Computer science

— Theory vs Experimentation —

Part 2: Experimental Analysis of Algorithms

Christine Solnon

INSA de Lyon - 5IF

2023 / 2024

What have we learned so far?

Theory may be used to study a few things...

- Complexity and decidability of a problem
- Complexity, correctness, completeness and termination of an algorithm

...with some limits

- A theoretical complexity gives a growth order
  $\leadsto$ Asymptotic convergence when the input size tends to infinity
- An hand-made proof may contain errors
- Static analysis may raise false alarms
- ...

It's time to compare theory with practice

But the experimental analysis must be rigorous if we want it to be useful, reliable, reproducible and efficient!
$\leadsto$ Use a scientific approach to experimentally analyse algorithms

Plan

Experimental Process [McGeoch 2012]

Step 1: Prepare the experiment

- Formulate a question
- Design the experiment
  $\rightsquigarrow$ What should we measure? On which test suite? ...
- Prepare the test environment
  $\rightsquigarrow$ Scripts, Computers, Data analysis tools, ...

Step 2: Perform the experiment

- Run scripts
- Collect results

Step 3: Analyse results

- If question not answered, then go back to Step 1
- If question answered, then publish!

Iterative process ...

Example: Evaluation of a new algorithm for SAT

SAT problem (recall):

- Input: A Boolean formula composed of m clauses and n variables
- Output: A Boolean value
- Postrelation: return true if the formula can be satisfied; false otherwise

Examples of questions:

- Average time to solve an instance?
- Empirical complexity of the program?
- Influence of n and m on the run time?
- Influence of the structure of instances on the run time?
- What are the instances for which the algorithm is (not) efficient?
- Influence of the algorithm parameters?
- Is the algorithm competitive with state-of-the-art approaches?
- What are the best data structures for implementing the algorithm?

Two types of experiments

Exploratory experiment:

Identify what should be intensively experimented:

- Relevant questions?
- Parameters which have an impact on the solution process?
- Relevant instances?
- ...

$\leadsto$ Short cycles for preparing an intensive experiment

Intensive experiment:

- Use an efficient and automated experimental process
  - Goals are well defined
  - Cycles may be quite long (up to several months in some cases...)

The vocabulary of experimentation

Performance criterion: What do we want to evaluate?
$\rightsquigarrow$ Duration, Solution quality, Memory, Robustness, ...

Performance measure: What should we measure to evaluate a criterion?
$\rightsquigarrow$ CPU time, Gap to a reference solution, ...

Parameter: Feature that may change the value of a performance measure

- Algorithm parameters
  $\rightsquigarrow$ Thresholds, Frequencies, Heuristics, ...
- Instance parameters
  $\rightsquigarrow$ Number of variables or clauses of a SAT formula, ...
- Environment parameters
  $\rightsquigarrow$ Compiler, OS, Computer, ...

Factor: Parameter changed during the experiment

Level: Value assigned to a factor

Design point: Combination of levels to be tested during an experiment

Run: Execution for one design point

Reproducibility of an experiment

Why reproducing an experiment?

- To check published results
- To compare a new algorithm with a published one
- To evaluate a published algorithm on new benchmarks
- ...

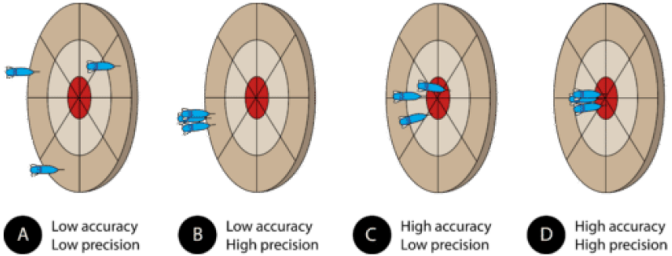
Why is it difficult to reproduce an experiment?

- All informations must be available (transparency):
  - Open source + Open data
  - Values of all parameters
  - Considered environment (processor, OS, compiler, ...)
  - Tools used to launch runs and analyse results
  - ...
- All tools must (still) be available $\leadsto$ Provide virtual machines
- Some measures may be uncertain

Uncertainty of Measures

Examples of uncertainty causes:

- Generator of (pseudo-)random numbers
  $\rightsquigarrow$ In C, the implementation of `rand()` depends on the compiler!
- CPU time depends on the computer load (among other things...)
- Programs may use several threads or processors
- Number of bits used to encode numbers
- ...
- and cosmic rays!



Different Reproducibility Levels [ACM 2016]

Repeatability

Same experimental conditions, same team

Replicability

Same experimental conditions, different team

Reproducibility

Different experimental conditions, different team

The Machine Learning Reproducibility Checklist (1/2)

https://www.cs.mcgill.ca/~jpineau/ReproducibilityChecklist.pdf

For all **models** and **algorithms** presented, check if you include:

- ❑ A clear description of the mathematical setting, algorithm, and/or model.
- ❑ A clear explanation of any assumptions.
- ❑ An analysis of the complexity (time, space, sample size) of any algorithm.

For any **theoretical claim**, check if you include:

- ❑ A clear statement of the claim.
- ❑ A complete proof of the claim.

For all **datasets** used, check if you include:

- ❑ The relevant statistics, such as number of examples.
- ❑ The details of train / validation / test splits.
- ❑ An explanation of any data that were excluded, and all pre-processing step.
- ❑ A link to a downloadable version of the dataset or simulation environment.
- ❑ For new data collected, a complete description of the data collection process, such as instructions to annotators and methods for quality control.

The Machine Learning Reproducibility Checklist (2/2)

https://www.cs.mcgill.ca/~jpineau/ReproducibilityChecklist.pdf

For all shared **code** related to this work, check if you include:

- ❑ Specification of dependencies.
- ❑ Training code.
- ❑ Evaluation code.
- ❑ (Pre-)trained model(s).
- ❑ README file includes table of results accompanied by precise command to run to produce those results.

For all reported **experimental results**, check if you include:

- ❑ The range of hyper-parameters considered, method to select the best hyper-parameter configuration, and specification of all hyper-parameters used to generate results.
- ❑ The exact number of training and evaluation runs.
- ❑ A clear definition of the specific measure or statistics used to report results.
- ❑ A description of results with central tendency (e.g. mean) & variation (e.g. error bars).
- ❑ The average runtime for each result, or estimated energy cost.
- ❑ A description of the computing infrastructure used.

How not to do it [Gent et al 1997] (1/2)

Getting started (Implementation step)

- Don't trust yourself
- Do make it fast enough
- Do use version control

Experimental design

- Do measure with many instruments
- Do use different hardware
- Do vary all relevant factors
- Don't change two things at once
- Do measure CPU time
- Do collect all data possible
- Do be paranoid
- Do check your solutions
- Do it all again
- Do use the same instances
- Don't ignore crashes
- Do it often and do it big
- Don't kill your machines
- Do look for scaling results
- Do be stupid

How not to do it [Gent et al 1997] (2/2)

Data analysis

- Do look at the raw data
- Do look for good views
- Don't discard data
- Do face up to the consequences of your results
- Don't reject the obvious

Presentation of results

- Do present statistics
- Do report negative results
- Don't push deadlines
- Do report important implementation details
- Do check your references

Problems with random numbers

- Don't trust your source of random numbers
- Do understand your instance generator
- Do control sources of variation

Random numbers should not be generated with a method chosen at random

D. Knuth

Plan

Choice of a Benchmark

The benchmark depends on the question addressed by the experiment

- Is my program correct?
  $\rightsquigarrow$ Stress-test instances (boundary instances, happy path, ...)
- How does it behave in the worst-case?
  $\rightsquigarrow$ Worst-case/bad-case instances
- What are its scale-up properties wrt some instance parameters?
  $\rightsquigarrow$ Random instances
- Does it scale well for a given application?
  $\rightsquigarrow$ Real-world instances
- Is it competitive with state-of-the-art approaches?
  $\rightsquigarrow$ Public benchmark

Homogeneous vs Heterogeneous Benchmarks

- Homogeneous benchmark $\Rightarrow$ The analysis of results is simplified
- Heterogeneous benchmark $\Rightarrow$ Results are more general
  $\rightsquigarrow$ Decompose benchmarks in homogeneous classes to analyse results

Hardness of Instances

Beware of ceil/floor effects!

- Extreme instances are useless for comparing algorithms
  - Too easy $\Rightarrow$ Quickly solved by all algorithms
  - Too hard $\Rightarrow$ No algorithm can solve them
- Reduce the number of instances that are too easy or too hard
- Gradually increase instance hardness

Factors that may influence hardness:

- Input size
- Structure of input data
  $\rightsquigarrow$ Example: Tree width of the constraint graph
- Constrainedness (for decision problems)
  $\rightsquigarrow$ Phase transition
- Distribution of local optima (for optimisation problems)
  $\rightsquigarrow$ Search landscape

Phase transition (1/2)

Ex.: Satisfiability of a Boolean formula with n var. and p clauses (SAT)

- Hardness depends on n...
- ... but also on the ratio between p and n
  - p/n small $\Rightarrow$ under-constrained instance $\Rightarrow$ Easy (except for rare cases which are exceptionally hard!)
  - p/n large $\Rightarrow$ over-constrained instance $\Rightarrow$ Easy
  - Between these two cases, things become difficult!

Experiment [Leyton-Brown et al 2014]:

- Randomly generate 3-SAT instances with $n = 400$
- Each instance = a point (x, y)
  - $x = p/n$
  - $y = \log_{10}(\text{runtime})$
  - colour=black if feasible
  - colour=pink if infeasible

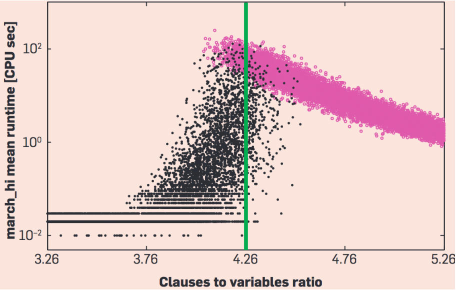
Phase transition (1/2)

Ex.: Satisfiability of a Boolean formula with n var. and p clauses (SAT)

- Hardness depends on n...
- ... but also on the ratio between p and n
  - p/n small $\Rightarrow$ under-constrained instance $\Rightarrow$ Easy (except for rare cases which are exceptionally hard!)
  - p/n large $\Rightarrow$ over-constrained instance $\Rightarrow$ Easy
  - Between these two cases, things become difficult!

Experiment [Leyton-Brown et al 2014]:

- Randomly generate 3-SAT instances with $n = 400$
- Each instance = a point (x, y)
  - $x = p/n$
  - $y = \log_{10}(\text{runtime})$
  - colour=black if feasible
  - colour=pink if infeasible



Phase Transition (2/2)

What is a phase transition?

- Abrupt state change (satisfiable vs unsatisfiable) wrt parameters
  $\rightsquigarrow$ For uniform 3-SAT: When $p/n = 4.26$
- Corresponding to a hardness pic independent from the solving approach

How to locate the phase transition?

Compute the probability that an instance is feasible:

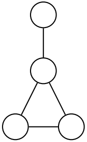
- Close to 0 $\rightsquigarrow$ Over-constrained instance (easy)
- Close to 0.5 $\rightsquigarrow$ Critically constrained instance (hard)
- Close to 1 $\rightsquigarrow$ Under-constrained instance (easy)

References:

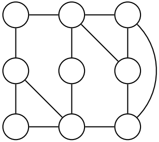
- P. Cheeseman, B. Kanefsky, W. Taylor (1991): *Where the Really Hard Problems Are*. IJCAI
- K. Leyton-Brown, H. Hoos, F. Hutter, L. Xu (2014): *Understanding the Empirical Hardness of NP-Complete Problems*. Communications of the ACM
- C. Solnon (2021): *Où sont les problèmes difficiles*. Tangente (Hors-série 75)

Illustration on the Subgraph Isomorphism Problem (SIP)

Goal: Search for a copy of a pattern graph G_p in a target graph G_t



$G_p = (N_p, E_p)$



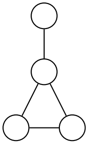
$G_t = (N_t, E_t)$

Find an injective mapping $f : N_p \rightarrow N_t$

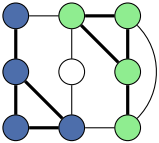
- Non-induced case:
  - $\forall (u, v) \in E_p : (f(u), f(v)) \in E_t$
- Induced case:
  - $\forall (u, v) \in E_p : (f(u), f(v)) \in E_t$
  - $\forall u, v \in N_p : (u, v) \notin E_p \Rightarrow (f(u), f(v)) \notin E_t$

Illustration on the Subgraph Isomorphism Problem (SIP)

Goal: Search for a copy of a pattern graph G_p in a target graph G_t



$G_p = (N_p, E_p)$



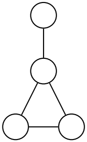
$G_t = (N_t, E_t)$

Find an injective mapping $f : N_p \rightarrow N_t$

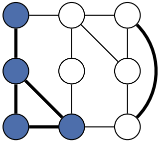
- Non-induced case:
  - $\forall (u, v) \in E_p : (f(u), f(v)) \in E_t$
- Induced case:
  - $\forall (u, v) \in E_p : (f(u), f(v)) \in E_t$
  - $\forall u, v \in N_p : (u, v) \notin E_p \Rightarrow (f(u), f(v)) \notin E_t$

Illustration on the Subgraph Isomorphism Problem (SIP)

Goal: Search for a copy of a pattern graph G_p in a target graph G_t



$G_p = (N_p, E_p)$



$G_t = (N_t, E_t)$

Find an injective mapping $f : N_p \rightarrow N_t$

- Non-induced case:
  - $\forall (u, v) \in E_p : (f(u), f(v)) \in E_t$
- Induced case:
  - $\forall (u, v) \in E_p : (f(u), f(v)) \in E_t$
  - $\forall u, v \in N_p : (u, v) \notin E_p \Rightarrow (f(u), f(v)) \notin E_t$

Experimental evaluation of VF3 in [Carletti et 2018]

Table VII shows an experimental comparison between VF2 and VF3 in terms of computational complexity.

Dataset	VF2		VF3	
	Labeled	Unlabeled	Labeled	Unlabeled
$\eta = 0.2$	$\theta(N^{5.7})$	$\theta(N^{6.3})$	$\theta(N^{3.5})$	$\theta(N^{3.6})$
$\eta = 0.3$	$\theta(N^{5.7})$	$\theta(N^{6.4})$	$\theta(N^{3.7})$	$\theta(N^{3.7})$
$\eta = 0.4$	$\theta(N^{5.7})$	$\theta(N^{6.6})$	$\theta(N^{4.5})$	$\theta(N^{4.7})$

TABLE VII: Comparison between VF2 and VF3 in terms of measured computational complexity.

Considered benchmark = Randomly generated instances:

- Randomly generate a target graph with $\eta \cdot n \cdot (n-1)$ edges
- Pattern graph = Connected subgraph of the target

$\rightsquigarrow$ What is the difficulty of these instances?

How to control hardness when generating instances?

McCreesh, Prosser, Solnon & Trimble (2018): *When Subgraph Isomorphism is Really Hard, and Why This Matters for Graph Databases*. Journal of Artificial Intelligence Research

Random generation of an SIP instance

Random generation of a graph $G(n, d)$ wrt Erdös-Rényi model:

- n = number of vertices
- d = probability of adding an edge between 2 vertices
  - d close to 0 $\leadsto$ Sparse graphs
  - d close to 1 $\leadsto$ Dense graphs

Random generation of an SIP instance:

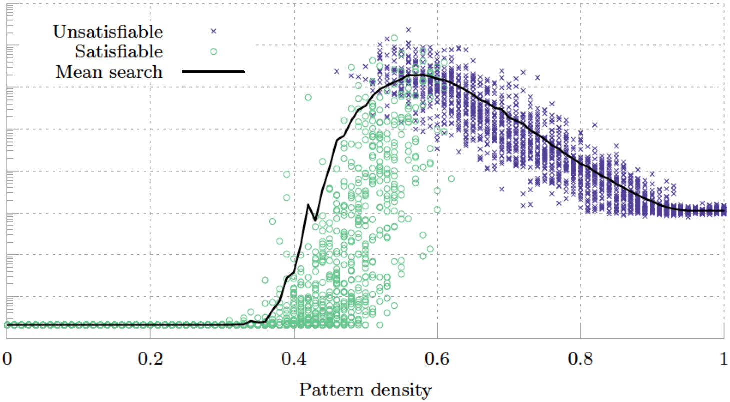
- Generation of a pattern graph $G(n_p, d_p)$ and a target graph $G(n_t, d_t)$
- Parameters = n_p, d_p, n_t, d_t

How can we control hardness for the non-induced case?

