Experimental Evaluation

Good Practices and Pitfalls to Avoid

Christine Solnon

INSA de Lyon - CITI - INRIA

27 November 2020

Overview of the talk

Theory versus Experimentation (1/2)

Some properties may be proven by theoretical analysis:

- Complexity and decidability of a problem
- Complexity, correctness, completeness, termination, ... of an algorithm
- Consistency level and time complexity of a constraint propagator
- ...

But theory has some limits:

- A theoretical complexity gives a growth order
  ...and all exact solvers for NP-hard problems have exp. time complexities
- An hand made proof may contain errors
- Static analysis may raise false alarms
- ...

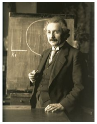
Experimentation is complementary to theoretical analysis:

It provides empirical insights into algorithm properties

Theory versus Experimentation (2/2)

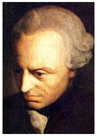
In theory, theory and practice are the same.
In practice, they are not.

(A. Einstein)



Experience without theory is blind,
but theory without experience is mere intellectual play.

(I. Kant)



If you find that you're spending almost all your time on theory, start turning some attention to practical things; it will improve your theories. If you find that you're spending almost all your time on practice, start turning some attention to theoretical things; it will improve your practice.

(D. Knuth)

Overview of the talk

Experimental process

Step 1: Prepare the experiment

- Formulate a question
  $\rightsquigarrow$ Influence of parameters? Solver competitive with state-of-the-art? ...
- Design the experiment
  $\rightsquigarrow$ What should we measure? On which benchmark? . . .
- Prepare the test environment
  $\rightsquigarrow$ Scripts for launching tests, Computing infrastructure, . . .

Step 2: Perform the experiment

- Run scripts and collect results

Step 3: Analyse results

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

Reference: A Guide to Experimental Algorithmics, C. McGeoch, 2012

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...)

Overview of the talk

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 and tools must be available:

- Open source + Open data
- Values of all parameters
- Considered environment (processor, OS, compiler, ...)
- Tools used to launch runs and analyse results
- ...

$\leadsto$ Provide virtual machines (see the Recomputation Manifesto, Gent 2013)

Different Reproducibility Levels [ACM 2016]

Repeatability

Same experimental conditions, same team

Replicability

Same experimental conditions, different team

Reproducibility

Different experimental conditions, different team

See https://www.acm.org/publications/policies/artifact-review-badging

The Machine Learning Reproducibility Checklist (1/2)

www.cs.mcgill.ca/∼jpineau/ReproducibilityChecklist.pdf (V1.2, Mar.27 2019)

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

- A clear description of the mathematical setting, algorithm, and/or model
- An analysis of the complexity (time, space, sample size) of any algorithm
- A link to a downloadable source code, with specification of all dependencies, including external libraries

For any theoretical claim, check if you include:

- A statement of the result
- A clear explanation of any assumptions
- A complete proof of the claim

The Machine Learning Reproducibility Checklist (2/2)

www.cs.mcgill.ca/∼jpineau/ReproducibilityChecklist.pdf (V1.2, Mar.27 2019)

For all figures and tables that present empirical results, check if you include:

- A complete description of the data collection process, including sample size
- A link to a downloadable version of the dataset or simulation environment
- An explanation of any data that were excluded, description of any pre-processing step
- An explanation of how samples were allocated for training / validation / testing
- 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 evaluation runs
- A description of how experiments were run
- A clear definition of the specific measure or statistics used to report results
- Clearly defined error bars
- A description of results with central tendency (e.g. mean) & variation (e.g. stddev)
- A description of the computing infrastructure used

Overview of the talk

Choice of a Benchmark

The benchmark depends on the question addressed by the experiment

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

Homogeneous vs Heterogeneous Benchmarks

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

Hardness of Instances

Beware of ceil/floor effects!

- Extreme instances are useless to compare 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 to study scale-up properties

Factors that may influence hardness:

- Input size
- Structure of input data
  $\leadsto$ Example: tree width of the constraint graph
- Constrainedness (for decision problems)
  $\leadsto$ Phase transition
- Distribution of local and global optima (for optimisation problems)
  $\leadsto$ 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 ⇒ under-constrained instance ⇒ Easy (except for rare cases which are exceptionally hard!)
    - p/n large ⇒ over-constrained instance ⇒ 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)
    - $x = p/n$
    - $y =$ solving time
    - colour=black if feasible
    - colour=pink if infeasible

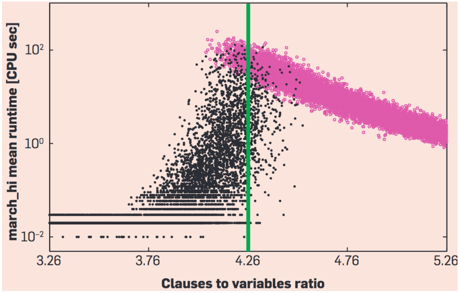
Phase transition (1/2)

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 = solving time
    - 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
  $\leadsto$ For uniform 3-SAT: When $p/n = 4.26$
- Corresponding to a hardness pic
- Independent from the solving approach

How to locate the transition phase?

Compute the expected number of solutions $\langle Sol \rangle$:

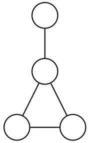
- $\langle Sol \rangle$ much smaller than 1 $\leadsto$ Over-constrained instance (easy)
- $\langle Sol \rangle$ close to 1 $\leadsto$ Critically constrained instance (hard)
- $\langle Sol \rangle$ much larger than 1 $\leadsto$ Under-constrained instance (easy)

References:

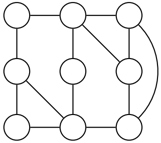
- P. Cheeseman, B. Kanefsky, W. Taylor (1991): *Where the Really Hard Problems Are*. IJCAI
- K. Leyton-Brown, H. Hoos, F. Hutter, L. Xu (2014): *Understanding the Empirical Hardness of NP-Complete Problems*. Communications of the ACM

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 $f : N_p \rightarrow N_t$ s.t. $\forall (u, v) \in E_p : (f(u), f(v)) \in E_t$

Question:

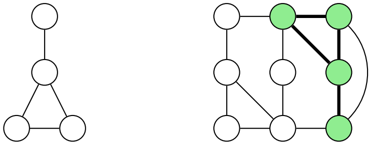
How to control hardness of randomly generated instances of SIP?

Reference:

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

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 $f : N_p \rightarrow N_t$ s.t. $\forall (u, v) \in E_p : (f(u), f(v)) \in E_t$

Question:

How to control hardness of randomly generated instances of SIP?

Reference:

C. McCreesh, P. Prosser, C. Solnon & J. 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 $\rightsquigarrow$ Sparse graphs
    - d close to 1 $\rightsquigarrow$ Dense graphs

Random generation of an SIP instance:

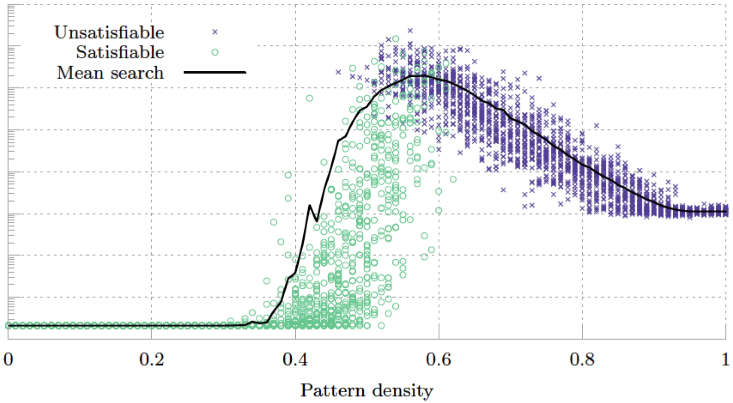
- Generation of a pattern graph $G(n_p, d_p)$ and a target graph $G(n_t, d_t)$
- Parameters = n_p, d_p, n_t, d_t

How can we control hardness?

