#post_modern Branch-and-Cut Implementation

Matteo Fischetti, University of Padova

ISMP 2018, Bordeaux, July 6, 2018

Why bothering about implementations at ISMP?

ISMP2018
BORDEAUX
23rd International Symposium on Mathematical Programming | July 1-6

MAIN MENU
- Home
- JuMP Workshop
- Poster ISMP 2018
- Gallery
- Clusters and Streams
- Schedule
- Program ▼
- Social Events ▼
- Prize Awards
- Committees ▼
- Attendees
- Submissions ▼
- Sponsors & Exhibits ▼
- Important Dates

Clusters and Streams

Cluster 1: Discrete Optimization and Integer Programming

a. **IPtheory**: Integer Programming Theory (Polyhedral Study, Lattices, Extented Formulations...)
b. **IPpractice**: Integer Programming Algorithms (Branch-and-cut, Reformulations and Decomposition, ...)
c. **MINLP**: Mixed Integer Non-linear Programming
d. **APPROX**: Complexity, Approximation and Online Algorithms
e. **COMB**: Combinatorial Optimization and Graph Theory
f. **CP**: Constraint Programming

Cluster 2: Optimization under Uncertainty

a. **Stoch:** Stochastic Optimization
b. **Robust:** Robust Optimization
c. **Markov:** Dynamic Programming, Markov Decision Processes, and Simulation
d. **Game:** Game theory, Bi-level and Multi-Objective Optimization

Cluster 3: Continuous Optimization

a. **NLP:** Linear and Nonlinear Optimization, Sparse Optimization and Applications
b. **Global:** Global Optimization
c. **NonSmooth:** Nonsmooth Optimization
d. **SDP:** Conic Programming, Quadratic Programming and Semi-Definite Programming
e. **Variat** : Variational Analysis, Variational Inequalities and Complementarity.
f. **RandomM**: Random Methods for Continuous Optimization (Stochastic Gradient, ...)
g. **DerFree:** Derivative-free and Simulation-based Optimization
h. **Control:** Optimal Control, PDE Constrained Optimization, and Multi-level Methods

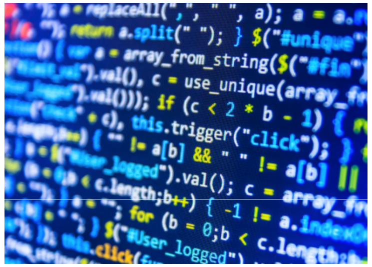
Cluster 4: Problem Specific Models, Algorithm Implementations, and Software

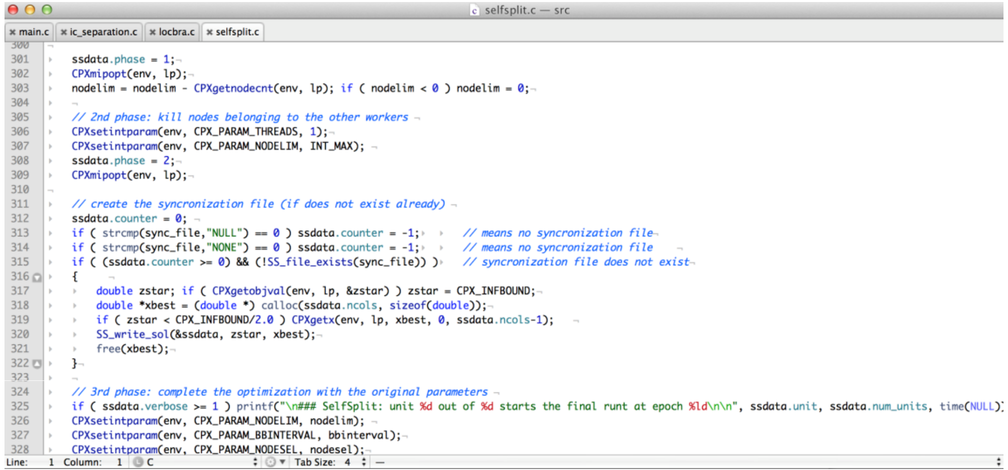
a. **Learning:** Machine Learning, Big Data, Cloud Computing, and Huge-Scale Optimization
b. **Network**: Network Flow, Network Design, and Applications in Telecom and Traffic Management
c. **Logistics:** Packing, Logistics, Location, and Routing
d. **Scheduling:** Scheduling, Planning and Applications in Manufacturing Systems and Healthcare
e. **Energy:** Optimization for Environmental, Energy and Engineering Systems
f. **Sciences:** Optimization in Sciences, Computational Biology, Societal Issues, Finance, and Economics

g. **Algo:** Math Programming Algorithm Implementations, Parallel Computing, and Software

Why bothering about implementations at ISMP?

- Implementation
  is **not** just coding!





