The simpler the better:

Thinning out MIP's by Occam's razor

Matteo Fischetti, University of Padova

Occam's razor

- **Occam's razor**, or law of parsimony (lex parsimoniae): a problem-solving principle devised by the English philosopher William of Ockham (1287–1347).

- Among competing hypotheses, the one with the fewest assumptions is more likely be true and should be preferred—the fewer assumptions that are made, the better.

- Used as a heuristic guide in the development of theoretical models (Albert Einstein, Max Planck, Werner Heisenberg, etc.)

- Not to misinterpreted and used as an excuse to address oversimplified models: "Everything should be kept as simple as possible, but no simpler" (Albert Einstein)

Overfitting and Integer Programming

- Complicated models/algorithms tend to involve many **parameters**
- **Overmodelling**: too many param.s → overfitting
- A case study:

  Support Vector Machine training by Mixed-Integer Programming

- Fuller details in:

  M. Fischetti, "Fast training of Support Vector Machines with Gaussian kernel", to appear in *Discrete Optimization*, 2015**.**

SVM training

- Input: a **training set** of points

$$(x_1, y_1), \cdots, (x_p, y_p) \text{ with } x_i \in \mathbb{R}^n \text{ and } y_i \in \{-1, +1\}$$

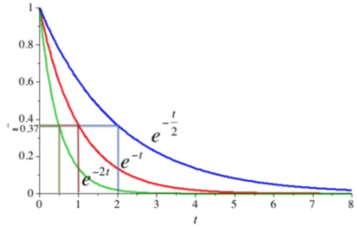
- For a generic point $x \in \mathbb{R}^n$ we want to estimate its unknown **classification** $y_x \in \{-1, +1\}$ through a function of the type

$$y(x) := sign(\sum_{i=1}^{p} \alpha_i y_i K(x, x_i) + \beta_0)$$

where $K(x, x_i)$ is a **kernel** scalar function that measures the "similarity" between x and x_i , and $\alpha_1, \cdots, \alpha_p \geq 0$ and β_0 are parameters that one can **tune** using the training set.

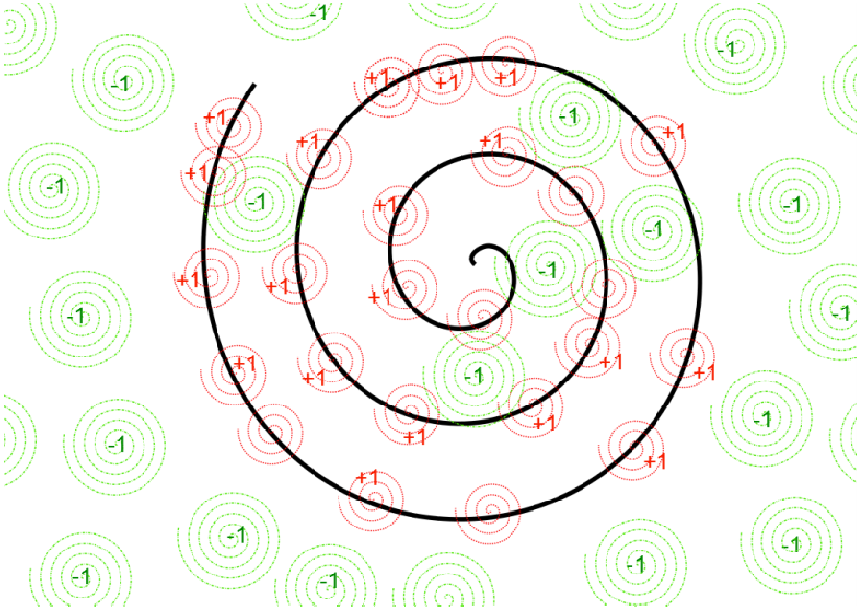
Gaussian kernel and its interpretation

- **Gaussian** kernel depending on parameter $\gamma > 0$



$$K(x, x_i) := e^{-\gamma \|x - x_i\|^2}$$

- **Telecommunication** interpretation of $y(x) := sign(\sum_{i=1}^{p} \alpha_i y_i K(x, x_i) + \beta_0)$



- Every training point x_i broadcasts its +1/-1 with power α_i
- Signal decays with distance *d* as $e^{-\gamma d^2}$
- Receiver seating in *x* measures total signal

$$\sum_{i=1}^{p} \alpha_i y_i e^{-\gamma \|x - x_i\|^2}$$

compares it with threshold $-\beta_0$ and decides between +1 (total signal larger than threshold) and -1

How to decide the SVM parameters?

- Parameters $\alpha_1, \cdots, \alpha_p \geq 0$, β_0 and $\gamma > 0$ to be determined in a preliminary **training phase** using the training set only

- Parameters are viewed as **variables** of an optimization model

- SVM classical model for a fixed kernel (i.e. for a given $\gamma > 0$)

$$
\begin{aligned}
\text{(HINGE)} \quad \min_{\alpha,\beta_0,\xi} \ & \tfrac{1}{2} \textstyle\sum_{i=1}^{p} \sum_{j=1}^{p} \alpha_i \alpha_j y_i y_j K(x_i, x_j) + C \sum_{j=1}^{p} \xi_j \\
& y_j \left(\textstyle\sum_{i=1}^{p} \alpha_i y_i K(x_j, x_i) + \beta_0\right) \geq 1 - \xi_j \quad \forall j = 1, \cdots, p \\
& \alpha_j \geq 0, \ \forall j = 1, \cdots, p \\
& \xi_j \geq 0, \ \forall j = 1, \cdots, p
\end{aligned}
$$

- Parameters $\gamma > 0$ and C determined in an outer loop (*k*-fold validation), **they are not part of the HINGE optimization!**

MIPing SVM training

- Why not using a Mixed-Integer Linear Programming (MILP) model like

$$
\begin{array}{ll}
\text{(NAIVE)} & \min_{\alpha,\beta_0,z} \ \frac{1}{p} \ \sum_{j=1}^{p} z_j \\
& y_j(\sum_{i=1}^{p} \alpha_i y_i K(x_j, x_i) + \beta_0) \geq \epsilon - M z_j \quad \forall j = 1, \cdots, p \\
& 0 \leq \alpha_i \leq 1, \quad \forall i = 1, \cdots, p \\
& z_i \in \{0, 1\}, \ \forall i = 1, \cdots, p
\end{array}
$$

or its “**leave-one-out**” improved version

$$
\begin{array}{ll}
\text{(LOO\_MILP)} & \min_{\alpha,\beta_0,z} \ \frac{1}{p} \ \sum_{j=1}^{p} z_j \\
& y_j(\sum_{i:i\neq j} \alpha_i y_i K(x_j, x_i) + \beta_0) \geq \epsilon - M z_j \quad \forall j = 1, \cdots, p \\
& 0 \leq \alpha_i \leq 1, \ \forall i = 1, \cdots, p \\
& z_i \in \{0, 1\}, \ \forall i = 1, \cdots, p
\end{array}
$$

whose parameters are determined by minimizing the number of **misclassified points** in the training set?

