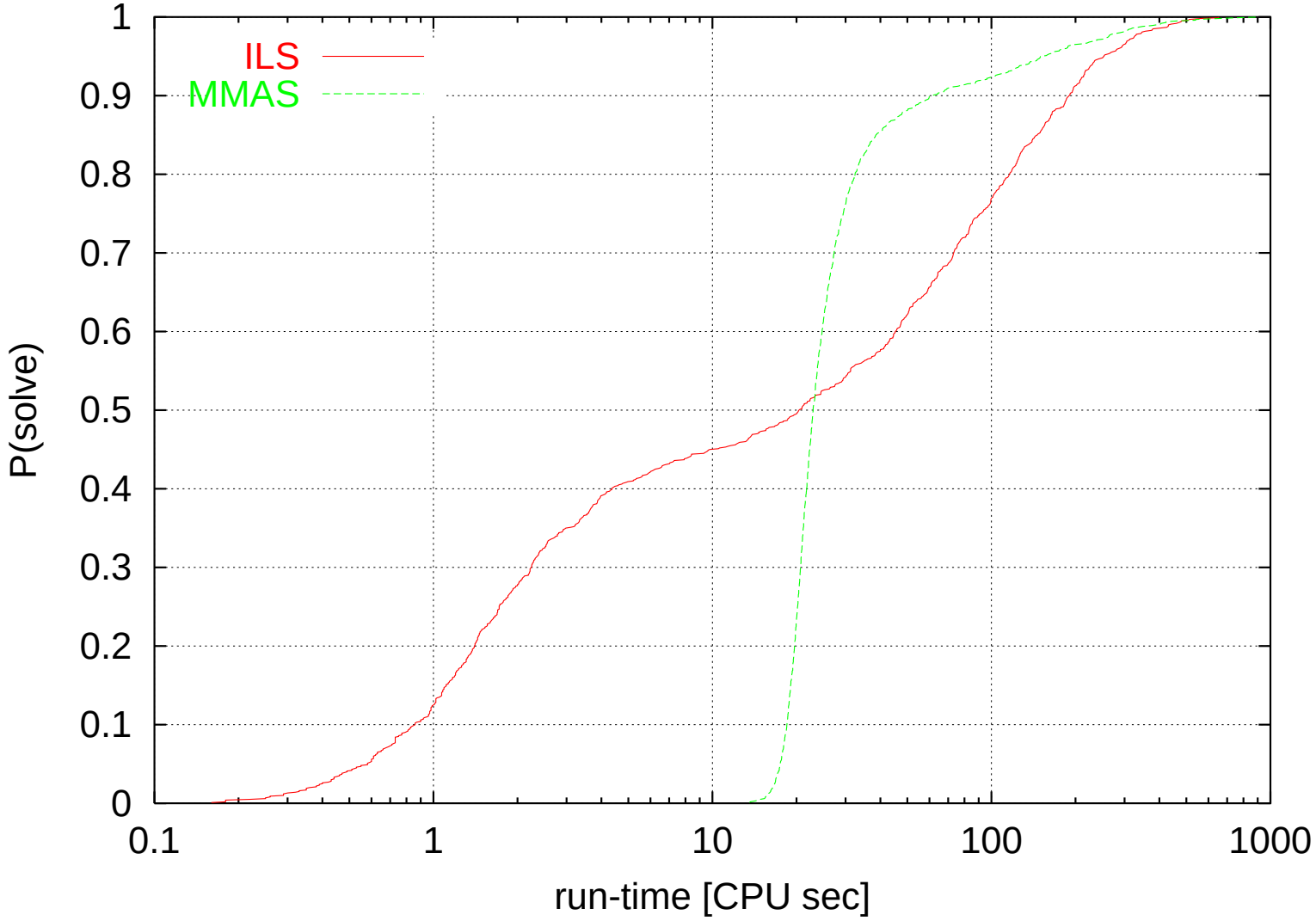
Note: Unlike the widely used t-test, the Mann-Whitney U-Test does *not* require the assumption that the given samples are normally distributed with identical variance.

Sample Sizes for Mann-Whitney U-Test

m_A/m_B: ratio between the medians of RTDs for A and B

sign. level 0.05, power 0.95		sign. level 0.01, power 0.99	
sample size	m_A/m_B	sample size	m_A/m_B
3010	1.10	5565	1.10
1000	1.18	1000	1.24
122	1.5	225	1.5
100	1.6	100	1.8
32	2.0	58	2
10	3.0	10	3.9

Performance comparison for ACO and ILS algorithm for TSP



Significance of Differences between RTDs

Given: RTDs for algorithms A, B on the same problem instance

Hypothesis: There is a significant difference between the RTDs (i.e., performance of algorithm A is different from that of B)