- Needed if we **#orms** want to have an impact in practical applications

- Ask yourself: would Artificial Intelligence (notably: deep learning) be so successful without gradient-descent **algorithms** served with their efficient **#backpropagation** **implementations**?

Algorithms without implementation

Theorem 2 *Assume w.l.o.g. that* $rank(A) = n$. *Given a vertex* x^* *of* P, *let the system* $Ax \geq b$ *be partitioned into* $Bx \geq b_B$ *and* $Nx \geq b_N$, *where* $Bx^* = b_B$ *and* B *is an* $n \times n$ *nonsingular matrix. Let* (u_B, v_B) *and* (u_N, v_N) *denote the Farkas multipliers associated with the rows of* B *and* N, *respectively. For a given disjunction* (2) *with* $\eta^* = \pi x^* - \pi_0 \in [0, 1]$, *let* $u_0^* = 1 - \eta^*$, $v_0^* = \eta^*$, $u_N^* = v_N^* = 0$, $u_B^* = [\pi B^{-1}]_+$ *and* $v_B^* = [-\pi B^{-1}]_+$, *while* γ^* *and* γ_0^* *are defined through* (4) *and* (6), *respectively. Then* $(\gamma^*, \gamma_0^*, u^*, v^*, u_0^*, v_0^*)$ *is an optimal CGLP solution w.r.t. the trivial normalization* (10).

Proof We first prove feasibility. Consistency between (4) and (5) requires $u^* A - u_0^* \pi = v^* A + v_0^* \pi$, i.e., $u_B^* - v_B^* = (u_0^* + v_0^*)\pi B^{-1} = \pi B^{-1}$, which follows directly from the definition of u_B^* and v_B^*. Analogously, consistency between (6) and (7) requires $(u_B^* - v_B^*) b_B = (u_0^* + v_0^*)\pi_0 + v_0^*$, i.e., $\pi B^{-1} b_B = \pi_0 + v_0^*$. This latter equation is indeed satisfied because $B^{-1} b_B = x^*$ and $v_0^* = \eta^* = \pi x^* - \pi_0$. As to optimality, we observe that $u_0^* + v_0^* = 1$ holds by definition. Because of (4) and (6), $\gamma x^* - \gamma_0 = u^*(Ax^* - b) - u_0^*(\pi x^* - \pi_0) = u_B^*(Bx^* - b_B) + u_N^*(Nx^* - b_N) - u_0^* \eta^* = 0 + 0 - (1 - \eta^*)\eta^*$, hence the cut violation attains bound UB3 of Lemma 1. □

Algorithms without implementation

Theorem 2 *Assume w.l.o.g. that* $rank(A) = n$*. Given a vertex* x^* *of* P*, let the system* $Ax \geq b$ *be partitioned into* $Bx \geq b_B$ *and* $Nx \geq b_N$*, where* $Bx^* = b_B$ *and* B *is an* $n \times n$ *nonsingular matrix. Let* (u_B, v_B) *and* (u_N, v_N) *denote the Farkas multipliers associated with the rows of* B *and* N*, respectively. For a given disjunction* (2) *with* $\eta^* = \pi x^* - \pi_0 \in [0, 1]$*, let* $u_0^* = 1 - \eta^*$, $v_0^* = \eta^*$, $u_N^* = v_N^* = 0$, $u_B^* = [\pi B^{-1}]_+$ *and* $v_B^* = [-\pi B^{-1}]_+$*, while* γ^* *and* γ_0^* *are defined through* (4) *and* (6)*, respectively. Then* $(\gamma^*, \gamma_0^*, u^*, v^*, u_0^*, v_0^*)$ *is an optimal CGLP solution w.r.t. the trivial normalization* (10).

Proof: omitted as of no interest to the typical MP reader.

Describing an **Algorithm** without **Implementation** is like stating a **Theorem** without **Proof**

#just_a_computational_conjecture

Branch & Cut ™

SIAM REVIEW
Vol. 33, No. 1, pp. 60–100, March 1991

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

MANFRED PADBERG† AND GIOVANNI RINALDI‡

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

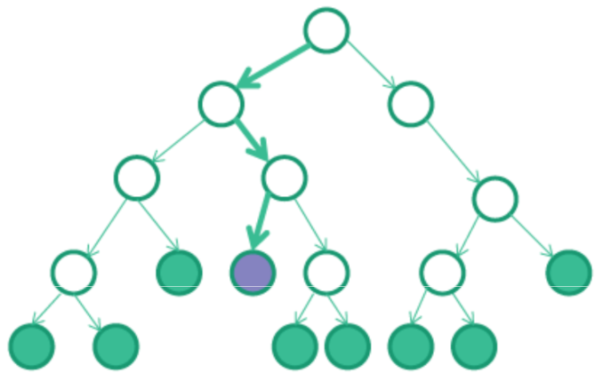
Theorem. *Using cuts within an enumerative scheme is good*.

