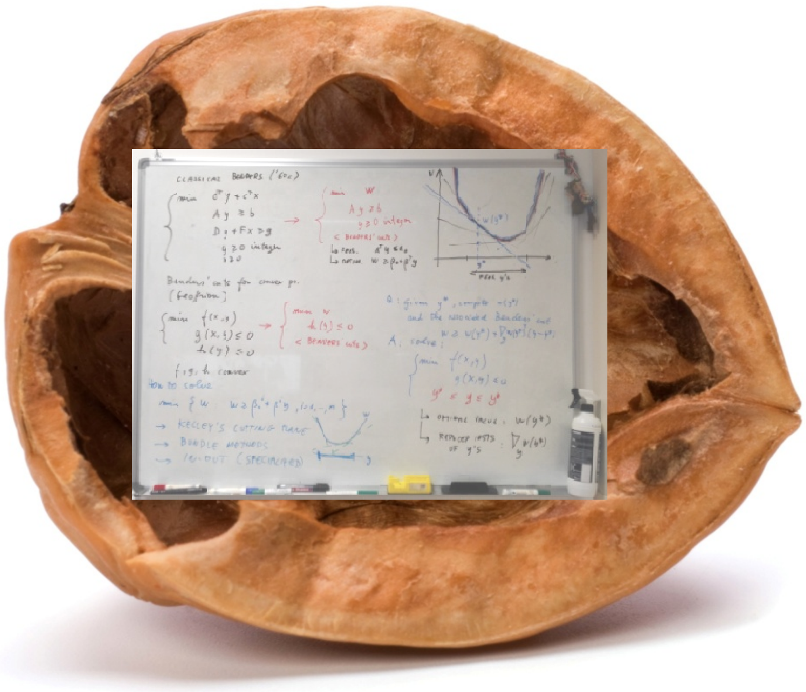
Branch-and-cut implementation of Benders' decomposition

Matteo Fischetti, University of Padova



Mixed-Integer Programming

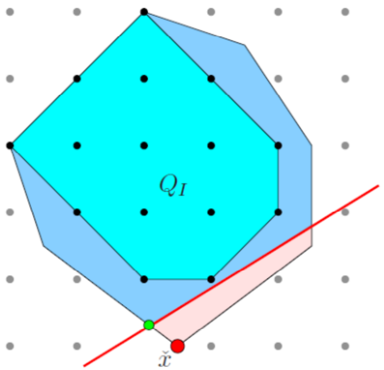
- We will focus on the MIP

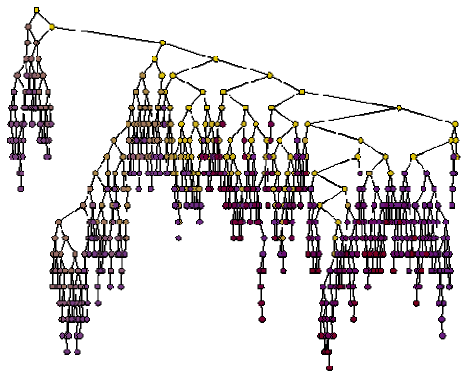
  where *f* and *g* are **convex functions**

$$\min f(x,y)$$
$$g(x,y) \leq 0$$
$$Ay \leq b$$
$$y \text{ integer}$$

- Non-convexity only comes from integrality requirement on *y*, so removing the latter produces an easy-to-solve **convex relaxation** → lower bound ***LB*** along with a **fractional solution** *x** to be used "somehow"

- **Cutting plane** method (Gomory 1958)





- **Branch-and-Bound** enumeration (Land and Doig, 1960)

Branch-and-Cut (B&C)

- B&C was proposed by Padberg and Rinaldi in the 1990s and is nowadays the **method of choice** for solving MIPs

- B&C is a clever **mixture** of cutting-plane and branch-and-bound methods

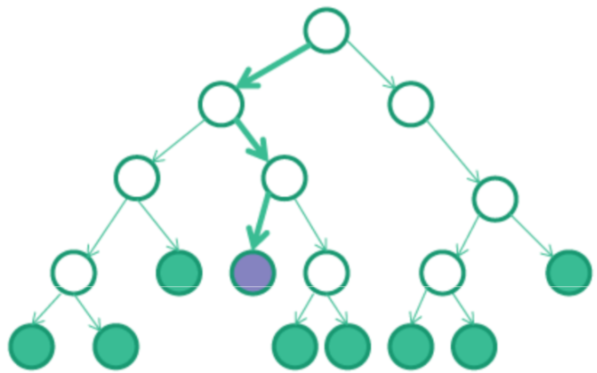
- **Cuts** are generated during B&B (potentially, at all nodes) with the aim of improving the lower bound and producing “more integral” solutions → better pruning, better heuristics, and (hopefully) better branching guidance

- **Convergence** relies on enumeration (inherited by the B&B scheme) → cut generation can safely be stopped at any time, to prevent e.g. shallow cuts, tailing off, numerical issues, etc.