(Un)surprising results

- Results on standard benchmark datasets
- **real**: “true” %**misclassification** on a separate test set
- **estim**: %misclassification on the training set
- **t.**: computing times in CPU sec.s (CPLEX 12.5)
- HINGE with 5-fold validation

	HINGE %miscl.		LOO_MILP %miscl. (p+2 freedom deg.s)		
	real	t.	estim	real	t.
Australian	18.2	728.3	2.5	22.7	435.1
Breast	3.2	627.5	0.0	6.2	41.4
Bupa	32.7	88.2	4.4	39.1	429.6
German	28.0	2453.2	19.8	29.2	462.8
Heart	23.0	47.9	1.3	23.2	141.9
Ionosphere	6.3	123.5	0.0	6.4	5.4
Pima	25.7	1231.0	11.8	28.2	450.4
Sonar	18.0	25.2	0.0	15.6	6.3
Wdbc	4.5	646.4	0.0	4.8	8.7
Wpbc	23.7	29.0	2.1	32.7	184.5
Average	**18.3**	**600.0**	**4.2**	**20.8**	**216.6**

* HINGE could be solved much faster using specialized codes

Keep it simple!

- How can we cure the huge **overfitting** of the MILP model?
- Shall we introduce a **normalization** (convex) term in the objective function, or **add** variables to the model, or go to **larger** kernel space, or what?
- Why not just **simplify** the MILP model instead? **#OccamRazor**
- Overfitting ← too many parameters (*p*+*2*): let's reduce them!
- Options **LOO_k** with just *k* degrees of freedom (including γ)
  - LOO_1: add constraint $\alpha_1 = \alpha_2 = \cdots = \alpha_p = 1 \;\text{ and }\; \beta_0 = 0$
  - LOO_2: add constraint $\alpha_1 = \alpha_2 = \cdots = \alpha_p = 1, \;\; \beta_0 \;\text{ free}$
  - LOO_3: add constraint $\alpha_i = \begin{cases} \alpha^+ \geq 0, & \forall i : y_i = +1 \\ \alpha^- \geq 0, & \forall i : y_i = -1 \end{cases}, \;\; \beta_0 \;\text{ free}$

Simpler, faster and better

#That's OccamBaby

	HINGE %miscl.		LOO_MILP %miscl. (p+2 freedom deg.s)			LOO_1 %miscl. (1 freedom deg.)			LOO_2 %miscl. (2 freedom deg.s)			LOO_3 %miscl. (3 freedom deg.s)		
	real	t.	estim	real	t.	estim	real	t.	estim	real	t.	estim	real	t.
Australian	18.2	728.3	2.5	22.7	435.1	16.0	16.3	0.4	14.7	15.3	0.8	14.6	15.7	10.6
Breast	3.2	627.5	0.0	6.2	41.4	3.2	3.6	0.4	3.0	4.0	0.8	2.4	3.5	9.8
Bupa	32.7	88.2	4.4	39.1	429.6	36.7	38.7	0.0	34.0	39.0	0.1	33.2	39.6	1.7
German	28.0	2453.2	19.8	29.2	462.8	27.2	26.5	1.0	26.6	26.5	2.2	24.1	25.7	22.9
Heart	23.0	47.9	1.3	23.2	141.9	17.9	19.1	0.0	17.1	19.1	0.1	16.5	18.8	1.0
Ionosphere	6.3	123.5	0.0	6.4	5.4	19.7	23.4	0.0	4.9	6.7	0.1	4.2	6.3	1.8
Pima	25.7	1231.0	11.8	28.2	450.4	25.2	25.6	0.6	24.4	25.8	1.2	23.4	25.7	14.8
Sonar	18.0	25.2	0.0	15.6	6.3	16.5	17.0	0.0	15.9	17.9	0.0	12.9	11.4	0.6
Wdbc	4.5	646.4	0.0	4.8	8.7	5.2	5.7	0.2	4.7	5.3	0.5	3.6	4.6	6.9
Wpbc	23.7	29.0	2.1	32.7	184.5	22.7	21.6	0.0	22.3	22.7	0.0	21.0	23.1	0.6
Average	**18.3**	**600.0**	**4.2**	**20.8**	**216.6**	**19.0**	**19.7**	**0.3**	**16.8**	**18.2**	**0.6**	**15.6**	**17.5**	**7.1**

- **LOO_1**: no optimization at all required (besides γ by an external bisection method): better than the too sophisticated LOO_MILP!!
- **LOO_2**: add sorting to determine β_0 (very fast, already comparable or better than HINGE)
- **LOO_3**: add enumeration of 10 values for α^+ in range [0,1]: best classifier on this (limited) data set