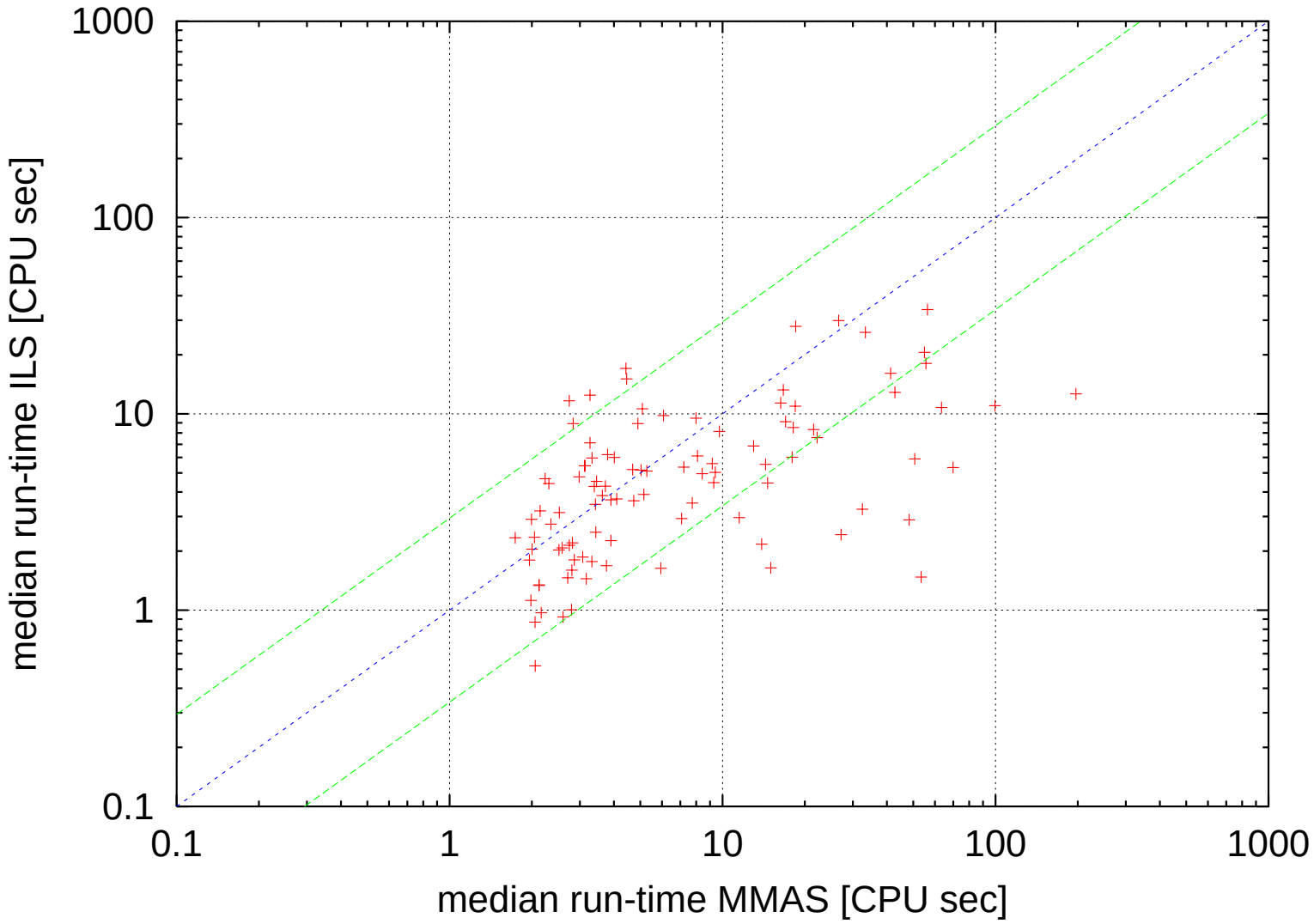
Test: Kolmogorov-Smirnov Test

Note: This test can also be used to test for significant differences between an empirical and a theoretical distribution.

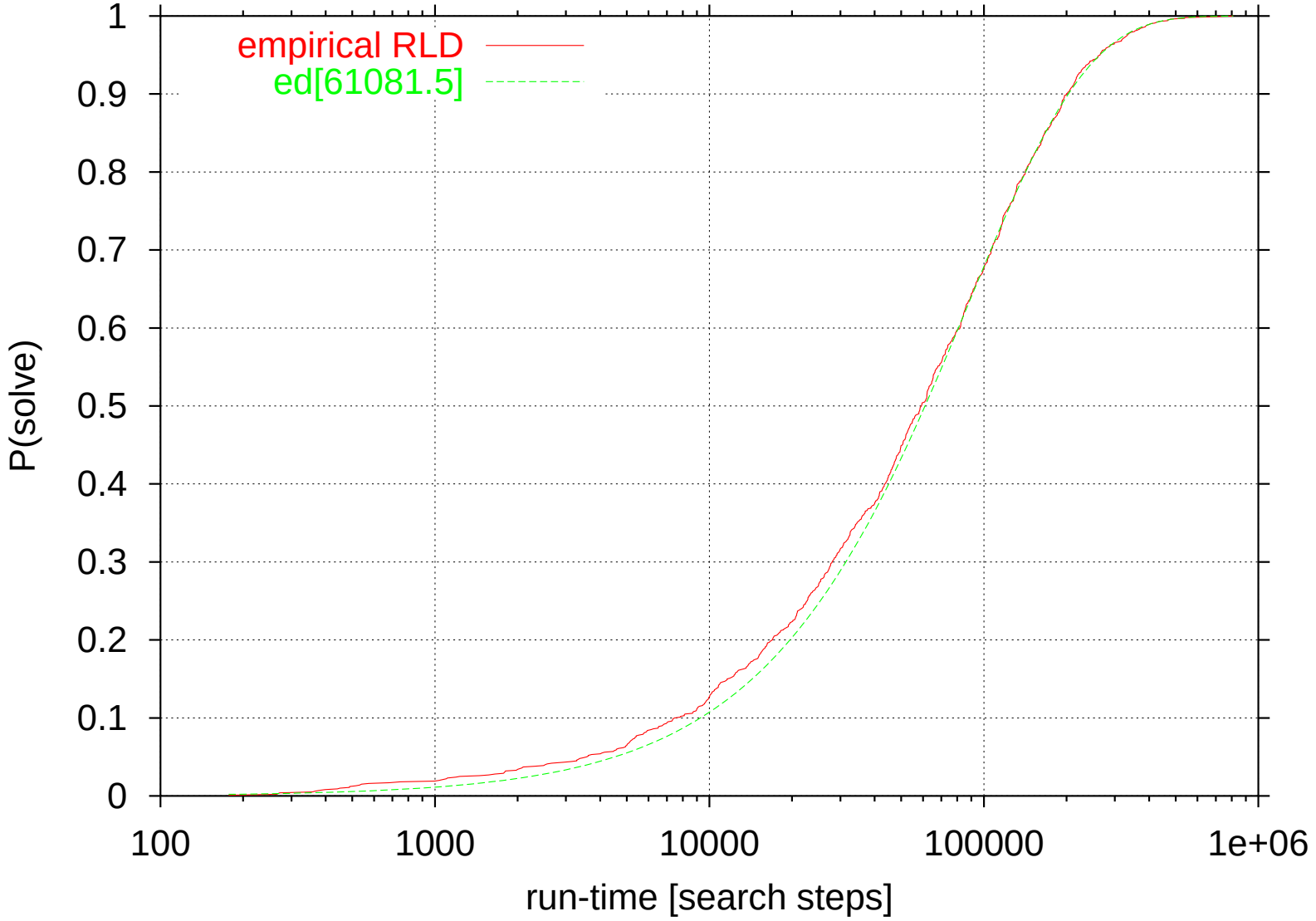
Comparative performance analysis for ensembles of instances:

- check for uniformity of RTDs
- partition ensemble according to probabilistic domination
- analyse correlation for (reasonably stable) RTD statistics
- use statistical tests to assess significance of performance differences across ensemble (e.g., Wilcoxon matched pairs signed-rank test)

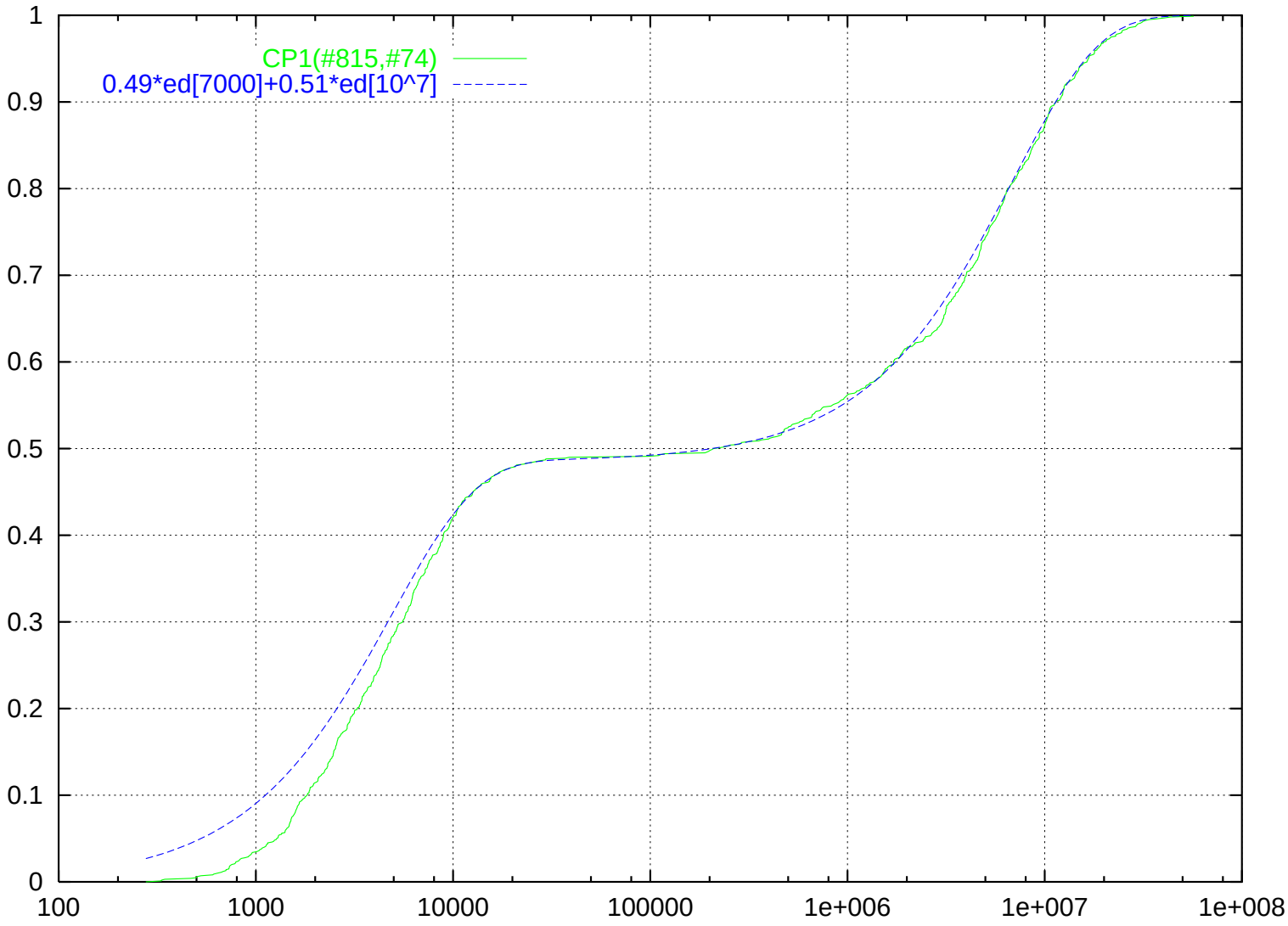
Peformance correlation for ACO and ILS algorithm for TSP