- Since the beginning, an highly-effective **implementation** was part of the B&C trademark (use of cut pool, global vs local cuts, variable pricing, etc.)

Modern B&C implementation

- Modern commercial B&C solvers such as IBM ILOG Cplex, Gurobi, XPRESS etc. can be fully **customized** by using ***callback functions***



- Callback functions are just **entry points** in the B&C code where an advanced user **(you!)** can add his/her customizations

- Most-used callbacks (using Cplex's jargon)
  - **Lazy constraint**: add "lazy constr.s" that should be part of the original model
  - **User cut**: add additional contr.s that hopefully help enforcing integrality
  - Heuristic: try to improve the incumbent (primal solution) as soon as possible
  - Branch: modify the branching strategy
  - …

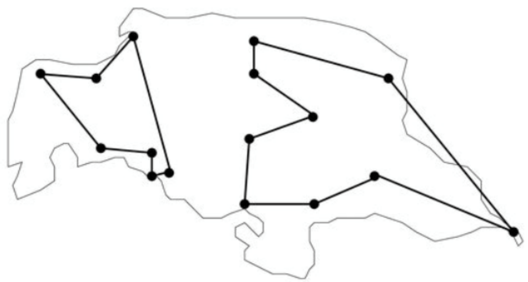
Lazy constraint callback

- Automatically invoked when a solution is going to update the **incumbent** (meaning it is **integer** and **feasible** w.r.t. current model)

- This is the **last checkpoint** where you can discard a solution for whatever reason (e.g., because it violates a constraint that is not part of the current model)

- To avoid be bothered by this solution again and again, you can/should return a **violated constraint (cut)** that is added (globally or locally) to the current model

- Cut generation is often **simplified** by the fact that the solution to be cut is known to be **integer** (e.g., SECs for TSP)

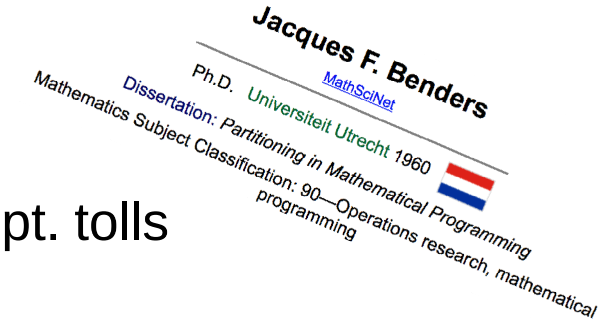


User cut callback

- Automatically invoked at every B&B node when the current solution is **not integer** (e.g., just before branching)
- A **violated cut** can possibly be returned, to be added (locally or globally) to the current model → often leads to an improved convergence to integer solutions
- If no cut is returned, **branching** occurs as usual

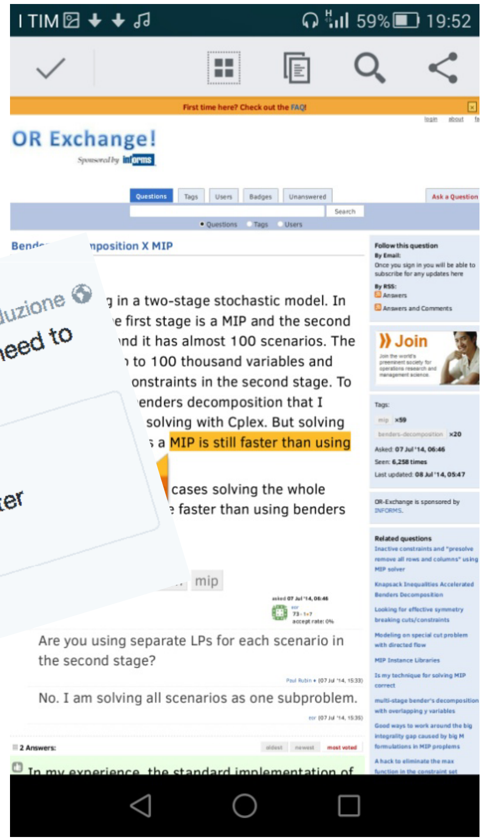
- Cut generation **can be hard** as the point is not integer (heuristic approaches can be used)
- User cuts are **not mandatory** for B&C correctness → insisting too much can actually **slow-down** the solver because of the overhead in generating and using the new cuts (larger/denser LPs etc.)

