CSE 6140/ CX 4140
Computational Science and Engineering
ALGORITHMS

Coping with NP-completeness - 8
Empirical Analysis, Vertex Cover approximation, ILP

Instructor: Xiuwei Zhang
Assistant Professor
School of Computational Science and Engineering

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

Today's plan

Finish empirical analysis

An approximation algorithm for vertex cover

Integer Linear Programming

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

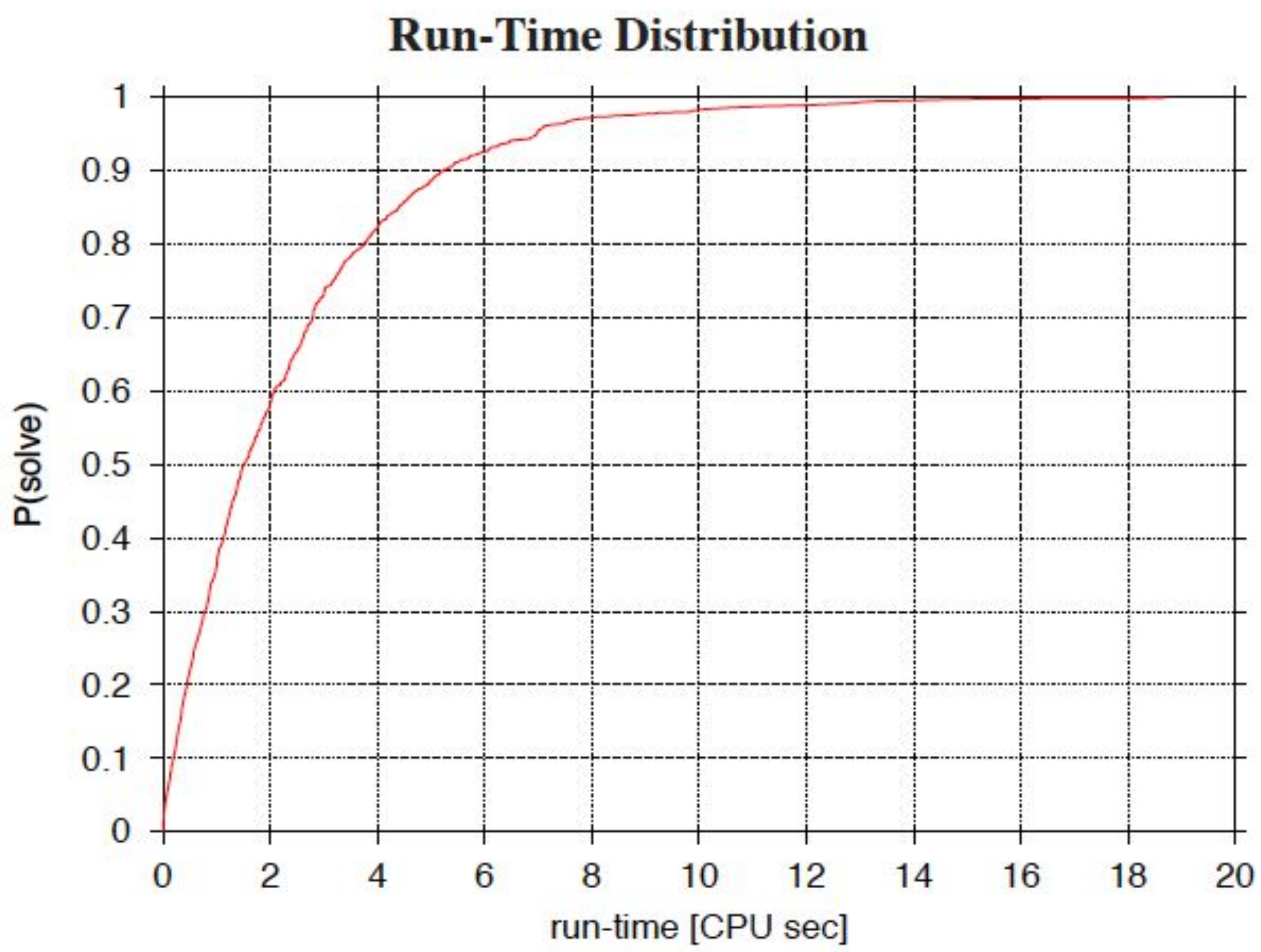
k'=8

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)


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'.

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.