$\leadsto$ Probabilities d_p and d_t control graph densities

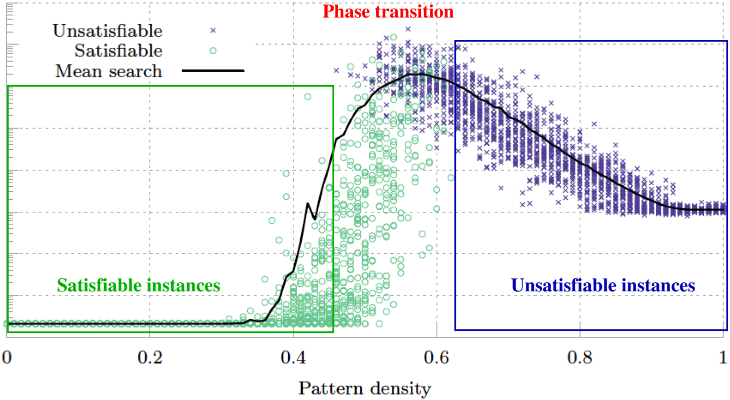
- Sparse pattern and dense target $\leadsto$ Easy to find a solution
- Dense pattern and sparse target $\leadsto$ Easy to prove inconsistency
- **Hard instances should be between these two extreme cases!?**

Phase transition from feasibility to infeasibility (non-induced case)



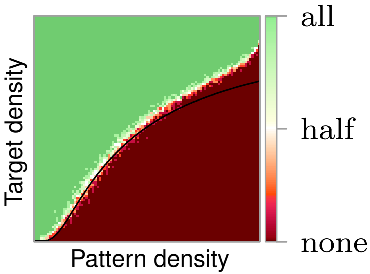
- We fix $n_p = 20$, $n_t = 150$, $d_t = 0.4$, and we vary d_p from 0 to 1
  $\rightsquigarrow$ Each point (x, y) is an instance generated with $d_p = x$
  - y = Search effort to solve the instance with Glasgow
  - Colour = Feasibility of the instance (green=yes; blue=no)

Phase transition from feasibility to infeasibility (non-induced case)



- $d_p \leq 0.44$: **Satisfiable** instances
  $\rightsquigarrow$ Most of them are trivial; a few of them are harder
- $d_p \geq 0.67$: **Unsatisfiable** instances
  $\rightsquigarrow$ Neither trivial, nor extremely hard
- $0.44 < d_p < 0.67$: **Phase transition** between sat and unsat
  $\rightsquigarrow$ Hardest instances

Phase transition when varying d_p and d_t (non-induced case)



- We fix $n_p = 30$, $n_t = 150$, and we vary d_p and d_t from 0 to 1
  $\leadsto$ Each point (x, y) = 10 instances generated with $d_p = x$ and $d_t = y$
- Colour = proportion of satisfiable instances
  - Top left: sparse patterns and dense targets $\leadsto$ All satisfiable
  - Bottom right: dense patterns and sparse targets $\leadsto$ All unsatisfiable
- Black line = Theoretical prediction of the phase transition location

Locating the phase transition (non-induced case)

How to compute the probability that an instance is feasible?

- Consider the random var. S corresponding to the number of solutions
- $p(S \geq 1) = 0.5 \Leftrightarrow \mathbb{E}(S) = 1$ if S has a normal distribution

Expected number of solutions for pattern $G(n_p, d_p)$ and target $G(n_t, d_t)$:

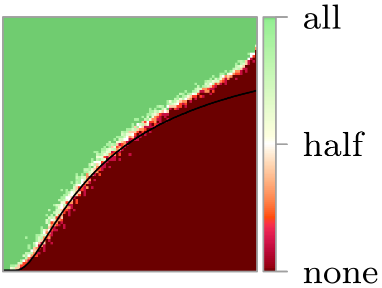
- Expected number of pattern edges = $d_p \cdot \frac{n_p(n_p-1)}{2}$
- Probability for one pattern edge to be mapped to a target edge = d_t
- Probability for one injective mapping to be a solution = $d_t^{d_p \cdot \frac{n_p(n_p-1)}{2}}$
- Number of possible injective mappings = $n_t \cdot (n_t - 1) \cdot \ldots \cdot (n_t - n_p + 1)$
- $\mathbb{E}(S) = n_t \cdot (n_t - 1) \cdot \ldots \cdot (n_t - n_p + 1) \cdot d_t^{d_p \cdot \frac{n_p(n_p-1)}{2}}$

Theoretical prediction of the phase transition location:

- $\mathbb{E}(S)$ much larger than 1 $\rightsquigarrow$ Easy to find a solution
- $\mathbb{E}(S)$ close to 0 $\rightsquigarrow$ Not very difficult to prove inconsistency
- $\mathbb{E}(S)$ close to 1 $\rightsquigarrow$ Really hard instance (black line)

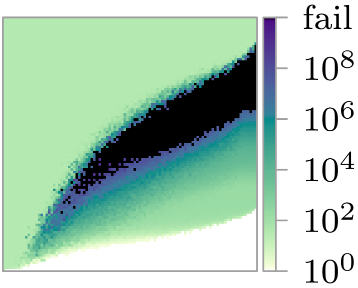
Feasibility vs Search Effort (non-induced case)

Feasibility:



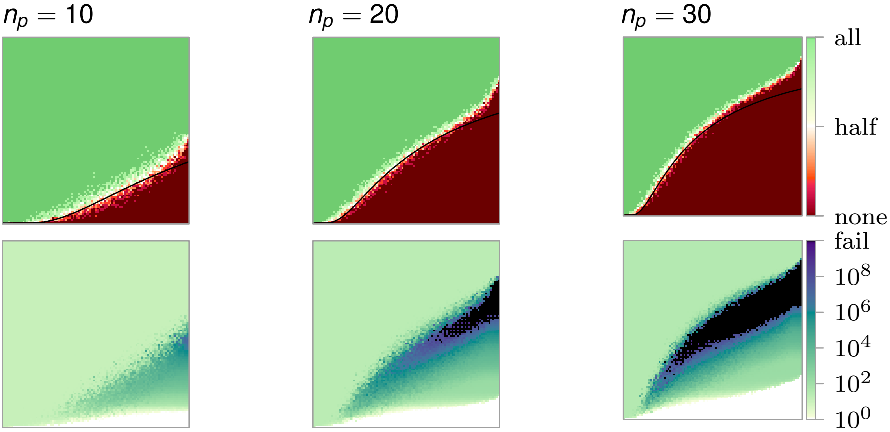
- Green = 100% feasible
- White = 50% feasible
- Brown = 0% feasible

Search Effort of Glasgow:



- Black = not solved in 1000s
- White = solved without backtracking

Scale-up properties when increasing n_p (non-induced case)



- The search effort slowly increases in easy regions
  $\leadsto$ Empirical polynomial time complexities on these instances
- The search effort strongly increases in the phase transition region
  $\leadsto$ Empirical exponential time complexities on these instances

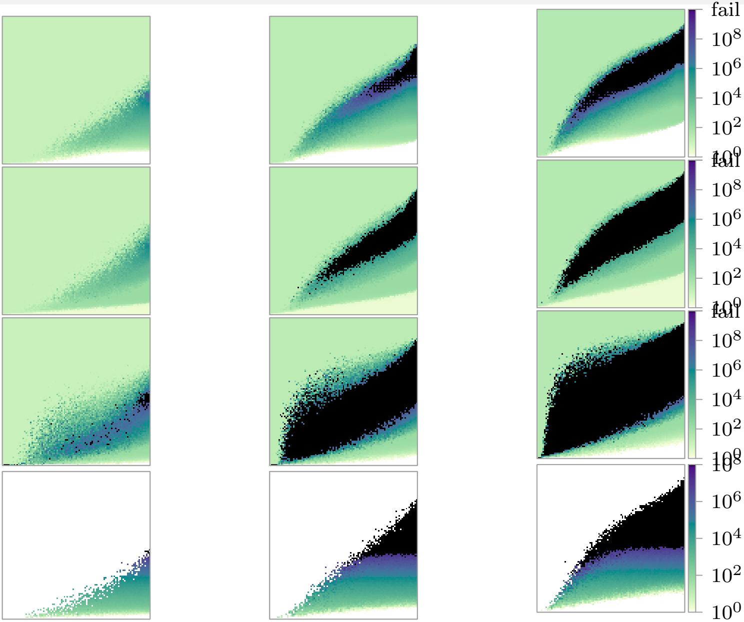
What about other solvers?

Glasgow:

LAD:

VF2:

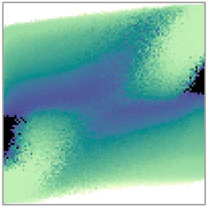
RI:



What about the induced case?

Feasibility ($n_p = 15$):

Glasgow search effort ($n_p = 15$):



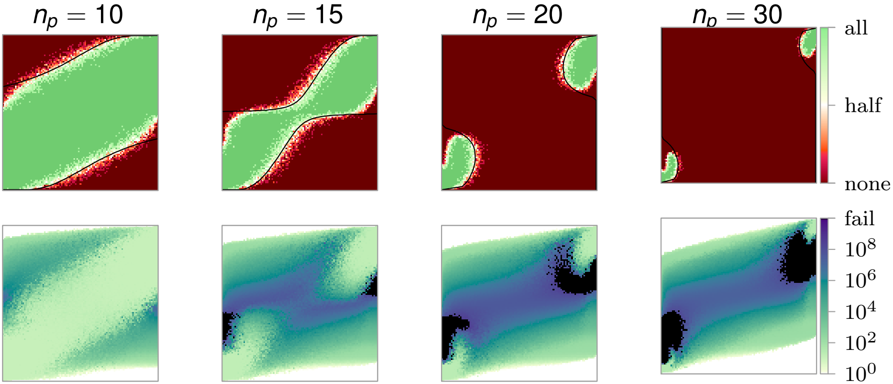
Theoretical prediction:

Expected number of solutions for pattern $G(n_p, d_p)$ and target $G(n_t, d_t)$:

$$\mathbb{E}(S) = n_t \cdot (n_t - 1) \cdot \ldots \cdot (n_t - n_p + 1) \cdot d_t^{d_p \cdot \frac{n_p(n_p-1)}{2}} \cdot (1 - d_t)^{(1-d_p) \cdot \frac{n_p(n_p-1)}{2}}$$

Phase transition when $\mathbb{E}(S)$ close to 1 (black line)

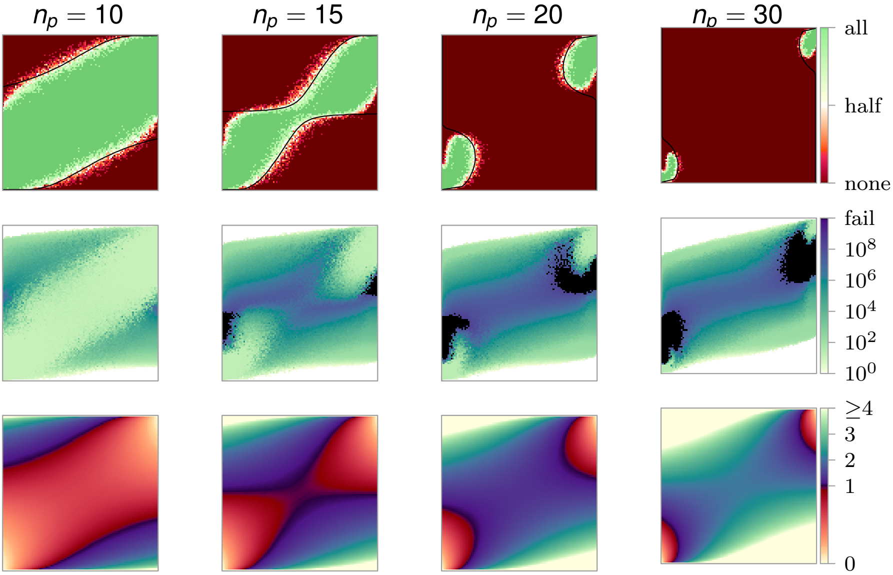
Scale-up properties when increasing n_p (induced case)



Some hard instances are far from the phase transition!

$\leadsto$ Prediction of hardness by means of constrainedness (see [JAIR 2018])

Scale-up properties when increasing n_p (induced case)



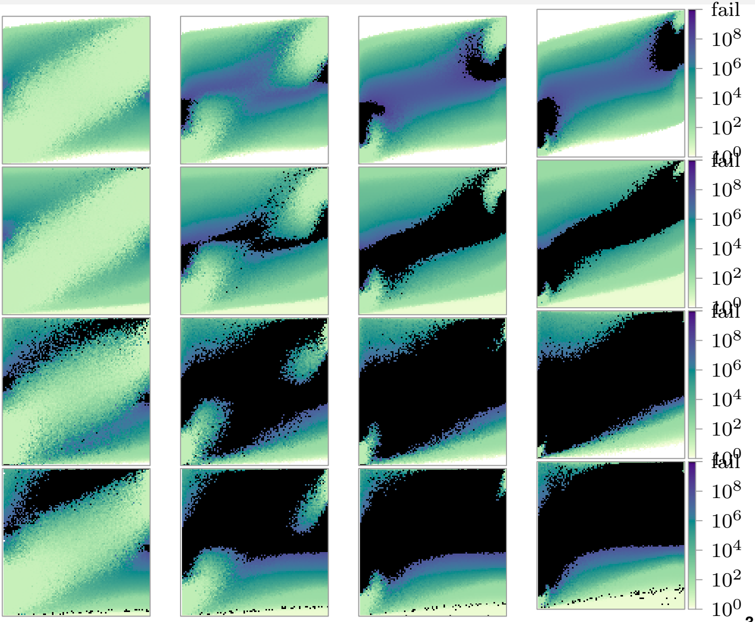
Induced case: Other solvers

Glasgow:

LAD:

VF2:

VF3:



What about Optimisation Problems? (1/3)

Definition of a Constrained Optimisation Problem (COP)

- Input:
  - A set X of variables
  - A set C of constraints
  - An objective function $f : X \to \mathbb{R}$
- Output: An assignment of X that satisfies C and maximises f

Classical approaches to solve $\mathcal{NP}$-hard COPs:

- Complete approaches: Iteratively solve decision problems
  $\leadsto$ Optimality proof but exponential time complexity
- Incomplete approaches: Heuristic exploration of the search space
  $\leadsto$ Polynomial time complexity but no guarantee on solution quality

What about Optimisation Problems? (2/3)

Complete approaches:

Most complete approaches solve sequences of decision problems

1. Search for an assignment a which satisfies C
   - Use heuristics to find "good" assignments
   - Use bounding functions to prune the search
   - ...
2. If there does not exist such an assignment, then stop
3. Add the constraint $f(X) > f(a)$ to C and go to (1)

Hardness of the successively solved instances:

The last two instances are the closest to the phase transition

- The penultimate one is the most constrained satisfiable instance
- The last one is the less constrained unsatisfiable instance

$\leadsto$ In general, at least one of these two instances is really hard

What about Optimisation Problems? (3/3)

Incomplete approaches (Tabu Search, Genetic Algo., ...)