(Over)fitting

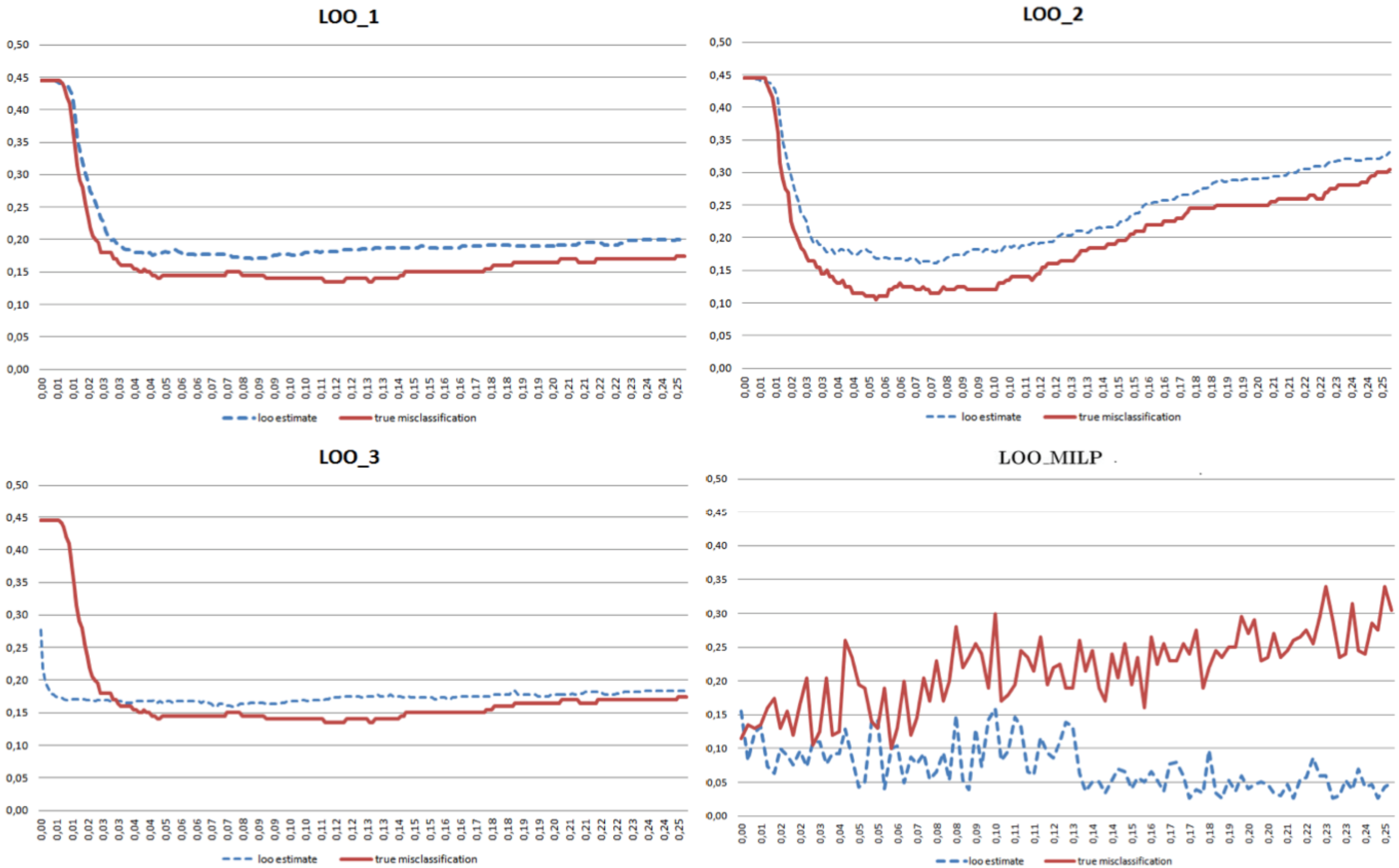


Figure 2: Optimal model values ("loo estimate", in dashed blue) and "true misclassification" rate on the test set (in red) as a function of γ, for instance `Australian`.

Leave one out!

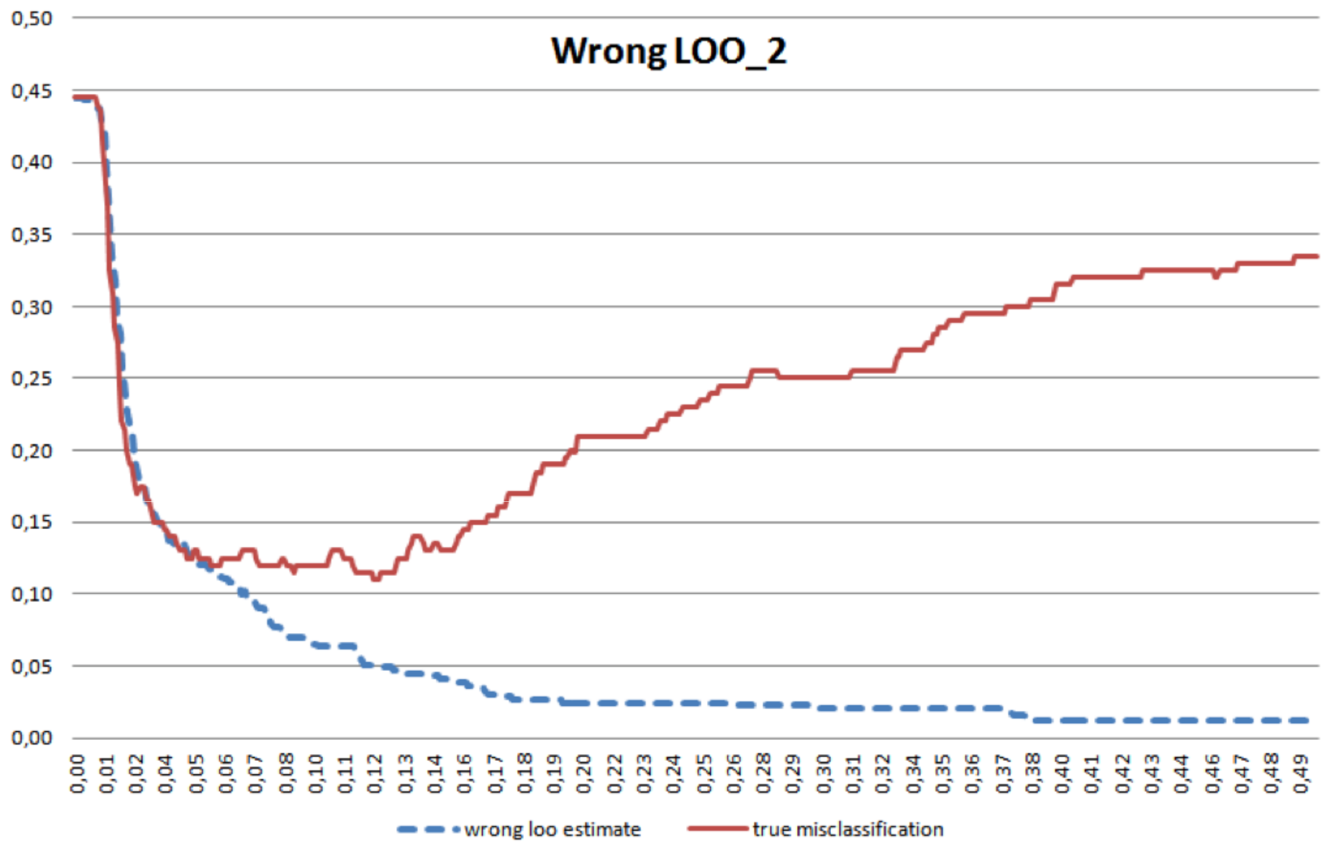


Figure 3: Wrong LOO_2 optimal value with total instead of net signal (dashed blue) and "true" misclassification rate on the test set (red) as a function of γ, for instance `Australian`.

