

Intersection Cuts for Quadratic Mixed-Integer Optimization

Matteo Fischetti, University of Padova

Michele Monaci, University of Bologna

ODS 2019

INTERNATIONAL CONFERENCE ON
OPTIMIZATION AND DECISION SCIENCE

XLIX ANNUAL MEETING OF AIRO - ITALIAN OPERATIONS RESEARCH SOCIETY

GENOVA, ITALY, SEPTEMBER 4-7, 2019



ODS 2019, Genova, September 4-7, 2019

Non-convex MIQP

- **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: ...
- This talk goes for 2.

MIQP as a MILP with bilinear eq.s

- The fully-general MIQP of interest reads

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

and can be restated as

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

McCormick inequalities

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

$$z = x y$$

$$\ell_x \leq x \leq u_x$$

$$\ell_y \leq y \leq u_y$$

- Obviously

$$\begin{array}{ll}
 (x - \ell_x)(y - \ell_y) \geq 0 & \text{mc1)} \quad z \geq \ell_y x + \ell_x y - \ell_x \ell_y \\
 (x - u_x)(y - u_y) \geq 0 & \rightarrow \text{mc2)} \quad z \geq u_y x + u_x y - u_x u_y \\
 (x - \ell_x)(y - u_y) \leq 0 & \text{mc3)} \quad z \leq u_y x + \ell_x y - \ell_x u_y \\
 (x - u_x)(y - \ell_y) \leq 0 & \text{mc4)} \quad z \leq \ell_y x + u_x y - u_x \ell_y
 \end{array}$$

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

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

$$z \geq x_0^2 + 2x_0(x - x_0), \quad \text{for each } x_0 \in \mathbb{R}$$

Spatial branching

- McCormick inequalities are not perfect
→ they are tight only when x and/or y are at their lower/upper bound
- $$\begin{aligned}(x - \ell_x)(y - \ell_y) &\geq 0 \\(x - u_x)(y - u_y) &\geq 0 \\(x - \ell_x)(y - u_y) &\leq 0 \\(x - u_x)(y - \ell_y) &\leq 0\end{aligned}$$

→ 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)

- **Standard Spatial Branching**: if $z^* = x^* y^*$ is violated, branch on **$(x \leq x^*)$ OR $(x \geq x^*)$**
to make the upper (resp. lower) bound on x tight at the left (resp. right) child node – thus improving the corresponding MC inequality

A new branching rule

- **Shifted Spatial Branching**: let $\rho^* := z^* - x^* y^*$; if $\rho^* > 0$, branch on

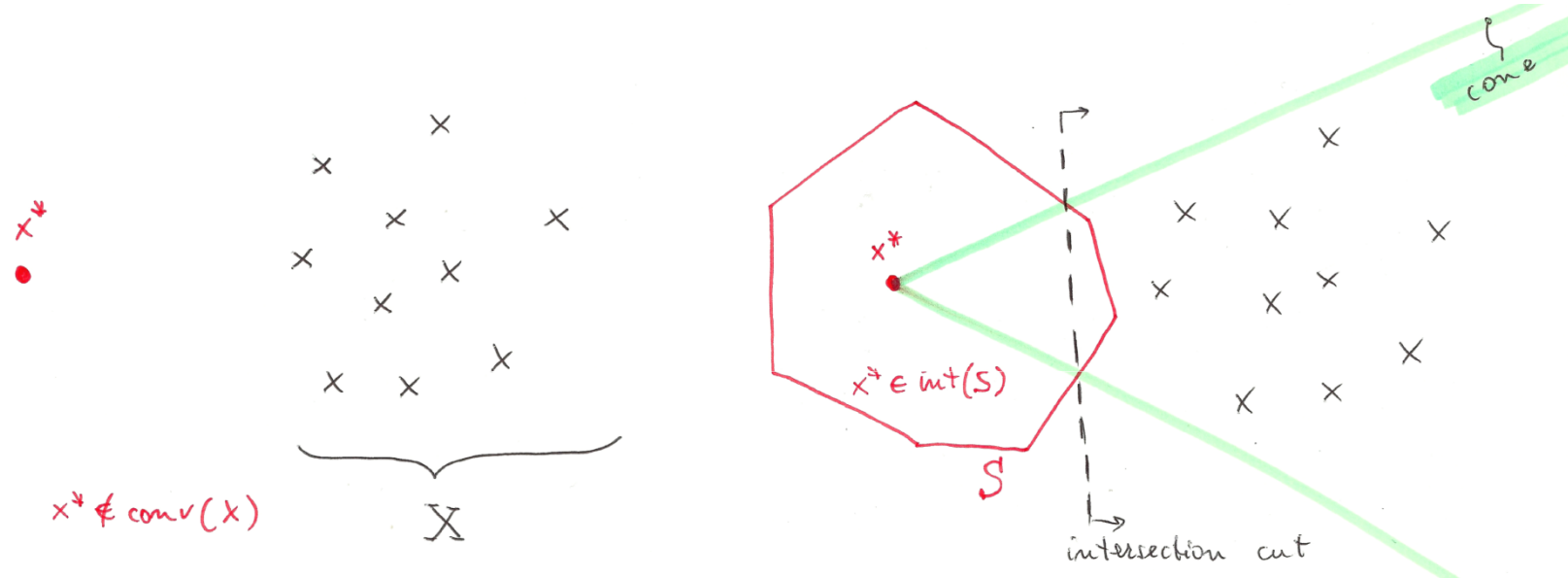
$$(x \leq x^* - \delta) \text{ OR } (x \geq x^* - \delta)$$

where δ is defined so as to **balance** the violation of the two child nodes (case $\rho^* < 0$ is similar)