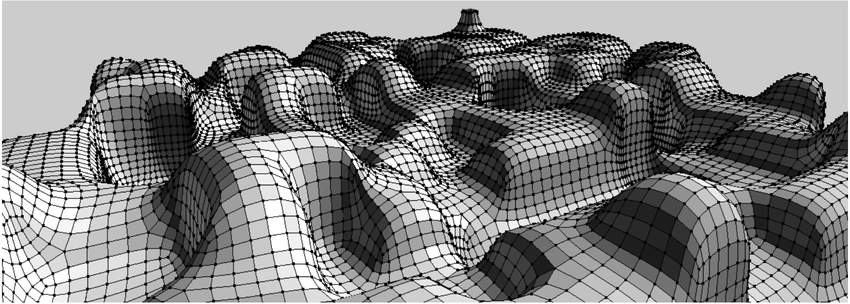
- Heuristic exploration of the search space: Use mechanisms to build new solutions from previously visited solutions
- Neighbourhood graph $G = (V, N)$ associated with an incomplete approach:
  - Vertices: V = set of all possible solutions
  - Edges: $N = \{(v_i, v_j) \in V \times V : v_j$ can be built from $v_i\}$
    $\leadsto$ Depends on mechanisms used to build solutions
  - Notation: neighbourhood of $v_i = N(v_i) = \{v_j \;/\; (v_i, v_j) \in N\}$

Hardness depends on the fitness landscape associated with G

Fitness Landscape (1/3)

Fitness landscape associated with a neighbourhood graph $G = (V, N)$:

- Each solution in V corresponds to a point
- The objective function f corresponds to the point height
- The neighbourhood N is used to position points wrt other dimensions



Fitness Landscape (2/3)

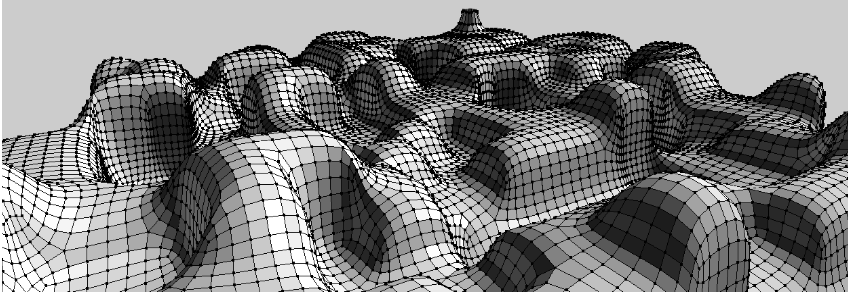
Topological features of a fitness landscape:

- Local optimum = Point with no neighbour strictly better

$$v_i \in V \text{ such that } \forall v_j \in N(v_i), f(v_j) < f(v_i)$$

- Plateau = Set of connected points in G which all have the same height
- Basin of attraction of a local optimum v_i = Set of all points from which v_i can be reached by hill-climbing

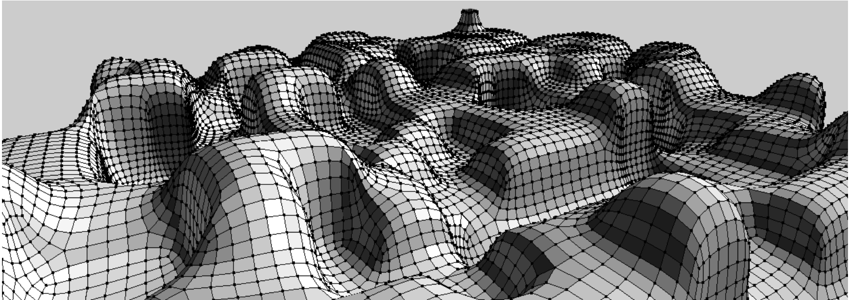
$\leadsto$ These features are used to study hardness



Fitness Landscape (3/3)

Influence of the landscape on performance:

- Landscape with a single local optimum (= global opt.) and no plateau
  $\rightsquigarrow$ A greedy (hill-climbing) algorithm is efficient and optimal
- Rugged landscape = many optima uniformly distributed
  $\rightsquigarrow$ No correlation between height and distance to the global optimum
  $\rightsquigarrow$ A pure random algorithm is probably the most efficient approach
- Massif Central landscape
  $\rightsquigarrow$ Find the right compromise between intensification and diversification



Plan

Choice of Factors, Levels and Design Points

Factors: Choose the most influential parameters

$\leadsto$ Exploit literature, knowledge on the algorithm, and exploratory experiments

Levels: Identify relevant values for each factor

- Symbolic factor: 1 level per value
- Numeric factor:
  - Identify intervals of relevant values
  - Sample with an exponential progression
    $\leadsto$ 1, 2, 4, 8, 16, ... or 1, 10, 100, 1000, ...

Design Points:

- Full factorial design = All possible Factor/Level combinations
  - Pros: Identify all factor effects, including interaction effects due to inter-dependency of factors
  - Cons: Exponential number of combinations wrt number of factors
- Fractional factorial design = Selection of a subset of combinations
  $\leadsto$ Choose representative and complementary combinations

Performance Criteria

Three most common criteria:

- Duration
- Memory
- Quality

Warning: These criteria are often inter-dependent...

Performance measures for duration (1/2)

Number of dominant operations:

- Identify dominant operations:
  - Number of comparisons for sorting algorithms
  - Number of constraint checks when solving constraint satisfaction pb
  - ...
- Count the number of times these operations are done

Number of Mems:

$\leadsto$ Number of memory accesses (load and store)

Pros:

Measures independent from the language, the OS, the processor, . . .

Cons:

Not always representative of duration...

Performance measures for duration (2/2)

Elapsed real time

- Difference of time between the beginning and the end of the run
- Not reliable because it depends on the CPU load

CPU time

- Total time of CPU utilisation
- Also depends on the CPU load!

Illustration [McGeoch 2012]

Experiment on an 8 core HP:	CPU time	Real time
1 process on 1 core:	= 27.9	= 28.2

Experiment on a 2 core MAC:	CPU time	Real time
1 process on 1 core:	= 67	= 79

Performance measures for duration (2/2)

Elapsed real time

- Difference of time between the beginning and the end of the run
- Not reliable because it depends on the CPU load

CPU time

- Total time of CPU utilisation
- Also depends on the CPU load!

Illustration [McGeoch 2012]

Experiment on an 8 core HP:	CPU time	Real time
1 process on 1 core:	$= 27.9$	$= 28.2$
9 concurrent processes on 8 cores:	$\in [36.0; 37.6]$	$\in [43.4; 43.6]$

Experiment on a 2 core MAC:	CPU time	Real time
1 process on 1 core:	$= 67$	$= 79$
9 concurrent processes on 2 cores:	$\in [97; 100]$	$\in [630; 649]$

Performance Measures for Optimisation Problems

Recall: Definition of a COP (X, C, f)

Find an assignment a^* of X that satisfies C and maximises f

Complete/exact algorithm:

- Find a^* and prove its optimality
- Performance measure: CPU time, or number of mems/operations

Question: What if some instances aren't solved within the time limit?

Anytime algorithm:

- Solution continuously improved $\rightsquigarrow$ may be stopped before the end
- Performance measures for a given time limit t:
  - Best objective function value $f(a')$
  - Approximation ratio $\frac{f(a')}{f(a^*)}$ or gap to optimality $\frac{f(a')-f(a^*)}{f(a^*)}$

Questions: How to choose t? How to compute $\frac{f(a')}{f(a^*)}$ if a^* isn't known?

Plan

Data Analysis

Goal of data analysis:

Transform raw data into information

Tools for data analysis:

- Descriptive statistics:
  $\rightsquigarrow$ Concise description of the main properties
- Graphical data analysis:
  $\rightsquigarrow$ Visualisation that highlights data properties
- Statistical tests:
  $\rightsquigarrow$ Procedure used to reject or not a statistical hypothesis

$\rightsquigarrow$ See the 4IF course on statistics

Descriptive statistics (recalls)

Central tendency measures:

- Mean: $\overline{X} = \frac{\sum x_i}{n}$
  $\leadsto$ May be sensible to outliers
- Median: Middle value in the ordered sequence of values
  $\leadsto$ Not sensible to outliers
- Mode: Most frequent value
  $\leadsto$ Used to describe non numerical data

In a normal distribution, these 3 measures have very close values

Dispersion measures:

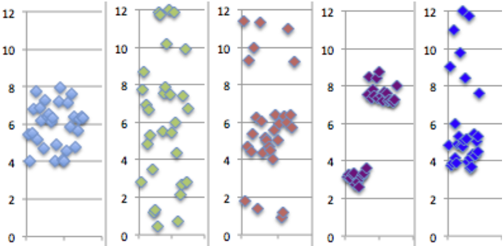
- Standard deviation: $\sigma = \sqrt{\frac{\sum (x_i - \overline{X})^2}{n}}$
- InterQuartile Range: $IQR = Q3 - Q1$ with
  - $Q1$ = largest value of the 25% lowest values
  - $Q3$ = smallest value of the 25% largest values

Look at Raw Data before starting Data Analysis!

	Mean	Median	StDv	IQR
Ciel	5.94	6.24	1.16	1.90
Vert	6.12	6.34	3.49	4.40
Rose	5.84	5.55	2.83	1.81
Violet	5.83	7.19	2.26	4.21
Bleu	5.88	4.91	2.52	2.21

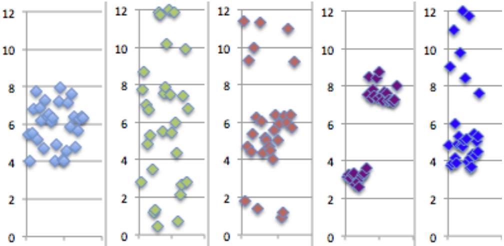
Look at Raw Data before starting Data Analysis!

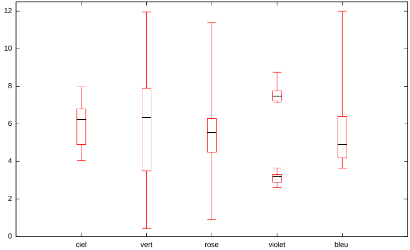
	Mean	Median	StDv	IQR
Ciel	5.94	6.24	1.16	1.90
Vert	6.12	6.34	3.49	4.40
Rose	5.84	5.55	2.83	1.81
Violet	5.83	7.19	2.26	4.21
Bleu	5.88	4.91	2.52	2.21



Look at Raw Data before starting Data Analysis!

	Mean	Median	StDv	IQR
Ciel	5.94	6.24	1.16	1.90
Vert	6.12	6.34	3.49	4.40
Rose	5.84	5.55	2.83	1.81
Violet	5.83	7.19	2.26	4.21
Bleu	5.88	4.91	2.52	2.21





. . . and use box plots!

[Datasaurus dataset]

Generate Data Easy to Analyse!

- Collect all relevant data (parameters, solutions, performance measures, etc) and format the output to ease the parsing
- Enlarge level intervals to amplify factor impacts
  $\leadsto$ Choose the right scale
- Use Variance Reduction Techniques (VRT):
  - Same reference benchmark for all experiments
  - Normalise Data before analysing them
    $\leadsto$ Percentage gap to optimal solution, for example
  - Cluster heterogeneous benchmarks into homogeneous classes
    $\leadsto$ Analyse each class separately

> *“If your experiment needs statistics then you ought to have done a better experiment.”*
>
> E. Rutherford

What are we going to see now?

Data analysis for four different kinds of experimental results:

- Non deterministic algorithms
  $\leadsto$ Illustration on the car sequencing problem
- Anytime (and non deterministic) algorithms
  $\leadsto$ Illustration on the maximum clique problem
- Large and heterogeneous benchmarks
  $\leadsto$ Illustration on the subgraph isomorphism problem
- Classifiers

All data and scripts are available on Moodle

And what shall we not see (among other things...)?

- Experimental evaluation of algorithms for multi-criteria optimisation
- Experimental evaluation of parallel algorithms
  $\leadsto$ See IF-5-PRJ33

Plan

Data Analysis for Non Deterministic Algorithms

What is a non deterministic algorithm?

Algorithm that uses a (pseudo-)random function $\Rightarrow$ independent runs on the same input data (except the random seed) do not necessarily return the same result

How to measure performance of non deterministic algorithms?

- Consider each measure as a random variable
  - Time $\leadsto$ Probability that time is smaller than a given bound
  - Quality $\leadsto$ Probability that quality is greater than a given bound
- Empirical estimation of the probability distribution of the measure $\leadsto$ Perform a large number of runs (with different random seeds)

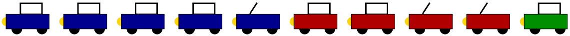
How to compare probability distributions?

$\leadsto$ Illustration on the *Car Sequencing Problem*

The Car Sequencing Problem

Description of the problem(1):

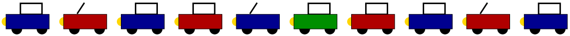
- Input: A set of cars to be produced



with spacing constraints between options

 $\leq 1/2$;  $\leq 2/5$;  $\leq 1/5$;  $\leq 1/3$

- Output: Permutation of cars that satisfy constraints



Question addressed by the experiment:

What is the best parameter setting of a non deterministic algorithm(2) for solving instance 26-82, among 5 given parameter settings?

(1) C. Solnon, VD. Cung, A. Nguyen, C. Artigues: *The car sequencing problem: Overview of state-of-the-art methods and industrial case-study of the ROADEF'2005 challenge*, European Journal of Operational Research (EJOR), 2008

(2) C. Solnon: *Combining two pheromone structures for solving the car sequencing problem with Ant Colony Optimization*, European Journal of Operational Research (EJOR), 2008

Experimental Data to Analyse

For each parameter setting $i \in \{1, 2, 3, 4, 5\}$, the file `resParam`*i*`.txt` contains traces of 100 runs with this setting

Example of trace (for 2 runs):

```
Parameters: alpha=4 beta=6 rho1=0.050000 rho2=0.030000 tau1Min=0.010000 tau1Max=4.000000
nbCycles=5000 nbAnts=30 verbose=1(5000) input=pb26-82 seed=1 strategy=1
iteration 1 time 0.000187s: new best = 6
iteration 6 time 0.000347s: new best = 5
iteration 114 time 0.003628s: new best = 2
iteration 827 time 0.026609s: new best = 1
iteration 91988 time 2.880917s: new best = 0
End of run
Parameters: alpha=4 beta=6 rho1=0.050000 rho2=0.030000 tau1Min=0.010000 tau1Max=4.000000
nbCycles=5000 nbAnts=30 verbose=1(5000) input=pb26-82 seed=2 strategy=1
iteration 1 time 0.000187s: new best = 5
iteration 3 time 0.000254s: new best = 4
iteration 9 time 0.000445s: new best = 2
iteration 255 time 0.007937s: new best = 1
End of run
```

$\rightsquigarrow$ The first run has found a solution at iteration 91988

$\rightsquigarrow$ The second run has not found a solution (best sol. violates 1 constraint)

Performance criterion and measure

Performance criterion:

Duration needed to solve the instance

Performance measure:

- Two possible measures: CPU time and number of iterations
  An iteration spends (nearly) always the same CPU time
  $\rightsquigarrow$ Measure the number of iterations
- What should we do for runs that did not solve the instance?
  $\rightsquigarrow$ Maximum number of iterations (=150000)
  $\rightsquigarrow$ **Warning: This is a lower bound of the actual measure**

Shell script for extracting performance measures from run traces:

```
for p in {1..5}; do
    cat resParam$p.txt | grep -B 1 End | grep best | \
        awk '($8==0){print $2}($8>0){print 150000}' > tmp$p.txt
done
```

Let's start with some descriptive statistics

Central tendency measures:

- Mean: $\overline{X} = \frac{\sum x_i}{n}$
- Median: Middle value in the ordered sequence of values

$\leadsto$ In a normal distribution, these 2 measures have very close values

	Mean	Median	
1	8657	8705	
2	5082	4743	
3	3055	3111	
4	56205	1378	
5	8746	1728	

- Mean ranking: 3, 2, 1, 5, 4
- Median ranking: 4, 5, 3, 2, 1
- σ not computed in case of TO
- IQR not computed in case of TO before Q3

Let's start with some descriptive statistics

Central tendency measures:

- Mean: $\overline{X} = \frac{\sum x_i}{n}$ $\leadsto$ **Lower bound in case of time out (TO)**
- Median: Middle value in the ordered sequence of values

$\leadsto$ In a normal distribution, these 2 measures have very close values

	Mean	Median	
1	8657	8705	
2	5082	4743	
3	3055	3111	
4	$\geq$56205	1378	
5	$\geq$8746	1728	

- Mean ranking: 3, 2, 1, 5, 4 or 3, 2, 1, 4, 5
- Median ranking: 4, 5, 3, 2, 1
- σ not computed in case of TO
- IQR not computed in case of TO before Q3

Let's start with some descriptive statistics

Central tendency measures:

- Mean: $\overline{X} = \frac{\sum x_i}{n} \leadsto$ **Lower bound in case of time out (TO)**
- Median: Middle value in the ordered sequence of values

$\leadsto$ In a normal distribution, these 2 measures have very close values

Dispersion measures:

- Standard deviation: $\sigma = \sqrt{\frac{\sum (x_i - \overline{X})^2}{n}}$
- Inter Quartile Range: $IQR = Q3 - Q1$ where $Q1$ (resp. $Q3$) is the largest value of the 25% lowest (resp. 75%) lowest values

	Mean	Median	σ	IQR
1	8657	8705	3323	4481
2	5082	4743	1813	2725
3	3055	3111	1053	1259
4	$\geq$56205	1378	-	-
5	$\geq$8746	1728	-	863

- Mean ranking: 3, 2, 1, 5, 4 or 3, 2, 1, 4, 5
- Median ranking: 4, 5, 3, 2, 1
- σ not computed in case of TO
- IQR not computed in case of TO before Q3