Thinning out MIP models

- The practical difficulty in solving hard problems sometimes comes for **overmodelling**:

  Too many vars.s and constr.s just

  stifle your model

  (and the cure is <u>not</u> to complicate it even more!)

Let your model breathe!

Example 1: QAP

- Quadratic Assignment Problem (QAP): extremely hard to solve
- Unsolved **esc*** instances from QAPLIB (attempted on constellations of thousand computers around the world for many CPU years)
- The thin out approach: *esc* instances are
  1. very symmetrical → find a cure and **simplify** the model through ***Orbital Shrinking*** to actually **reduce** the size of the instances
  2. very large → use **slim MILP** models with high node throughput
  3. decomposable → solve pieces **separately**
- **Outcome**:

  a. all esc* but two instances solved in **minutes** on a notebook

  b. **esc128** (by far the largest ever attempted) solved in just **seconds**

M. Fischetti, M. Monaci, D. Salvagnin, "Three ideas for the Quadratic Assignment Problem", Operations Research 60 (4), 954-964, 2012.

M. Fischetti, L. Liberti, "Orbital shrinking", Lecture Notes in Computer Science, Vol. 7422, 48-58, 2012.

Example 2: Steiner Trees

- Recent **DIMACS 11 (2014)** challenge on Steiner Tree: various versions and categories (exact/heuristic/parallel/…) and scores (avg/formula 1/ …)
- Many very hard (unsolved) instances available on STEINLIB
- Standard MILP models use *x* var.s (arcs) and *y* var.s (nodes)
- **Observation**: many hard instances have uniform arc costs
- **Thin out**: remove *x* var.s and work on the *y*-space (Benders' projection)
- Heuristics based on the **blur** principle: initially forget about details…
- **Outcome:**
  - Some open instances solved in a few **seconds**
  - Our codes (StayNerd, MozartBalls) won most DIMACS categories

M. Fischetti, M. Leitner, I. Ljubic, M. Luipersbeck, M. Monaci, M. Resch, D. Salvagnin, M. Sinnl, "Thinning out Steiner trees: a node-based model for uniform edge costs", Tech.Rep., 2014

Example 3: Facility Location

- **Uncapacitated facility location** with linear (UFL) and quadratic (qUFL) costs
- **Huge** MILP models involving y var.s (selection) and x var.s (assignment)
- **Thin out**: x var.s **suffocate** the model, just remove them..
- A perfect fit with **Benders decomposition**, but … not sexy nowadays as more complicated schemes are preferred **#paperability?**
- **Outcome**:
  - Many hard UFL instances solved very quickly
  - Seven open instances solved to optimality, 22 best-known improved
  - Speedup of 4 orders of magnitude for qUFL up to size 150x150
  - Solved qUFL instances up to 2,000x10,000 in 5 min.s **(MIQCP's with 20M SOC constraints and 40M var.s)**

M. Fischetti, I. Ljubic, M. Sinnl, "Thinning out facilities: a Benders decomposition approach for the uncapacitated facility location problem with separable convex costs", TR 2015.

Thin out your favorite model

call Benders toll free

Benders decomposition well known

… but not so many MIPeople actually use it

… besides Stochastic Programming guys of course

Create account Log in

Article Talk Read Edit View history

Benders' decomposition

From Wikipedia, the free encyclopedia

Benders' decomposition (or **Benders's decomposition**) is a technique in mathematical programming that allows the solution of very large linear programming problems that have a special block structure. This block structure often occurs in applications such as stochastic programming. The technique is named after Jacques F. Benders.

As it progresses towards a solution, Benders' decomposition adds new *constraints*, so the approach is called "*row* generation". In contrast, Dantzig–Wolfe decomposition uses "*column* generation".

See also [edit]

- FortSP solver uses Benders' decomposition for solving stochastic programming problems

References [edit]

- Benders, J. F. (Sept. 1962), "Partitioning procedures for solving mixed-variables programming problems", *Numerische Mathematik* 4(3): 238–252.
- Lasdon, Leon S. (2002), *Optimization Theory for Large Systems* (reprint of the 1970 Macmillan ed.), Mineola, New York: Dover Publications, pp. xiii+523, MR 1888251.

Categories: Mathematical optimization | Decomposition methods

This page was last modified on 19 May 2015, at 16:43.

Text is available under the Creative Commons Attribution-ShareAlike License; additional terms may apply. By using this site, you agree to the Terms of Use and Privacy Policy. Wikipedia® is a registered trademark of the Wikimedia Foundation, Inc., a non-profit organization.

