

Branch-and-Cut is our swiss army knife

Matteo Fischetti, University of Padova



Branch & Cut TM

A BRANCH-AND-CUT ALGORITHM
FOR THE RESOLUTION OF LARGE-SCALE
SYMMETRIC TRAVELING SALESMAN PROBLEMS *

MANFRED PADBERG[†] AND GIOVANNI RINALDI[‡]

- A “trademark” by Manfred Padberg and Giovanni Rinaldi
- Proposed in the 1990’s for the TSP (and soon extended)
- Comes as an **algorithm** entangled with its **implementation**

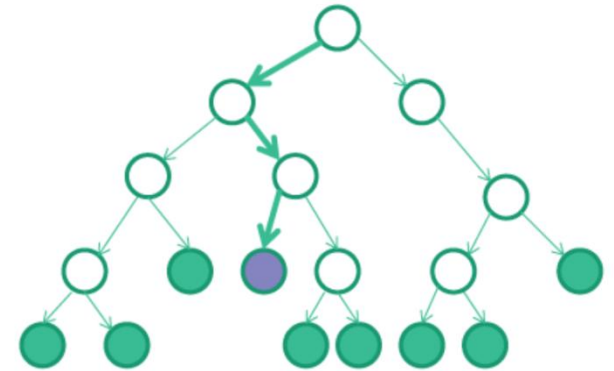
Conjecture: *Using cuts within an enumerative scheme is good.*

Proof. Assume w.l.o.g. a good LP solver. Then apply B&Bound but

- make use of families of (problem dependent) globally-valid inequalities
- perform efficient exact/heuristic cut separation on the fly
- use a data-structure (cut pool) to effectively share cuts among nodes
- price variables in a dynamic way (well before branch-and-price!)
- alternate row and column generation in a sound way ...
- suspend a node if “unattractive”
- ...