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 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
  – …

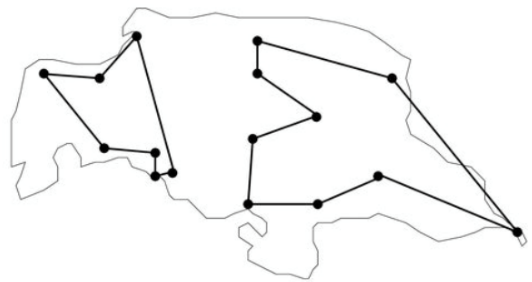
Lazy constraint callback

- 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 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)



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 → 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
- **Incumbent callback**: invoked just before updating the incumbent (after the lazy constraint callback) to possibly kill a solution without providing any violated cut
- **Heuristic callback**: used to build new (possibly problem-specific) feasible integer solutions
- **Informative**: to just compute/print internal statistics
- etc. etc.

Application: non-convex MIQP

(based on ongoing work with Michele Monaci, U. Bologna,
and Domenico Salvagnin, U. Padova)

- **Goal: implement a Mixed-Integer (non-convex) Quadratic solver**

- Two approaches:

  1. start with a continuous QP solver and add enumeration on top of it
  → implement B&B to handle integer var.s

  2. start with a MILP solvers (B&C) and customize it to handle the
  non-convex quadratic terms → add McCormick & spatial branching

     PROS: …

     CONS: …

MIQP as a MILP with bilinear eq.s

- The fully-general MIQP of interest reads

$$
(MIQP) \qquad \begin{aligned}
&\min a_0^T x + x^T Q^0 x \\
&a_k^T x + x^T Q^k x \,@\, b, && k = 1, \ldots, m \\
&\ell_j \le x_j \le u_j, && j = 1, \ldots, n \\
&x_j \text{ integer}, && j \in \mathcal{I}, \\
&x_j \text{ continuous}, && j \in \mathcal{C},
\end{aligned}
$$

and can be restated as

$$
(MIBLP) \qquad \begin{aligned}
&\min_x c^T x \\
&Ax = b \\
&\ell_j \le x_j \le u_j, && j = 1, \ldots, n \\
&x_j \text{ integer}, && j \in \mathcal{I} \\
&x_j \text{ continuous}, && j \in \mathcal{C} \\
&x_{r_k} = x_{p_k} x_{q_k}, && k = 1, \ldots, K,
\end{aligned}
$$

McCormick inequalities

- To simplify notation, rewrite the generic bilevel eq. $x_{r_k} = x_{p_k} x_{q_k}$ as:

$$z = x\,y$$
$$\ell_x \le x \le u_x$$
$$\ell_y \le y \le u_y$$

- Obviously

$$\begin{aligned} (x-\ell_x)(y-\ell_y) &\ge 0 \\ (x-u_x)(y-u_y) &\ge 0 \\ (x-\ell_x)(y-u_y) &\le 0 \\ (x-u_x)(y-\ell_y) &\le 0 \end{aligned} \quad \rightarrow \quad \begin{array}{lll} \text{mc1)} & z & \ge \ell_y x + \ell_x y - \ell_x \ell_y \\ \text{mc2)} & z & \ge u_y x + u_x y - u_x u_y \\ \text{mc3)} & z & \le u_y x + \ell_x y - \ell_x u_y \\ \text{mc4)} & z & \le \ell_y x + u_x y - u_x \ell_y \end{array}$$

(just replace *xy* by *z* in the products on the left)

- Note: *mc1)* and *mc2)* can be improved in case *x=y* → **gradients cuts**

$$z \ge x_0^2 + 2x_0(x - x_0), \quad \text{for each } x_0 \in \Re$$

Spatial branching

- McCormick inequalities are not perfect
  → they are tight only when *x* and/or *y* are at their lower/upper bound

$$(x-\ell_x)(y-\ell_y) \geq 0$$
$$(x-u_x)(y-u_y) \geq 0$$
$$(x-\ell_x)(y-u_y) \leq 0$$
$$(x-u_x)(y-\ell_y) \leq 0$$

→ at some B&C nodes, it may happen that the current (fractional or integer) solution satisfies **all** MC inequalities but some bilinear eq.s $z = xy$ are still violated (we call this **#bilinear_infeasibility**)

→ we need a **bilinear-specific branching** (the usual MILP branching on integrality does not work if all var.s are integer already)

- **Spatial branching**: if $z^* = x^* y^*$ is an offended bilinear eq., branch on

(x ≤ x*) OR (x ≥ x*)

to make the upper (resp. lower) bound on *x* tight at the left (resp. right) child node – thus improving the corresponding MC inequality