Privacy policy About Wikipedia Disclaimers Contact Wikipedia Developers Mobile view

WIKIPEDIA The Free Encyclopedia

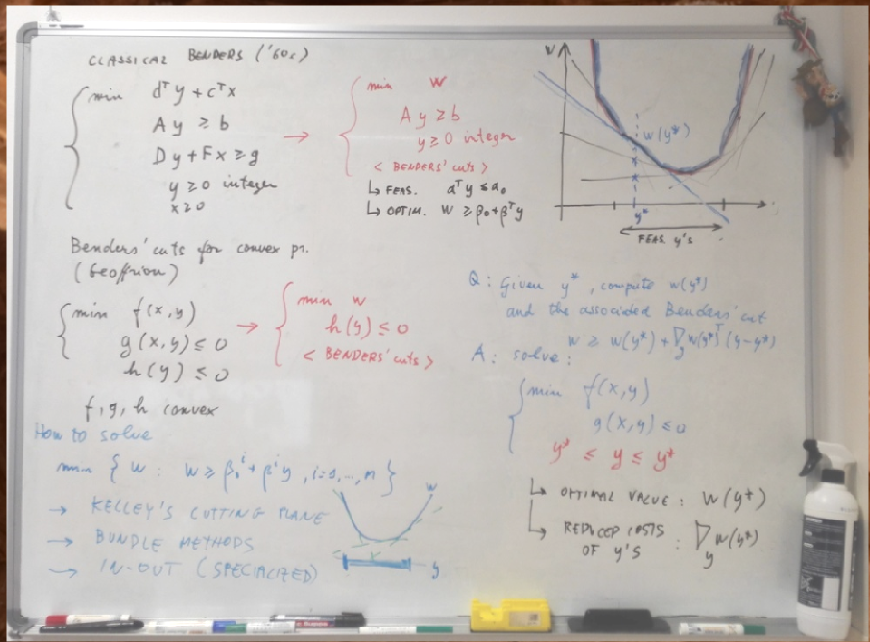
Main page Contents Featured content Current events Random article Donate to Wikipedia Wikipedia store

Interaction: Help About Wikipedia Community portal Recent changes Contact page

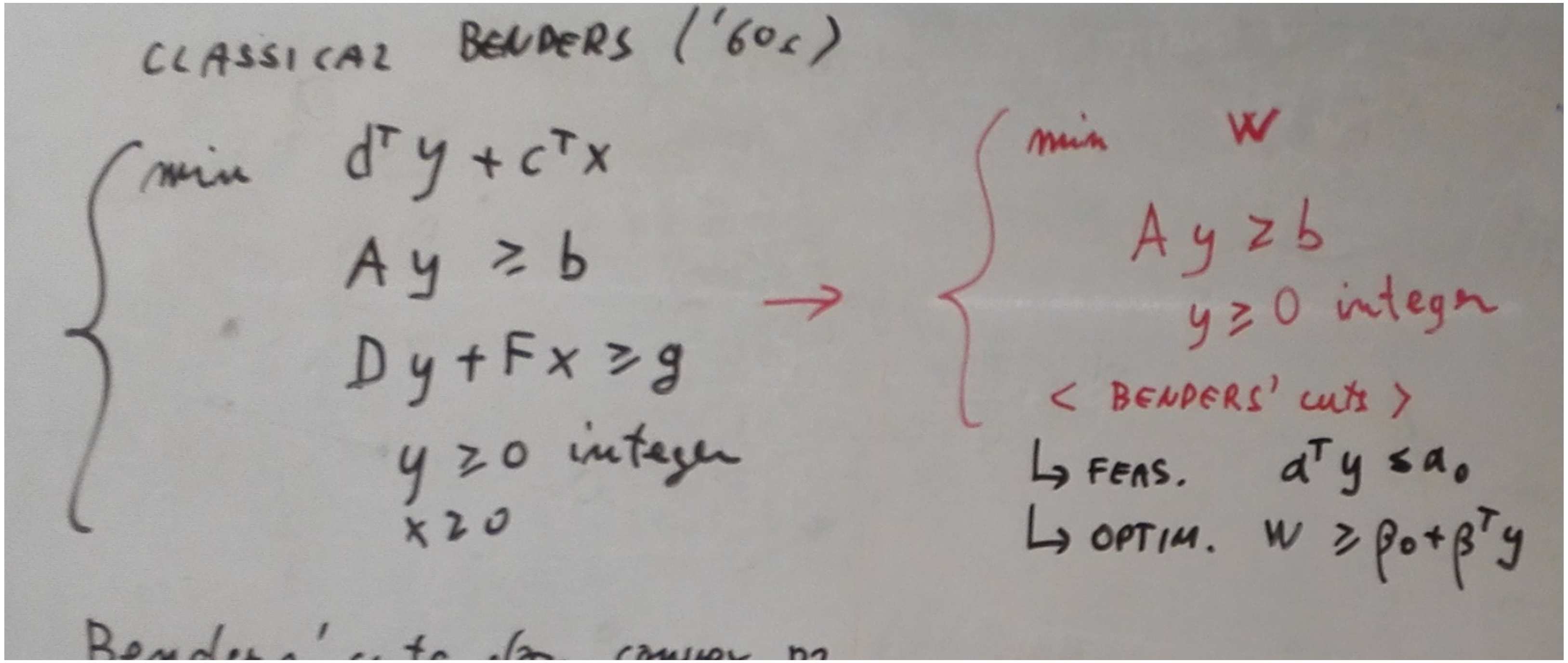
Tools: What links here Related changes Upload file Special pages Permanent link Page information Wikidata item Cite this page

Print/export: Create a book

Benders in a nutshell



#BendersToTheBone



Original problem (left) vs Benders' master problem (right)

Benders after Padberg&Rinaldi

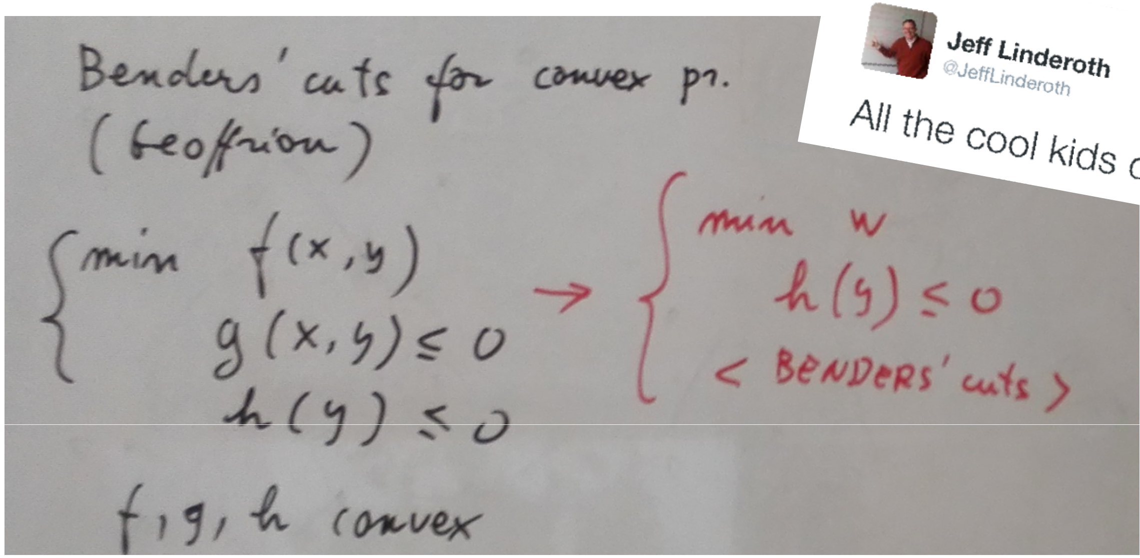
- The original ('60s) recipe was to solve the master to optimality by enumeration (integer y*), to generate B-cuts for y*, and to repeat