Modern B&C implementation

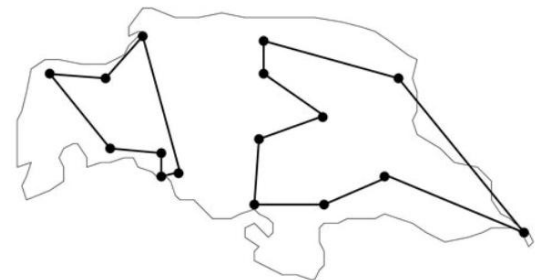
- Modern B&C solvers such as Cplex, Gurobi, Express, SCIP etc. can be fully **customized** by using **callback functions**
 - Callback functions are just **entry points** in the B&C code where an advanced user (you!) can add his/her customizations
 - Most-used callbacks (using old-style Cplex's jargon)
 - **Lazy constraint**: add “lazy constr.s” that should be part of the original model
 - **User cut**: add additional contr.s that hopefully help enforcing feasibility/integrality
 - Heuristic: try to improve the incumbent (primal solution) as soon as possible
 - Branch: modify the branching strategy
 - ...
-
- ```
graph TD; N1(()) --> N2(()); N1 --> N3(()); N2 --> N4(()); N2 --> N5(()); N3 --> N6(()); N3 --> N7(()); N4 --> N8(()); N4 --> N9(()); N5 --> N10(()); N5 --> N11(()); N6 --> N12(()); N6 --> N13(()); N7 --> N14(()); N7 --> N15(()); N8 --> N16(()); N8 --> N17(()); N9 --> N18(()); N9 --> N19(()); N10 --> N20(()); N10 --> N21(()); N11 --> N22(()); N11 --> N23(()); N12 --> N24(()); N12 --> N25(()); N13 --> N26(()); N13 --> N27(()); N14 --> N28(()); N14 --> N29(()); N15 --> N30(()); N15 --> N31(()); N16 --> N32(()); N16 --> N33(()); N17 --> N34(()); N17 --> N35(()); N18 --> N36(()); N18 --> N37(()); N19 --> N38(()); N19 --> N39(()); N20 --> N40(()); N20 --> N41(()); N21 --> N42(()); N21 --> N43(()); N22 --> N44(()); N22 --> N45(()); N23 --> N46(()); N23 --> N47(()); N24 --> N48(()); N24 --> N49(()); N25 --> N50(()); N25 --> N51(()); N26 --> N52(()); N26 --> N53(()); N27 --> N54(()); N27 --> N55(()); N28 --> N56(()); N28 --> N57(()); N29 --> N58(()); N29 --> N59(()); N30 --> N60(()); N30 --> N61(()); N31 --> N62(()); N31 --> N63(()); N32 --> N64(()); N32 --> N65(()); N33 --> N66(()); N33 --> N67(()); N34 --> N68(()); N34 --> N69(()); N35 --> N70(()); N35 --> N71(()); N36 --> N72(()); N36 --> N73(()); N37 --> N74(()); N37 --> N75(()); N38 --> N76(()); N38 --> N77(()); N39 --> N78(()); N39 --> N79(()); N40 --> N80(()); N40 --> N81(()); N41 --> N82(()); N41 --> N83(()); N42 --> N84(()); N42 --> N85(()); N43 --> N86(()); N43 --> N87(()); N44 --> N88(()); N44 --> N89(()); N45 --> N90(()); N45 --> N91(()); N46 --> N92(()); N46 --> N93(()); N47 --> N94(()); N47 --> N95(()); N48 --> N96(()); N48 --> N97(()); N49 --> N98(()); N49 --> N99(()); N50 --> N100(()); N50 --> N101(()); N51 --> N102(()); N51 --> N103(()); N52 --> N104(()); N52 --> N105(()); N53 --> N106(()); N53 --> N107(()); N54 --> N108(()); N54 --> N109(()); N55 --> N110(()); N55 --> N111(()); N56 --> N112(()); N56 --> N113(()); N57 --> N114(()); N57 --> N115(()); N58 --> N116(()); N58 --> N117(()); N59 --> N118(()); N59 --> N119(()); N60 --> N120(()); N60 --> N121(()); N61 --> N122(()); N61 --> N123(()); N62 --> N124(()); N62 --> N125(()); N63 --> N126(()); N63 --> N127(()); N64 --> N128(()); N64 --> N129(()); N65 --> N130(()); N65 --> N131(()); N66 --> N132(()); N66 --> N133(()); N67 --> N134(()); N67 --> N135(()); N68 --> N136(()); N68 --> N137(()); N69 --> N138(()); N69 --> N139(()); N70 --> N140(()); N70 --> N141(()); N71 --> N142(()); N71 --> N143(()); N72 --> N144(()); N72 --> N145(()); N73 --> N146(()); N73 --> N147(()); N74 --> N148(()); N74 --> N149(()); N75 --> N150(()); N75 --> N151(()); N76 --> N152(()); N76 --> N153(()); N77 --> N154(()); N77 --> N155(()); N78 --> N156(()); N78 --> N157(()); N79 --> N158(()); N79 --> N159(()); N80 --> N160(()); N80 --> N161(()); N81 --> N162(()); N81 --> N163(()); N82 --> N164(()); N82 --> N165(()); N83 --> N166(()); N83 --> N167(()); N84 --> N168(()); N84 --> N169(()); N85 --> N170(()); N85 --> N171(()); N86 --> N172(()); N86 --> N173(()); N87 --> N174(()); N87 --> N175(()); N88 --> N176(()); N88 --> N177(()); N89 --> N178(()); N89 --> N179(()); N90 --> N180(()); N90 --> N181(()); N91 --> N182(()); N91 --> N183(()); N92 --> N184(()); N92 --> N185(()); N93 --> N186(()); N93 --> N187(()); N94 --> N188(()); N94 --> N189(()); N95 --> N190(()); N95 --> N191(()); N96 --> N192(()); N96 --> N193(()); N97 --> N194(()); N97 --> N195(()); N98 --> N196(()); N98 --> N197(()); N99 --> N198(()); N99 --> N199(()); N100 --> N200(()); N100 --> N201(()); N101 --> N202(()); N101 --> N203(()); N102 --> N204(()); N102 --> N205(()); N103 --> N206(()); N103 --> N207(()); N104 --> N208(()); N104 --> N209(()); N105 --> N210(()); N105 --> N211(()); N106 --> N212(()); N106 --> N213(()); N107 --> N214(()); N107 --> N215(()); N108 --> N216(()); N108 --> N217(()); N109 --> N218(()); N109 --> N219(()); N110 --> N220(()); N110 --> N221(()); N111 --> N222(()); N111 --> N223(()); N112 --> N224(()); N112 --> N225(()); N113 --> N226(()); N113 --> N227(()); N114 --> N228(()); N114 --> N229(()); N115 --> N230(()); N115 --> N231(()); N116 --> N232(()); N116 --> N233(()); N117 --> N234(()); N117 --> N235(()); N118 --> N236(()); N118 --> N237(()); N119 --> N238(()); N119 --> N239(()); N120 --> N240(()); N120 --> N241(()); N121 --> N242(()); N121 --> N243(()); N122 --> N244(()); N122 --> N245(()); N123 --> N246(()); N123 --> N247(()); N124 --> N248(()); N124 --> N249(()); N125 --> N250(()); N125 --> N251(()); N126 --> N252(()); N126 --> N253(()); N127 --> N254(()); N127 --> N255(()); N128 --> N256(()); N128 --> N257(()); N129 --> N258(()); N129 --> N259(()); N130 --> N260(()); N130 --> N261(()); N131 --> N262(()); N131 --> N263(()); N132 --> N264(()); N132 --> N265(()); N133 --> N266(()); N133 --> N267(()); N134 --> N268(()); N134 --> N269(()); N135 --> N270(()); N135 --> N271(()); N136 --> N272(()); N136 --> N273(()); N137 --> N274(()); N137 --> N275(()); N138 --> N276(()); N138 --> N277(()); N139 --> N278(()); N139 --> N279(()); N140 --> N280(()); N140 --> N281(()); N141 --> N282(()); N141 --> N283(()); N142 --> N284(()); N142 --> N285(()); N143 --> N286(()); N143 --> N287(()); N144 --> N288(()); N1
```