$\rightsquigarrow$ Probabilities d_p and d_t control graph densities

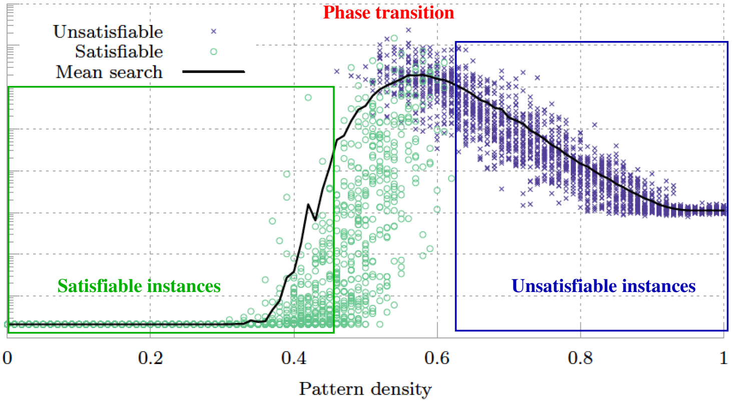
- Sparse pattern and dense target $\rightsquigarrow$ Easy to find a solution
- Dense pattern and sparse target $\rightsquigarrow$ Easy to prove inconsistency
- **Hard instances should be between these two extreme cases!?**

Phase transition from feasibility to infeasibility



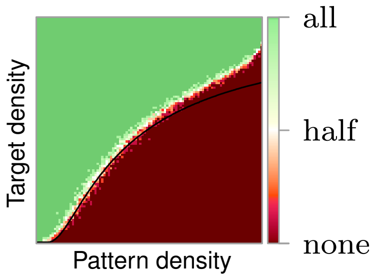
- We fix $n_p = 20$, $n_t = 150$, $d_t = 0.4$, and we vary d_p from 0 to 1
  $\rightsquigarrow$ Each point (x, y) is an instance generated with $d_p = x$
  - y = Search effort to solve the instance with Glasgow
  - Colour = Feasibility of the instance (green=yes; blue=no)

Phase transition from feasibility to infeasibility



- $d_p \leq 0.44$: **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



- We fix $n_p = 30$, $n_t = 150$, and we vary d_p and d_t from 0 to 1
  $\rightsquigarrow$ 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 $\rightsquigarrow$ All satisfiable
  - Bottom right: dense patterns and sparse targets $\rightsquigarrow$ All unsatisfiable
- Black line = Theoretical prediction of the phase transition location

Locating the phase transition

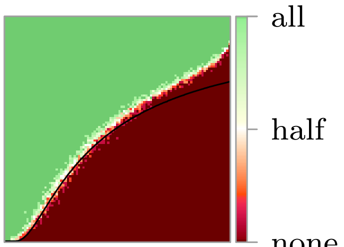
Expected number of solutions for pattern $G(n_p, d_p)$ and target $G(n_t, d_t)$:

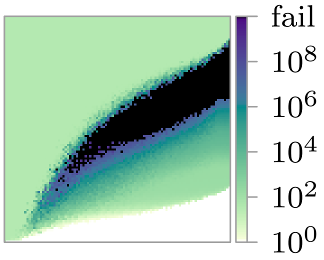
- Expected number of pattern edges = $d_p \cdot \frac{n_p(n_p-1)}{2}$
- Probability for one pattern edge to be mapped to a target edge = d_t
- Probability for one injective mapping to be a solution = $d_t^{d_p \cdot \frac{n_p(n_p-1)}{2}}$
- Number of possible injective mappings = $n_t \cdot (n_t - 1) \cdot ... \cdot (n_t - n_p + 1)$
- Expected number of solutions:
  $\langle Sol \rangle = n_t \cdot (n_t - 1) \cdot ... \cdot (n_t - n_p + 1) \cdot d_t^{d_p \cdot \frac{n_p(n_p-1)}{2}}$

Theoretical prediction of the phase transition location:

- $\langle Sol \rangle$ larger than 1 $\leadsto$ Easy to find a solution
- $\langle Sol \rangle$ smaller than 1 $\leadsto$ Not very difficult to prove inconsistency
- $\langle Sol \rangle$ close to 1 $\leadsto$ Really hard instances (black line)

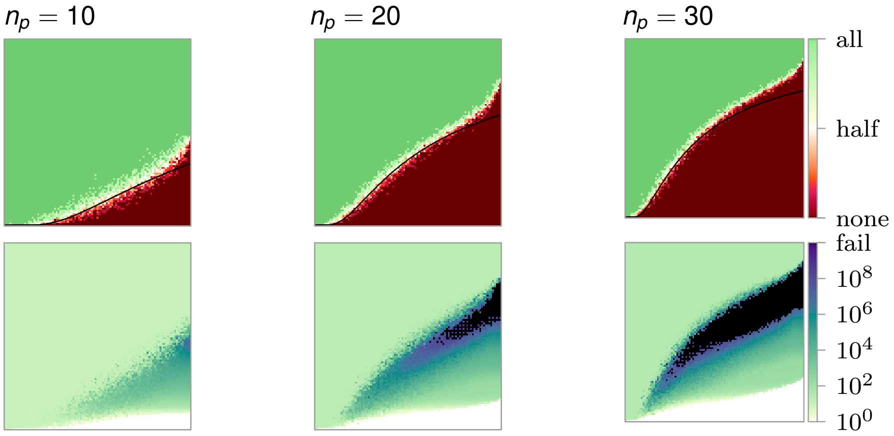
Phase transition vs Search effort





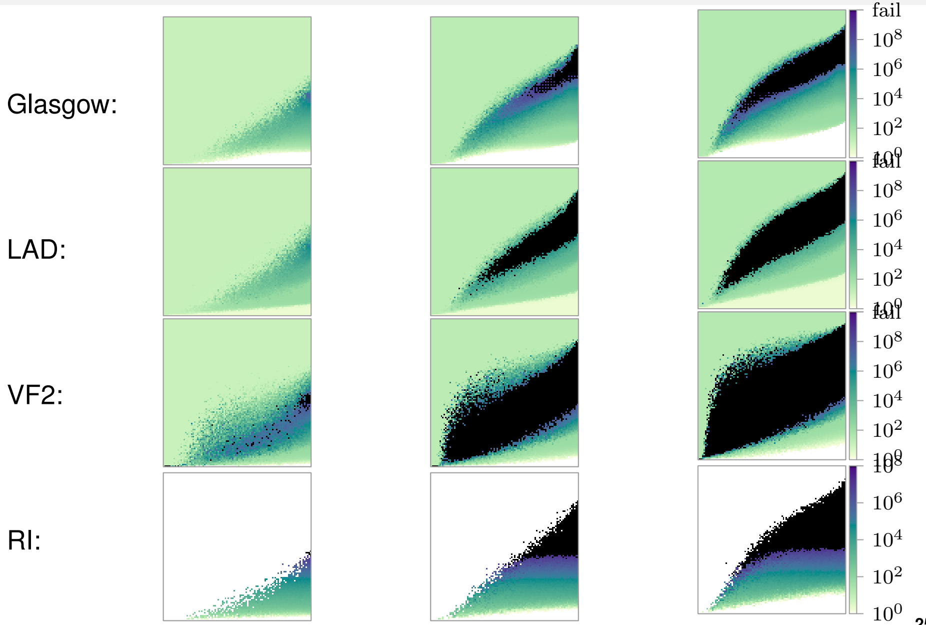
- Black point = Instance not solved by Glasgow within 1000s
- White point = Instance solved by Glasgow without backtracking

Scale-up properties when increasing n_p



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

What about other solvers?



Hardness for Optimisation Problems

$\rightsquigarrow$ **Case of complete/exact approaches**

Most complete approaches solve sequences of decision problems:

1. Search for an assignment *a* which satisfies the set *C* of constraints
   - 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

$\rightsquigarrow$ These two instances are usually the hardest of the sequence

Hardness for Optimisation Problems

$\leadsto$ **Case of incomplete/meta-heuristic approaches**

Heuristic exploration of the search space:

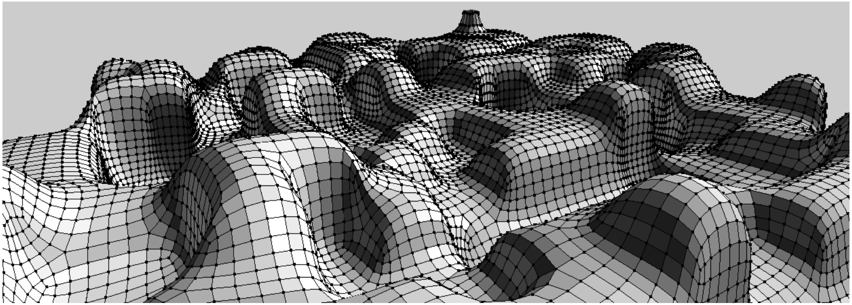
- Use mechanisms to build new solutions from previously visited solutions
- Neighbourhood graph $G = (V, N)$ associated with an incomplete approach:
  - Vertices: V = set of all possible solutions
  - Edges: $N = \{(v_i, v_j) \in V \times V : v_j$ can be built from $v_i\}$
    $\leadsto$ Depends on mechanisms 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 (1/2)

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 (2/2)

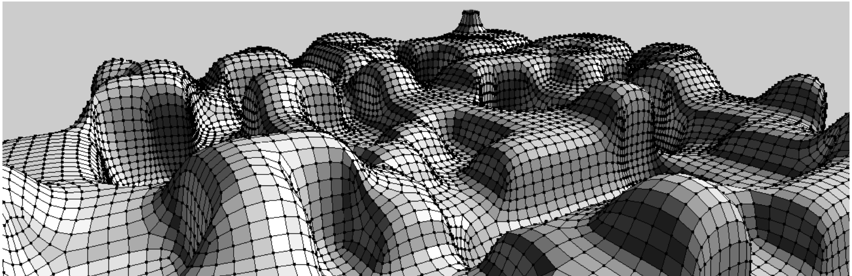
Topological features of a fitness landscape:

- 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
- ...