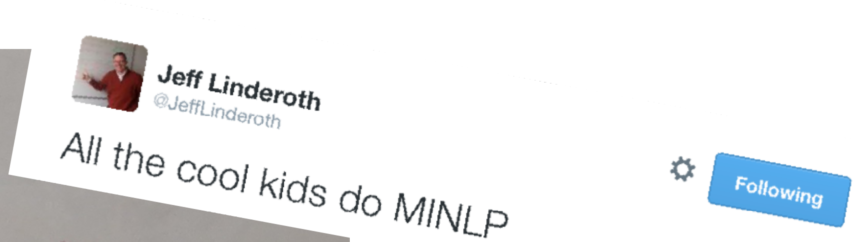
  → This is what we call "**Old Benders**" within our group

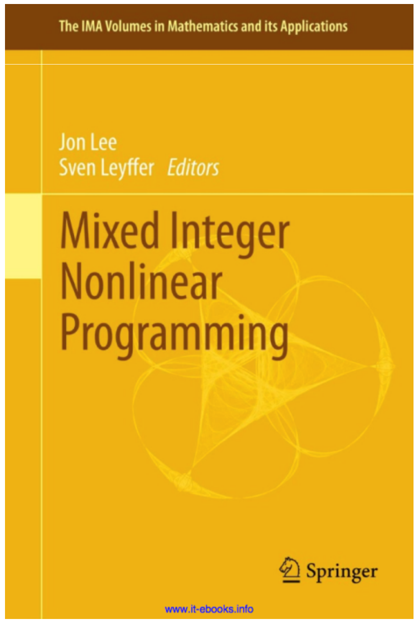
  → **still the best option for some problems!**
- Folklore (Miliotios for TSP?): generate B-cuts for any integer y* that is going to update the incumbent
- McDaniel & Devine (1977) use of B-cuts to cut (root node) fractional y*'s
- …
- Everything fits very naturally within modern **Branch-and-Cut**
  - **Lazy constraint** callback for integer y* (needed for correctness)
  - **User cut** callback for any y* (useful but not mandatory)

- Feasibility cuts → we know how to handle (minimal infeasibility etc.)
- **Optimality cuts → often a nightmare even after MW improvements (pareto-optimality) and alike → THE TOPIC OF THE PRESENT TALK**

Benders for convex MINLP



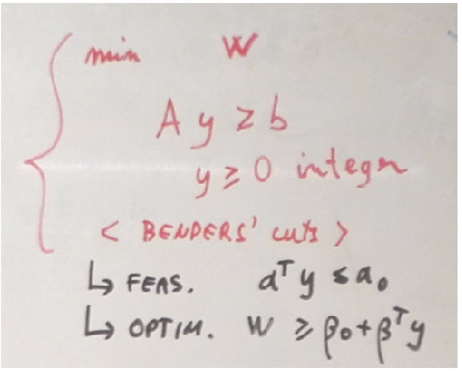


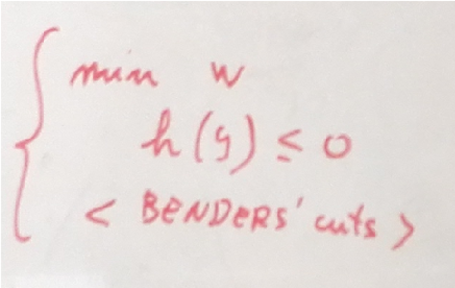


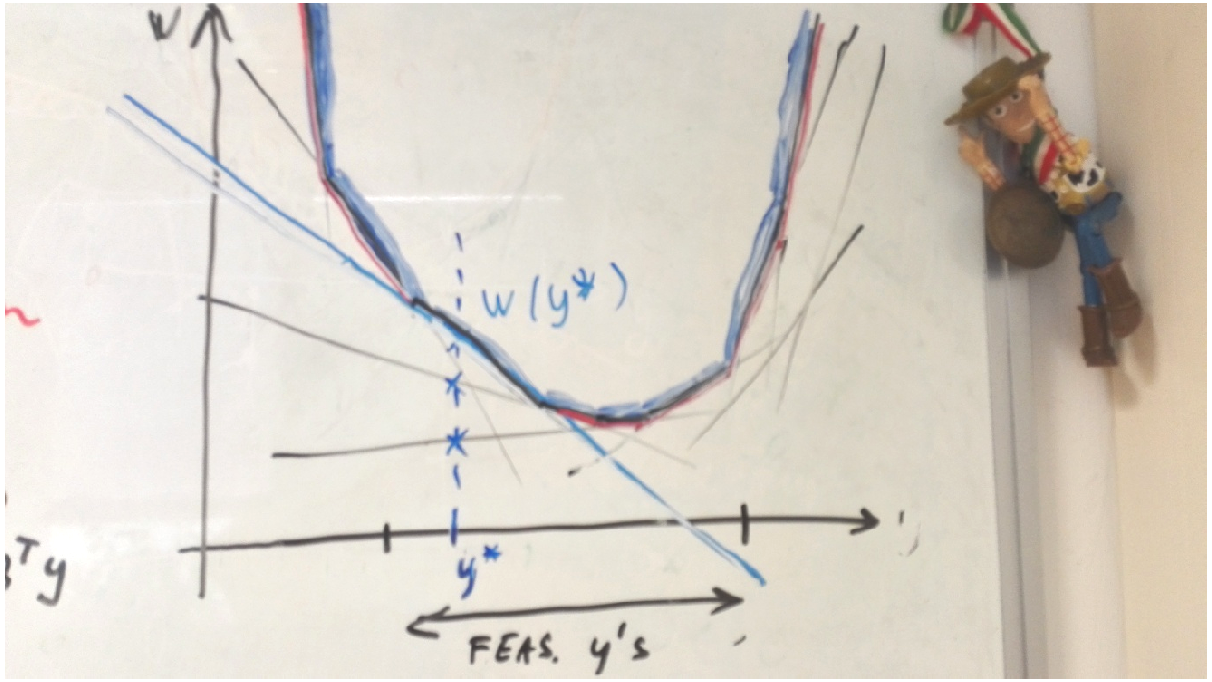
- Benders cuts can be generalized to convex MINLP
  - → Geoffrion via Lagrangian duality
  - → resulting **Generalized Benders** cuts still linear