# Lazy constraint callback

## CPX\_CALLBACKCONTEXT\_CANDIDATE

- Automatically invoked when a solution is going to update the **incumbent** (meaning it is **integer** and **feasible** w.r.t. current model, often coming from an internal primal heuristic)
- This is the **last checkpoint** where we 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, we 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)



# Usercut callback

## CPX\_CALLBACKCONTEXT\_RELAXATION

- Automatically invoked at every B&B node when the current solution is **noninteger** (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 noninteger (heuristic approaches can be used)
- User cuts are **not mandatory** for B&C correctness → being too clever on them can actually **slow-down** the solver because of the overhead in generating and using them (larger/denser LPs etc.)



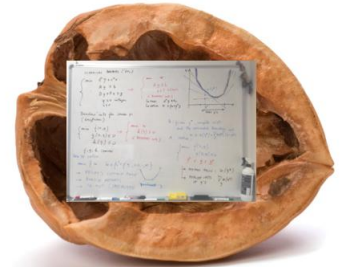
# Other callbacks

- **Branch callback:** invoked at the end of each node (even when the LP solution is integer and apparently does not require any cut/branching) and used to impose/customize branching
- **Heuristic callback:** used to build new (possibly problem-specific) feasible integer solutions to be **posted**, i.e., passed to the solver which will use them (at the appropriate time) to possibly update the incumbent
- etc. etc.

# But... how do we generate cuts?

- **Problem-specific cuts**
- **General-purpose** MIP cuts trying to enforce integrality (e.g. Gomory cuts)
  - Modern solvers already have a lot of them, implemented in the most effective way...
- Cuts coming from an alternative **extended formulation**
  - **Benders cuts**
  - ...
- Cuts dealing with nonlinearities
  - **Intersection cuts** (e.g., for **bilevel optimization**)
  - ...

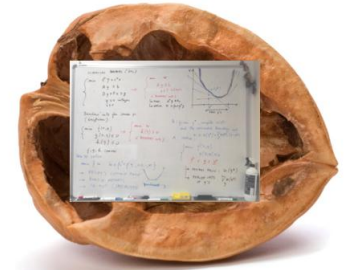
# Benders' cuts for dummies



- The original Benders' paper from the '60s uses **two** distinct ingredients for solving a Mixed-Integer Linear Program (MILP):
  - 1) A **cut loop strategy** where a relaxed (**NP-hard**) MILP 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)



# Benders cuts for dummies



- **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  
→ **lazycuts**
  - McDaniel & Devine (1977)
    - use Benders cuts to cut **fractional sol.s** as well  
→ **usercuts**
- Everything fits very naturally within a modern **Branch-and-Cut** (B&C) framework!

# Modern Benders

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

$$\min f(x, y)$$

$$g(x, y) \leq 0$$

$$Ay \leq b$$

$$y \text{ integer}$$

Warning: the  
important var.s  
are the  $y$ 's !

- For the sake of simplicity, assume that:
  - the set  $S := \{y : Ay \leq b\}$  is nonempty and bounded
  - the **convex function**

$$\Phi(y) := \min_x f(x, y)$$

$$g(x, y) \leq 0$$

is well defined for all  $y \in S$

# Working on the y-space (projection)

(1)

$$\min_y \min_x f(x, y)$$

$$g(x, y) \leq 0$$

$$Ay \leq b$$

$y$  integer

(2)

“isolate the inner minimization over  $x$ ”

$$\Phi(y) := \min_x f(x, y)$$

$$g(x, y) \leq 0$$

(3)

$$\min \Phi(y)$$

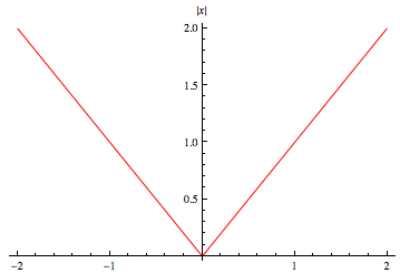
$$Ay \leq b$$

$y$  integer

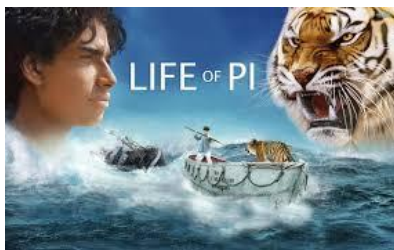
**Original** MINLP in the  $(x, y)$  space  $\rightarrow$  Benders' **master** problem in the  $y$  space