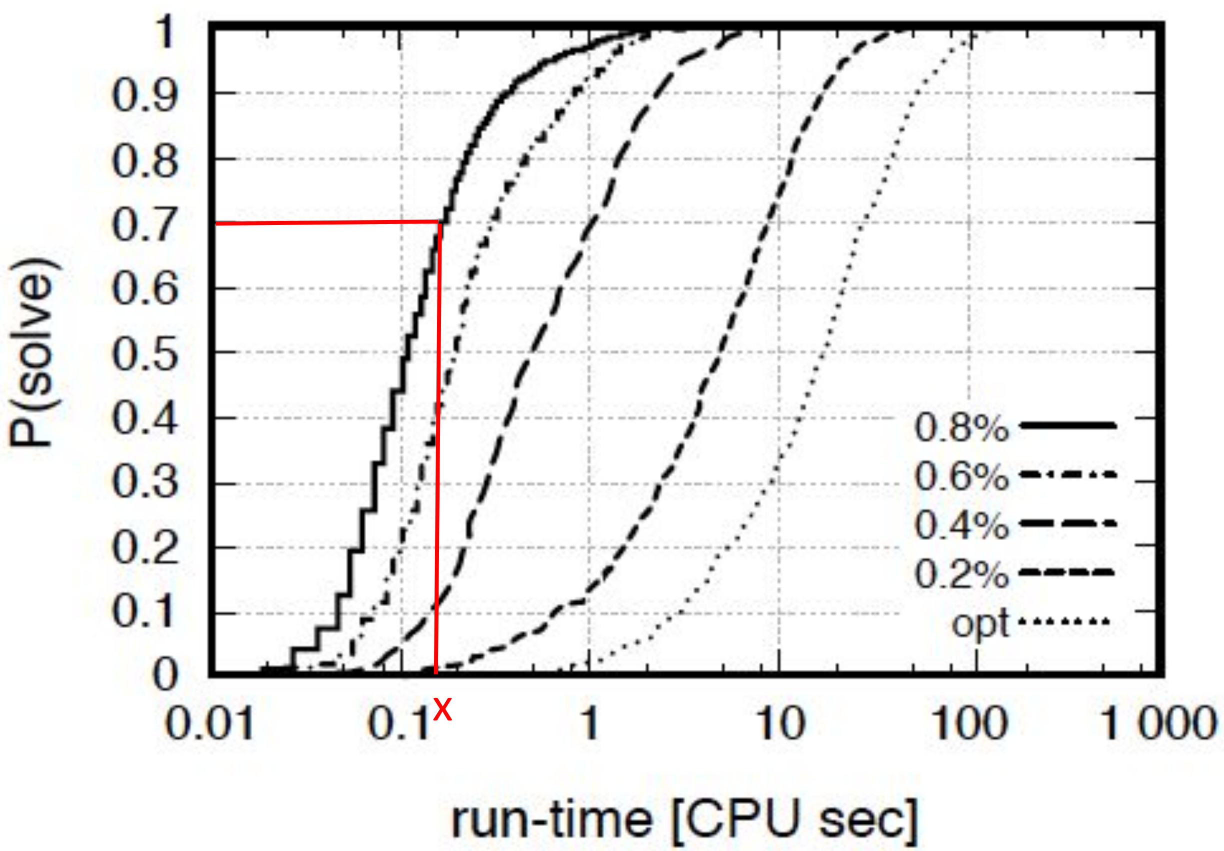
Qualified RunTime Distribution

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

Qualified RTDs for various solution qualities:

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?