Ready for Benders?

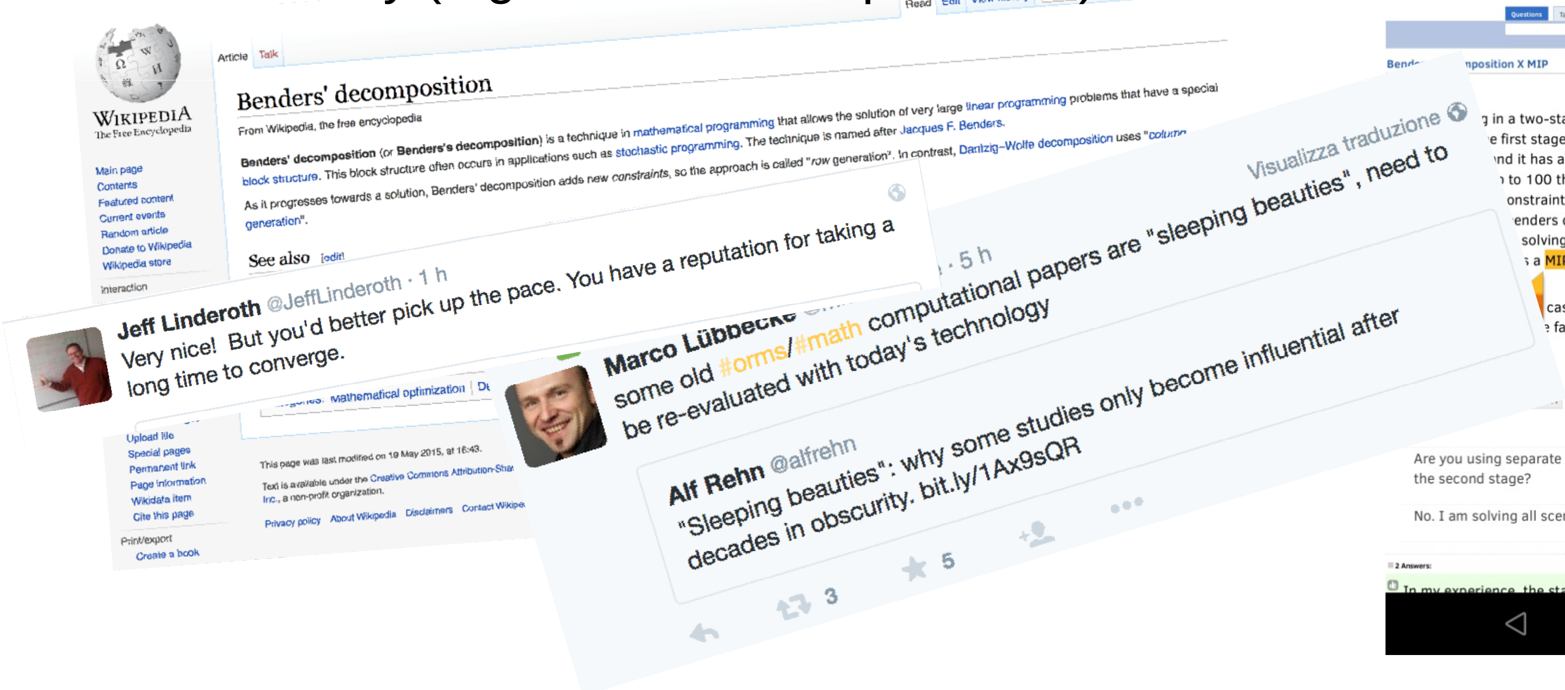


Benders decomposition is one of basic Math.Opt. tolls

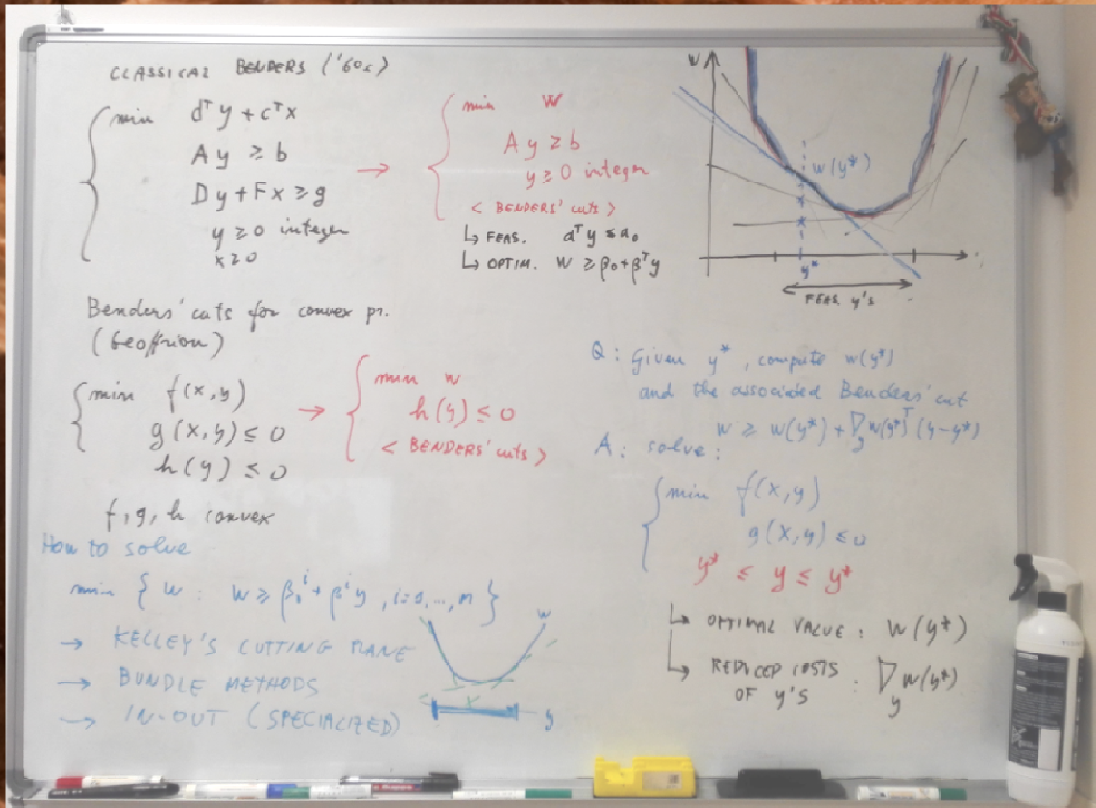
… but not so many MIPeople are willing to implement it because of its bad reputation (instability, slow convergence, etc.)



… till recently (e.g., it is now in Cplex 12.7)



Benders in a nutshell



What do you actually mean by "Benders decomposition"?

- The original Benders decomposition from the '60s uses **two** distinct ingredients for solving a Mixed-Integer Linear Program (MILP):
  1) A **search strategy** where a relaxed **(NP-hard)** MILP on a variable **subspace** is solved exactly (i.e., to **integrality**) by a black-box solver, and then is iteratively tightened by means of additional **"Benders" linear cuts**
  2) The **technicality** of how to actually compute those cuts (Farkas' projection)
  - Papers proposing "a new Benders-like scheme" typically refer to 1)
  - Students scared by "Benders implementations" typically refer to 2)

  Later developments in the '70s:
  - Folklore (Miliotios for TSP?): generate Benders cuts within a **single B&B tree** to cut any infeasible integer solution that is going to update the incumbent
  - McDaniel & Devine (1977): use Benders cuts to cut **fractional sol.s** as well (root node only)
- Everything fits very naturally within a modern **Branch-and-Cut** (B&C) framework.