Vanilla B&C implementation

- **Lazy constraint callback**: separation of MC inequalities
- **Usercut callback**: not needed (and sometimes detrimental)
- **Branch callback:** spatial branching on the "most offended" $z = xy$
- **Incumbent callback**: very-last resort to kill a bilinear-infeasible integer solution (when everything else fails e.g. because of tolerances)
- **Precision**: LP precision higher (more restrictive) than bilinear tolerance
- **MILP heuristics** (kindly provided by the MILP solver): active at their default level
- **MIQP-specific heuristics**: not implemented

- Implemented but not used in the vanilla version:
  - additional bilinear-specific cuts → Balas' **Intersection Cuts (ICs)**
  - **semi-spatial** branching (branch threshold $x^*+\delta$ → x^* violates the x-bound in one of the two children, MC only needed in the other one)

Does it work?

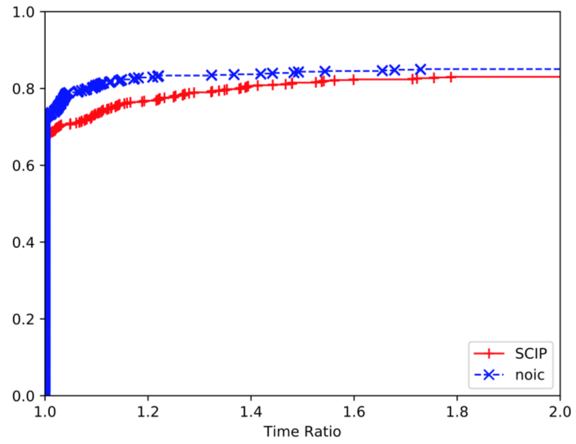
- Comparison with the SCIP 5.0 MIQP solver using CPLEX 12.8 as **LP solver** + internal nonlinear solver

- Preliminary test on the quadratic **MINLPlib** (700+ instances) …
  … but some instances removed as root LP was **unbounded**
  → they need bound tightening by preprocessing (TODO)

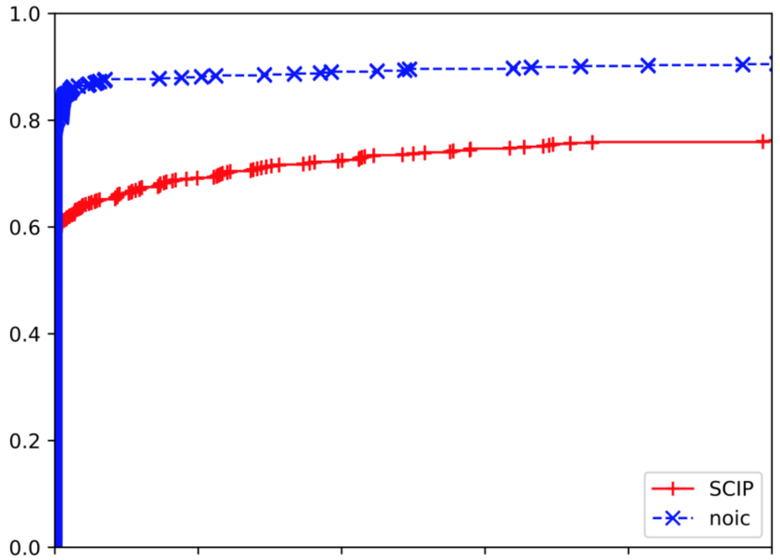
- Results of our B&C callback-based vanilla implementation using CPLEX 12.8 as **MILP solver**; 1-thread runs (parallel runs not allowed in SCIP); only instances solved by both codes in the 1-hour time limit.
  – Overall, we are as fast as SCIP (but the latter solves more instances within the time limit → SCIP qualifies as a more robust solver).
  – We are 2 to 10 times faster than SCIP when the optimal/best-known solution from MINLPlib is used as a warm-start for both codes → evidently, we miss a sound bilinear-specific heuristic (TODO)

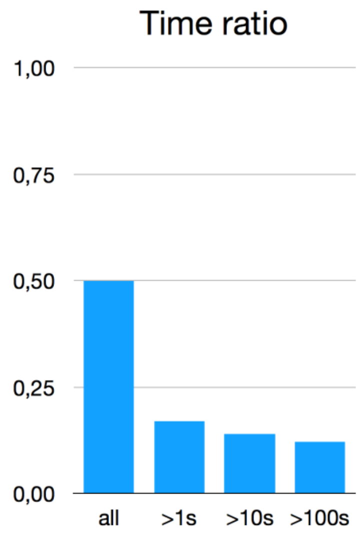
More detailed comparison

- **SCIP** vs **noic** (our "vanilla" version with no ICs and classical spatial branching) →



- Results with incumbent warm-start (only instances solved by both codes)





Thanks for your attention!

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