Scripts for Drawing Box Plots

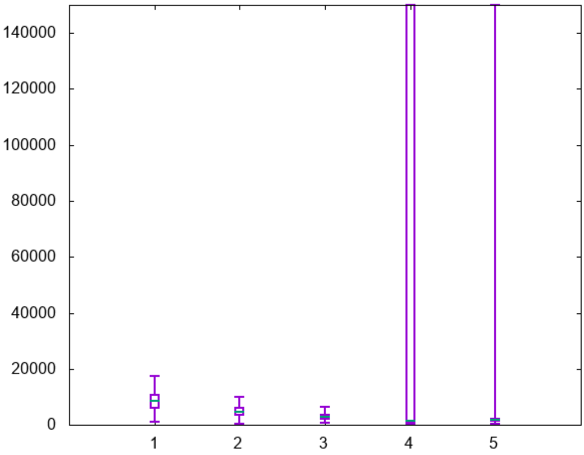
Shell script for computing Mean, Min, Q1, Median, Q3 and Max:

```
for p in {1..5}; do
    echo -n "$p `awk 'BEGIN{tot=0}
                      {tot+=$1}
                      END{print tot/100}' tmp$p.txt` " >> stats.txt
    for l in 1 25 50 75 100; do
        echo -n "`cat tmp$p.txt | sort -g | head -$l | tail -1` " >> stats.txt
    done
    echo >> stats.txt
done
```

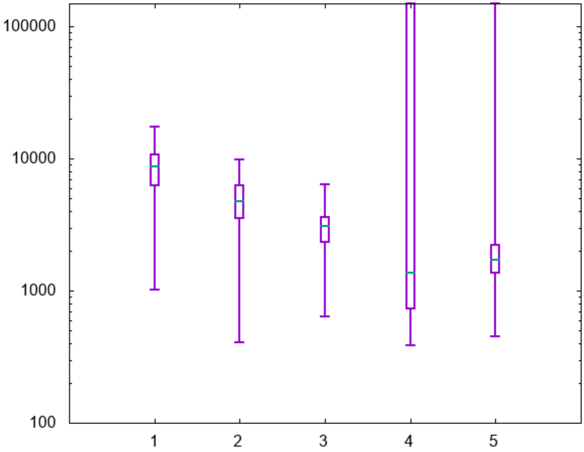
Gnuplot script to build box plots::

```
set terminal png; set output 'boxplot-carseq.png'
set xrange [0:6]; set bars 4.0
plot 'stats.txt' using 1:4:3:7:6:xticlabels(1) with candlesticks notitle whiskerbars lw 2,
     'stats.txt' using 1:5:5:5:5 with candlesticks lw 2 notitle
```

Box Plots



Box Plots



With a log scale!
(`set logscale y`)

Utilisation of a Statistical Test

Distributions are not normal

$\leadsto$ Use a non parametric test, *e.g.*, Mann–Whitney U test

Null hypothesis H_0 for a couple of parameters (p_i, p_j):

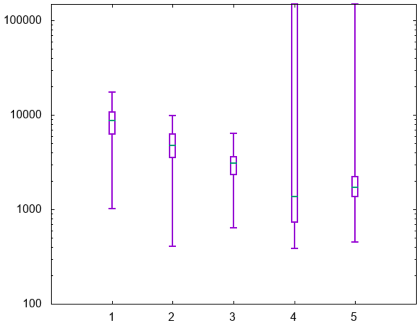
Proba(time with p_i > time with p_j) = Proba(time with p_i < time with p_j)

Python script to use the Mann–Whitney U test:

```
from scipy.stats import mannwhitneyu
val=[]
for p in range(1, 6):
    f = open('tmp'+str(p)+'.txt', 'r+')
    val.append([int(i) for i in f.readlines()])
for p1 in range(1, 5):
    for p2 in range(p1+1, 6):
        u, pValue = mannwhitneyu(val[p1-1],val[p2-1])
        print p1, p2, u, pValue
```

Results of the Statistical Test

p1	p2	U	p-value
1	2	1690.5	3.104e-16
1	3	542.5	6.418e-28
1	4	4307.0	0.045
1	5	974.0	3.944e-23
2	3	1675.0	2.273e-16
2	4	4412.0	0.075
2	5	1164.0	3.571e-21
3	4	4509.0	0.115
3	5	2218.0	5.366e-12
4	5	4826.0	0.335



Conclusions:

- Reject the null hypothesis when p-value $\leq \alpha$
- How to choose the threshold value α?
  - Use the Bonferroni correction in case of multiple hypothesis
    $\rightsquigarrow$ Divide α by the number of hypothesis

Comparison of Cumulative Distribution Functions (CDFs)

What is the CDF of a random variable X?

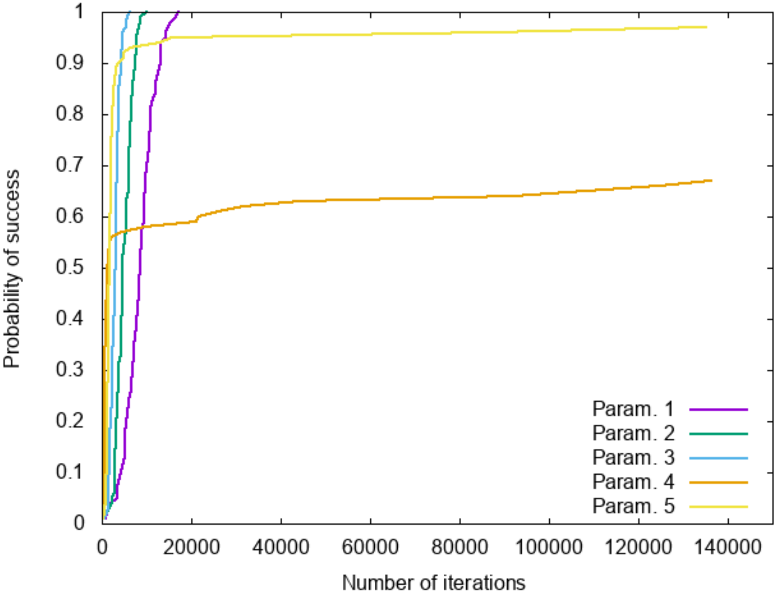
- $f_X(x)$ = Probability that X is smaller than or equal to x
- If X is the random variable associated with the number of iterations:
  - x is a number of iterations
  - $f_X(x)$ is the probability that the number of iterations needed to solve the instance is smaller than or equal to x
- Empirical estimation by considering a large number of runs

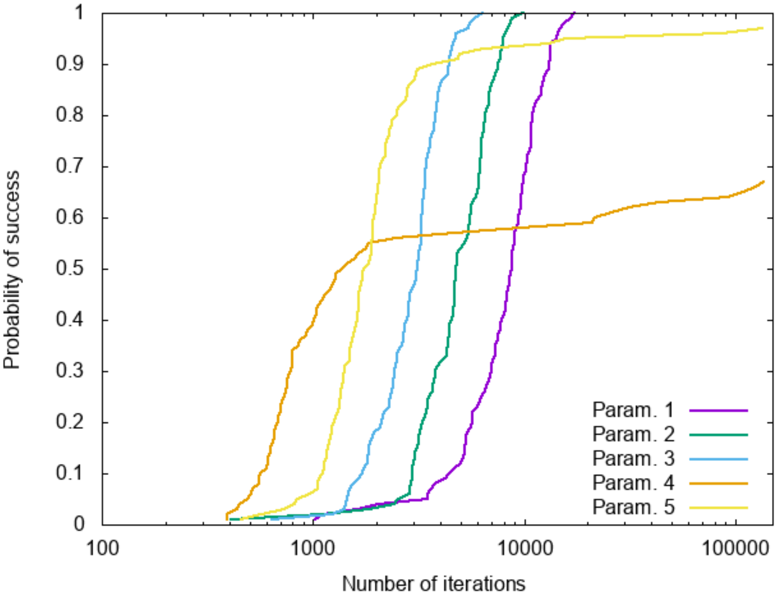
Shell script to compute CDFs:

```
for p in {1..5}; do
    cat tmp$p.txt | grep -v 150000 | sort -g | \
        awk 'BEGIN{i=0}{i++; printf("%d %f\n",$1,i/100)}' > Param-$p
done
```

Gnuplot script to visualise CDFs:

```
set terminal png; set output 'CDF.png'
set xlabel 'Number of iterations'; set ylabel 'Probability of success'
set xrange [100:150000]; set key bottom
plot for [p=1:5] "Param-".p title "Param. ".p with lines linewidth 2
```

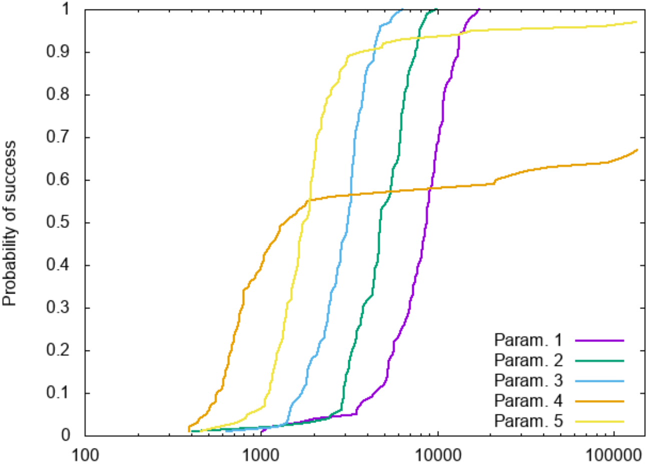

Probability of success
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
0
20000
40000
60000
80000
100000
120000
140000
Number of iterations
Param. 1
Param. 2
Param. 3
Param. 4
Param. 5



With a log scale! (`set logscale x`)

Conclusions

- Param. 1 and Param. 2 are dominated by Param. 3
- Choice between Param. 3, 4 and 5 depends on the nb of iterations we are willing to do $\leadsto$ Compromise between time and solution quality



Plan

Data Analysis for Anytime Algorithms

What is an anytime algorithm?

- Algorithm that produces a sequence of solutions of increasing quality
- The longer the time limit, the better the last solution in the sequence

$\rightsquigarrow$ Many algorithms for optimisation problems are anytime algorithms

Question addressed by the experiment:

Given 4 parameter settings of an anytime and non deterministic algorithm for the maximum clique problem[(1)], what is the best parameter setting for 16 graphs coming from 3 different benchmarks (C, gen, and brock)?

Reference:

(1) C. Solnon & S. Fenet: *A study of ACO capabilities for solving the Maximum Clique Problem*, Journal of Heuristics, 12(3):155-180, Springer, 2006

Input Data

- For each parameter setting $p \in \{1, 2, 3, 4\}$ and each instance i, `res-`p`-`i`.txt` contains the trace of 100 runs with setting p on instance i
- Heuristic algorithm $\Rightarrow$ No optimality proof

Example: file `res-3-C500.9.txt`

```
Params: alpha=1 best=57 rho=0.020000 tauMin=0.010000 tauMax=6.000000 nbCycles=3000 nbAnts=30 verbose=1200
input=instances/C500.9 mustRepair=1 seed=1 mode=v
Best clique size = 49 at cycle 0 0.070056 seconds
Best clique size = 50 at cycle 16 0.085924 seconds
Best clique size = 51 at cycle 141 0.210307 seconds
Best clique size = 52 at cycle 162 0.231288 seconds
Best clique size = 53 at cycle 219 0.287652 seconds
Best clique size = 54 at cycle 224 0.292766 seconds
solution: 1 28 33 40 49 50 52 61 68 84 94 98 102 108 110 113 132 141 146 149 153 157 168 182 194 226 256 266 294
295 300 310 319 323 329 332 336 340 374 385 404 405 407 413 415 433 435 461 468 480 481 484 490 493
Params: alpha=1 best=57 rho=0.020000 tauMin=0.010000 tauMax=6.000000 nbCycles=3000 nbAnts=30 verbose=1200
input=instances/C500.9 mustRepair=1 seed=2 mode=v
Best clique size = 47 at cycle 0 0.069913 seconds
Best clique size = 49 at cycle 1 0.070969 seconds
Best clique size = 50 at cycle 42 0.112515 seconds
Best clique size = 51 at cycle 98 0.169504 seconds
Best clique size = 53 at cycle 145 0.215919 seconds
Best clique size = 54 at cycle 230 0.299549 seconds
Best clique size = 55 at cycle 264 0.332908 seconds
Best clique size = 56 at cycle 342 0.409441 seconds
solution: 6 21 22 44 46 53 68 87 92 97 110 112 114 121 134 137 140 145 147 148 149 160 181 182 207 212 223 238
244 245 268 284 290 294 319 321 326 364 381 393 398 404 405 411 415 427 442 447 451 454 467 484 488 490 491 493
```

Performance Measures

- Duration measure = Number of iterations of the algorithm
- Quality measure = Size of the largest clique found

Shell script for extracting performance measures:

```
for p in 1 2 3 4; do
    for i in `cat instances.txt`; do
        cat resultats-bruts/res-$p-$i.txt | awk '
            ($1=="Best"){printf("%d %d\n",$5,$8)}
            ($1=="solution:"){printf("end\n")}' > perf/res-$p-$i.txt
    done
done
```

Example:

```
49 0
50 16
51 141
52 162
53 219
54 224
end
47 0
49 1
50 42
51 98
53 145
54 230
55 264
56 342
end
```

Can we study each criterion separately?

Let's fix the quality and plot the CDF associated with duration

$\leadsto$ Probability of finding a clique of size *k* wrt number of iterations

Shell script for computing CDFs:

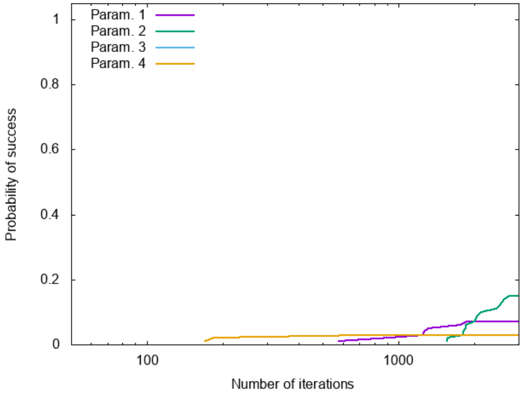
```
for p in 1 2 3 4; do
    for i in `cat instances.txt`; do
        cat perf/res-$p-$i.txt | awk -v k=$k '
            BEGIN{found=0}
            ($1!="end" && found==0 && $1>=k){print $2; found=1}
            ($1=="end"){found=0}' \
        | sort -g | awk '
            BEGIN{i=0; printf("0 0\n")}
            {i++; printf("%d %f\n",$1,i/100)}
            END{printf("3000 %f\n",i/100)}' > Param-$p
    done
done
```

Can we study each criterion separately?

Let's fix the quality and plot the CDF associated with duration

$\leadsto$ Probability of finding a clique of size k wrt number of iterations

Visualisation of CDFs for instance C500.9:



Result for k=57
(Size of the maximum clique of C500.9)

What can we do if the algorithm does not find a clique of size 57?

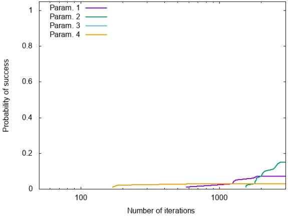
Can we study each criterion separately?

Let's fix the quality and plot the CDF associated with duration

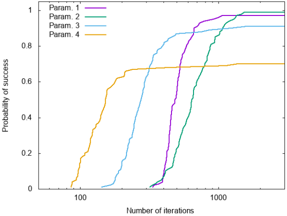
$\leadsto$ Probability of finding a clique of size k wrt number of iterations

Results for different values of k:

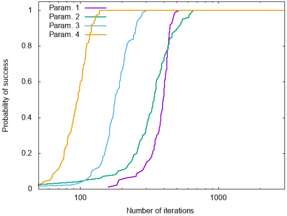
k=57:



k=55:



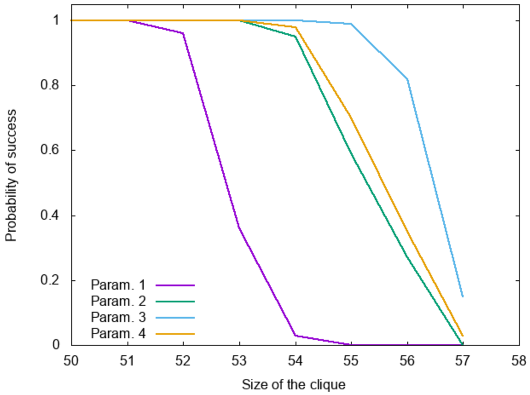
k=53:



How to choose k?

What if we fix the number of iterations and plot the CDF associated with quality?

Probability of finding a clique of size *x* in less than *t* iterations:

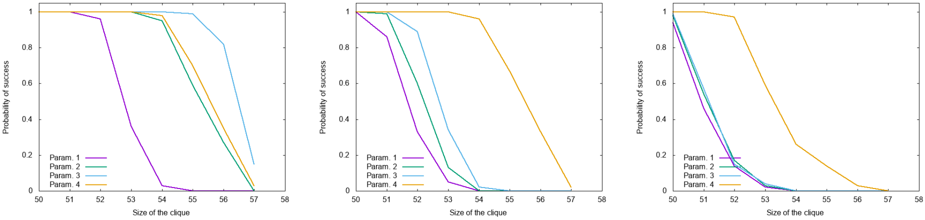


Result for instance C500.9 when t=3000

What if we fix the number of iterations and plot the CDF associated with quality?