- Potentially very useful **to remove nonlinearity** from the master by using kind of “surrogate cone” cuts → hide nonlinearity where it does not hurt…

Optimality cut geometry

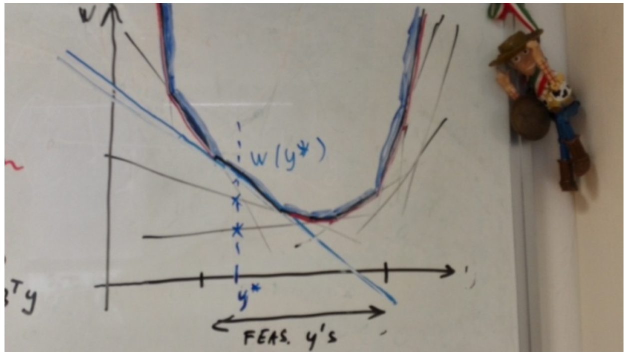




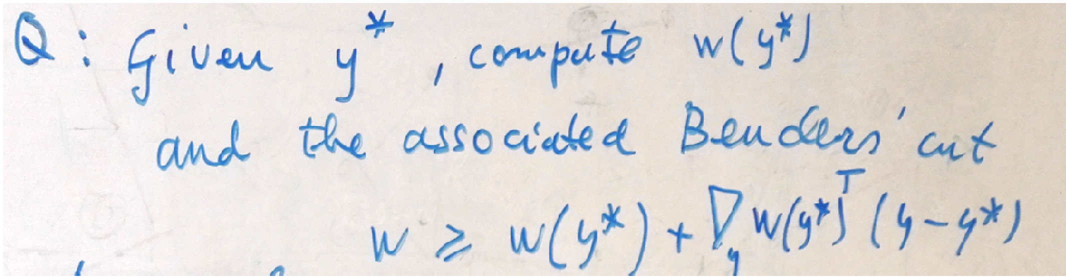


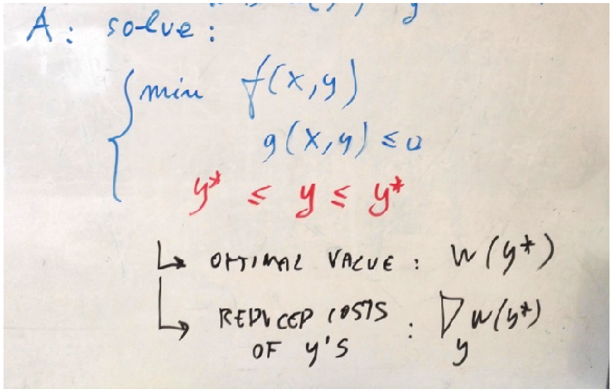
Solving the master LP relaxation → minimization of a convex function w(y) → a very familiar setting for people working with **Lagrange** duality (Dantzig-Wolfe decomposition and alike)

Optimality cut generation



Given y*, how to compute the supporting hyperplane (in blue)?





1-2-3 Benders optimality cut computation

1) solve the original convex problem with new var. bounds $y^* \leq y \leq y^*$

2) take *opt_val* and reduced costs r_j's

3) write $w \geq opt\_val + \sum_j r_j (y_j - y_j^*)$

Benders++ cuts

• We have seen that Benders cuts are obtained by solving the **original problem** after fixing y=y*, thus voiding the information that y must be integer

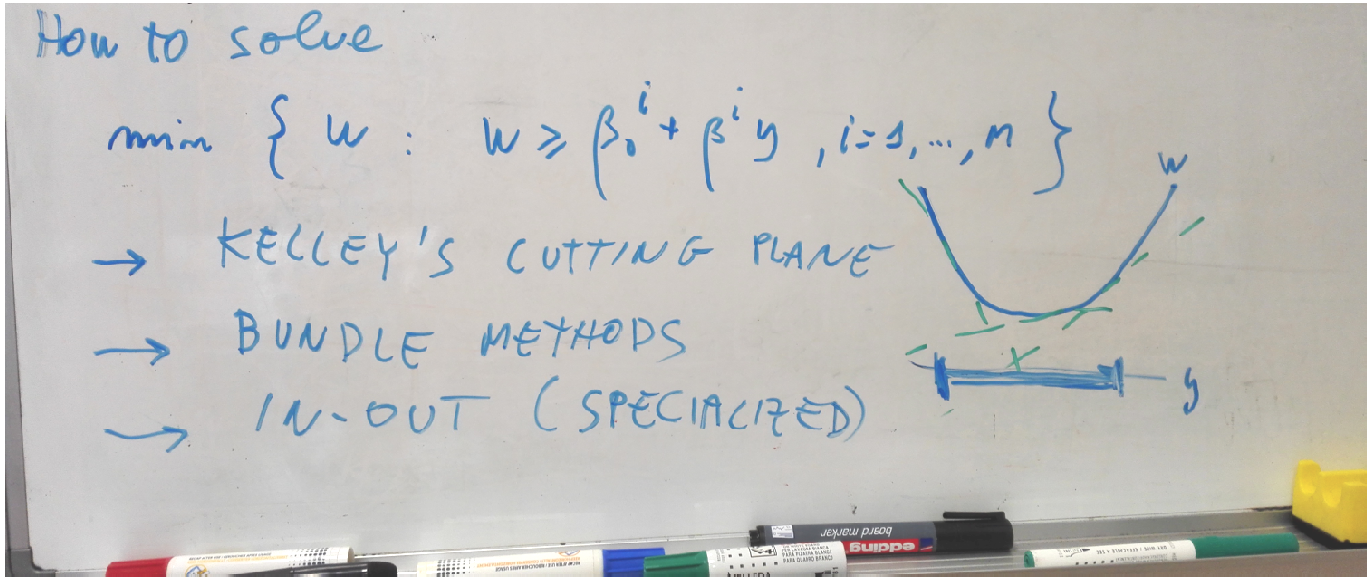
• Full primal optimal sol. **(y*,x*)** available for generating MIP cuts exploiting the integrality of y

• However **(y*,x*)** is not a vertex → no cheap “tableau cuts” (GMI and alike) available …
… while any black-box **separation function** that receives the original model and the pair (y*,x*) on input can be used (MIR heuristics, CGLP’s, half cuts, etc.)