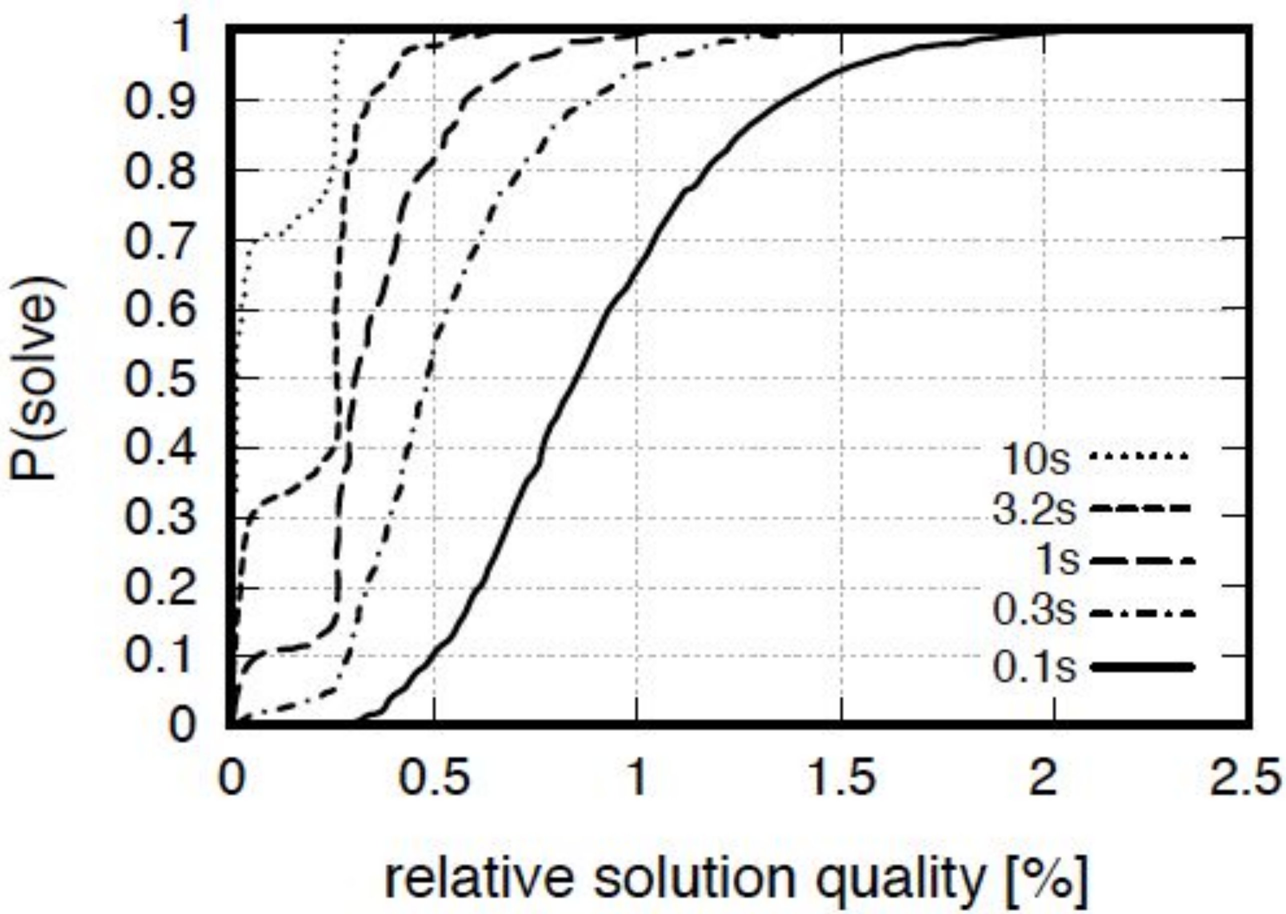
Solution quality distributions (SQDs)

- 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).

Solution quality distributions for various run-times:



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.

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.

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.

Probabilistic Domination

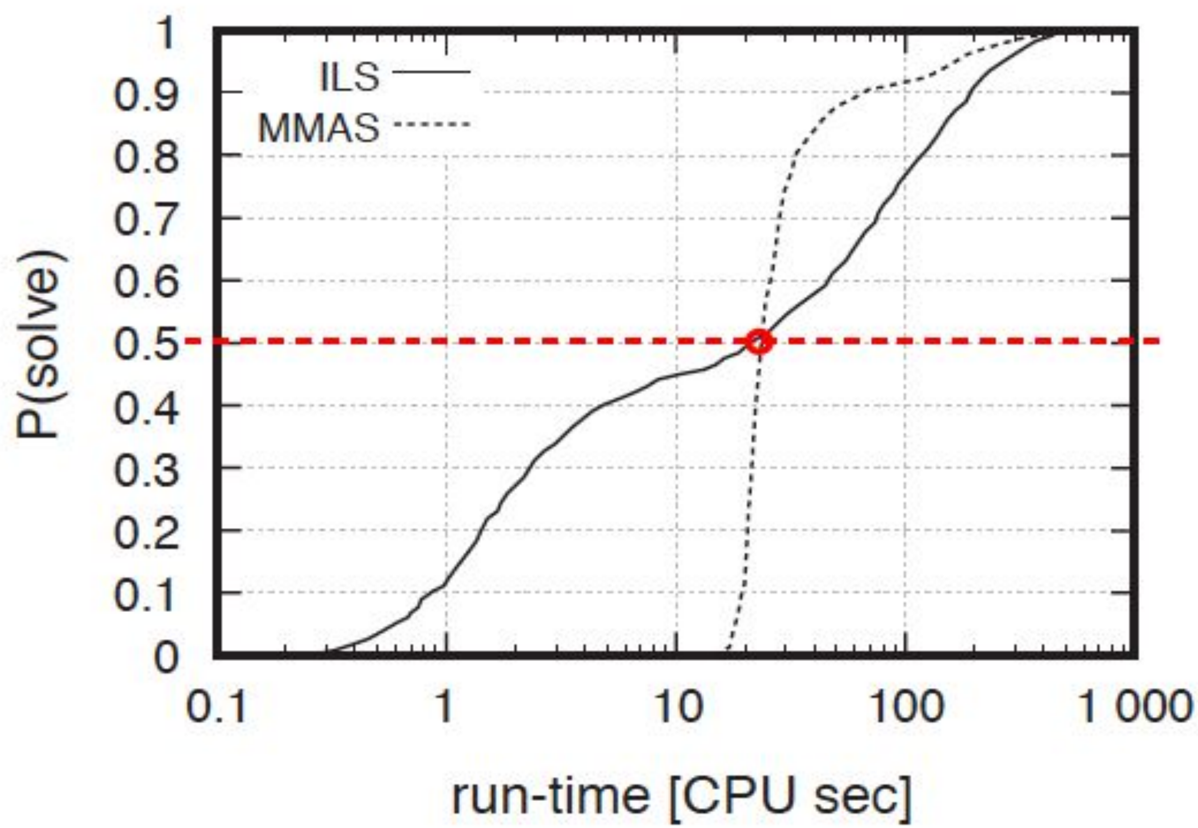
Definition: Algorithm A *probabilistically dominates* algorithm B on problem instance π, iff