CDFs for different values of t:

t=3000: t=1000: t=100:



How to choose the number of iterations?

Evolution of Quality with respect to Duration

- Plot $f(x)$ = size of the best clique found within x iterations
- Non deterministic algorithm $\leadsto$ Empirical estimation of the expected size by considering a large number of runs

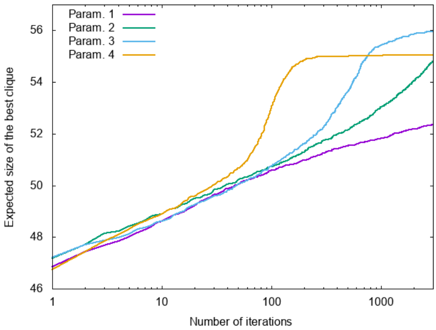
Shell script:

```
for p in 1 2 3 4; do
    for i in `cat instances.txt`; do
        for t in `cat perf/res-$p-$i.txt | grep -v end | sort -g -k 2 -u | awk '{print $2}'`; do
            cat perf/res-$p-$i.txt | awk -v t=$t '
                BEGIN{y=0; total=0.0}
                ($1=="end"){total+=y; y=0}
                ($1!="end" && $2<=t && y<$1){y=$1}
                END{printf("%d %f\n",t,total/100);}
            ' >> evolBest-$p-$i
        done
    done
done
```

Evolution of Quality with respect to Duration

- Plot $f(x)$ = size of the best clique found within x iterations
- Non deterministic algorithm $\rightsquigarrow$ Empirical estimation of the expected size by considering a large number of runs

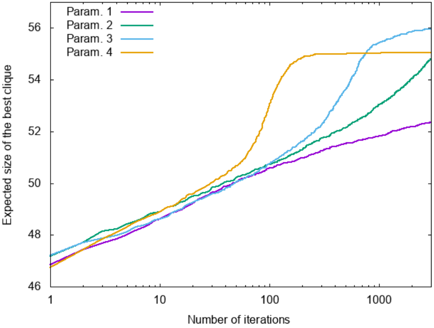
Visualisation for C500.9:



Evolution of Quality with respect to Duration

- Plot $f(x)$ = size of the best clique found within x iterations
- Non deterministic algorithm $\rightsquigarrow$ Empirical estimation of the expected size by considering a large number of runs

Visualisation for C500.9:



How to aggregate plots of different instances (that have maximum cliques of different sizes)?

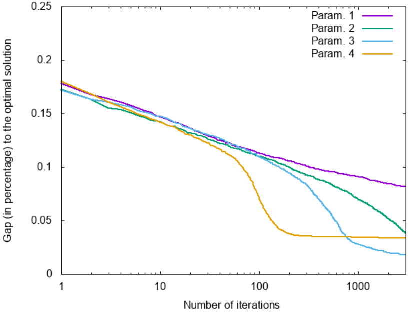
Let's normalise the measure!

Gap to the optimal solution (%):

- gap = $\frac{f(s^*)-f(s)}{f(s^*)}$

$\leadsto$ gap = 0 when $f(s) = f(s^*)$

$\leadsto$ gap $>$ 0 when $f(s) < f(s^*)$

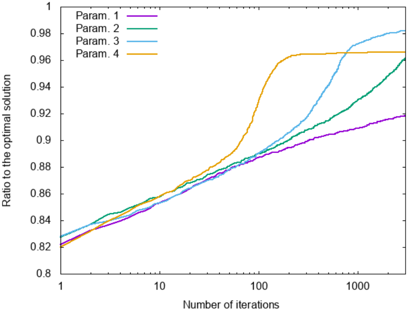


Ratio to the optimal solution:

- ratio = $\frac{f(s)}{f(s^*)}$

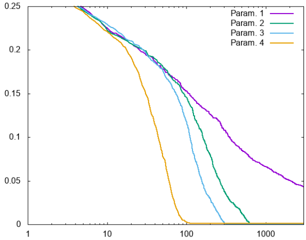
$\leadsto$ ratio = 1 when $f(s) = f(s^*)$

$\leadsto$ ratio $<$ 1 when $f(s) < f(s^*)$

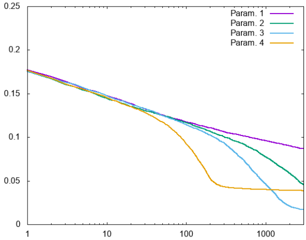


Average Gap for each Class of Graphs

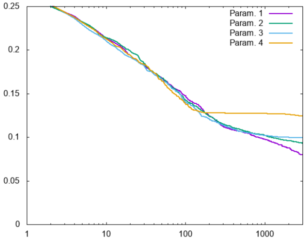
Gen graphs:



C graphs:



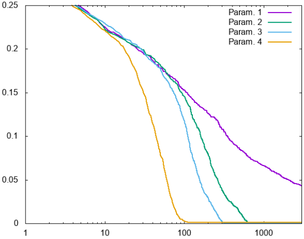
Brock graphs:



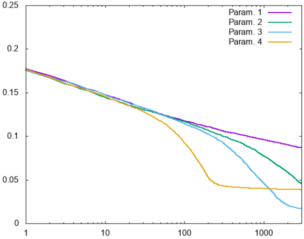
- Can we explain why results are different from a class to another?
  $\leadsto$ Correlation between clique size and distance to the max clique

Average Gap for each Class of Graphs

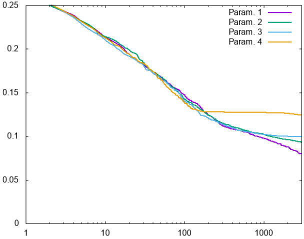
Gen graphs:



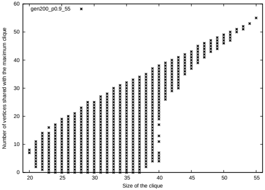
C graphs:

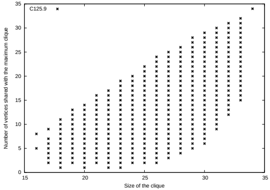


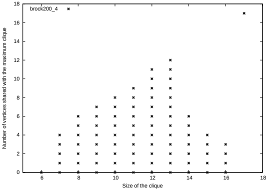
Brock graphs:



- Can we explain why results are different from a class to another?
  $\leadsto$ Correlation between clique size and distance to the max clique







Conclusion

Duration and quality are inter-dependent criteria for anytime algorithms

- Plot the evolution of quality wrt duration
- Normalise measures to compare results of different instances
  $\leadsto$ Gap or ratio to the optimal solution

Performance changes from an instance to another

- Analyse performance for each instance separately
  $\leadsto$ Aggregate results by grouping similar instances
- Use automatic selection and configuration technics
  $\leadsto$ See part 3 of this course

Plan

Illustration: Large scale evaluation

Subgraph Isomorphism Problem (non-induced case):

- Search for a copy of $G_p = (N_p, E_p)$ in $G_t = (N_t, E_t)$
  $\rightsquigarrow$ Find an injection $f : N_p \rightarrow N_t$ s.t. $\forall (u, v) \in E_p : (f(u), f(v)) \in E_t$
- See reference (1) for the induced case

Question addressed by the experiment:

What is the best among LAD[(2)], Glasgow[(3)], RI[(4)], PathLAD+[(5)], and LAD'23?

References:

(1) Solnon: *Experimental Evaluation of Subgraph Isomorphism Solvers*. In Graph-based Representations in Pattern Recognition. 2019

(2) Solnon: *Alldifferent-based filtering for subgraph isomorphism*, in AI 2010

(3) McCreesh, Prosser: *A parallel, backjumping subgraph isomorphism algorithm using supplemental graphs*, in CP 2015

(4) Bonnici, Giugno: *On the variable ordering in subgraph isomorphism algorithms*, in IEEE/ACM Trans. Comput. Biology Bioinform. 2017

(5) Wang, Jin, Cai, Lin: *PathLAD+: An Improved Exact Algorithm for Subgraph Isomorphism Problem*, in IJCAI 2023

Performance criteria

Memory usage:

- All algorithms have polynomial space complexities

$\leadsto$ Memory is not an issue for graphs with thousands of nodes

Solving time:

- In theory: All algorithms have exponential time complexities
- In practice: Some very large instances can be very quickly solved...
  ...But some small instances cannot be solved within days

$\leadsto$ **Time is the big issue**

Performance measure:

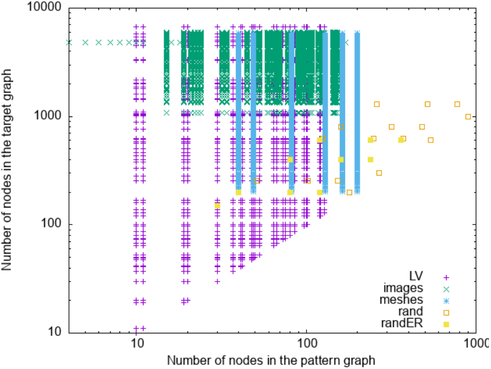
- CPU time on Apple M1 Pro
- Each run is limited to 1000 seconds
  - Some instances are not solved within this limit
  - Some instances are still not solved when the limit is 100,000s

Benchmark Description

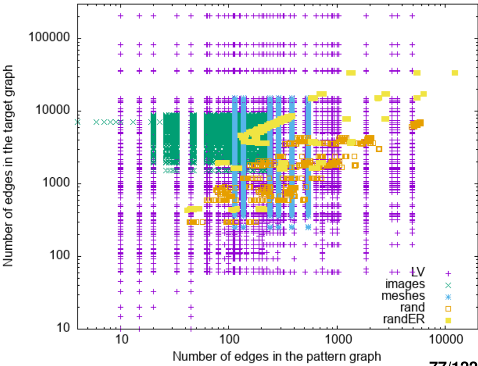
15,128 instances coming from 8 existing benchmarks

- Instances coming from real applications: Images and Meshes
- Random instances: randER (Erdös-Rényi) and rand (other models)
- Instances generated from the Stanford graph base: LV

Number of nodes:



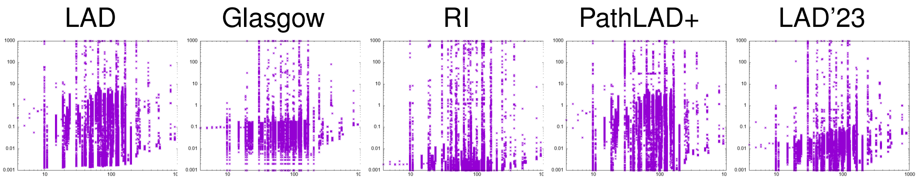
Number of edges:



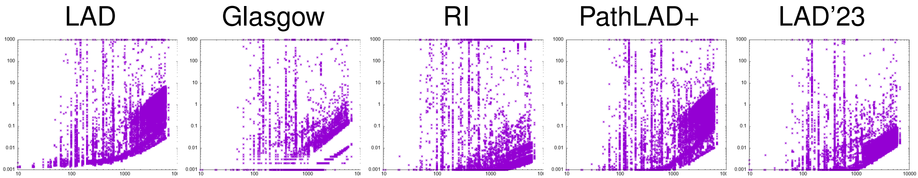
Let's start by looking at raw data...

For each instance, plot a point (x, y) where x = instance parameter and y = time

Plot when x = number of pattern vertices



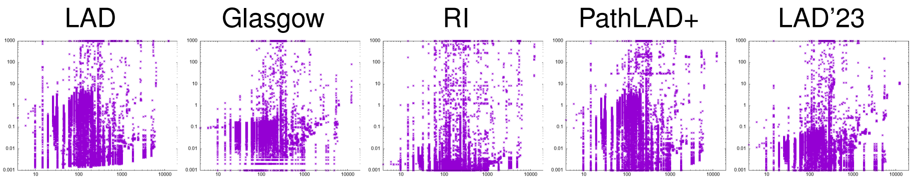
Plot when x = number of target vertices



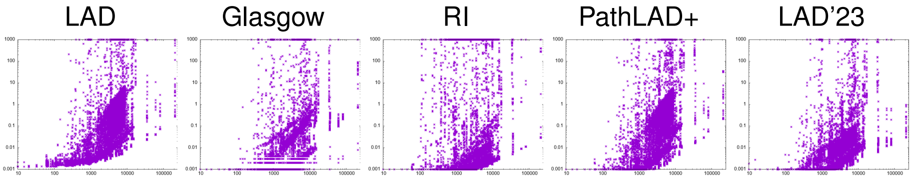
Let's start by looking at raw data...

For each instance, plot a point (x, y) where x = instance parameter and y = time

Plot when x = number of pattern edges



Plot when x = number of target edges

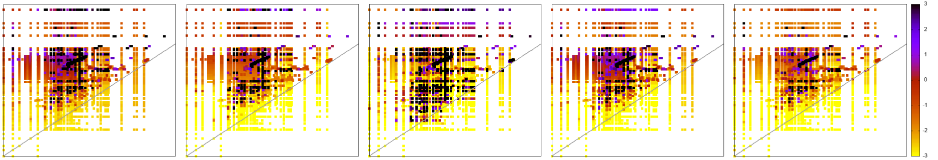


How to study interaction effects between two parameters?

$\rightsquigarrow$ **Plot a point (x, y) for each instance: x, y = instance parameters; colour = time**

Example: x = nb of pattern edges; y = nb of target edges

LAD | Glasgow | RI | PathLAD+ | LAD'23



Log scale for time: yellow if $t < 10^{-3}$s; red if $t = 1$s; black if $t > 10^3$s

First conclusions:

- No correlation between the number of edges and the CPU time
- Standard deviations are very high
- Some instances are not solved within the time limit (black points)

It's meaningless to plot the average solving time wrt the nb of nodes or edges!

What statistic can we compute to find the best solver?

	LAD	Glasgow	RI	PathLAD+	LAD'23
# fastest	4	4 224	11 169	2 458	3 433

1. Number of instances for which a solver is the fastest
   $\rightsquigarrow$ Ranking = RI, Glasgow, LAD'23, PathLAD+, LAD
2. Number of instances whose solving time is lower than $1,000$s
   $\rightsquigarrow$ Ranking = LAD'23, PathLAD+, Glasgow, LAD, RI
3. Number of instances whose solving time is lower than 1s
   $\rightsquigarrow$ Ranking = LAD'23, Glasgow, RI, PathLAD+, LAD
4. Number of instances whose solving time is lower than or equal to 0.01s
   $\rightsquigarrow$ Ranking = RI, LAD'23, Glasgow, PathLAD+, LAD
5. Time to solve an instance (average on instances solved within 1000s)
   $\rightsquigarrow$ Ranking = Glasgow, LAD'23, LAD, PathLAD+, RI
6. Lower bound of the average solving time on the $15,128$ instances
   (the time of unsolved instances is lower bounded by the time limit)
   $\rightsquigarrow$ Ranking = LAD'23, PathLAD+, Glasgow, LAD, RI

What statistic can we compute to find the best solver?

	LAD	Glasgow	RI	PathLAD+	LAD'23
# fastest	4	4 224	11 169	2 458	3 433
# solved in 1 000s	14 678	14 858	14 242	14 916	14 972

1. Number of instances for which a solver is the fastest
   $\rightsquigarrow$ Ranking = RI, Glasgow, LAD'23, PathLAD+, LAD
2. Number of instances whose solving time is lower than $1,000$s
   $\rightsquigarrow$ Ranking = LAD'23, PathLAD+, Glasgow, LAD, RI
3. Number of instances whose solving time is lower than 1s
   $\rightsquigarrow$ Ranking = LAD'23, Glasgow, RI, PathLAD+, LAD
4. Number of instances whose solving time is lower than or equal to 0.01s
   $\rightsquigarrow$ Ranking = RI, LAD'23, Glasgow, PathLAD+, LAD
5. Time to solve an instance (average on instances solved within 1000s)
   $\rightsquigarrow$ Ranking = Glasgow, LAD'23, LAD, PathLAD+, RI
6. Lower bound of the average solving time on the $15,128$ instances
   (the time of unsolved instances is lower bounded by the time limit)
   $\rightsquigarrow$ Ranking = LAD'23, PathLAD+, Glasgow, LAD, RI

What statistic can we compute to find the best solver?