Modern Benders

- Consider again the convex MINLP in the *(x,y)* space

$$\begin{aligned} \min f(x,y) \\ g(x,y) \le 0 \\ Ay \le b \\ y \text{ integer} \end{aligned}$$

and assume for the sake of simplicity that $S := \{y : Ay \le b\}$ is nonempty and bounded, and that

$$X(y) := \{x : g(x,y) \le 0\}$$

is **nonempty**, closed and bounded for all y ∈ S

→ the **convex function** $\Phi(y) := \min_{x \in X(y)} f(x,y)$ is well defined for all y ∈ S

→ no "feasibility cuts" needed (this kind of cuts will be discussed later on)

Working on the y-space (projection)

(1)

$$\min_{y} \min_{x} f(x,y)$$
$$g(x,y) \le 0$$
$$Ay \le b$$
$$y \text{ integer}$$

(2)

"isolate the inner minimization over *x*"

$$\Phi(y) := \min_{x} f(x,y)$$
$$g(x,y) \le 0$$

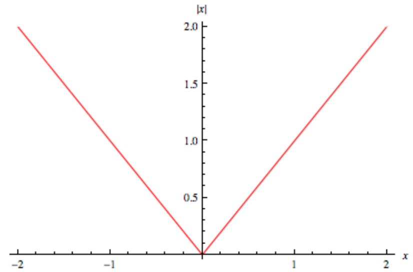
(3)

$$\min \Phi(y)$$
$$Ay \le b$$
$$y \text{ integer}$$

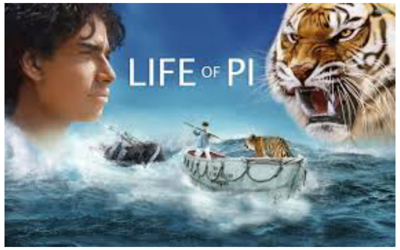
Original MINLP in the (x,y) space → Benders' **master** problem in the *y* space

Warning: projection changes the objective function (e.g., linear → convex nonlinear)

$$\min x$$
$$x \ge y$$
$$x \ge -y$$
$$y \in [-1,1]$$

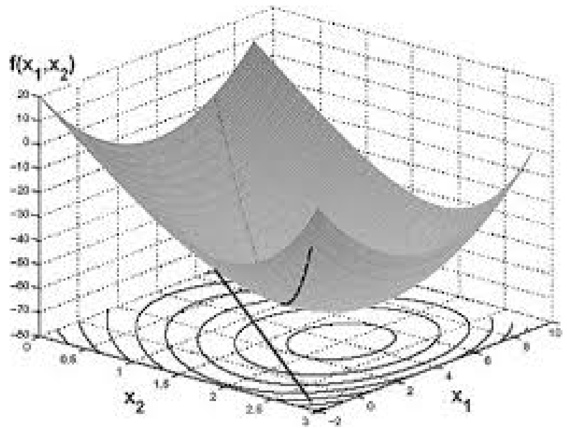


$$\min \Phi(y) = |y|$$
$$y \in [-1,1]$$



Life of P(H)I

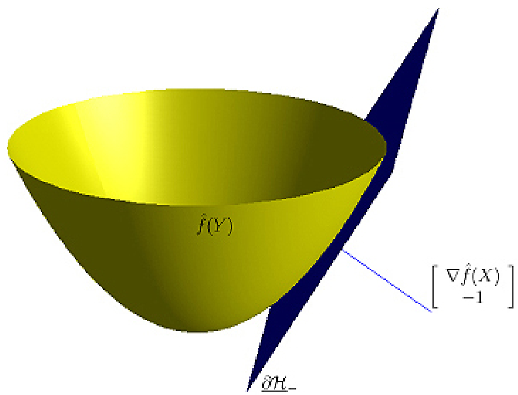
- Solving Benders' master problem calls for the minimization of a **nonlinear** convex function **(even if you start from a linear problem!)**



- Branch-and-cut MINLP solvers generate a sequence of **linear cuts** to approximate this function from below (**outer-approximation**)

$$\begin{aligned} \min\, & w \\ \text{s.t.}\ w & \geq \Phi(y) \\ Ay & \leq b \\ y & \text{ integer} \end{aligned}$$

subgradient
(aka Benders) cut →



$$w \geq \Phi(y) \geq \Phi(y^*) + \xi(y^*)^T (y - y^*)$$

Benders cut computation

- **Benders** (for linear) and **Geoffrion** (general convex) told us how to compute a **subgradient** to be used in the cut derivation, by using the optimal primal-dual solution *(x*,u*)* available after computing $\Phi(y^*)$

$$\xi(y^*) = \nabla_y f(x^*, y^*) + u^* \nabla_y g(x^*, y^*)$$

- The above formula is **problem-specific** and perhaps **#scaring**
- Introduce an **artificial variable vector q** (acting as a copy of *y*) to get

$$\Phi(y^*) = \min\{f(x, \mathbf{q}) \mid g(x, \mathbf{q}) \leq 0,\ y^* \leq \mathbf{q} \leq y^*\}$$

and to obtain the following **simpler** and **completely general** cut-recipe:

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 feasibility cuts

- For some important applications, the set

$$X(y) := \{x : g(x, y) \leq 0\}$$

can be empty for some "**infeasible**" $y \in S$

→ $$\Phi(y) := \min_{x \in X(y)} f(x, y)$$ undefined

- This situation can be handled by considering the "phase-1" feasibility condition

$$0 \geq \Psi(y) := \min\{1^T s \,|\, g(x, y) \leq s,\ s \geq 0\}$$

where the function $\Psi(y)$ is **convex**
→ it can be approximated by the usual subgradient **"Benders feasibility cut"**

$$0 \geq \Psi(y) \geq \Psi(y^*) + \xi(y^*)^T (y - y^*)$$

to be computed as in the previous **"Benders optimality cut"**

$$w \geq \Phi(y) \geq \Phi(y^*) + \xi(y^*)^T (y - y^*)$$

Successful Benders applications

- Benders decomposition works well when fixing *y* = *y** for computing $\Phi(y^*)$ makes the problem **much simpler to solve**.

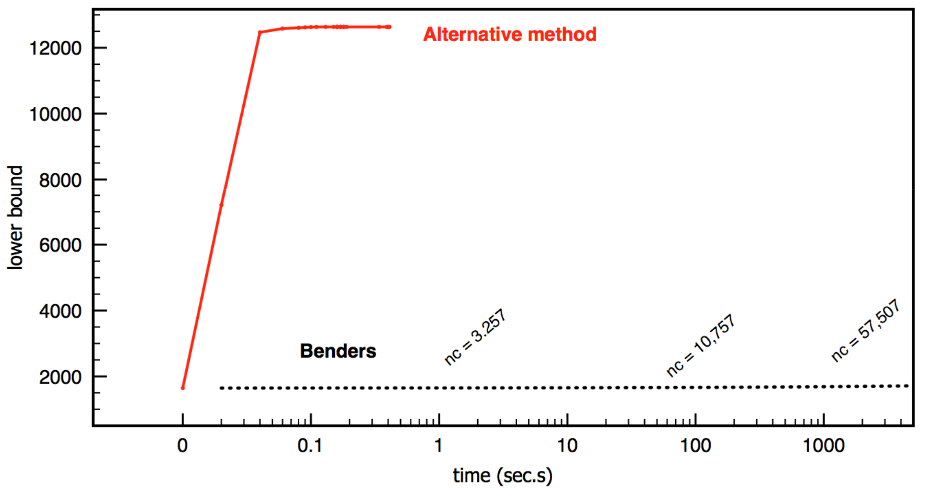
- This usually happens when
  - The problem for *y* = *y** decomposes into a number of **independent subproblems**
    - Stochastic Programming
    - Uncapacitated Facility Location
    - etc.

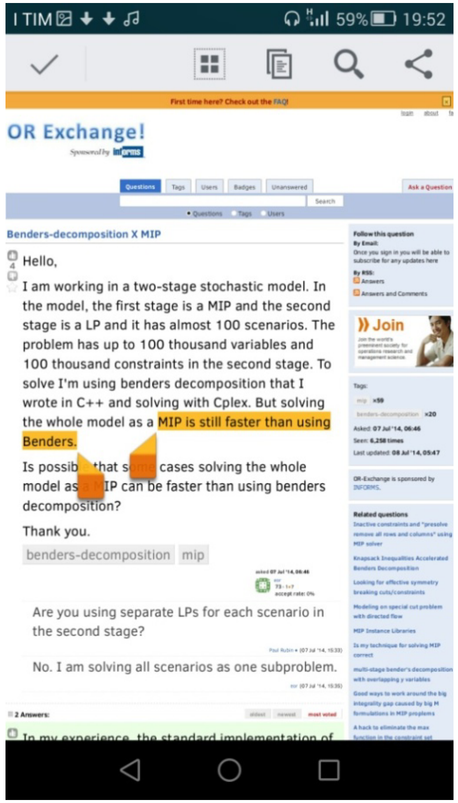
$$\begin{aligned} \min \sum_{i\in I} f_i y_i + \sum_{i\in I}\sum_{j\in J} c_{ij} x_{ij} & \\ \text{s.t. } \sum_{i\in I} x_{ij} = 1 & \quad \forall j \in J \\ x_{ij} \le y_i & \quad \forall i \in I, j \in J \\ x_{ij} \ge 0 & \quad \forall i \in I, j \in J \\ y_i \in \{0,1\} & \quad \forall i \in I \end{aligned}$$