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

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

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

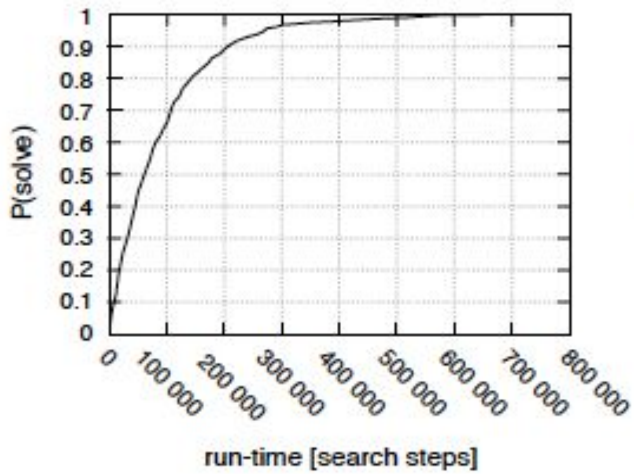
Example of crossing RTDs for two SLS algorithms for the TSP applied to a standard benchmark instance (1000 runs/RTD):

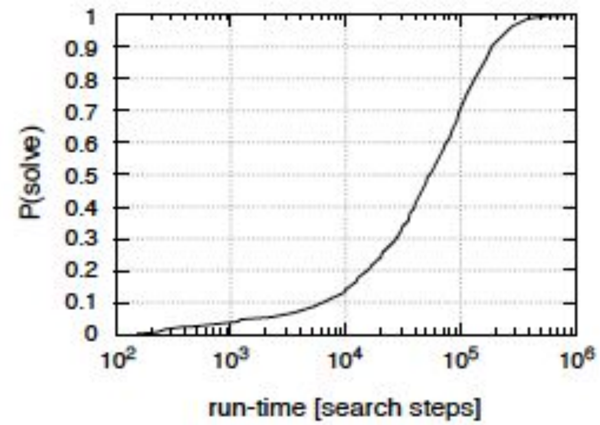


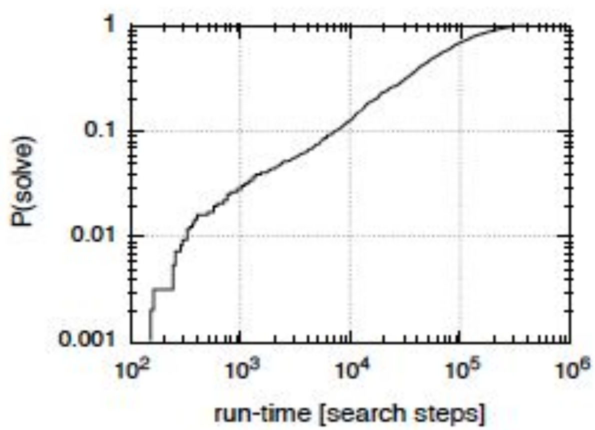
RTD plots are useful for the *qualitative analysis* of LVA behaviour:

- *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:



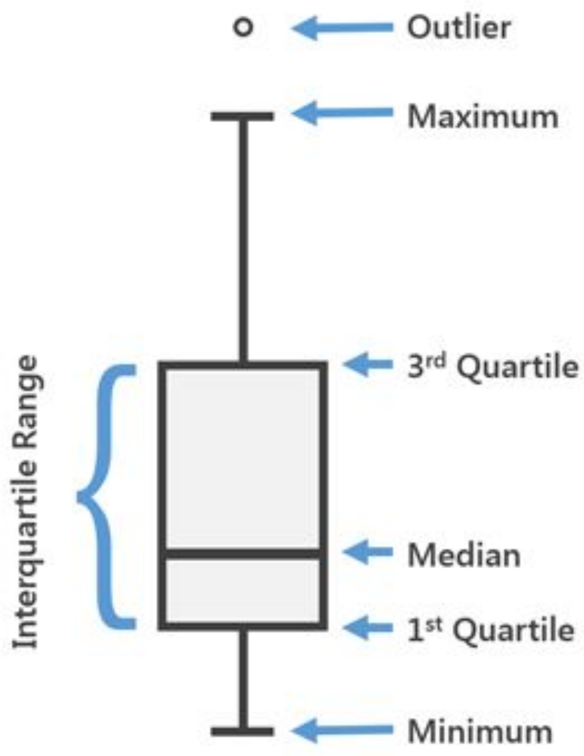




A few general guidelines:

- 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!