	LAD	Glasgow	RI	PathLAD+	LAD'23
# fastest	4	4 224	11 169	2 458	3 433
# solved in 1 000s	14 678	14 858	14 242	14 916	14 972
# solved in 1s	11 878	14 493	13 632	12 273	14 594

1. Number of instances for which a solver is the fastest
   $\leadsto$ Ranking = RI, Glasgow, LAD'23, PathLAD+, LAD
2. Number of instances whose solving time is lower than $1,000$s
   $\leadsto$ Ranking = LAD'23, PathLAD+, Glasgow, LAD, RI
3. Number of instances whose solving time is lower than 1s
   $\leadsto$ Ranking = LAD'23, Glasgow, RI, PathLAD+, LAD
4. Number of instances whose solving time is lower than or equal to 0.01s
   $\leadsto$ Ranking = RI, LAD'23, Glasgow, PathLAD+, LAD
5. Time to solve an instance (average on instances solved within 1000s)
   $\leadsto$ Ranking = Glasgow, LAD'23, LAD, PathLAD+, RI
6. Lower bound of the average solving time on the $15,128$ instances
   (the time of unsolved instances is lower bounded by the time limit)
   $\leadsto$ Ranking = LAD'23, PathLAD+, Glasgow, LAD, RI

What statistic can we compute to find the best solver?

	LAD	Glasgow	RI	PathLAD+	LAD'23
# fastest	4	4 224	11 169	2 458	3 433
# solved in 1 000s	14 678	14 858	14 242	14 916	14 972
# solved in 1s	11 878	14 493	13 632	12 273	14 594
# solved in 0.01s	4 918	5 831	12 438	5 799	7 483

1. Number of instances for which a solver is the fastest
   $\rightsquigarrow$ Ranking = RI, Glasgow, LAD'23, PathLAD+, LAD
2. Number of instances whose solving time is lower than $1,000$s
   $\rightsquigarrow$ Ranking = LAD'23, PathLAD+, Glasgow, LAD, RI
3. Number of instances whose solving time is lower than 1s
   $\rightsquigarrow$ Ranking = LAD'23, Glasgow, RI, PathLAD+, LAD
4. Number of instances whose solving time is lower than or equal to 0.01s
   $\rightsquigarrow$ Ranking = RI, LAD'23, Glasgow, PathLAD+, LAD
5. Time to solve an instance (average on instances solved within 1000s)
   $\rightsquigarrow$ Ranking = Glasgow, LAD'23, LAD, PathLAD+, RI
6. Lower bound of the average solving time on the $15,128$ instances
   (the time of unsolved instances is lower bounded by the time limit)
   $\rightsquigarrow$ Ranking = LAD'23, PathLAD+, Glasgow, LAD, RI

What statistic can we compute to find the best solver?

	LAD	Glasgow	RI	PathLAD+	LAD'23
# fastest	4	4 224	11 169	2 458	3 433
# solved in 1 000s	14 678	14 858	14 242	14 916	14 972
# solved in 1s	11 878	14 493	13 632	12 273	14 594
# solved in 0.01s	4 918	5 831	12 438	5 799	7 483
avg time (solved)	3.69	1.62	5.82	4.91	1.98

1. Number of instances for which a solver is the fastest
   $\leadsto$ Ranking = RI, Glasgow, LAD'23, PathLAD+, LAD
2. Number of instances whose solving time is lower than $1,000$s
   $\leadsto$ Ranking = LAD'23, PathLAD+, Glasgow, LAD, RI
3. Number of instances whose solving time is lower than 1s
   $\leadsto$ Ranking = LAD'23, Glasgow, RI, PathLAD+, LAD
4. Number of instances whose solving time is lower than or equal to 0.01s
   $\leadsto$ Ranking = RI, LAD'23, Glasgow, PathLAD+, LAD
5. Time to solve an instance (average on instances solved within 1000s)
   $\leadsto$ Ranking = Glasgow, LAD'23, LAD, PathLAD+, RI
6. Lower bound of the average solving time on the $15,128$ instances
   (the time of unsolved instances is lower bounded by the time limit)
   $\leadsto$ Ranking = LAD'23, PathLAD+, Glasgow, LAD, RI

What statistic can we compute to find the best solver?

	LAD	Glasgow	RI	PathLAD+	LAD'23
# fastest	4	4 224	11 169	2 458	3 433
# solved in 1 000s	14 678	14 858	14 242	14 916	14 972
# solved in 1s	11 878	14 493	13 632	12 273	14 594
# solved in 0.01s	4 918	5 831	12 438	5 799	7 483
avg time (solved)	3.69	1.62	5.82	4.91	1.98
bound on avg time	33.33	19.44	64.05	18.85	12.27

1. Number of instances for which a solver is the fastest
   $\leadsto$ Ranking = RI, Glasgow, LAD'23, PathLAD+, LAD
2. Number of instances whose solving time is lower than $1,000$s
   $\leadsto$ Ranking = LAD'23, PathLAD+, Glasgow, LAD, RI
3. Number of instances whose solving time is lower than 1s
   $\leadsto$ Ranking = LAD'23, Glasgow, RI, PathLAD+, LAD
4. Number of instances whose solving time is lower than or equal to 0.01s
   $\leadsto$ Ranking = RI, LAD'23, Glasgow, PathLAD+, LAD
5. Time to solve an instance (average on instances solved within 1000s)
   $\leadsto$ Ranking = Glasgow, LAD'23, LAD, PathLAD+, RI
6. Lower bound of the average solving time on the $15,128$ instances
   (the time of unsolved instances is lower bounded by the time limit)
   $\leadsto$ Ranking = LAD'23, PathLAD+, Glasgow, LAD, RI

What statistic can we compute to find the best solver?

	LAD	Glasgow	RI	PathLAD+	LAD'23
# fastest	4	4 224	11 169	2 458	3 433
# solved in 1 000s	14 678	14 858	14 242	14 916	14 972
# solved in 1s	11 878	14 493	13 632	12 273	14 594
# solved in 0.01s	4 918	5 831	12 438	5 799	7 483
avg time (solved)	3.69	1.62	5.82	4.91	1.98
bound on avg time	33.33	19.44	64.05	18.85	12.27

First conclusions:

- LAD'23 is able to solve more instances when time limit = 1000s or 1s
  But RI is able to solve more instances when time limit = 0.01s
- Glasgow has the smallest average solving time on solved instances
  But the subset of solved instances is different from a solver to another
- LAD'23 has the smallest lower bound on the average solving time
  But RI is the fastest for a wide majority of instances
- The ranking strongly depends on the considered statistic...

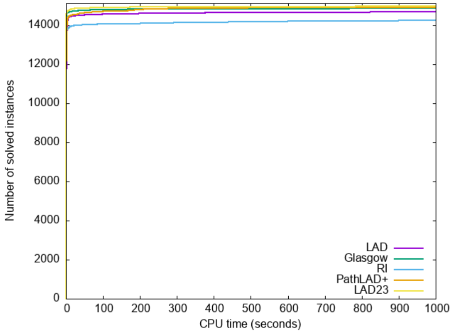
There is no clear winner $\leadsto$ We need to analyse results more carefully

Cactus Plot: Number of solved instances wrt time

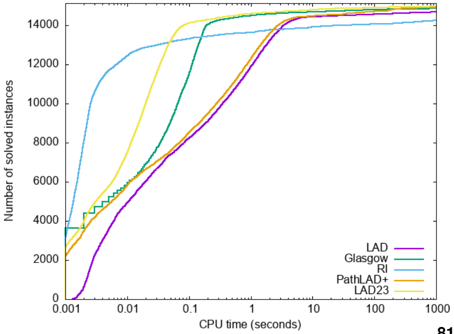
How to produce a cactus plot for a solver s?

- For each instance i, let t_i be the time spent by s to solve i
- Initialise a counter c to 0
- For each solving time t_i, taken by increasing order:
  Increase c and plot the point (t_i, c)

With a linear scale:



With a logarithmic scale:

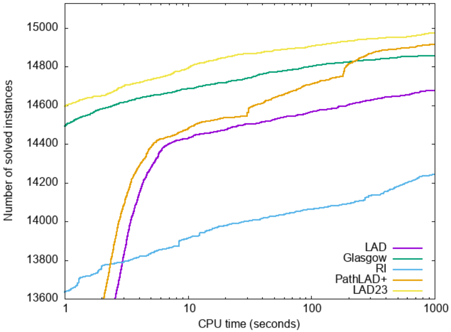


Cactus Plot: Number of solved instances wrt time

How to produce a cactus plot for a solver *s*?

- For each instance *i*, let t_i be the time spent by *s* to solve *i*
- Initialise a counter *c* to 0
- For each solving time t_i, taken by increasing order:
  Increase *c* and plot the point (t_i, c)

With a logarithmic scale + zoom:



With a logarithmic scale:

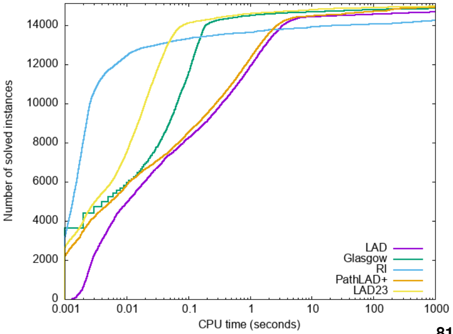


Figure from the paper that introduced PathLAD+

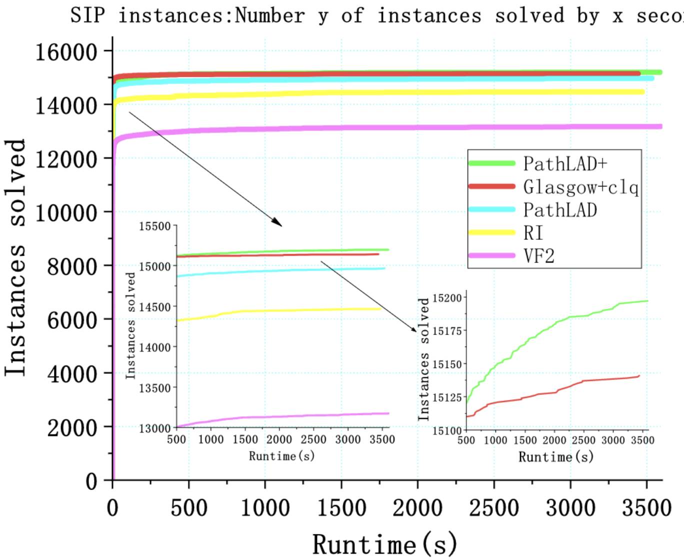


Table from the paper that introduced PathLAD+

Benchmark	#inst	PathLAD+ #solved	Glasgow+Clq #solved	PathLAD #solved	RI #solved	VF2 #solved
images-CVIU11	6278	**6278**	**6278**	**6278**	**6278**	**6278**
meshes-CVIU11	3018	**3008**	2987	2983	2695	2647
images-PR15	24	**24**	**24**	**24**	**24**	**24**
scalefree	100	**100**	**100**	**100**	82	21
si	1170	**1170**	**1170**	1109	1163	886
phase-transition	200	**134**	128	44	31	0
LV	1176	**1139**	1136	1130	1039	811
LargerLV	3430	**3344**	3318	3300	3154	2505
#total	15396	**15197**	15141	14968	14466	13172

Quote from the paper:

Overall, the performance of PathLAD+ totally dominates Glasgow+Clq, PathLAD, RI, and VF2.

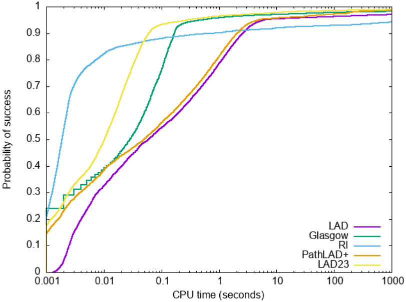
Any reaction?

CDF: Probability of success wrt time

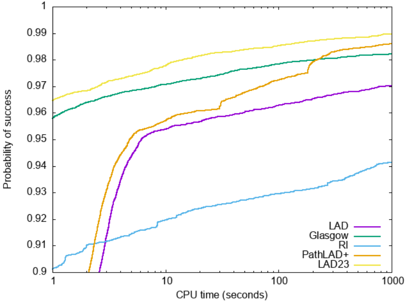
How to obtain a CDF from a cactus plot?

- Divide the number of solved instances by the total number of instances

With a logarithmic scale:



With a logarithmic scale + Zoom:

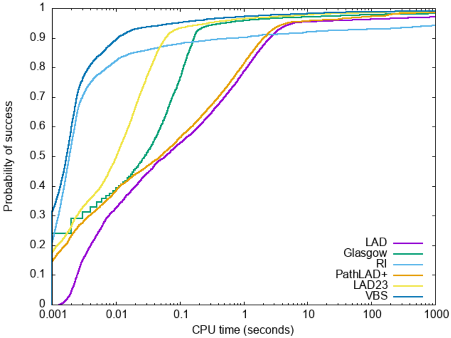


Virtual Best Solver (VBS)

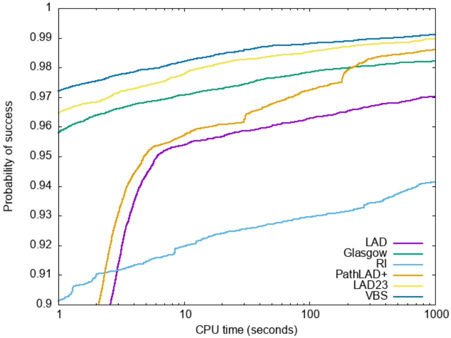
VBS associated with a set $\mathcal{S}$ of solvers and a set $\mathcal{I}$ of instances:

- $\forall s \in \mathcal{S}, \forall i \in \mathcal{I}$, let t_i^s be the runtime of s on i
- $\forall i \in \mathcal{I}$, runtime of VBS on i: $t_i^{VBS} = \min_{s \in \mathcal{S}} t_i^s$

With a logarithmic scale:



With a logarithmic scale + Zoom:



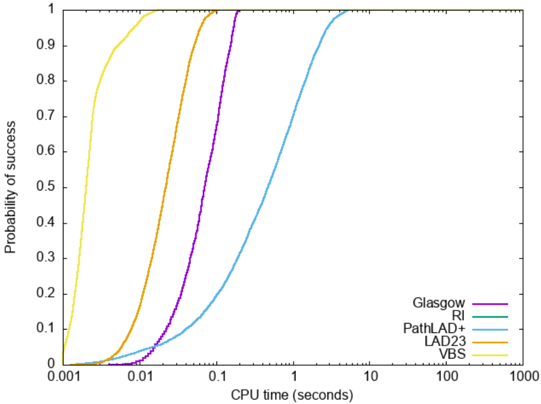
What have we learned so far?

- For very small runtime limits (<0.04s), RI has the highest success probability
- For larger runtime limits (>0.04s), LAD'23 has the highest success probability
- PathLAD+ has a higher success probability than Glasgow only when the runtime limit is larger than 200s

Warning:

This is a global picture for an unbalanced benchmark that contains a lot of easy instances and a few very hard instances
$\leadsto$ **Analyse results for each class separately**

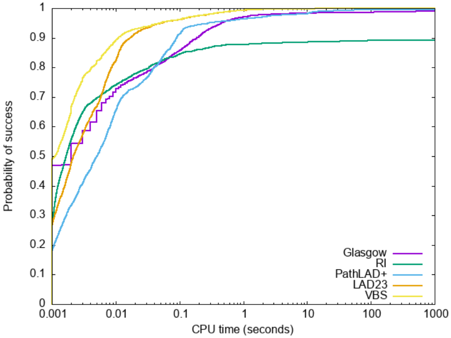
Results on the 6302 instances of Class *Images*



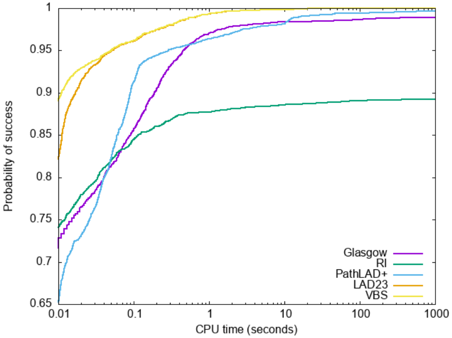
- RI = VBS for all instances of this class
  $\rightsquigarrow$ It never needs more than 0.02s to solve an instance
- Other approaches also solve all instances but they need more time
- Ranking on this benchmark: RI, LAD'23, Glasgow, PathLAD+

Results on the 3018 instances of Class *Meshes*

Big picture:



With a zoom:

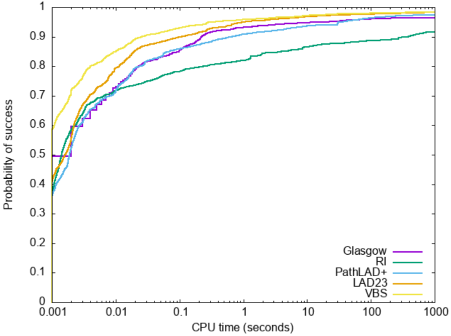


Conclusions:

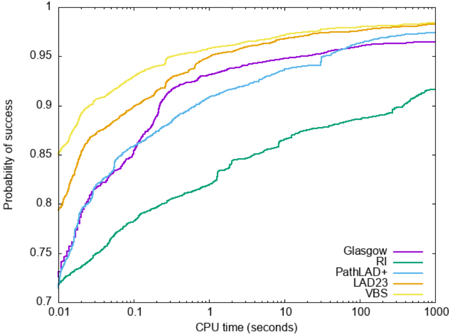
- RI is the most successful when time < 0.007s
- LAD'23 is the most successful when time > 0.007s
  $\leadsto$ It is the only one able to solve all instances

Results on the 4338 instances of Class *LV*

Big picture:



With a zoom:

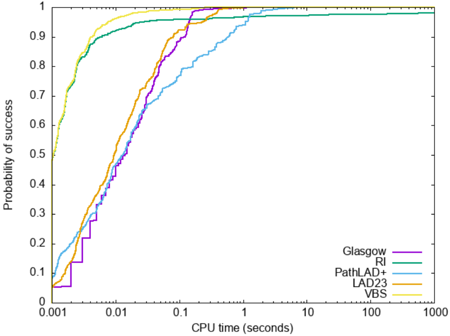


Conclusions:

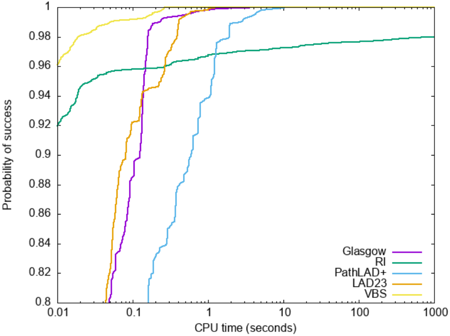
- LAD'23 is the most successful, except for very short time limits (< 0.003)
- 80% of the instances are solved within 0.01s by LAD'23
  But some instances are really hard (75 unsolved instances)

Results on the 1000 instances of *Rand*

Big picture:



With a zoom:

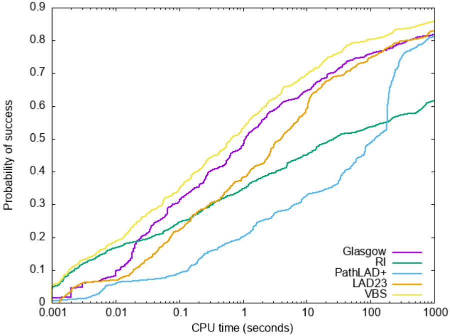


Conclusions:

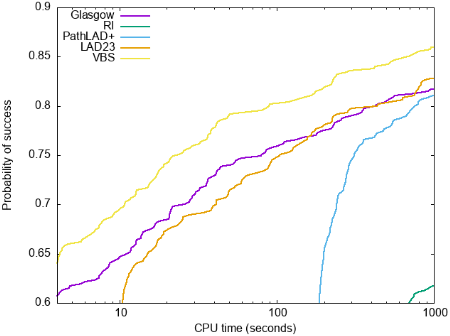
- RI is the most successful when time < 0.15s
- Glasgow is the most successful when time > 0.15s
- Glasgow, LAD'23 and PathLAD+ solve all instances within 10s whereas RI fails on 20 instances within 1000s

Results on the 470 instances of Class *RandER*

Big picture:



With a zoom:

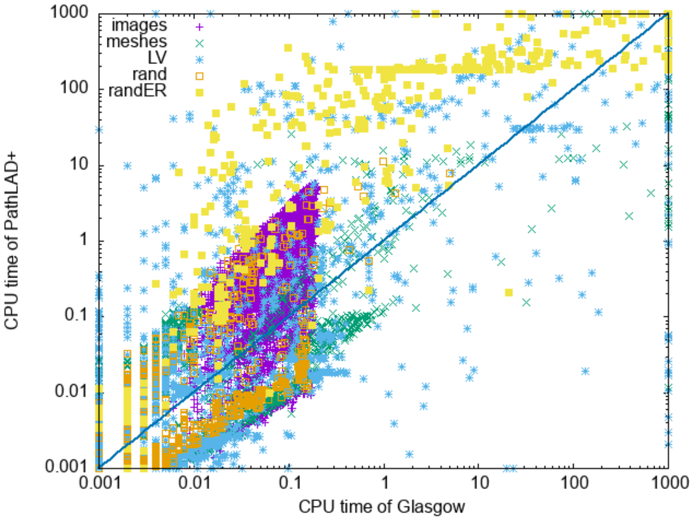


Conclusions:

- RI is the most successful when time < 0.03s
  But it is the less successful when time > 200s
- Glasgow is the most successful when time $\in$ [0.03s,150s]
- Glasgow and LAD'23 have similar performance when time > 150s

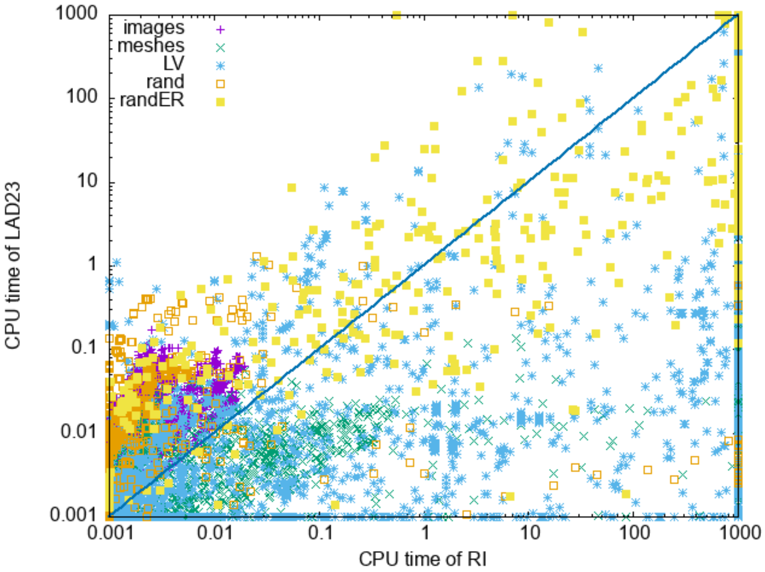
Per instance comparison of Glasgow and PathLAD+

Scatter plot: plot a point (x, y) for each instance with $x = t^{\text{Glasgow}}$ and $y = t^{\text{PathLAD+}}$



Can we really say that "PathLAD+ totally dominates Glasgow"?

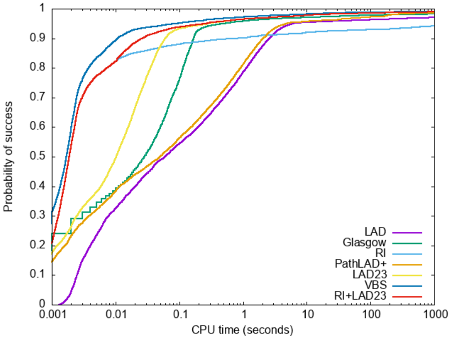
Scatter plot of RI and LAD'23



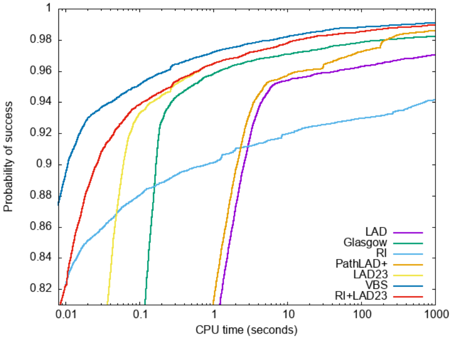
Let’s combine RI and LAD’23 sequentially

$\leadsto$ **Run RI for 0.01s and if the instance isn’t solved, run LAD’23**

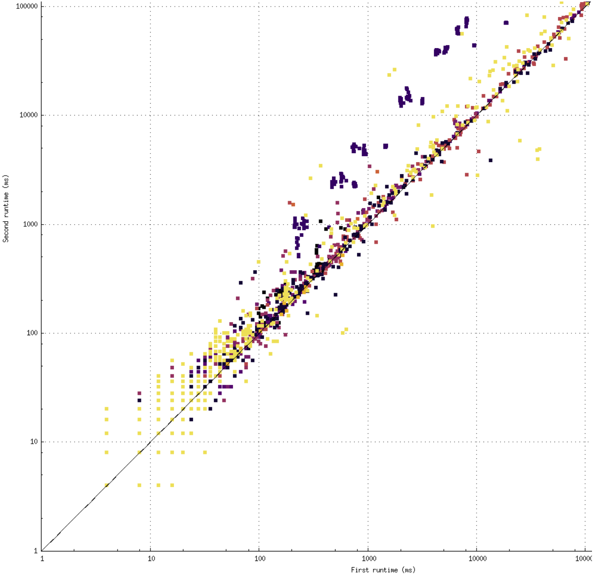
Big picture:



With a zoom:



Using Scatter plots to evaluate sensibility to randomness



Conclusion of this experiment

Modern solvers are able to quickly solve large instances...

...But there are small instances that are still very challenging
$\leadsto$ Don't forget to evaluate your favorite solver on these instances too!

Plotting the evolution of time wrt size is not very meaningful

Better pictures are given by plotting CDF, and scatter plots

Advertisement: Have a look at Metrics Studio (http://crillab-metrics.cloud/dash/)

Conclusions are different from a benchmark to another

- Consider as many benchmarks as possible
- Analyse results for each benchmark separately, especially in case of unbalanced benchmarks
- Use automatic selection tools to improve performance

Plan

Supervised Classification (recalls)

Input data:

- A set O of objects
  $\rightsquigarrow$ For ex: O = set of images of dogs, cats, and birds
- A set C of classes
  $\rightsquigarrow$ For ex: $C = \{\text{dog, cat, bird}\}$
- A training set $S_t \subset O$ and a ground truth function $c : S_t \rightarrow C$ that returns the class $c(o)$ of every objet $o \in S_t$

Dogs

Cats

Birds

Output model:

A classifier $\kappa : O \rightarrow C$ such that, $\forall o \in O$, $\kappa(o)$ = predicted class for o

$\rightsquigarrow$ For ex: $\kappa($$) = ?$

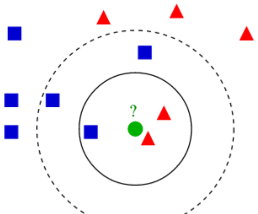
Example 1: k Nearest Neighbours (kNN)

Model = S_t + distance $d : O \times O \rightarrow \mathbb{R}^+$

- In general: Definition of d in a vector space $\mathbb{R}^n$
  $\leadsto$ Each object is described by a numerical vector
- Ideally: Choose d st $c(o_i) = c(o_j) \neq c(o_k) \Rightarrow d(o_i, o_j) < d(o_i, o_k)$
  $\leadsto$ d may be learned (metric learning)

Classification of an object $o \in S_t$:

- Compute the set A of the k objects of S_t which are the nearest to o
- $\kappa(o)$ = most frequent class in A (possibility to weight with d)



(image: Wikipedia)

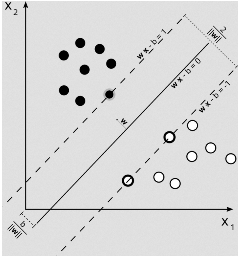
Example 2: SVM (*Support Vector Machine*)

Model in the case of 2 classes:

- Search for the hyperplan that maximises the gap between the classes
  $\rightsquigarrow$ Quadratic optimisation problem
- Projection ϕ of the objects in a space of larger dimension
  $\rightsquigarrow$ Use a kernel $\kappa : O \times O \rightarrow \mathbb{R}^+$ such that $\kappa(x_i, x_j) = \phi(x_i)^T \cdot \phi(x_j)$

Classification of an object $o \in O$:

Compute class by looking at the position of $\phi(o)$ wrt the hyperplan



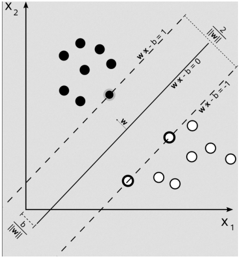
Example 2: SVM (*Support Vector Machine*)

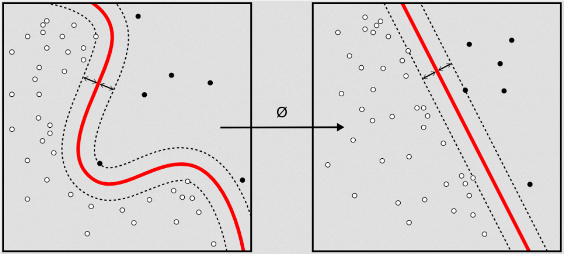
Model in the case of 2 classes:

- Search for the hyperplan that maximises the gap between the classes
  $\leadsto$ Quadratic optimisation problem
- Projection ϕ of the objects in a space of larger dimension
  $\leadsto$ Use a kernel $\kappa : O \times O \to \mathbb{R}^+$ such that $\kappa(x_i, x_j) = \phi(x_i)^T \cdot \phi(x_j)$

Classification of an object $o \in O$:

Compute class by looking at the position of $\phi(o)$ wrt the hyperplan





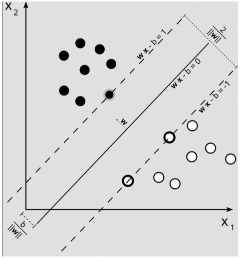
Example 2: SVM (*Support Vector Machine*)

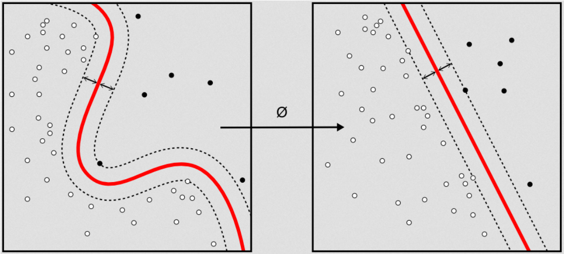
Model in the case of 2 classes:

- Search for the hyperplan that maximises the gap between the classes
  $\rightsquigarrow$ Quadratic optimisation problem
- Projection ϕ of the objects in a space of larger dimension
  $\rightsquigarrow$ Use a kernel $\kappa : O \times O \rightarrow \mathbb{R}^+$ such that $\kappa(x_i, x_j) = \phi(x_i)^T \cdot \phi(x_j)$

Classification of an object $o \in O$:

Compute class by looking at the position of $\phi(o)$ wrt the hyperplan





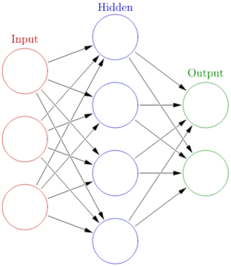
Example 3: Neural Networks

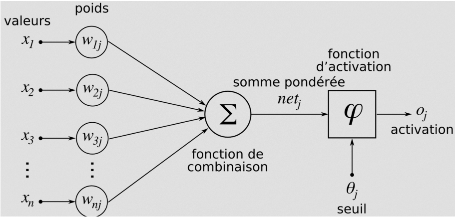
Model:

- Network structure (number of layers, connections between the layers)
  $\rightsquigarrow$ First layer = 1 neuron per feature of the objects of O
  $\rightsquigarrow$ Last layer = 1 neuron per class
- Connection weights are learned on S_t by using backpropagation

Classification of an object $o \in O$

Activation of input neurons / features of o





Training and Test Sets

Training set:

Data used to learn a model (*i.e.*, search for parameter values)

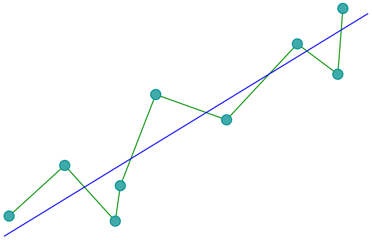
Test set:

Data used to evaluate the learned model

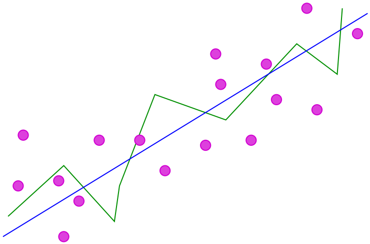
Why testing on data different from those used to train the model?

$\leadsto$ To check that the learned model is not overfitted

Models learned on a training set:



Evaluation on a test set:



Hyper-parameter Tuning and Validation Set

What is an hyper-parameter?

- Parameter that must be assigned before learning a model
  $\leadsto$ Number of layers in a neural network, SVM kernel, etc
- Each hyper-parameter has a (possibly infinite) set of possible values

Which combinations of values should be tested?