RTD Approximation with Exponential Distribution



RTD Approximation with Mixture of Exponential Distributions



Randomised Algorithms with One-Sided Error

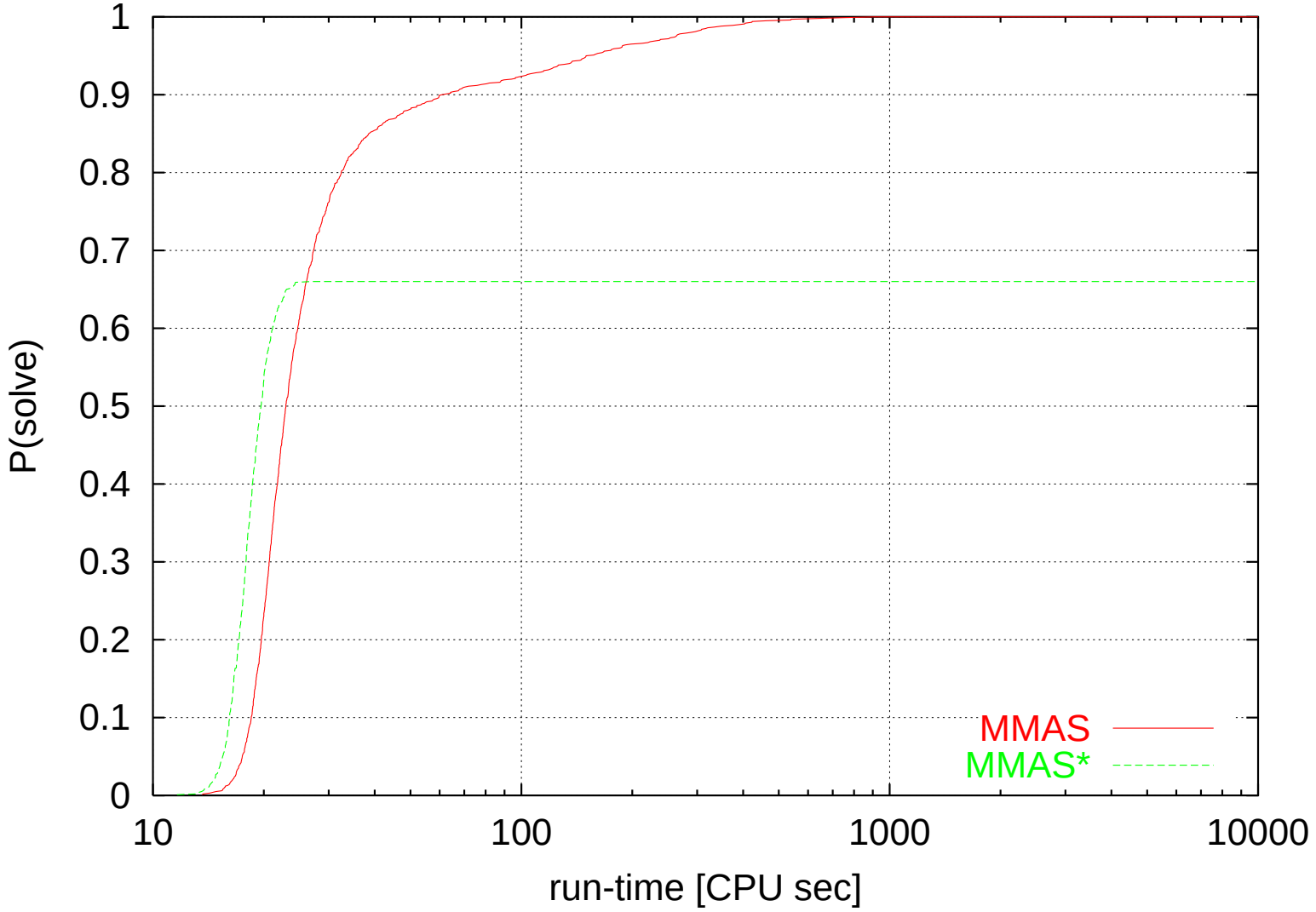
Types of Errors:

- false negatives (FN): incorrectly return “no” answer
- false positives (FP): incorrectly return “yes” answer

Monte Carlo Algorithm (MCA) with one-sided error:

- decision algorithm without false positives,
  i.e., “yes” answers are guaranteed to be correct
- false negatives may occur
- run-time for given problem instance (and parameter settings)
  is a random variable

Qualititative Differences between RTDs of two TSP Algorithms



Speed *vs.* Error Rate

Performance criteria:

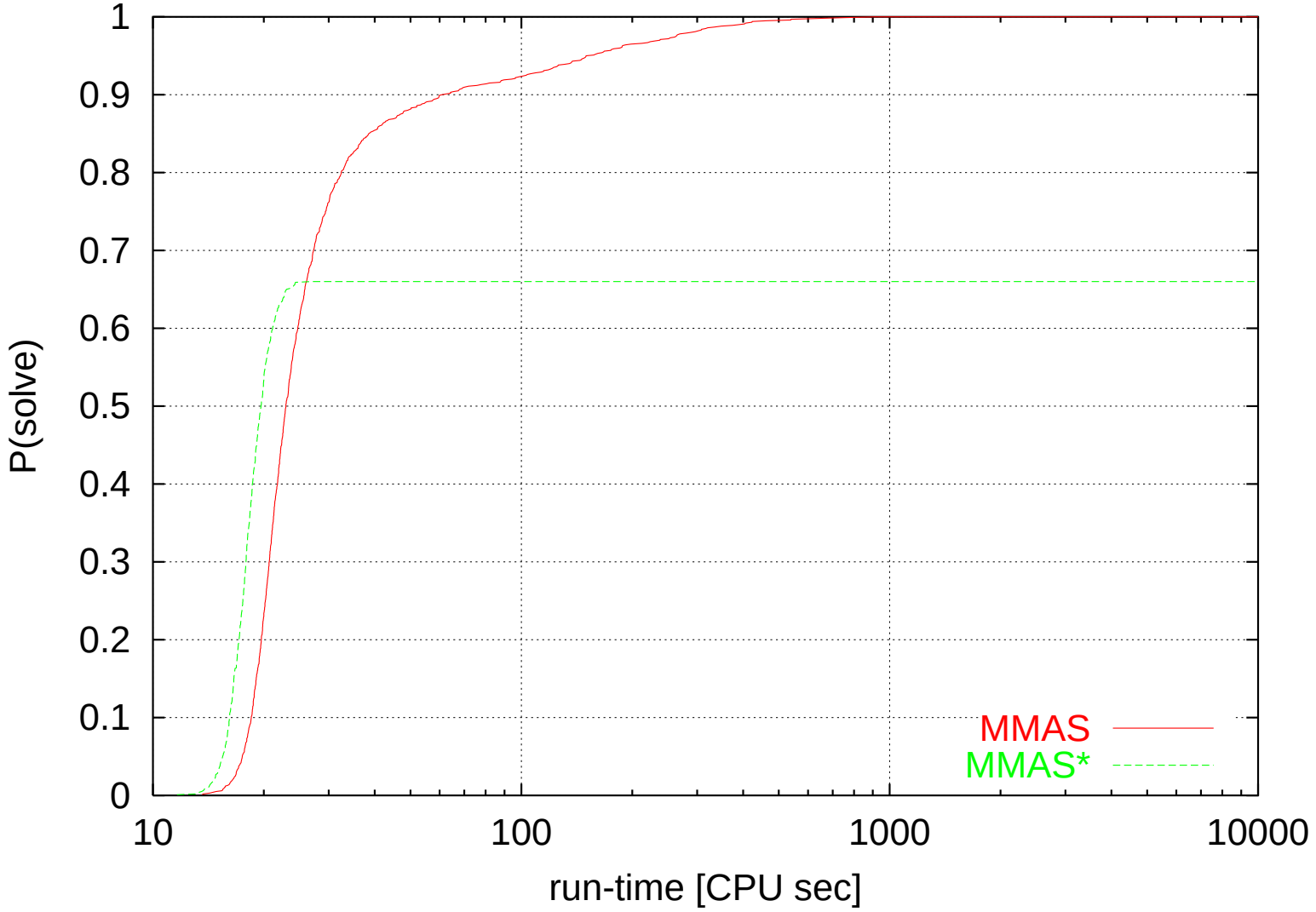
- run-time (distributions)
- success probability = 1 - error probability = limit of probability for producing correct "yes" answer for run-time $\to \infty$