Boxplot of runtime



Measure of dispersion

- Sample range

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

- Sample variance

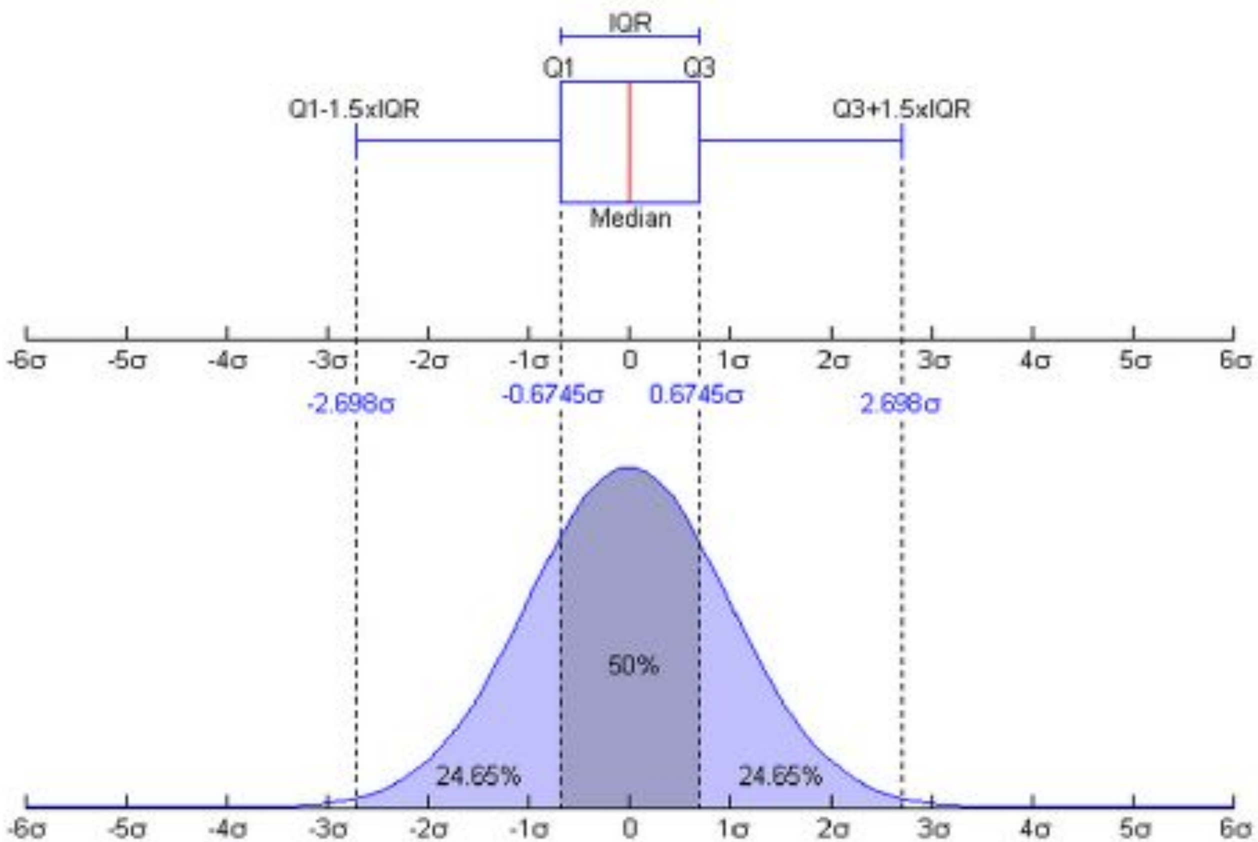
$$s^2 = \frac{1}{n-1}\sum(x_i - \bar{X})^2$$

- Standard deviation

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

- Inter-quartile range

$$IQR = Q_3 - Q_1$$



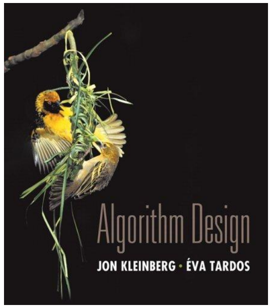
Boxplot and a probability density function (pdf) of a Normal N(0,1s2) Population. (source: Wikipedia)

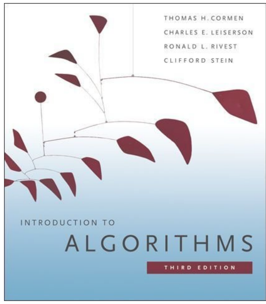
[see also: `http://informationandvisualization.de/blog/box-plot`]

VERTEX COVER APPROXIMATION – [CLRS 37.1]

Adapted from Slides by
Kevin Wayne.