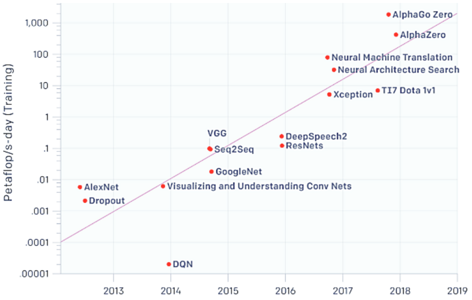
- Grid search: Choose a subset of values for each parameter, and consider all possible combinations of these subsets
- Random search: Consider k randomly chosen combinations
  $\leadsto$ k controls the proba. of choosing at least 1 among the x% best ones

Use a validation set to choose the best combination

- For each combination, learn a model
  $\leadsto$ Use a training set to learn these models
- Compare the learned models on the validation set
  $\leadsto$ Select the best one
- Evaluate the selected model on the test set

Training and Hyper-parameter tuning costs:

Training time of AI models:



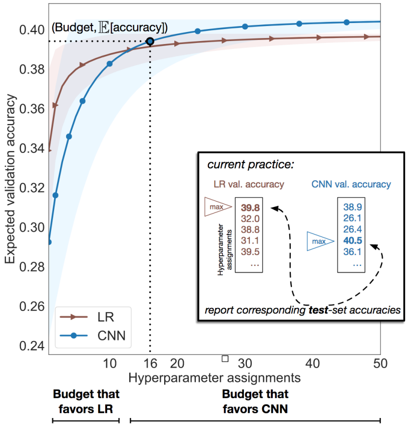
But that's just the tip of the iceberg!

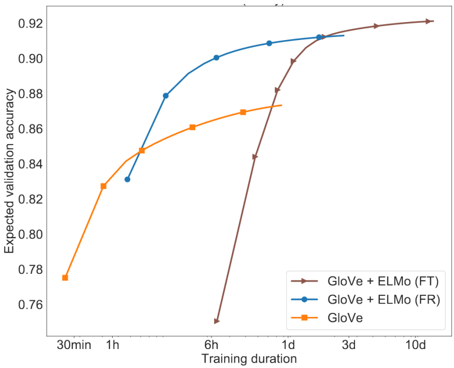
Consumption	CO_2e (lbs)
Air travel, 1 passenger, NY↔SF	1984
Human life, avg, 1 year	11,023
American life, avg, 1 year	36,156
Car, avg incl. fuel, 1 lifetime	126,000
Training one model (GPU)	
NLP pipeline (parsing, SRL)	39
w/ tuning & experimentation	78,468
Transformer (big)	192
w/ neural architecture search	626,155

Reference:

Strubell, Ganesh, & McCallum (2019): Energy and Policy Considerations for Deep Learning in NLP

Evolution of Performance wrt Training Cost





Reference:

J. Dodge, S. Gururangan, D. Card, R. Schwartz, N. A. Smith. Show Your Work (2019): Improved Reporting of Experimental Results

Choice of Training, Validation and Test sets

In theory:

Training, validation and test sets should:

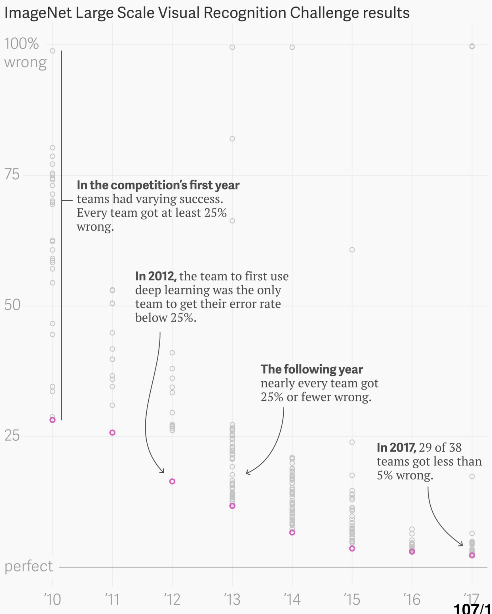
- be independent
- follow a same distribution and not be biased
- contain enough examples to avoid overfitting

In practice:

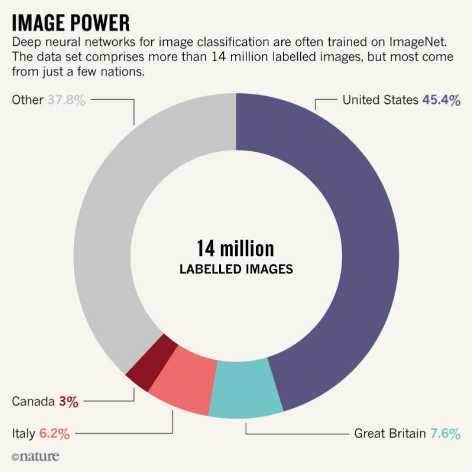
- We usually do not know the probability distribution of data on which the model will be used
  $\rightsquigarrow$ Domain adaptation
- Classes may not be balanced (e.g., anomaly detection)
- Data may contain bias
- ...

Example 1: ImageNet (1/2)

2009 : ImageNet



Example 1: ImageNet (2/2)



Which of these two photos represents a bride?

Learned models reproduce biases of training data!

Zou and Schiebinger: *AI can be sexist and racist — it's time to make it fair*, in Nature 559, 324-326 (2018)

Example 2: Word Embedding (*e.g.*, Word2vec)

Basic principle of word embedding:

- Embed words in a vectorial space
  $\leadsto$ Distance between words in this space reflects context similarity
- Used for inferring analogies
  $\leadsto$ king - man + woman = ?

Results obtained when training on *news.google.com*:

- doctor - man + woman =
- computer programmer - man + *woman* =

Example 2: Word Embedding (*e.g.*, Word2vec)

Basic principle of word embedding:

- Embed words in a vectorial space
  $\rightsquigarrow$ Distance between words in this space reflects context similarity
- Used for inferring analogies
  $\rightsquigarrow$ king - man + woman = queen

Results obtained when training on *news.google.com*:

- doctor - man + woman = ?
- computer programmer - man + *woman* = ?

Example 2: Word Embedding (*e.g.*, Word2vec)

Basic principle of word embedding:

- Embed words in a vectorial space
  $\rightsquigarrow$ Distance between words in this space reflects context similarity
- Used for inferring analogies
  $\rightsquigarrow$ king - man + woman = queen

Results obtained when training on *news.google.com*:

- doctor - man + woman = nurse
- computer programmer - man + *woman* = homemaker

$\rightsquigarrow$ The learned embedding reproduces biases

Bolukbasi et al: *Man is to Computer Programmer as Woman is to Homemaker? Debiasing Word Embeddings*, in NIPS 2016

Example 3: ChatGPT

Peux tu traduire en français le paragraphe suivant : The surgeon is assisted by a nurse. The nurse has a beautiful beard. The surgeon has a dress.

Bien sûr, voici la traduction en français du paragraphe : Le chirurgien est assisté par une infirmière. L'infirmière a une belle barbe. Le chirurgien porte une robe.

Question :

Should training sets be representative of the world as it is, or the world we'd like to live in?

Example 3: ChatGPT

Peux tu traduire en français le paragraphe suivant : The surgeon is assisted by a nurse. The nurse has a beautiful beard. The surgeon has a dress.

Bien sûr, voici la traduction en français du paragraphe : Le chirurgien est assisté par une infirmière. L'infirmière a une belle barbe. Le chirurgien porte une robe.

Notez cependant que ce paragraphe ne semble pas très cohérent, car il est inhabituel pour une infirmière d'avoir une barbe et pour un chirurgien de porter une robe.

Question :

Should training sets be representative of the world as it is, or the world we'd like to live in?

Example 3: ChatGPT

Peux tu traduire en français le paragraphe suivant : The surgeon is assisted by a nurse. The nurse has a beautiful beard. The surgeon has a dress.

Bien sûr, voici la traduction en français du paragraphe : Le chirurgien est assisté par une infirmière. L'infirmière a une belle barbe. Le chirurgien porte une robe.

Notez cependant que ce paragraphe ne semble pas très cohérent, car il est inhabituel pour une infirmière d'avoir une barbe et pour un chirurgien de porter une robe.

Question :

Should training sets be representative of the world as it is, or the world we'd like to live in?

Experimental Protocol

k-fold cross-validation:

Given a set $S \subseteq O$ of objects that have known classes

- Partition S in k folds $S_1, \ldots, S_k$
  (or randomly select k folds)
- For each fold $S_i \in \{S_1, \ldots, S_k\}$:
  - Learn a model on the training set $S_T = S \setminus S_i$
  - Evaluate the model on the testing set S_i
- Statistical analysis of performance measures for the k folds

Leave-one-out:

Cross-validation with $k = |S|$
$\rightsquigarrow$ Used when there are very few objects in S

Evaluation of the Quality of a Binary Classifier

Confusion matrix:

	Actual Class +	Actual Class -
Predicted Class +	TP	FP
Predicted Class -	FN	TN
Total per column	P	N

- TP = *True Positive*
- TN = *True Negative*
- FP = *False Positive* $\rightsquigarrow$ False alarm
- FN = *False Negative* $\rightsquigarrow$ Missed
- P = *total Positive* = TP+FN
- N = *total Negative* = FP+TN

Some measures:

- Accuracy = $\frac{TP+TN}{P+N}$
- Precision = $\frac{TP}{TP+FP}$
- True Positive Rate (Recall) = $\frac{TP}{P}$ $\rightsquigarrow$ Detection probability
- False Positive Rate = $\frac{FP}{N}$ $\rightsquigarrow$ False alarm probability
- F-measure = $\frac{2TP}{2TP+FP+FN}$ $\rightsquigarrow$ Harmonic mean of Precision and Recall

Example of Results for Unbalanced Classes

$\leadsto$ **Fraud or anomaly detection, etc**

Results of A:

	Actual Class +	Actual Class -
Predict +	44	150
Predict -	156	2850
Total	200	3000

- Acc. = $\frac{2894}{3200}$ = **90.44%**
- Prec. = $\frac{44}{194}$ = 22.68%
- TPR = $\frac{44}{200}$ = 22%
- FPR = $\frac{150}{3000}$ = **5%**
- F = $\frac{88}{394}$ = 22.34%

Results of B:

	Actual Class +	Actual Class -
Predict +	83	375
Predict -	117	2625
Total	200	3000

- Acc. = $\frac{2708}{3200}$ = 84.62%
- Prec. = $\frac{83}{458}$ = 18.12%
- TPR = $\frac{83}{200}$ = **41.5%**
- FPR = $\frac{375}{3000}$ = 12.5%
- F = $\frac{166}{658}$ = 25.23%

Results of C:

	Actual Class +	Actual Class -
Predict +	60	185
Predict -	140	2815
Total	200	3000

- Acc. = $\frac{2875}{3200}$ = 89.84%
- Prec. = $\frac{60}{245}$ = **24.49%**
- TPR = $\frac{60}{200}$ = 30%
- FPR = $\frac{185}{3000}$ = 6.17%
- F = $\frac{120}{445}$ = **26.97%**

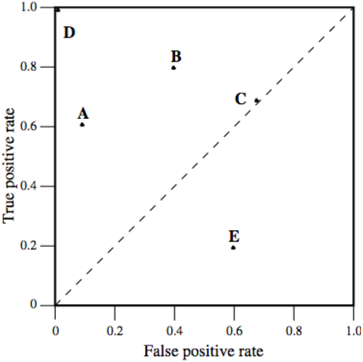
Measures may be weighted (see IF-4-FD)

$\leadsto$ Associate a weight with every case $\in \{TP, TN, FP, FN\}$

ROC (Receiver Operating Characteristic) Space

Compromise between income (y=TPR) and cost (x=FPR):

- $(0, 0) \rightsquigarrow$ Classifier always answering -
- $(1, 1) \rightsquigarrow$ Classifier always answering +
- $(0, 1) \rightsquigarrow$ Perfect classifier
- $x = y \rightsquigarrow$ Random classifier ($p(+) = x$; $p(-) = 1 - x$)



Bi-criteria optimisation:

(x_1, y_1) dominate (x_2, y_2) if $x_1 \leq x_2$ and $y_1 \geq y_2$
$\rightsquigarrow$ Strict domination if $(x_1, y_1) \neq (x_2, y_2)$

Reference:

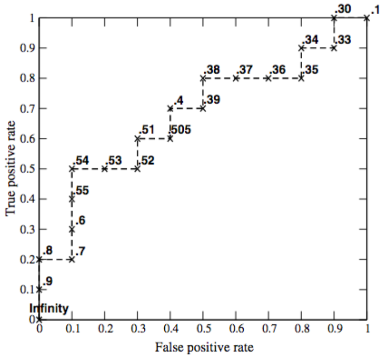
T. Fawcett: *An introduction to ROC analysis*. PRL 2006

ROC Curve

- Some classifiers may return a numerical value (probability, score, ...) instead of a class $\rightsquigarrow$ Class determined wrt a threshold
- ROC curve = Points obtained when considering different thresholds $\rightsquigarrow$ Cumulative distribution function of TPR wrt FPR

Example from [Fawcett 2006]:

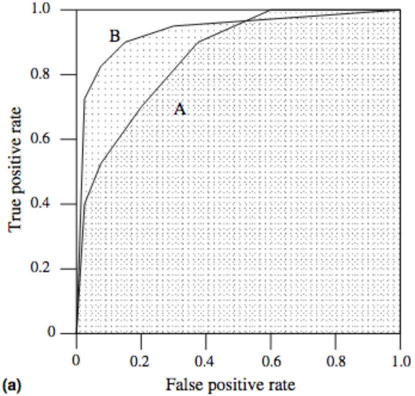
Inst#	Class	Score	Inst#	Class	Score
1	p	.9	11	p	.4
2	p	.8	12	n	.39
3	n	.7	13	p	.38
4	p	.6	14	n	.37
5	p	.55	15	n	.36
6	p	.54	16	n	.35
7	n	.53	17	p	.34
8	n	.52	18	n	.33
9	p	.51	19	p	.30
10	n	.505	20	n	.1



Area under the ROC curve (AUC)

- Between 0 and 1
  $\rightsquigarrow$ AUC of a random classifier = 0.5
- P(score of a positive instance > score of a negative instance)

Example from [Fawcett 2006]:



Evaluation of the Quality of a non Binary Classifier

Confusion Matrix and Accuracy:

	Actual Class A	Actual Class B	Actual Class C	Actual Class D
Predicted Class A	12	4	2	0
Predicted Class B	3	8	3	1
Predicted Class C	0	1	9	0
Predicted Class D	0	2	1	14

$$\text{Accuracy} = \frac{12+8+9+14}{60}$$

Measures associated with *n* binary classifiers:

- A vs NonA: TPR=$\frac{12}{15}$; FPR=$\frac{6}{45}$; ...
- B vs NonB: TPR=$\frac{8}{15}$; FPR=$\frac{7}{45}$; ...
- C vs NonC: TPR=$\frac{9}{15}$; FPR=$\frac{1}{45}$; ...
- D vs NonD: TPR=$\frac{14}{15}$; FPR=$\frac{3}{45}$; ...

Other Performance Criteria

Size of the training set and number of tested configurations to tune hyper-parameters:

- Criteria that impact the classifier quality
- Plot the evolution of quality measures wrt these criteria

Duration / Cost:

- Offline: Cost to train the model
  $\rightsquigarrow$ Usually dependent from the size of the training set...
- Online: Cost to make a forecast

... and many other properties:

- Equity and Fairness
- Robustness, Security, and Confidentiality
- ...
- Explainability

Why should we explain?

Europe, 2018: Article 22 of General Data Protection Regulation

The data subject shall have the right not to be subject to a decision based solely on automated processing, including profiling, which produces legal effects concerning him or her or similarly significantly affects him or her. (...) The data controller shall implement suitable measures to safeguard the data subject's rights and freedoms and legitimate interests, at least the **right to obtain human intervention on the part of the controller, to express his or her point of view and to contest the decision**.

France, 2016: Loi pour une République Numérique

Algorithms used by French administration:

- Explicitly mention that the decision is based on an algorithmic treatment
- Communication of the source code and the main features of the algorithm
- Right to appeal against an individual administrative decision based on algorithmic processing

What is a (good) explanation?

It depends on the goal of the user!

Understand a system before using it

Describe the algorithm and/or open source code, training data, ...
$\rightsquigarrow$ Transparency

Check that a decision is fair, or contest a decision

Identify the features that most influenced the decision
$\rightsquigarrow$ Counterfactuals

Assist the decision making process in case of recommandations

Justify the decision

Understand the causes of an error to correct it

Identify the causes of the decision

Why is it difficult to explain?

Code may not be open

$\rightsquigarrow$ Competition issues

Data may not be open

$\rightsquigarrow$ Competition and privacy issues

Explanations must be understandable

Some machine learning approaches are easy to interpret (e.g. decision trees)
But most of them are not (e.g. CNN)

- We may learn explanations during the training process
  But can we trust these explanations?
  See Andreou et al, 2018: Investigating ad transparency mechanisms in social media: A case study of Facebook's explanations
- Is it possible to explain an algorithm by observing it as a black box?

See REGALIA on the regulation of algorithms