$\rightsquigarrow$ These features are used to study hardness



Overview of the talk

Performance Criteria

Three most common criteria:

- Duration
- Memory consumption
- Quality

Warning: These criteria are often inter-dependent

- Duration may be reduced by using more data structures
  $\leadsto$ Ex: Maintain values instead of recomputing them from scratch
- Quality may be improved by spending more time
  $\leadsto$ Ex: Anytime solvers

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 (used by Knuth in TAOCP):

$\leadsto$ 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

Exact algorithm that finds the optimal solution a^* and proves optimality

- Performance measure: CPU time, or number of mems/operations
- Question: What if some instances aren't solved within the time limit?

Anytime algorithm that continuously improves the solution

- 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?

Overview of the talk

Data Analysis

Goal of data analysis:

Transform raw data into information

Tools for data analysis:

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

Warning:

Do look at raw Data before starting Data analysis

What are we going to see now?

Data analysis for three different kinds of experimental results:

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

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

- Data analysis for multi-criteria optimisation problems
- Data analysis of parallel algorithms

Overview of the talk

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
- Empirical estimation of its probability distribution by collecting a large number of runs (with different random seeds)

Illustration on the *Car Sequencing Problem*

Question addressed by the experiment: What is the best parameter setting (among 5 given settings) of a non deterministic algorithm$^{(1)}$ for solving instance 26-82?

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

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

$\leadsto$ Measure the number of iterations

Duration limit:

- Instances of NP-hard problems can't always be solved within a reasonable amount of time (unless P=NP...)
- The duration of a run must be limited
  $\leadsto$ In our case: every run is limited to 150000 iterations
- What do we measure when the duration limit is reached?
  $\leadsto$ Maximum number of iterations (150000)
  $\leadsto$ **Warning: This is a lower bound of the actual measure**

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 failures
- IQR not computed in case of failures before Q3

Let's start with some descriptive statistics

Central tendency measures:

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

$\rightsquigarrow$ 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
- Median ranking: 4, 5, 3, 2, 1
- σ not computed in case of failures
- IQR not computed in case of failures 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 failures
- Median: Middle value in the ordered sequence of values

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

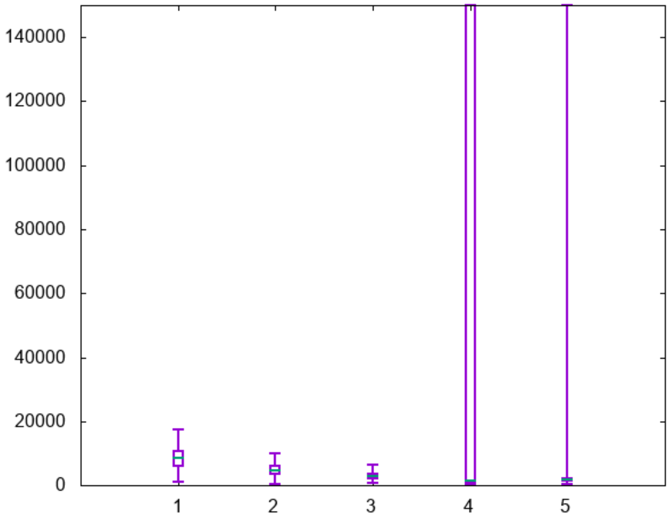
Dispersion measures:

- 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

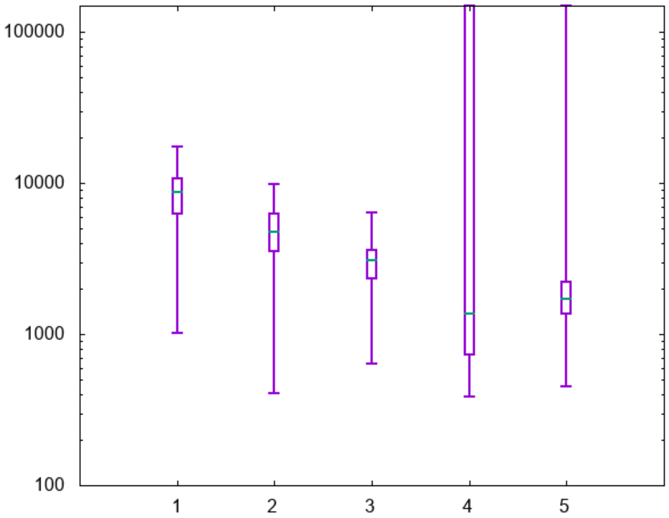
	Mean	Median	σ	IQR
1	8657	8705	3323	4481
2	5082	4743	1813	2725
3	3055	3111	1053	1259
4	$\geq$56205	1378	-	-
5	$\geq$8746	1728	-	863

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

Visualisation of quartiles with Box Plots



Visualisation of quartiles with Box Plots



With a log scale!

Utilisation of a Statistical Test

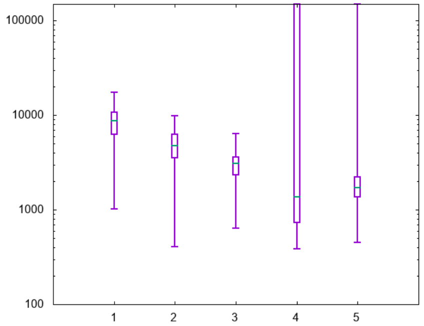
Distributions are not normal

$\rightsquigarrow$ 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)

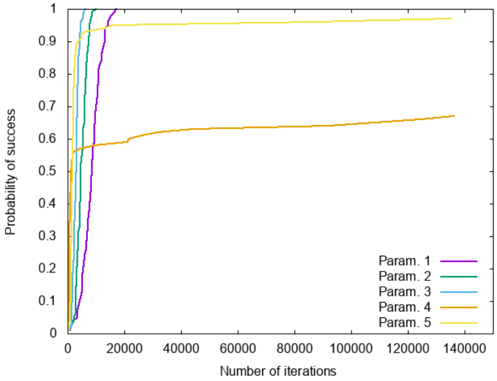
p_i	p_j	U	p-value	
1	2	1690.5	3.1e-16	Reject H_0
1	3	542.5	6.4e-28	Reject H_0
1	4	4307.0	0.045	Reject ?
1	5	974.0	3.9e-23	Reject H_0
2	3	1675.0	2.2e-16	Reject H_0
2	4	4412.0	0.075	
2	5	1164.0	3.5e-21	Reject H_0
3	4	4509.0	0.115	
3	5	2218.0	5.3e-12	Reject H_0
4	5	4826.0	0.335	



Comparison of Cumulative Distribution Functions (CDF)

What is the CDF of a random variable X?

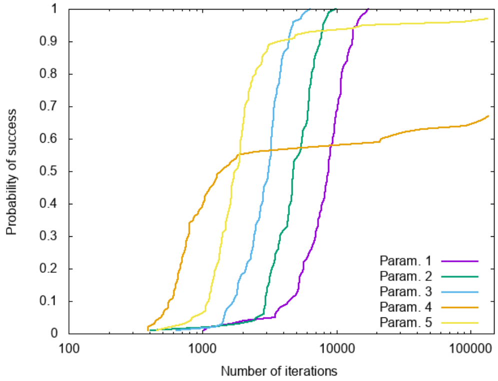
- $F_X(x)$ = Probability that X is smaller than or equal to x
- If X = random variable associated with the number of iterations: $F_X(x)$ = Proba. that the instance is solved in at most x iterations
- Empirical estimation by considering a large number of runs



Comparison of Cumulative Distribution Functions (CDF)

What is the CDF of a random variable X?

- $F_X(x)$ = Probability that X is smaller than or equal to x
- If X = random variable associated with the number of iterations: $F_X(x)$ = Proba. that the instance is solved in at most x iterations
- Empirical estimation by considering a large number of runs

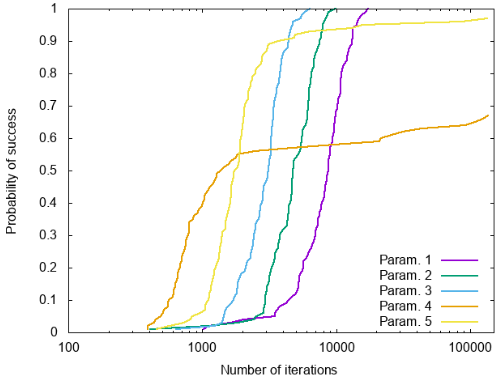


With a logscale...

Comparison of Cumulative Distribution Functions (CDF)

How to compute the CDF for a solver s?

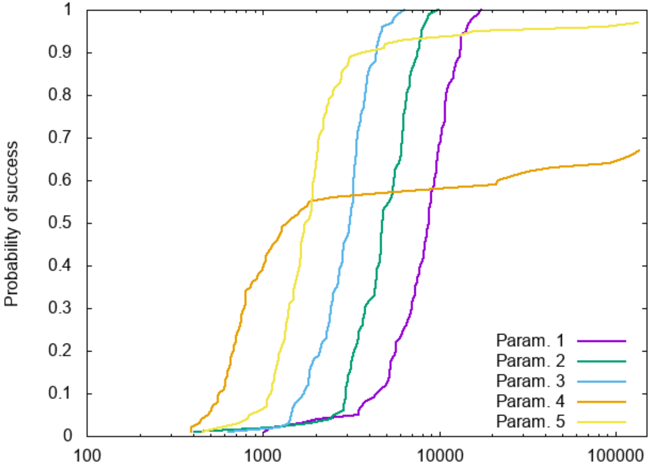
- For each run i of s, let m_i be the measure for this run
- Initialise a counter c to 0
- Sort all measures by increasing order and for each measure m_i: Increase c and plot the point $(m_i, \frac{c}{n})$ where n = total number of runs



With a logscale...