And Bistra Dilkina, Anne Benoit





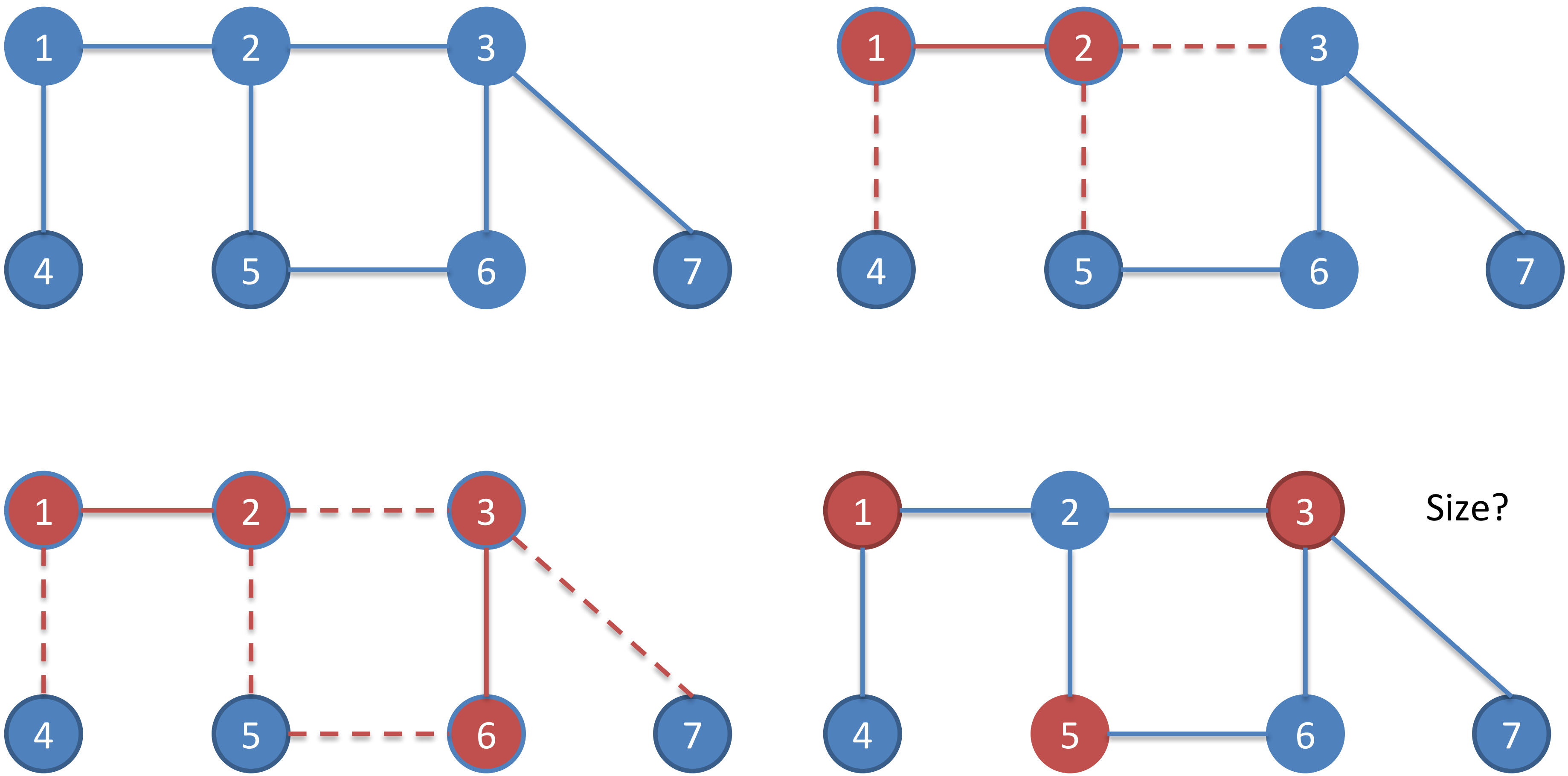
Approximate vertex-cover algorithm

Vertex cover: given a graph G=(V,E), find the *smallest* number of vertices that cover *each edge*
(each edge has at least one endpoint in the vertex cover set)

Approximation algorithm:

1. $C \leftarrow \varphi$ (the vertex cover)
2. $E' \leftarrow E$ (uncovered edges)
3. **while** $E' \neq \varphi$
4. **do** let (u,v) be an arbitrary edge of E'
5. $C \leftarrow C \cup \{u,v\}$
6. remove from E' every edge incident to either u or v.
7. return C

Example



Solution = 4

Optimal = 3

Ratio=1.33

2-approximate Vertex Cover

- Theorem.
  - APPROX-VERTEX-COVER is a poly-time 2-approximate algorithm, i.e., the size of returned vertex cover set is at most twice of the size of optimal vertex-cover.
- Proof:
  - It runs in poly time (linear time)
  - The returned set C is a vertex cover
    - every selected or deleted edge has endpoint in C,
    - and we continue until every edge is either selected or deleted