Question: How to evaluate tradeoff between run-time and success probability?

Asymptotic Run-Time Behaviour

- *completeness*
  — for each problem instance P there is a time bound $t_{max}(P)$ for the time required by A to produce a correct answer
- *probabilistic approximate completeness (PAC property)*
  — for each "yes" problem instance the correct answer is produced by A with probability $\to 1$ as run-time $\to \infty$.
- *essential incompleteness*
  — for some "yes" instances, the probability for producing a "yes" answer is < 1 for run-time $\to \infty$.

Qualititative Differences between RTDs of two TSP Algorithms



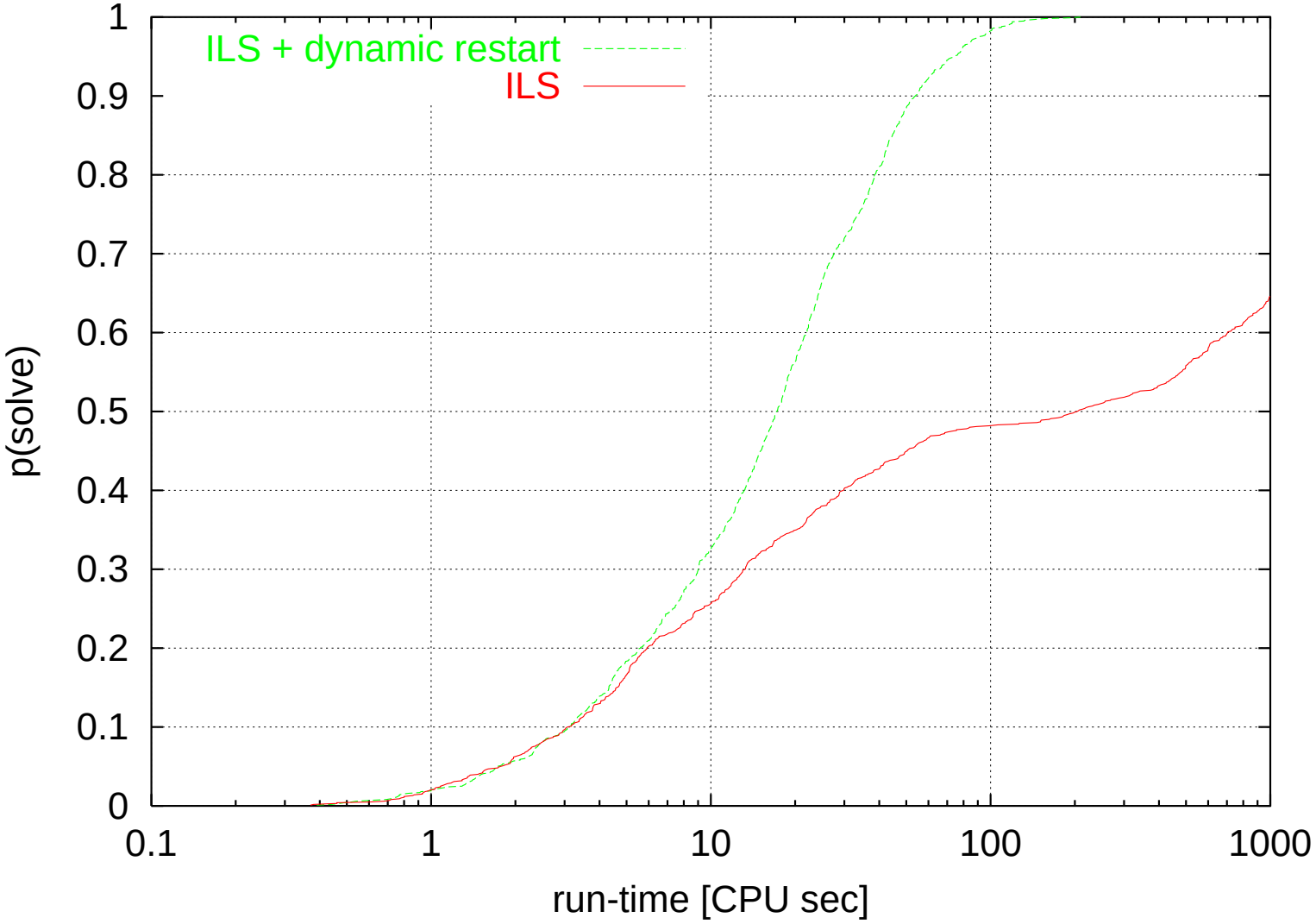
Multiple Independent Runs

Key Insight: By performing multiple independent runs of algorithm, we can trade off error probability against run-time.