Conclusion of this experiment

- 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 $\rightsquigarrow$ Compromise between time and solution quality



Overview of the talk

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 solution

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

Illustration on the Maximum Clique Problem

Question addressed by the experiment: Given 4 parameter settings of a non deterministic algorithm(1), what is the best setting for three classes of graphs (C, gen, and brock)

(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

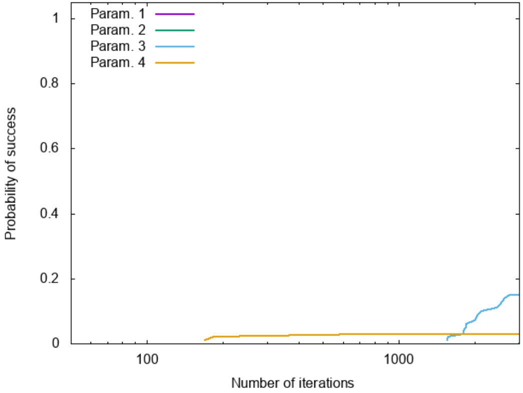
Performance measures:

- Duration measure: Number of iterations
- Quality measure: Size of the clique

Can we study each criterion separately?

$\rightsquigarrow$ **Fix the quality and plot the CDF associated with duration**

Probability of finding a clique of size *k* wrt number of iterations:



Result for instance C500.9 when *k*=57

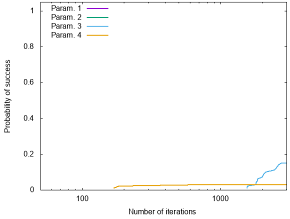
What if an algorithm does not find a clique of size 57?

Can we study each criterion separately?

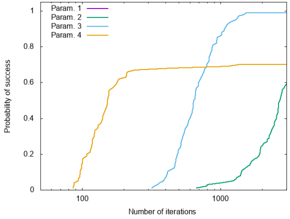
$\leadsto$ **Fix the quality and plot the CDF associated with duration**

CDFs for different values of *k*:

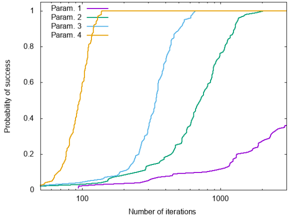
k=57:



k=55:



k=53:

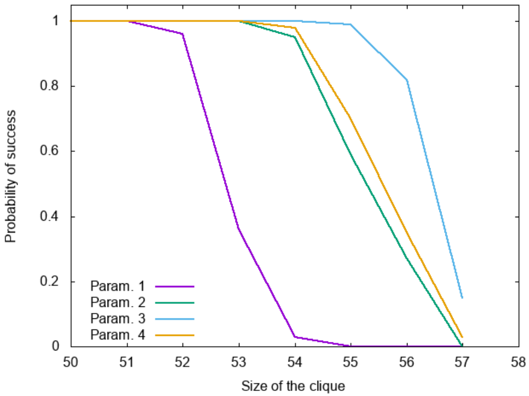


How to choose *k*?

Can we study each criterion separately?

$\rightsquigarrow$ **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:



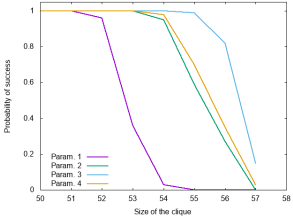
Result for instance C500.9 when *t*=3000

Can we study each criterion separately?

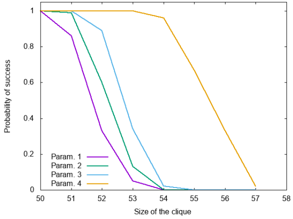
$\rightsquigarrow$ **Fix the number of iterations and plot the CDF associated with quality**

CDFs for different values of t:

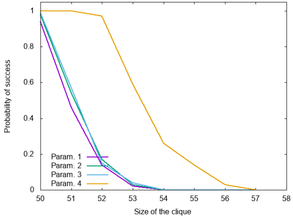
t=3000:



t=1000:



t=100:

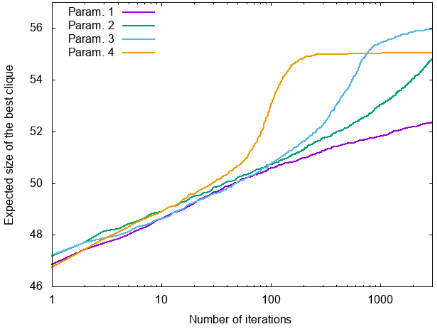


How to choose the number of iterations?

Evolution of Quality with respect to Duration

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

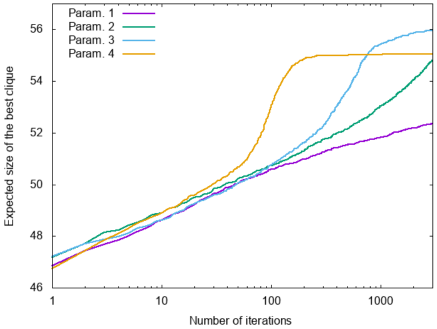
Visualisation for C500.9:



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)?

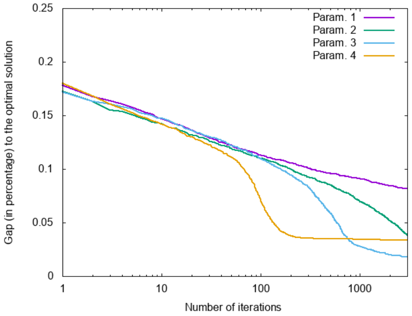
Let's normalise the measure!

Gap to the optimal solution:

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

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

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

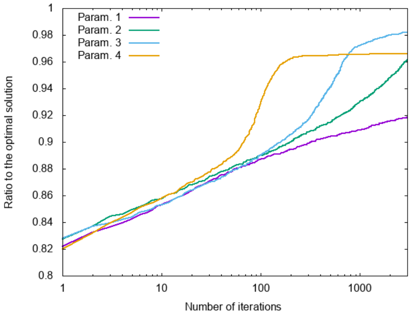


Ratio to the optimal solution:

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

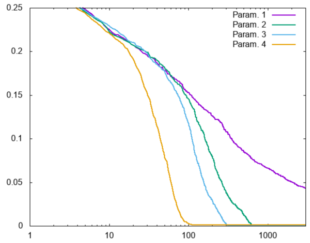
$\rightsquigarrow$ ratio = 1 when $f(s) = f(s^*)$

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

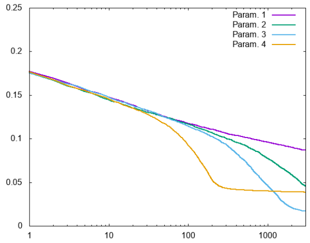


Average Gap for each Class of Graphs

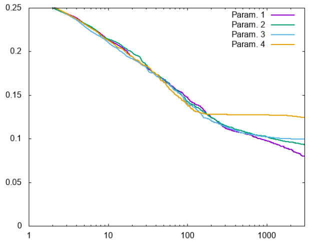
Gen graphs:



C graphs:



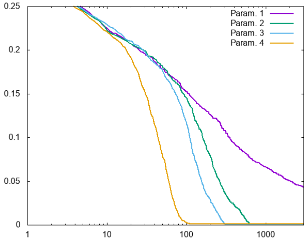
Brock graphs:



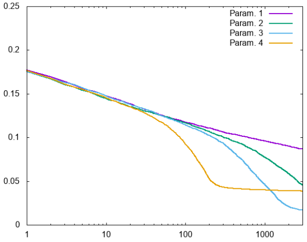
- Can we explain why results are different from a class to another?
  $\rightsquigarrow$ Correlation between clique size and distance to the max clique

Average Gap for each Class of Graphs

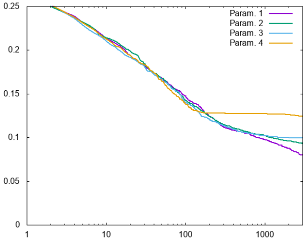
Gen graphs:



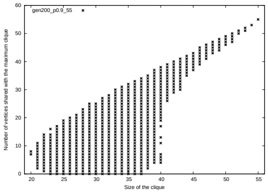
C graphs:

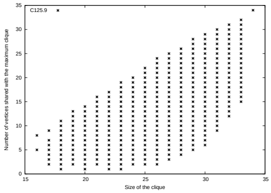


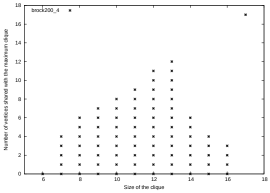
Brock graphs:



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







Conclusion of this experiment

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
  $\rightsquigarrow$ Gap or ratio to the optimal solution

Performance changes from an instance to another

- Analyse performance for each instance separately
  $\rightsquigarrow$ Aggregate results by grouping similar instances
- Use automatic selection and configuration technics
  $\rightsquigarrow$ see point 4 of this lecture

Overview of the talk

Illustration on the Subgraph Isomorphism Problem (SIP)

Question addressed by the experiment:

What is the best solver among VF2(1), LAD(2), Glasgow(3) and RI(4)?