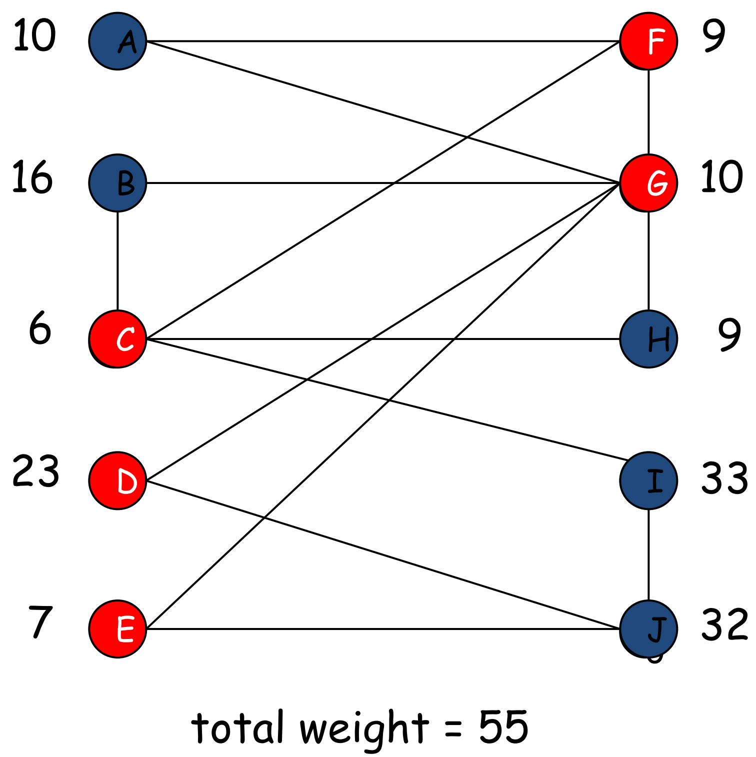
2-approximate Vertex Cover

- Proof continued
  - We will show $|C| \leq 2|C^*|$
  - Let A be the set of edges picked by the Approx. Algorithm and C* be the optimal vertex cover.
    - C* must include at least one end of each edge in A, since C* is a vertex cover
    - no two edges in A are covered by the same vertex in C*, since edges in A do not share endpoints (due to line 6)
    - so $|C^*| \geq |A|$ (at least one vertex from every edge in A)
    - Moreover, $|C|=2|A|$
    - (for each edge in A, we add 2 nodes to C, and edges in A do not share endpoints so each endpoint counts towards $|C|$)
    - so $|C|=2|A| \leq 2|C^*|$

Integer Linear Programming (ILP)

KT 11.6: Weighted Vertex Cover

<u>Weighted vertex cover</u> Given an undirected graph G = (V, E) with vertex weights $w_i \geq 0$, find a minimum weight subset of nodes S such that every edge is incident to at least one vertex in S.



Weighted Vertex Cover: IP Formulation

Weighted vertex cover Given an undirected graph G = (V, E) with vertex weights $w_i \geq 0$, find a minimum weight subset of nodes S such that every edge is incident to at least one vertex in S.

Integer linear programming formulation

- Model inclusion of each vertex i using a 0/1 variable x_i.

$$x_i = \begin{cases} 0 & \text{if vertex } i \text{ is not in vertex cover} \\ 1 & \text{if vertex } i \text{ is in vertex cover} \end{cases}$$

- Vertex covers in 1-1 correspondence with 0/1 assignments: $S = \{i \in V : x_i = 1\}$
- Objective function: minimize $\Sigma_i w_i x_i$.
- Constraints: must take either i or j for each edge (i,j) in E: $x_i + x_j \geq 1$.

Weighted Vertex Cover: ILP Formulation

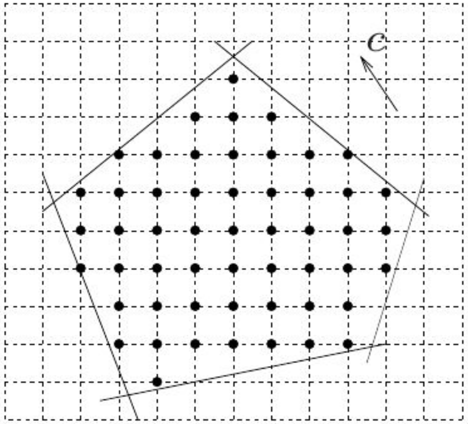
Weighted vertex cover. Integer linear programming (ILP) formulation.

$$
\begin{array}{llllll}
(ILP) & \min & \sum_{i \in V} w_i \, x_i & & & \\
 & \text{s. t.} & x_i + x_j & \geq & 1 & (i,j) \in E \\
 & & x_i & \in & \{0,1\} & i \in V
\end{array}
$$

Observation. If x* is optimal solution to (ILP), then $S = \{i \in V : x^*_i = 1\}$ is a min weight vertex cover.