• Generated cuts to be added to the original model (i.e. to the “slave”) in case they involve the x’s

• Very good results with split cuts for Stochastic Integer Programming recently reported by Bodur, Dash, Gunluck, Luedtke (2014)

#TheCurseOfKelley



Now that you have seen the plot of **w(y)**, you understand that a main reason for Benders slow convergence is the use of **Kelley's cutting plane scheme**

→Stabilization required as in

Column Generation and
Lagrangian Relaxation

Escaping the #CurseOfKelley

• Root node LP bound **very critical** → many ships sank here!

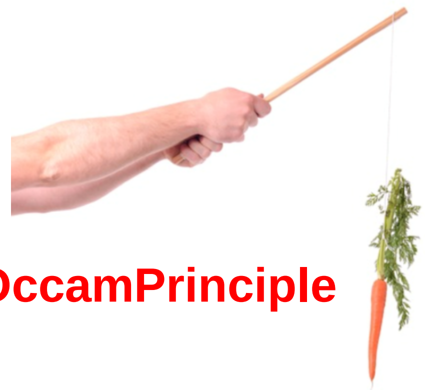
• **Kelley's** cutting plane can be desperately slow, bundle/interior points methods required

• For (q)UFL, at the root node we implemented our own "interior point" method inspired by (Ben-Ameur and Neto 2007, Fischetti and Salvagnin 2010, Naoum-Sawaya and Elhedhli 2013).

• **We want to work on the y-space** (as any honest bundle would do)

• In-out/analytic center methods work on the (y,w) space → adaptation needed

• As a quick shot, we implemented a very simple **"chase the carrot"** heuristic to determine an internal path towards the optimal y



• Our very first implementation worked so well that we did not have an incentive to try and improve it **#OccamPrinciple**

Our #ChaseTheCarrot dual heuristic

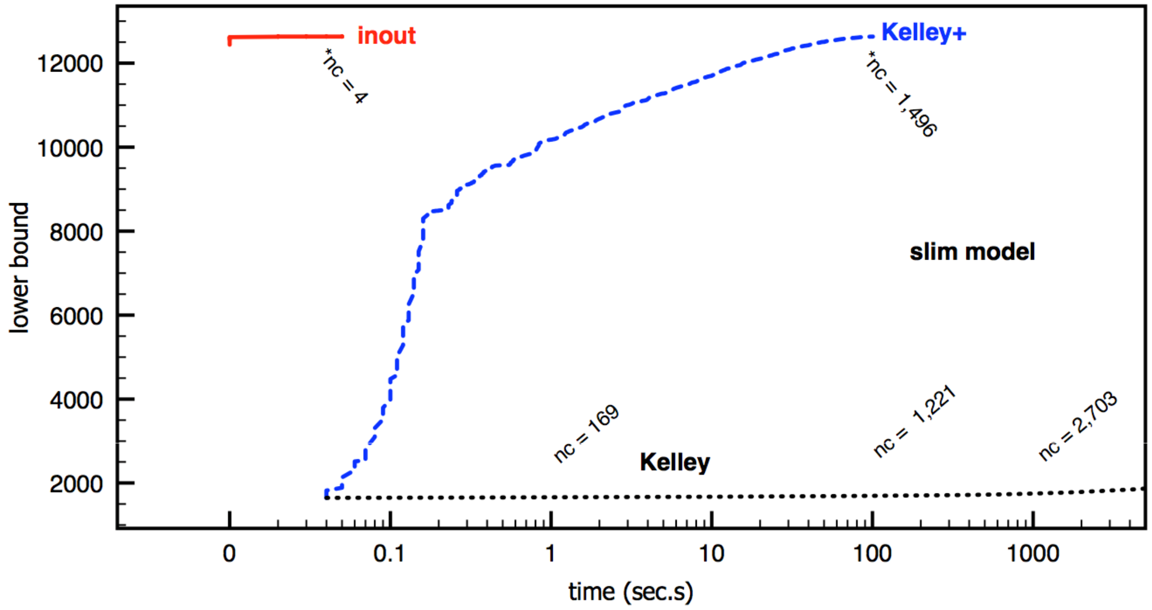
- We (the donkey) start with y=(1,1,…) and optimize the master LP as in Kelley, to get optimal y* (the carrot on the stick).

- We move y half-way towards y*. We then separate a point y' in the segment y-y* close to y. The generated optimality cut(s) are added to the master LP, which is reoptimzied to get the new optimal y* (carrot moves).

- Repeat until bound improves, then switch to Kelley for final bound refinement (cross-over like)

- **Warning: adaptations needed if feasibility cuts can be generated…**

Effect of the improved cut-loop



- Comparing **Kelley** cut loop at the root node with **Kelley+** (add epsilon to y*) and with our chase-the-carrot method (**inout**)
- Koerkel-Ghosh **qUFL** instance gs250a-1 (250x250, quadratic costs)
- ***nc** = n. of Benders cuts generated at the end of the root node
- times in **logarithmic scale**

Conclusions

- I wanted to write a very elaborated and convincing conclusion section …
- … so I started with a first version **#toolong**
- … and then I **simplified** it and then I **simplified** it and …
- This is what remains

Be simple (if you can)! #OccamRazor

Thank you for your attention

- Full papers and slides available at
  http://www.dei.unipd.it/~fisch/papers/
  http://www.dei.unipd.it/~fisch/papers/slides/