**Warning:** projection changes the objective function (e.g., linear  $\rightarrow$  convex nonlinear)

$$\begin{aligned} \min x \\ x &\geq y \\ x &\geq -y \\ y &\in [-1, 1] \end{aligned}$$



$$\begin{aligned} \min \Phi(y) &= |y| \\ y &\in [-1, 1] \end{aligned}$$



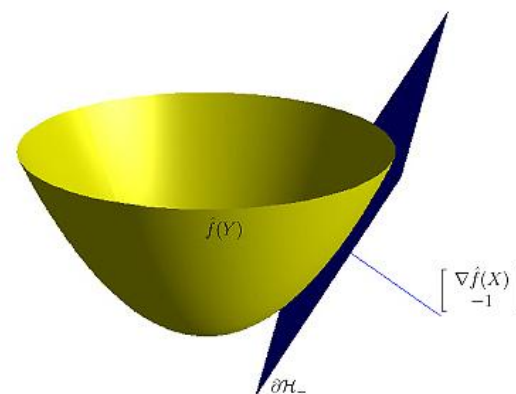
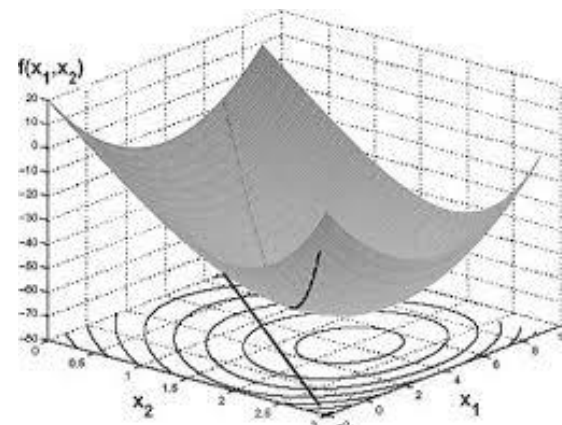
# Life of P(H)I

- 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  $\rightarrow$

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



# Benders cut computation

- **Benders** (for linear) and **Geoffrion** (general convex) use Linear/Lagrangian duality to compute a **subgradient** to be used in the cut derivation
- Given an optimal primal-dual solution  $(x^*, u^*)$  available after computing  $\Phi(y^*)$ , a subgradient of  $\Phi(y)$  in  $y^*$  is computed as
$$\begin{array}{l} \Phi(y) := \min_x f(x, y) \\ g(x, y) \leq 0 \end{array} \quad \rightarrow \quad \xi(y^*) = \nabla_y f(x^*, y^*) + u^* \nabla_y g(x^*, y^*)$$
- The above formula is **problem-specific** and sometimes cumbersome
- This is maybe the reason why Benders cuts are considered “too sophisticated” by students

# 1-2-3: Benders!

- But ... you can kindly ask your solver to make all calculations for you!

$$\min f(x, y)$$

$$g(x, y) \leq 0$$

- Here is the **recipe**:

$$Ay \leq b$$

- 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  $w \geq \text{opt\_val} + \sum_j r_j (y_j - y_j^*)$

# Benders feasibility cuts

- For some important applications, the set

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

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

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

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

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

where the function  $\Psi(y)$  is **convex**

$\rightarrow$  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 using reduced costs as before

# Successful Benders applications

- Benders cuts work 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**

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

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



# Benders cuts instability

- 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.

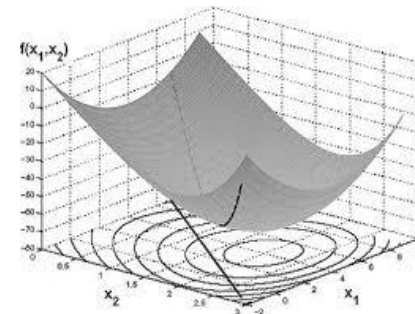
- Find an optimal **vertex**  $y^*$  of the current LP relaxation
- Invoke a separation function on  $y^*$ , add the returned violated cut (if any) to the current LP, and repeat

# Benders cuts instability

- This cut loop can be very **ineffective** in the **first iterations** when few Benders cuts have been generated, and the system  $Ay \leq b$  contains just few constraints (often, only variable bounds)

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

- In this situation:
  - the current master sol.  $y^*$  is almost unconstrained
    - **zig-zagging phenomenon**
  - Benders cuts convey information around “erratic” points  $y^*$  far from the region of interest



→ **Stabilization is needed**, e.g. through **Frank-Wolfe** cut loop

# 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**

**Benders** implemented in CPLEX **general** MIP solver since version 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", Management Science 63 (7), 2146-2162, 2017.