Performance measure:

- CPU time on dual Intel Xeon E5-2695 v4 CPUs and 256GBytes RAM
- 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

References:

(1) L. Cordella, P. Foggia, C. Sansone, M. Vento: *A (sub)graph isomorphism algorithm for matching large graphs*, in PAMI 2004

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

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

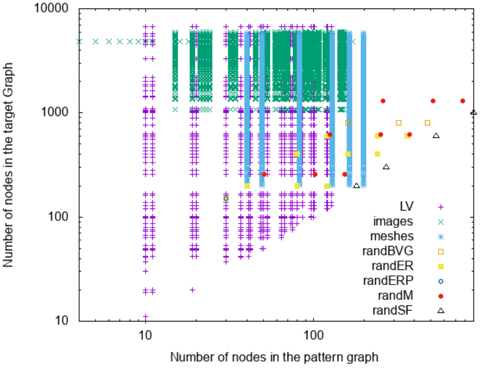
(4) V. Bonnici, R. Giugno: *On the variable ordering in subgraph isomorphism algorithms*, in IEEE/ACM Trans. Comput. Biology Bioinform. 2017

Benchmark Description

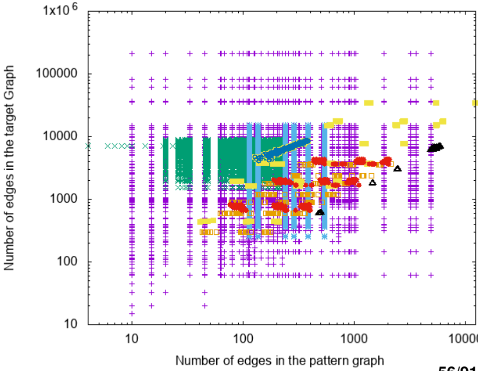
14,621 instances coming from 8 existing benchmarks

- Instances coming from real applications: Images and Meshes
- Random instances: randBVG, randER, randERP, randM, randSF
- Instances generated from a graph database: LV

Number of nodes:



Number of edges:

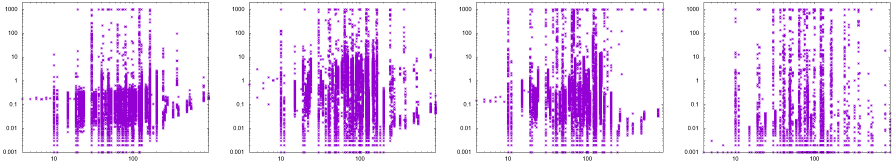


Can we compare scale-up properties?

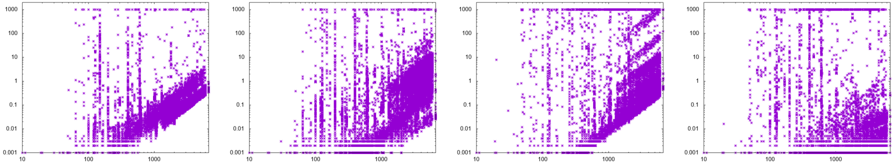
No: plotting the evolution of time wrt graph sizes is meaningless

- Standard deviations are very high
- Some instances are not solved within the CPU time limit

Time (*y*-axis) wrt nb of pattern nodes (*x*-axis) for Glasgow, LAD, VF2, and RI:



Time (*y*-axis) wrt nb of target nodes (*x*-axis) for Glasgow, LAD, VF2, and RI:



What statistic can we compute to answer the question?

	Glasgow	LAD	VF2	RI
# fastest	1 464	2 581	1 245	11 890

1. Number of instances for which a solver is the fastest
2. Number of instances whose solving time is lower than $1,000$s
3. Number of instances whose solving time is lower than or equal to .001s
4. Time (in seconds) to solve an instance
   $\leadsto$ Average on solved instances (different sets depending on the solver)
5. Lower bound of the average solving time on the $14,621$ instances
   $\leadsto$ The time for an unsolved instance is bounded by the time limit

What statistic can we compute to answer the question?

	Glasgow	LAD	VF2	RI
# fastest	1 464	2 581	1 245	11 890
# solved in 1 000s	14 356	14 176	12 528	13 725

1. Number of instances for which a solver is the fastest
2. Number of instances whose solving time is lower than $1,000$s
3. Number of instances whose solving time is lower than or equal to .001s
4. Time (in seconds) to solve an instance
   $\leadsto$ Average on solved instances (different sets depending on the solver)
5. Lower bound of the average solving time on the $14,621$ instances
   $\leadsto$ The time for an unsolved instance is bounded by the time limit

What statistic can we compute to answer the question?

	Glasgow	LAD	VF2	RI
# fastest	1 464	2 581	1 245	11 890
# solved in 1 000s	14 356	14 176	12 528	13 725
# solved in 0.001s	441	1 540	1 102	9 014

1. Number of instances for which a solver is the fastest
2. Number of instances whose solving time is lower than $1,000$s
3. Number of instances whose solving time is lower than or equal to .001s
4. Time (in seconds) to solve an instance
   $\rightsquigarrow$ Average on solved instances (different sets depending on the solver)
5. Lower bound of the average solving time on the $14,621$ instances
   $\rightsquigarrow$ The time for an unsolved instance is bounded by the time limit

What statistic can we compute to answer the question?

	Glasgow	LAD	VF2	RI
# fastest	1 464	2 581	1 245	11 890
# solved in 1 000s	14 356	14 176	12 528	13 725
# solved in 0.001s	441	1 540	1 102	9 014
avg time for solved instances	1.61	4.73	9.41	4.93

1. Number of instances for which a solver is the fastest
2. Number of instances whose solving time is lower than $1,000$s
3. Number of instances whose solving time is lower than or equal to .001s
4. Time (in seconds) to solve an instance
   $\rightsquigarrow$ Average on solved instances (different sets depending on the solver)
5. Lower bound of the average solving time on the $14,621$ instances
   $\rightsquigarrow$ The time for an unsolved instance is bounded by the time limit

What statistic can we compute to answer the question?

	Glasgow	LAD	VF2	RI
# fastest	1 464	2 581	1 245	11 890
# solved in 1 000s	14 356	14 176	12 528	13 725
# solved in 0.001s	441	1 540	1 102	9 014
avg time for solved instances	1.61	4.73	9.41	4.93
bound on the avg solving time	19.71	35.03	151.21	65.91

1. Number of instances for which a solver is the fastest
2. Number of instances whose solving time is lower than $1,000$s
3. Number of instances whose solving time is lower than or equal to .001s
4. Time (in seconds) to solve an instance
   $\leadsto$ Average on solved instances (different sets depending on the solver)
5. Lower bound of the average solving time on the $14,621$ instances
   $\leadsto$ The time for an unsolved instance is bounded by the time limit

What statistic can we compute to answer the question?

	Glasgow	LAD	VF2	RI
# fastest	1 464	2 581	1 245	11 890
# solved in 1 000s	14 356	14 176	12 528	13 725
# solved in 0.001s	441	1 540	1 102	9 014
avg time for solved instances	1.61	4.73	9.41	4.93
bound on the avg solving time	19.71	35.03	151.21	65.91

First conclusions:

- Glasgow is able to solve more instances within a time limit of 1000s
  But RI is able to solve more instances within a time limit of 0.001s
- Glasgow has the smallest average solving time
  But RI is the fastest for a wide majority of instances
- LAD is always the second best solver, for the 5 considered statistics

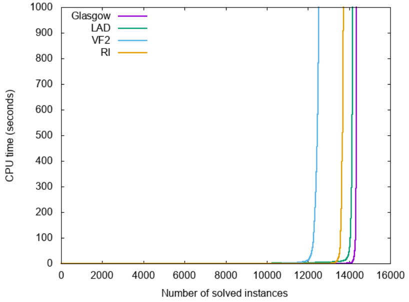
There is no clear winner!
$\leadsto$ Let's visualise Data

Cactus Plot: Number of solved instances wrt time

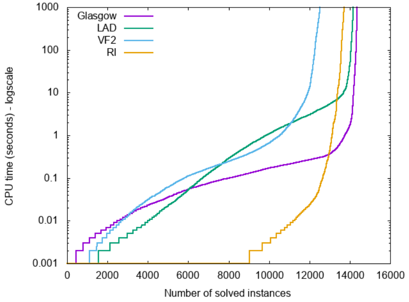
How to produce a cactus plot for a solver *s*?