  - Fixing *y* = *y** **changes the nature** of some constraints:
    - in Capacitated Facility Location, tons of constr.s of the form $x_{ij} \le y_j$ become just variable bounds
    - Second Order Constraints $x_{ij}^2 \le z_{ij}\, y_i$ become quadratic constr.s
    - etc.

That’s it … or not?

- In practice, Benders decomposition can work quite well, but sometimes it is **desperately slow**
  … as the root node bound does not improve even after the addition of tons of Benders cuts





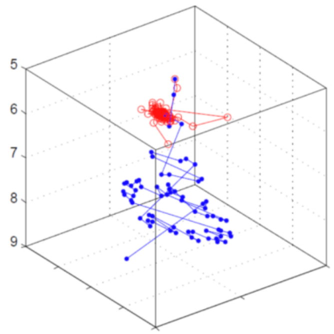
- Slow convergence is generally attributed to the **poor quality** of Benders cuts, to be cured by a more clever **selection policy** (Pareto optimality of Magnanti and Wong, 1981, etc.) but **there is more**…

Role of the cut loop

- B&C codes generate cuts, on the fly, in a **sequential** fashion
- Consider e.g. the **root B&C node** (arguably, the most critical one)
- A classical **cut-loop scheme** (described here for MILPs)

  J. E. Kelley. The cutting plane method for solving convex programs, Journal of the SIAM, 8:703-712, 1960.

  – Find an optimal **vertex** x^* of the current LP relaxation
  – Invoke a separation function on x^*, add the returned violated cut (if any) to the current LP, and repeat

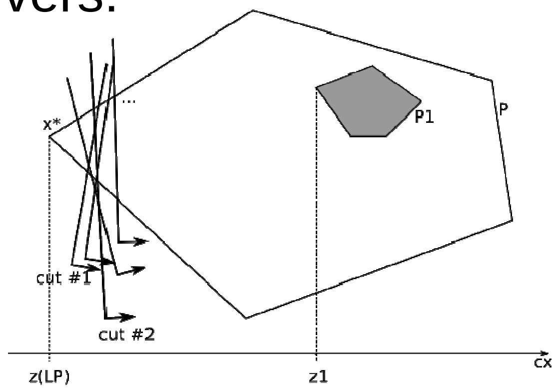


- Can be very **ineffective** in the **first iterations** when few constraints are specified, and x^* moves along an **unstable zig-zag trajectory**

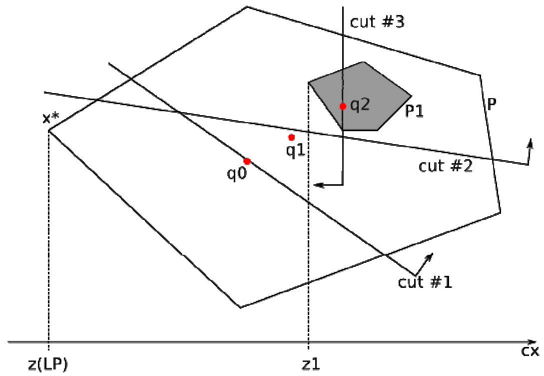
  ... which is precisely what often happens with Benders cuts

But… alternative cut loops do exist!

- **Kelley's** cut loop implemented in standard MI(L)P solvers:
  - PROS: natural, efficient reopt., often works well
  - CONS: can be VERY ineffective, e.g., in
    column generation or in some
    under-constrained cutting plane methods



- **Ellipsoid & Analytic Center** cut loops:
  kind of **binary search** in the multi-dimensional space:
  at each iteration, a **core point** *q* "well inside" the
  current relaxation is computed and separated
  - CONS: q can be difficult to find and to separate
  - PROS: overall convergence does not depend
    on the quality of the cut (facets not required here!)



- Cheaper alternatives often preferred: **bundle** (Lemaréchal) or **in-out** (Ben-Ameur and Neto) methods

Stabilizing Benders can be easy!