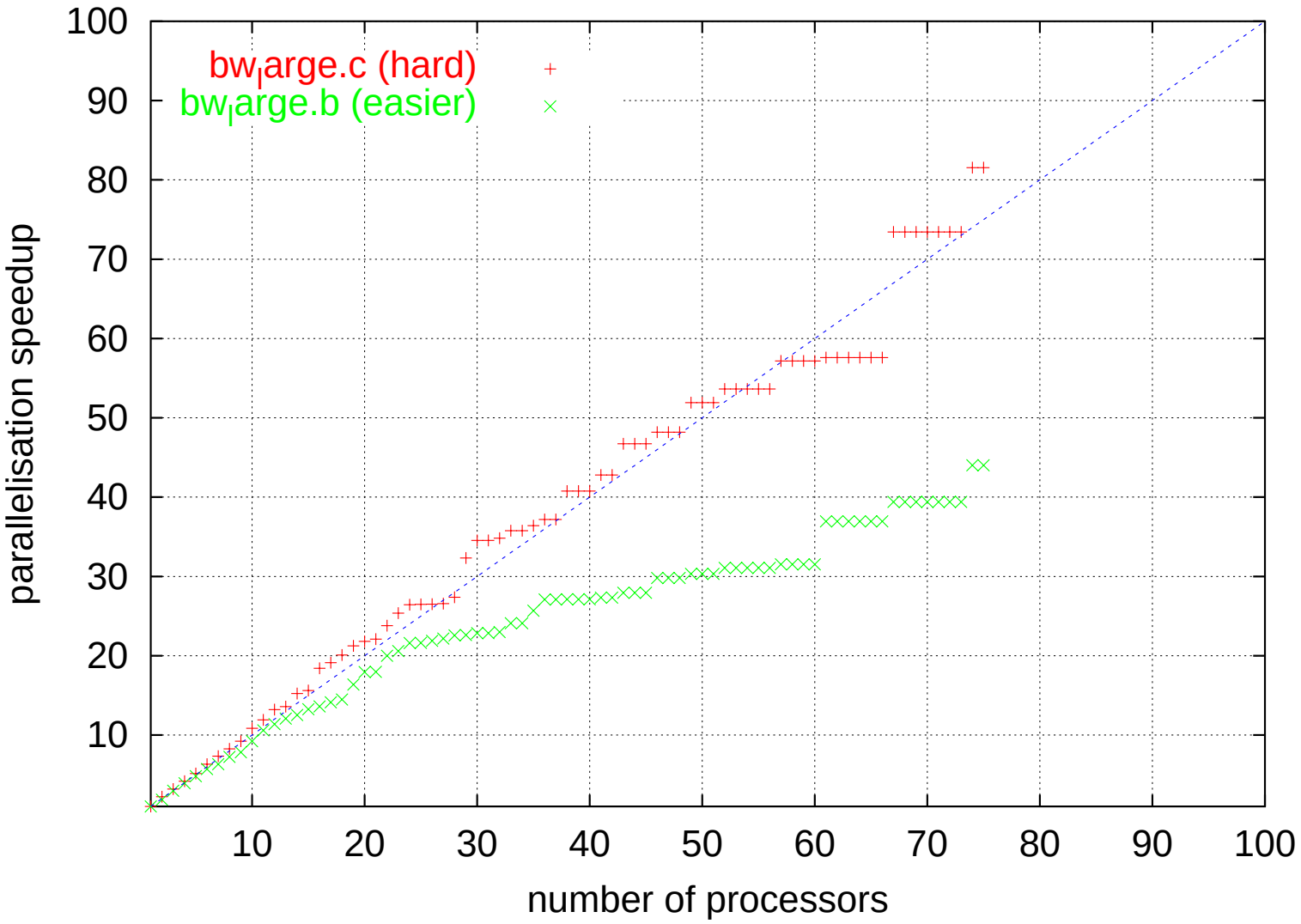
Practical Realisation:

- Run multiple copies of MCA in parallel on same problem instance (parallel processors, cluster of workstations, single CPU machine w/ time-sharing)
- Run multiple independent runs sequentially using cutoff and restart strategy

Effect of Dynamic Restart on ILS algorithm for TSP



Efficiency of multiple independent tries parallalelisation



Randomised Algorithms with Two-Sided Error

Monte Carlo Algorithm (MCA) with two-sided error:

- false positives and false negatives may occur
- run-time for given problem instance (and parameter settings) is a random variable

Sensitivity *vs.* Specificity

Sensitivity = TP/(TP+FN)

= fraction of “yes” instances correctly solved

Specificity = TP/(TP+FP)

= fraction of “yes” answers that are correct

Trade-offs between . . .