- 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 (c, t_i)

With a linear scale:



With a logarithmic scale:

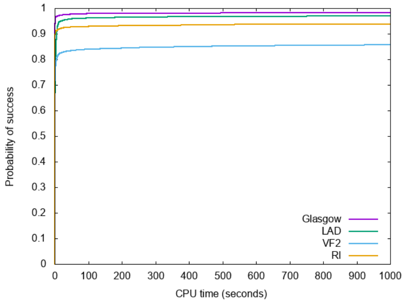


CDF: Probability of success wrt time

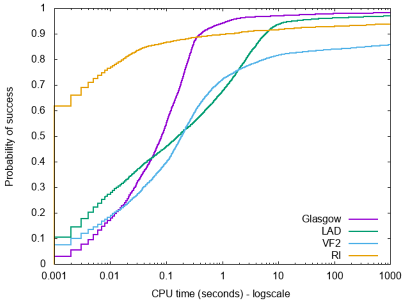
How to obtain a CDF from a cactus plot?

- Divide the number of solved instances by the total number of instances
- Invert the two axis

With a linear scale:



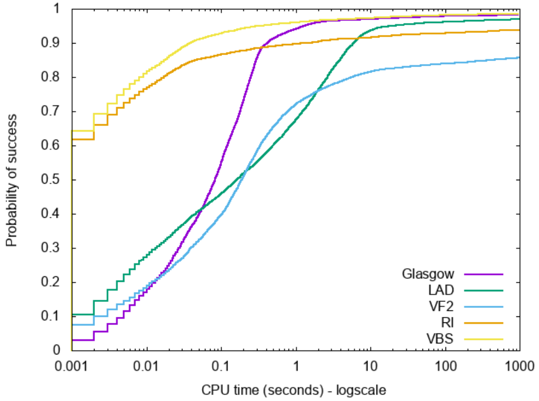
With a logarithmic scale:



Virtual Best Solver (VBS)

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

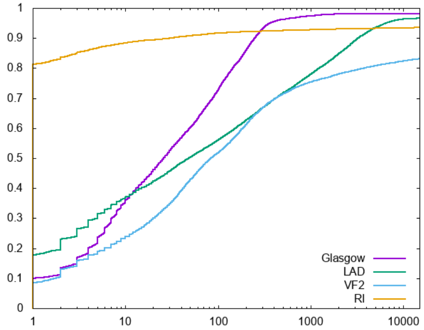
- $\forall s \in \mathcal{S}, \forall i \in \mathcal{I}$, let t_i^s be the time of s on i
- $\forall i \in \mathcal{I}$, time of VBS on s: $t_i^{VBS} = \min_{s \in \mathcal{S}} t_i^s$



Performance profiles

Performance profile of a solver *s*:

- Performance ratio of s on an instance i: $r_i^s = t_i^s / t_i^{VBS}$
  $\leadsto$ If $r_i^s > 1$ then s is r_i^s times as long as VBS
- Performance profile of s = CDF of r^s
  $\leadsto$ Probability that s is within a factor x of VBS

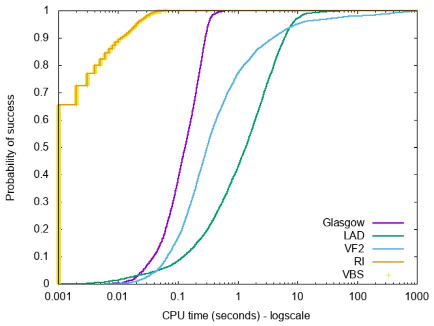


- $t^{RI} = t^{VBS}$ for 81% of the instances
- $t^{RI} \leq 1000 * t^{VBS}$ for 93% of the instances
- $t^{Glasgow} = t^{VBS}$ for 10% of the instances
- $t^{Glasgow} \leq 1000 * t^{VBS}$ for 97% of the inst.

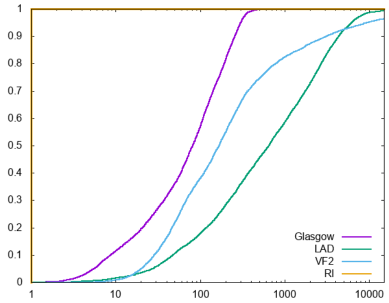
Warning: This is a global picture for an unbalanced benchmark that contains a lot of easy instances and a few very hard instances
$\leadsto$ **Analyse results for each class separately**

Results on the 6302 instances of Class *Images*

CDF:



Performance profile:

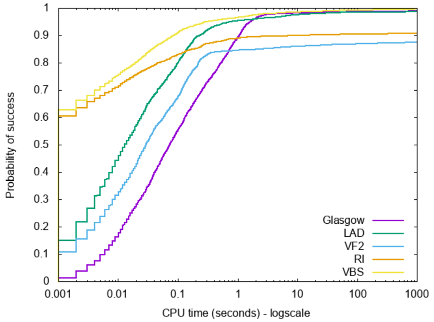


Conclusions:

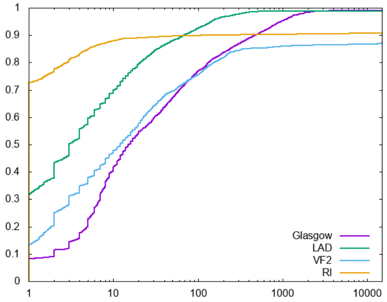
- RI = VBS for all instances of this class
  $\rightsquigarrow$ It never needs more than 0.1s to solve an instance
- Glasgow, LAD and VF2 are also able to solve all instances but they are several orders longer

Results on the 3018 instances of Class *Meshes*

CDF:



Performance profile:

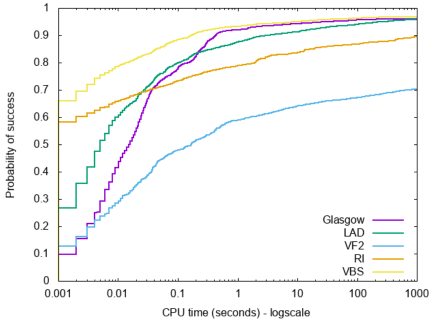


Conclusions:

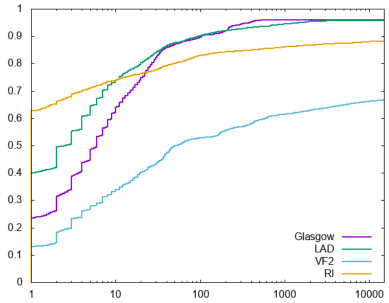
- RI is the most successful when time < 0.1s
- LAD is the most successful when 0.1s $<$ time $<$ 2s
- Glasgow is the most successful when time $>$ 2s

Results on the 3828 instances of Class *LV*

CDF:



Performance profile:

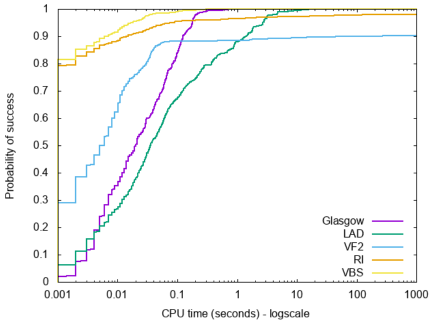


Conclusions:

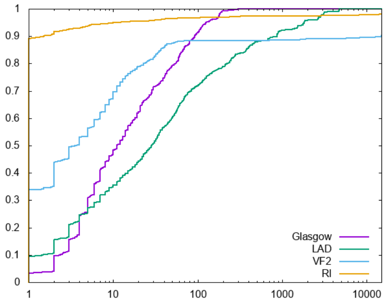
- RI is the most successful when time < 0.03s
- LAD is the most successful when 0.03s $<$ time $<$ 2s
- Glasgow is the most successful when time $>$ 2s

Results on the 1000 instances of *RandBVG*, *RandM*, and *RandSF*

CDF:



Performance profile:

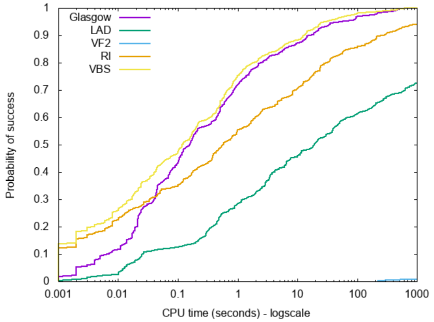


Conclusions:

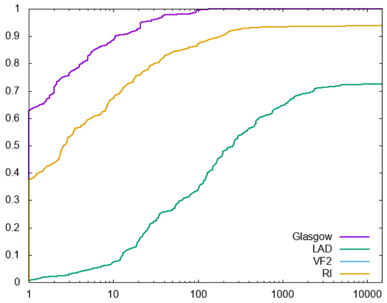
- RI is the most successful when time $<$ 0.15s
- Glasgow is the most successful when time $>$ 0.15s

Results on the 270 instances of Class *RandER*

CDF:



Performance profile:

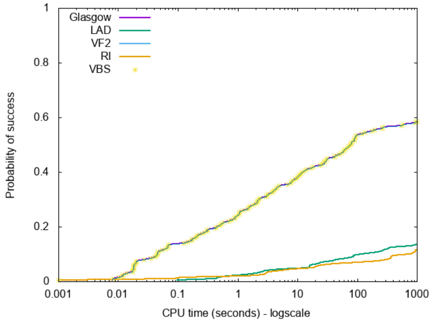


Conclusions:

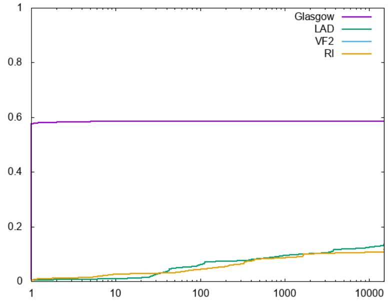
- RI is the most successful when time < 0.02s
- Glasgow is the most successful when time > 0.02s
  $\rightsquigarrow$ It is the only solver able to solve all instances within 1000s
- VF2 solves only 2 instances within 1000s

Results on the 200 instances of Class *RandPhase*

CDF:



Performance profile:

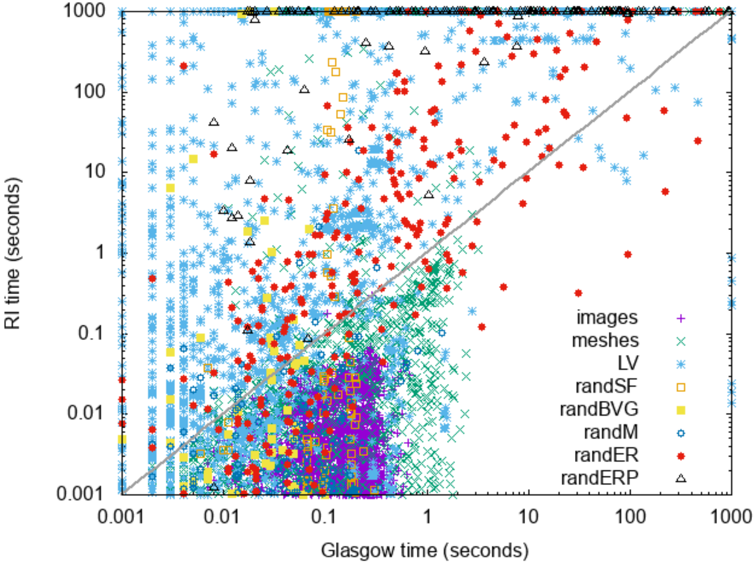


Conclusions:

- Glasgow = VBS for all solved instances of this class
  $\rightsquigarrow$ It solves 59% of the instances within 1000s
- LAD and RI solve less than 15% instances and are several orders longer
- VF2 is not able to solve any instance of this class

Comparison of RI and Glasgow for each instance separately

$\rightsquigarrow$ **Scatter plot: each instance i is a point (x, y) with $x = t_i^{\text{Glasgow}}$ and $y = t_i^{\text{RI}}$**



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, perf. profiles 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

Overview of the talk

Overview of the talk

Parameters and Hyper-Parameters

Parameters = Variables that define thresholds, weights, frequencies, . . .

- A parameter changes the algorithm performance
- Examples:
  - Initial temperature, or Cooling rate for Simulated Annealing
  - Tabu list length for Tabu Search
  - Population size, Cross-over rate, or Mutation rate for GAs

Hyper-parameters = Variables that correspond to design choices

- An hyper-parameter changes the algorithm
- Examples:
  - Bound function for Branch & Bound
  - Neighborhood function for Local Search
  - Filtering algorithm for Constraint Programming

Both param. and hyper-param. are called "Parameters" in what follows

The Vocabulary of Experimentation

Factors = Parameters that are studied in the experiment

$\leadsto$ Identify “important” parameters, and fix the other parameters

Levels = Set of possible values for a factor

- Symbolic factor: 1 level per value
- Numeric factor: Identify intervals of relevant values by sampling
  $\leadsto$ Use a geometric serie to sample: 1, 2, 4, 8, . . . or 1, 10, 100, . . .

Configuration = An assignment of one level to each factor

Design Point = Configuration that must be experimentally evaluated

- Full factorial design = All Factor/Level combinations (grid search)
  - 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 configurations
  $\leadsto$ **How to select configurations that must be evaluated?**

Manual Tuning vs Automatic Configuration

Main drawbacks of manual parameter tuning:

- It is time consuming
- Intuitions may be misleading
- It may not be fair
  $\rightsquigarrow$ The tuning effort may be different from a solver to another
- The tuning protocol is not reproducible

Programming by Optimisation [Hoos 2012]:

Developers specify a potentially large design space of programs that accomplish a given task, from which versions of the program optimised for various use contexts are generated automatically.

[Hoos 2012]: Communications of the ACM 55(2), pp. 70-80, February 2012

Automatic Configuration

Definition of the problem:

Given:

- A set of configurations Θ of an algorithm A
- A distribution $\mathcal{D}$ over the set of instances $\mathcal{I}$ of the problem solved by A
- A performance measure $m : \Theta \times \mathcal{I} \to \mathbb{R}$

Search for $\theta^* \in \Theta$ which optimises the expectation of $m(\theta^*, i)$ when $i \sim \mathcal{D}$

How to define the distribution $\mathcal{D}$?

- $\mathcal{D}$ should be representative of the actual instances that must be solved
  $\rightsquigarrow$ Gather a set $\mathcal{S}$ of representative instances

How to obtain training instances from $\mathcal{S}$?

- Solution 1: Design a model for randomly generating instances that have the same distribution as $\mathcal{S}$
- Solution 2: Use $\mathcal{S}$ as a finite support definition of $\mathcal{D}$
  $\rightsquigarrow$ Split $\mathcal{S}$ into training and test sets for cross-validation

Example of Automatic Configuration Tool

$\rightsquigarrow$ **Sequential Model-based Algorithm Configuration (SMAC)**

Basic Idea:

- Perform an initial set $\mathcal{R}$ of runs and select a first configuration θ^*
- Iterate the following steps:
  - Use $\mathcal{R}$ to build a model for predicting configuration performances
  - Use that model to select promising configurations
  - For each selected configuration θ:
    - Compare θ with θ^* using Random Online Agressive Racing (ROAR)
    - Update θ^* if θ wins the race, and update the set $\mathcal{R}$ of runs

Reference:

F. Hutter, H. Hoos, K. Leyton-Brown (2011): *Sequential Model-Based Optimization for General Algorithm Configuration*. LION

Source code available at http://www.cs.ubc.ca/labs/beta/Projects/SMAC/

Some other Automatic Configuration Tools

ParamILS: Greedy Local Search with Restarts

F. Hutter, H. Hoos, K. Leyton-Brown, T. Stützle (2009): *ParamILS: An Automatic Algorithm Configuration Framework*. JAIR

Source code available at http://www.cs.ubc.ca/labs/beta/Projects/ParamILS/

Iterated F-race: Iteratively sample configurations to race

M. Lopez-Ibanez, J. Dubois-Lacoste, L. Perez Caceres, M. Birattari, T. Stuetzle (2016): *The irace package: Iterated racing for automatic algorithm configuration*. Operations Research Perspectives

Available as a R package

Overview of the talk

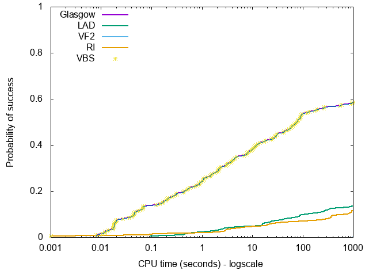
From Configuration to Selection

Automatic configuration finds the Single Best Solver (SBS)...

...But SBS may be far from VBS when instances are heterogeneous

Illustration on the subgraph isomorphism problem

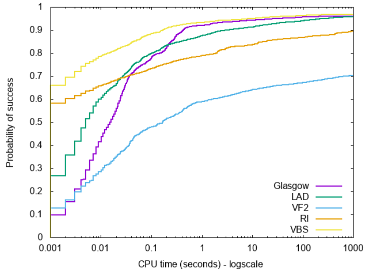
CDF for RandPhase instances:



- SBS = VBS = Glasgow

$\rightsquigarrow$ No need for automatic selection

CDF for LV instances:



- SBS depends on time limit
- VBS outperforms SBSs

$\rightsquigarrow$ Use automatic selection!