Integer Linear Programming

$$\begin{array}{ll} \text{minimize} & c^T x \\ \text{subject to} & Ax \leq b \\ & x \in \mathbf{Z}^n \end{array}$$



Observation. Vertex cover formulation proves that integer linear programming is NP-hard search problem.

ILP for SAT

$$(x_1 \vee x_2 \vee x_3) \wedge \ldots \wedge (x_3 \vee \overline{x_4} \vee \overline{x_1})$$

Goal: Find a truth assignment to satisfy all clauses

Variables: x_1, x_2, x_3, x_4

Constraints:

$$\begin{aligned} x_1 + x_2 + x_3 &\geq 1 \\ x_3 + (1 - x_4) + (1 - x_1) &\geq 1 \\ x_i &= \{0, 1\} \end{aligned}$$

Objective function: max 1

ILP for Knapsack

KNAPSACK: Given a finite set X (with n items), nonnegative weights w_i, nonnegative values v_i, a weight limit W, find a subset S ⊆ X such that the value of S is maximum.

Variables: x_1 to x_n

Objective function:

$$\max \sum_{i=1..n} v_i x_i$$

Constraints:

$$\sum_{i=1..n} w_i x_i \leq W$$

$$x_i \in \{0,1\}, \ \text{for } i = 1..n$$

How does ILP help us find the vertex cover

Solving the ILP:

Relax to LP (linear programming)

Linear Programming

Linear programming. Max/min linear objective function subject to linear inequalities.

- Input: parameters c_j, b_i, a_{ij} .
- Output: real numbers x_j.

$$\begin{array}{llllll} \text{(P)} & \min & \sum_{j=1}^{n} c_j x_j & & & \\ & \text{s.t.} & \sum_{j=1}^{n} a_{ij} x_j & \geq & b_i & 1 \leq i \leq m \\ & & x_j & \geq & 0 & 1 \leq j \leq n \end{array}$$

$$\begin{array}{lllll} \text{(P)} & \min & c^t x & & \\ & \text{s.t.} & Ax & \geq & b \\ & & x & \geq & 0 \end{array}$$

Linear. No x^2, xy, $\arccos(x)$, $x(1-x)$, etc.

Simplex algorithm. [Dantzig 1947] Can solve LP in practice.

Ellipsoid algorithm. [Khachian 1979] Can solve LP in poly-time.

Weighted Vertex Cover: LP Relaxation

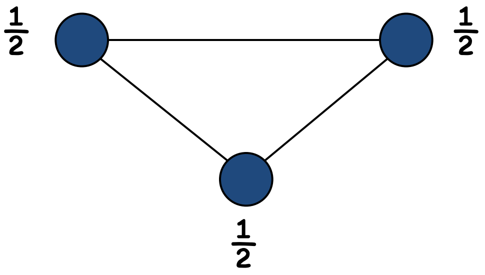
Weighted vertex cover. Linear programming formulation.

$$\begin{array}{llll} (LP)\ \min & \sum_{i \in V} w_i x_i & & \\ \text{s. t.} & x_i + x_j & \geq 1 & (i,j) \in E \\ & x_i & \geq 0 & i \in V \end{array}$$

Observation. Optimal value of (LP) is ≤ optimal value of (ILP).

Pf. LP has fewer constraints. Any solution to ILP is also solution to LP

Note. LP is not equivalent to vertex cover.



Q. How can solving LP help us find a small vertex cover?

A. Solve LP and round fractional values: x_i>=1/2 become 1, x_i < ½ become 0

Weighted Vertex Cover

Theorem. If x* is optimal solution to (LP), then $S = \{i \in V : x^*_i \geq ½\}$ is a vertex cover whose weight $\Sigma_{i\in S} w_i$ is at most twice OPT(Vertex Cover).

Pf. [S is a vertex cover]

- Consider an edge $(i, j) \in E$.
- Since $x^*_i + x^*_j \geq 1$, either $x^*_i \geq ½$ or $x^*_j \geq ½$ $\Rightarrow$ (i, j) covered.

Pf. [S has desired cost, $w(S) <= 2w(S^{VCOPT})$]

- Let S^{VCOPT} be optimal vertex cover. Corresponds to a soln of LP with x_i=1 if i in S^{VCOPT}, and 0 otherwise. Then

$$w(S^{VCOPT}) \geq \sum_{i\in V} w_i x_i^* \geq \sum_{i\in S} w_i x_i^* \geq \tfrac{1}{2}\sum_{i\in S} w_i = \tfrac{1}{2} w(S)$$

soln corresponding to S^{VCOPT} cannot be better than opt LP solution x*, since LP is a relaxation

Drop i with $x^*_i < \frac{1}{2}$, Keep $x^*_i >= \frac{1}{2}$

$x^*_i \geq \frac{1}{2}$ For all i in S

Theorem. 2-approximation algorithm for weighted vertex cover.