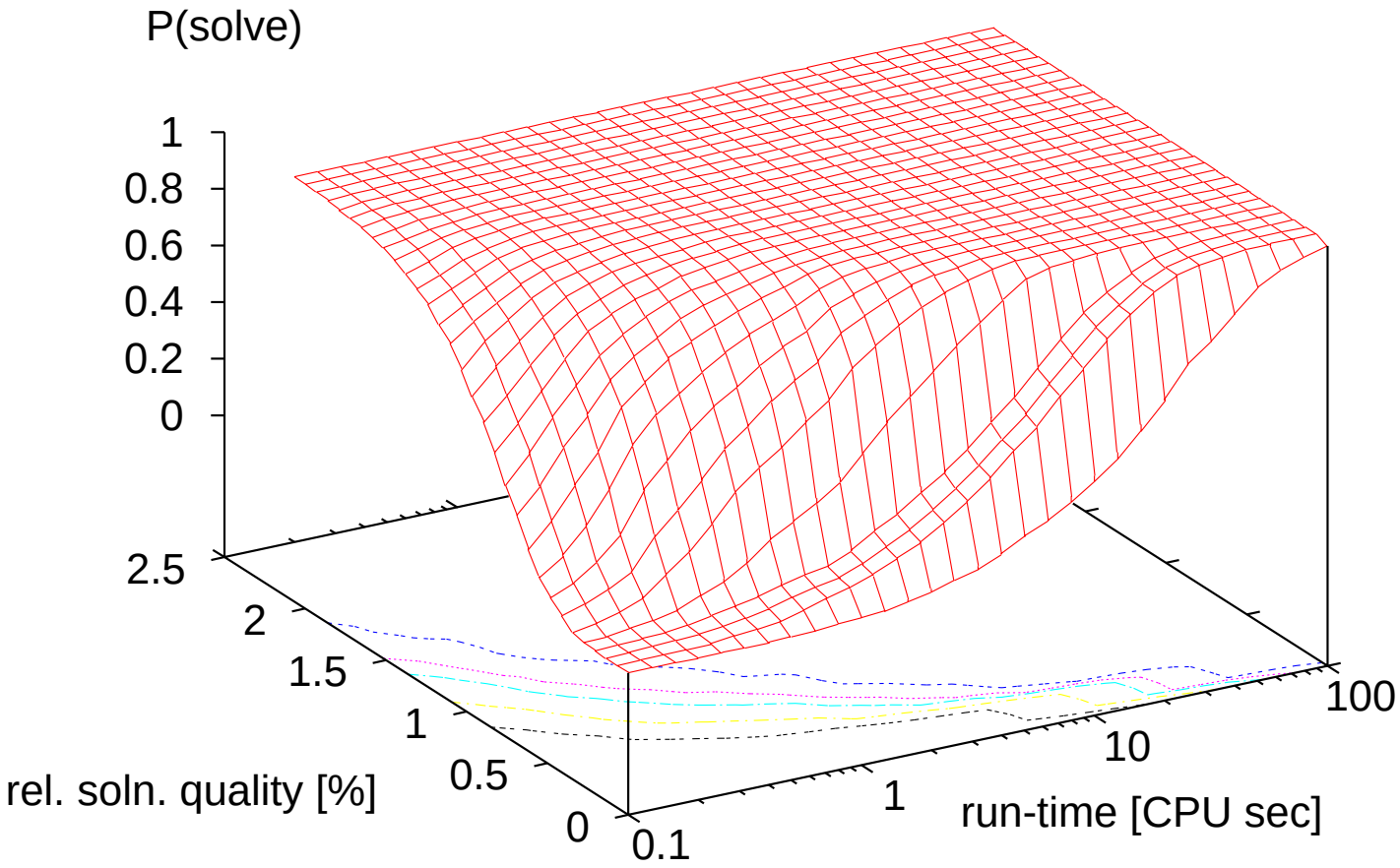
- sensitivity and specificity
- run-time and sensitivity/specificity

Optimisation Problems

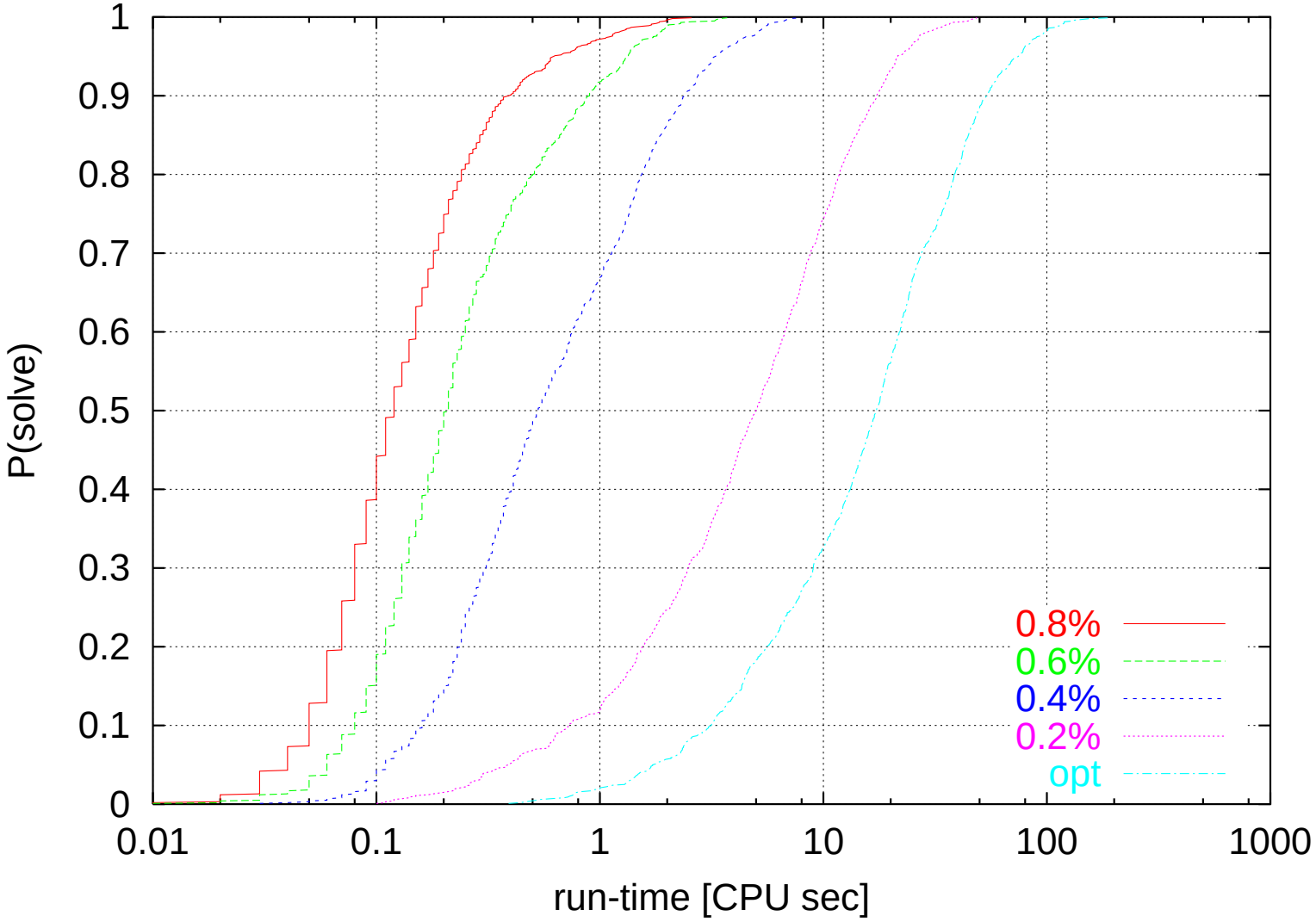
Given: Input data (e.g., graph G) and objective function f
(e.g., number of colours used in a given colouring of G)