Per Instance Algorithm Selection

Definition of the problem:

Given a portfolio $\mathcal{P}$ of algorithms (or of algorithm configurations) and an instance i, select an algorithm $A \in \mathcal{P}$ expected to perform best on i

Offline training:

Given:

- A distribution $\mathcal{D}$ over the set of instances $\mathcal{I}$
- A performance measure $m : \mathcal{P} \times \mathcal{I} \to \mathbb{R}$
- An embedding function $f : \mathcal{I} \to \mathcal{F}$ where $\mathcal{F} \subseteq \mathbb{R}^m$ is the feature space
  $\leadsto$ Each instance $i \in \mathcal{I}$ is described by $f(i) \in \mathcal{F}$

Build a selector $S : \mathcal{F} \to \mathcal{P}$ which optimises $m(S(f(i)), i)$ when $i \sim \mathcal{D}$

Online selection of an algorithm for an instance $i \in \mathcal{I}$

Return $S(f(i))$

Examples of existing Automatic Selection Approaches

SATzilla 2009:

- Offline: Learn a model for each algorithm
  $\leadsto$ Prediction of performance given instance features
- Online selection of an algorithm to solve a new instance *i*:
  $\leadsto$ Predict performance for each algorithm
  $\leadsto$ Select the algorithm with the best predicted performance

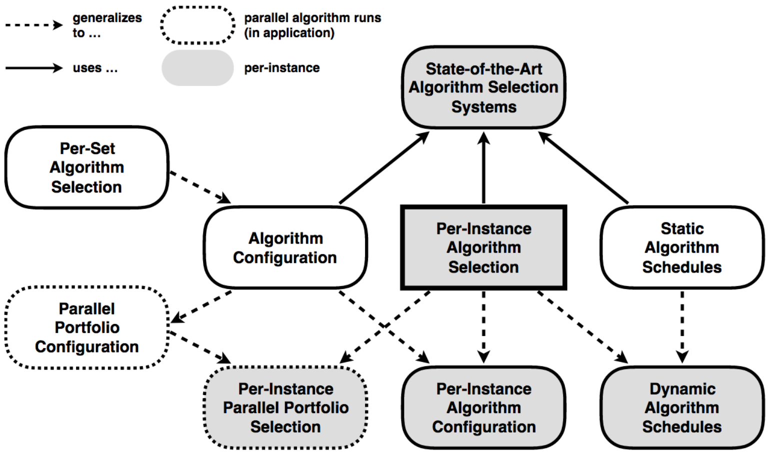
See L. Xu, F. Hutter, H. H. Hoos, K. Leyton-Brown (2009): SATzilla2009: an Automatic Algorithm Portfolio for SAT . SAT Competition 2009

ISAC [Kadioglu et al. 2010]:

- Offline: Partition instances into homogeneous clusters and use automatic configuration to determine the best algorithm for each cluster
- Online selection of an algorithm to solve a new instance *i*:
  $\leadsto$ Search for the cluster of *i* and select the corresponding algorithm

See S Kadioglu, Y Malitsky, M Sellmann, K Tierney (2010): ISAC-Instance-Specific Algorithm Configuration. ECAI

Related Problems

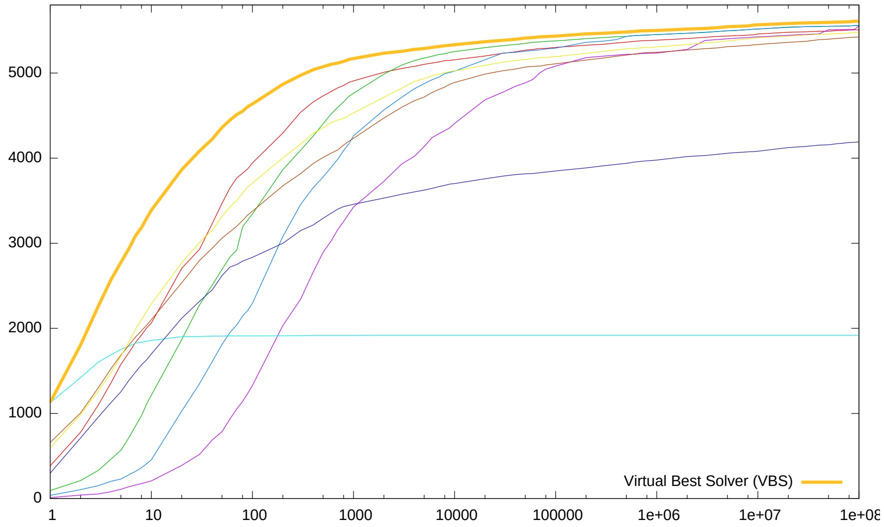

generalizes to ...
uses ...
parallel algorithm runs (in application)
per-instance
State-of-the-Art Algorithm Selection Systems
Per-Set Algorithm Selection
Algorithm Configuration
Per-Instance Algorithm Selection
Static Algorithm Schedules
Parallel Portfolio Configuration
Per-Instance Parallel Portfolio Selection
Per-Instance Algorithm Configuration
Dynamic Algorithm Schedules

Reference:

P. Kerschke, H. Hoos, F. Neumann, H. Trautmann (2019): *Automated Algorithm Selection: Survey and Perspectives*. ECJ

Illustration: Algorithm Selection for Subgraph Isomorphism

⤳ **CDF of 8 algorithms + VBS**



Overview of the process

Offline:

- Describe each training instance by a feature vector
- Train a model that predicts the best algorithm for each training instance

Online: Solve a new instance $i \in \mathcal{I}$

- Sequentially run 2 very fast and complementary algorithms
  $\leadsto$ Solve very easy instances
  $\leadsto$ Collect dynamic features for instances that are not solved
- If instance not solved:
  - Extract features from i
  - Ask the model to select an algorithm given the features
  - Run the algorithm

Feature extraction

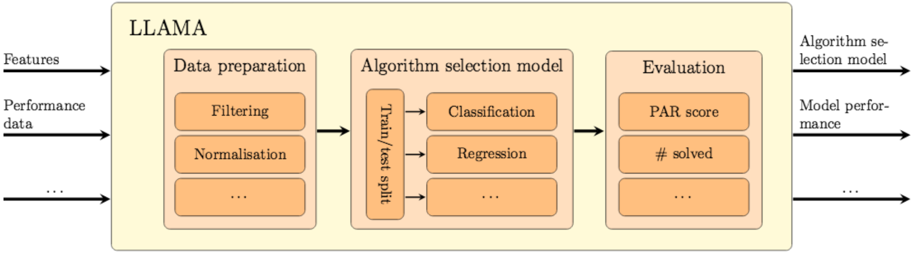
Static features extracted from the graphs

- Number of vertices and edges
- Density
- Number of loops
- Mean and max. degrees
- Mean and max. distance between all pairs of vertices
- Proportion of vertex pairs which are at least 2, 3 and 4 apart
- Binary features: Regular? Connected?

Dynamic features collected when running the 2 algorithms

- Number of value removals
- Percentage (average, min and max) of removed values per variable
- Algorithm solving time

Selection model: LLAMA [Khotthoff 2013]



- R package for designing algorithm selectors
- Includes different models
  $\leadsto$ Best results: Pairwise regression approach with random forest regression
  - For each pair of algorithm, train a model to predict performance difference
  - Choose algorithm with highest cumulative performance difference

Experimental evaluation (1/2)

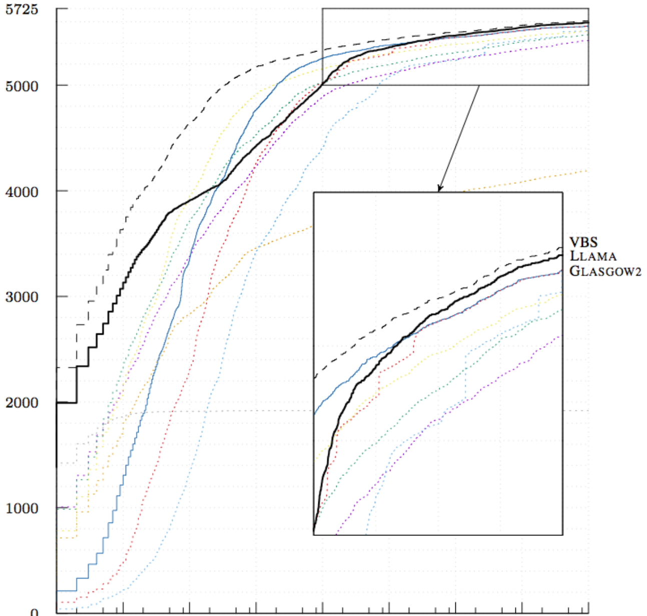
Experimental setup:

- 10-fold cross-validation
- Performance measures:
  - MCP: MisClassification Penalty
    $\rightsquigarrow$ Additional time required to solve an instance wrt VBS
  - # solved = number of instances that are solved
  - Time: time required to solve the instance, or 10^8 if not solved
    $\rightsquigarrow$ Lower bound of the actual time

Results:

Model	Mean MCP	# solved	Mean time
VBS	0	5,608	2,375,913
LLAMA	287,704	5,592	2,664,293
SBS	798,660	5,562	3,174,573

Experimental evaluation (2/2)



Overview of the talk

Conclusion

It's quite easy to obtain huge amounts of experimental Data

$\rightsquigarrow$ Much easier than in other sciences such as biology, for example

But do not kill machines!

$\rightsquigarrow$ Carefully prepare your experiment before running tests

It's more difficult to extract knowledge from experimental Data

$\rightsquigarrow$ Use appropriate Data analysis tools

Use automatic configuration to tune parameters...

$\rightsquigarrow$ Fair and reproducible experimental process

...And algorithm selection to exploit algorithm complementarity!