- To summarize:

  - Benders cut machinery is easy to implement …

  … but the root node cut loop can be **very critical**
  → many implementations sank here!

  - Kelley's cut loop can be **desperately slow**

  - Stabilization using "interior points" is a **must**
    → this is well-known in subgradient optimization and Dantzig-Wolfe decomposition (column generation), but holds for Benders as well

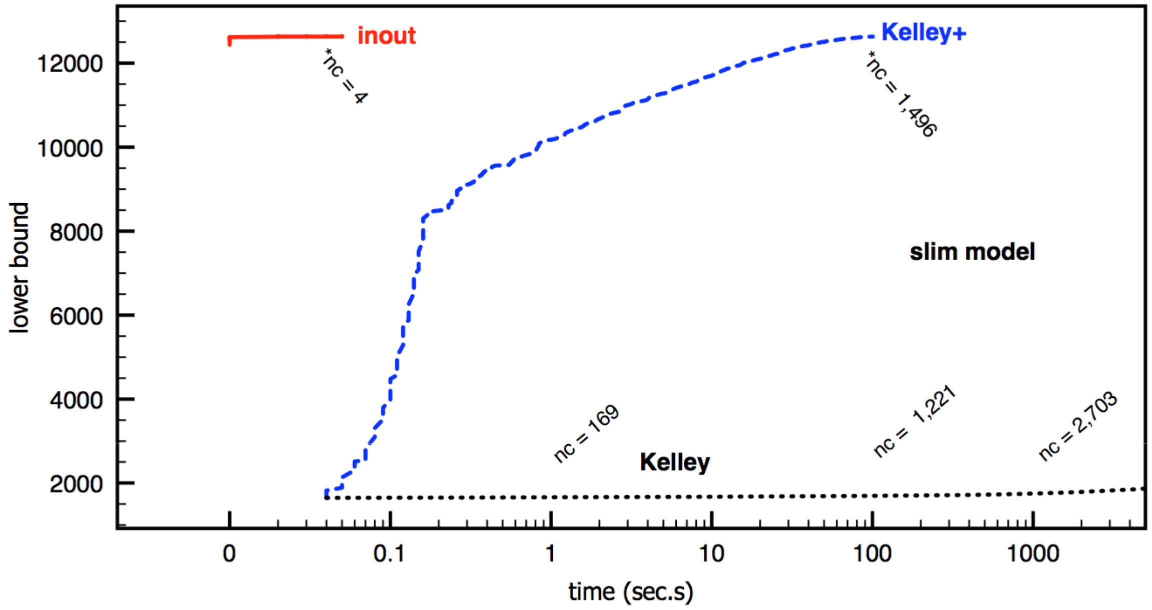
- E.g., for facility location problems, we implemented a very simple **"chase the carrot**" heuristic to determine a stabilized path towards the optimal *y*

- Akin to **Nesterov's Accelerated Gradient** descent method

Our #ChaseTheCarrot heuristic

- We (the donkey) start with $\boldsymbol{y = (1,1,\ldots,1)}$ and optimize the master LP as in Kelley, to get optimal $\boldsymbol{y^*}$ (the carrot on the stick).
- We move $\boldsymbol{y}$ just **half-way** towards $\boldsymbol{y^*}$. We then separate a point y' in the segment $\boldsymbol{[y, y^*]}$ close to the new $\boldsymbol{y}$.
- The generated Benders cut is added to the master LP, which is reoptimized to get the new optimal $\boldsymbol{y^*}$ (carrot moves).
- Repeat until bound improves, then switch to Kelley for final bound refinement (kind of cross-over)
- **Warning: adaptations needed if feasibility Benders 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

To summarize:

- Benders cuts are **easy** to implement within modern B&C (just use a callback where you solve the problem for $y = y^*$ and compute reduced costs)

- Kelley's cut loop can be **desperately slow** hence stabilization is a **must**

- Implementations in **general** MIP solver already in Cplex 12.7

Slides available at http://www.dei.unipd.it/~fisch/papers/slides/

Reference papers:

M. Fischetti, I. Ljubic, M. Sinnl, "Benders decomposition without separability: a computational study for capacitated facility location problems", European Journal of Operational Research, 253, 557-569, 2016.

M. Fischetti, I. Ljubic, M. Sinnl, "Redesigning Benders Decomposition for Large Scale Facility Location", to appear in Management Science, 2016.