Objective: Output optimal objective function value
(e.g., minimal number of colours required for a feasible colouring of G, i.e., chromatic number of G)

Bivariate RTD for ILS algorithm for TSP



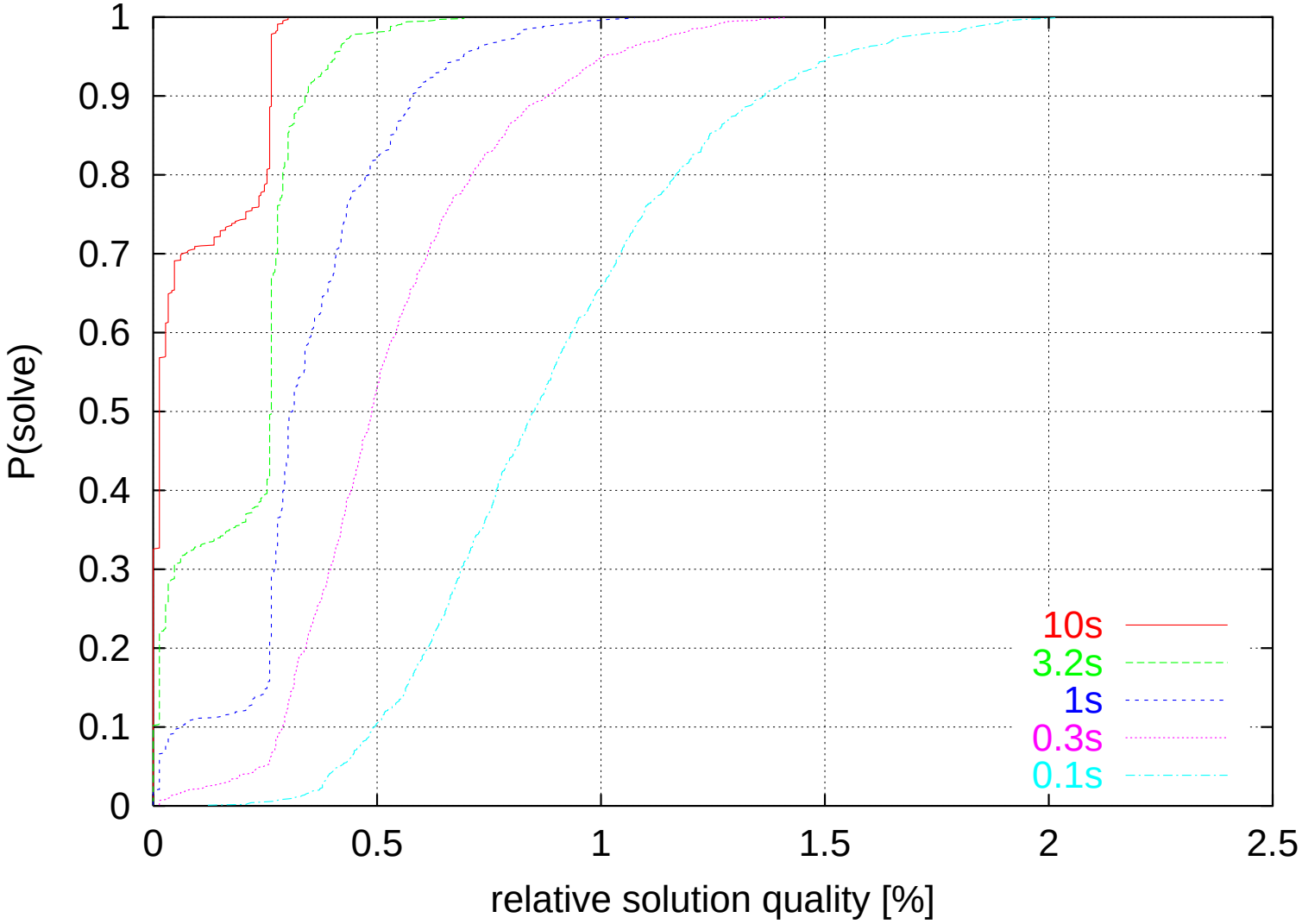
Qualified RTDs for ILS algorithms for TSP



RTD-based analysis of randomised optimisation algorithms:

- additionally, solution quality has to be considered
- introduce bounds on the desired solution quality
  $\leadsto$ qualified RTDs
- bounds can be chosen w.r.t. best-known or optimal solutions, lower bounds of the optimal solution cost etc.
- estimate run-time distributions for several bounds on the solution quality

SQDs for ILS algorithms for TSP



SQD-based methodology:

- run algorithm multiple times on given problem instance(s)
- estimate empirical solution quality distributions (SQDs) for different run-times
- get simple descriptive statistics (mean, stddev, percentiles, ...) from SQD data
- approximate empirical SQD with known distribution functions
- check statistical significance using goodness-of-fit test

Some questions in SQD analysis:

- How do the SQDs scale with increasing run-time?
- What is the limiting shape of the SQDs with increasing instance size?

Beyond Comparative Performance Analysis

Goal: Understand factors underlying algorithm's performance

$\rightsquigarrow$ typically requires domain and algorithm-specific approaches

Two general approaches:

- Systematically study variants of algorithms and problem instances
  ("study mutants")
- Build and analyse abstract models of algorithm (analytically/empirically)

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!