- **Left branch** ($u_x = x^* - \delta$) \rightarrow violation of δ of the upper bound u_x
- **Right branch** ($l_x = x^* - \delta$) \rightarrow violation of δ for the MC ineq.
 $(x - x^* + \delta)(y - u_y) \leq 0$ by choosing $\delta = \rho^* / (1 + u_y - y^*)$
- **New Branching Rule**: among all violated $z^* = x^* y^*$, select the one maximizing the balanced violation δ

Intersection Cuts (ICs)

- **Intersection cuts** (Balas, 1971): a powerful tool to separate a point x^* from a set X by a liner cut



- All you need is (love, but also)
 - a **cone** pointed at x^* containing all $x \in X$
 - a **convex set S** with x^* (but no $x \in X$) in its interior
- If x^* **vertex** of an LP relaxation, a suitable cone comes for the **LP basis**

Bilinear-free sets

- **Observation:** given an infeasible point x^* , any branching disjunction violated by x^* implicitly defines a **convex set S** with x^* (but no feasible x) in its interior

$$\bigvee_{i=1}^k (g_i^T x \geq g_{i0}) \quad \rightarrow \quad S = \{x : g_i^T x \leq g_{0i}, \quad i = 1, \dots, k\}$$

- Thus, **in principle**, one could always generate an IC instead of branching \rightarrow not always advisable because of numerical issues, slow convergence, tailing off, cut saturation, etc. **#LikeGomoryCuts**
- **Candidate branching disjunctions** (supplemented by MC cuts) are the 1- and 2-level (possibly shifted) spatial branching conditions:

$$(x \leq x^*) \vee (x \geq x^*)$$

$$(x \leq x^*, y \leq y^*) \vee (x \leq x^*, y \geq y^*) \vee (x \geq x^*, y \leq y^*) \vee (x \geq x^*, y \geq y^*)$$

IC separation issues

- IC separation can be problematic, as we need to read the cone rays from the LP tableau → **numerical accuracy** can be a big issue here!
- For **MILPs**, ICs like Gomory cuts are **not mandatory** (so we can skip their generation in case of numerical problems), but for **MIBLPs** they are more instrumental **#SeparateOrPerish**
- **Notation:** consider w.l.o.g. an LP in standard form and no var. ub's

$\min\{\hat{c}^T \xi : \hat{A}\xi = \hat{b}, \xi \geq 0\}$ be the LP relaxation at a given node

$S = \{\xi : g_i^T \xi \leq g_{i0}, i = 1, \dots, k\}$ be the bilinear-free set

$\bigvee_{i=1}^k (g_i^T \xi \geq g_{i0})$ be the disjunction to be satisfied by all feas. sol.s

Numerically safe ICs

A **single** valid inequality can be obtained by taking, for each variable, the worst LHS Coefficient (and RHS) in each disjunction

$$\bigvee_{i=1}^k (g_i^T \xi \geq g_{i0})$$

$$\bigvee_{i=1}^k (\bar{g}_i^T \xi \geq \bar{g}_{i0})$$

To be applied to a **reduced form** of each disjunction where the coefficient of all basic variables is zero (kind of LP reduced costs)

$$\bigvee_{i=1}^k \left(\frac{\bar{g}_i^T}{\bar{g}_{i0}} \xi \geq 1 \right)$$

Algorithm 1: Intersection cut separation

Input : An LP vertex ξ^* along with its associated LP basis \hat{B} ;

the feasible-free polyhedron $S = \{\xi : g_i^T \xi \leq g_{i0}, i = 1, \dots, k\}$ and the associated valid disjunction $\bigvee_{i=1}^k (g_i^T \xi \geq g_{i0})$ whose members are violated by ξ^* ;

Output: A valid intersection cut violated by ξ^* ;

```
1 for  $i := 1$  to  $k$  do
2   |  $(\bar{g}_i^T, \bar{g}_{i0}) := (g_i^T, g_{i0}) - u_i^T(\hat{A}, \hat{b})$ , where  $u_i^T = (g_i)^T \hat{B}^{-1}$ 
3 end
4 for  $j := 1$  to  $n$  do  $\gamma_j := \max\{\bar{g}_{ij}/\bar{g}_{i0} : i \in \{1, \dots, k\}\}$ ;
5 return the violated cut  $\gamma^T \xi \geq 1$ 
```

B&C implementation

- Implementation using **IBM ILOG Cplex 12.8** using callbacks:
 - **Lazy constraint callback**: separation of MC inequalities for integer sol.s
 - **Usercut callback**: separation of fractional sol.s → not mandatory (and sometimes even detrimental)
 - **Branch callback**: spatial branching (in a new variants)
 - **Incumbent callback**: very-last resort to kill a bilinear-infeasible integer solution (when everything else fails e.g. because of tolerances)
 - **MILP heuristics** (kindly provided by the MILP solver): active at their default level
 - **MIQP-specific heuristics**: could be very useful, but not implemented yet

Computational analysis

- **Three** algorithms under comparison
 - ✓ **SCIP**: the general-purpose solver SCIP (vers. 5.0.1 using CPLEX 12.8 as LP solver + IPOPT 3.12.9 as nonlinear solver)
 - ✓ **basic**: our branch-and-cut algorithm without intersection cuts
 - ✓ **with-IC**: intersection cuts separated at each node where the LP solution is integral
- Single-thread runs (parallel runs not allowed in SCIP) with a time limit of **1 hour** on a standard PC Intel @ 3.10 GHz with 16 GB ram
- **Testbed**: all quadratic instances in **MINLPlib** (700+ instances) ...
... but some instances removed as root LP was **unbounded**
→ **620** instances left, **408** of which solved by all methods in 1 hour

Results

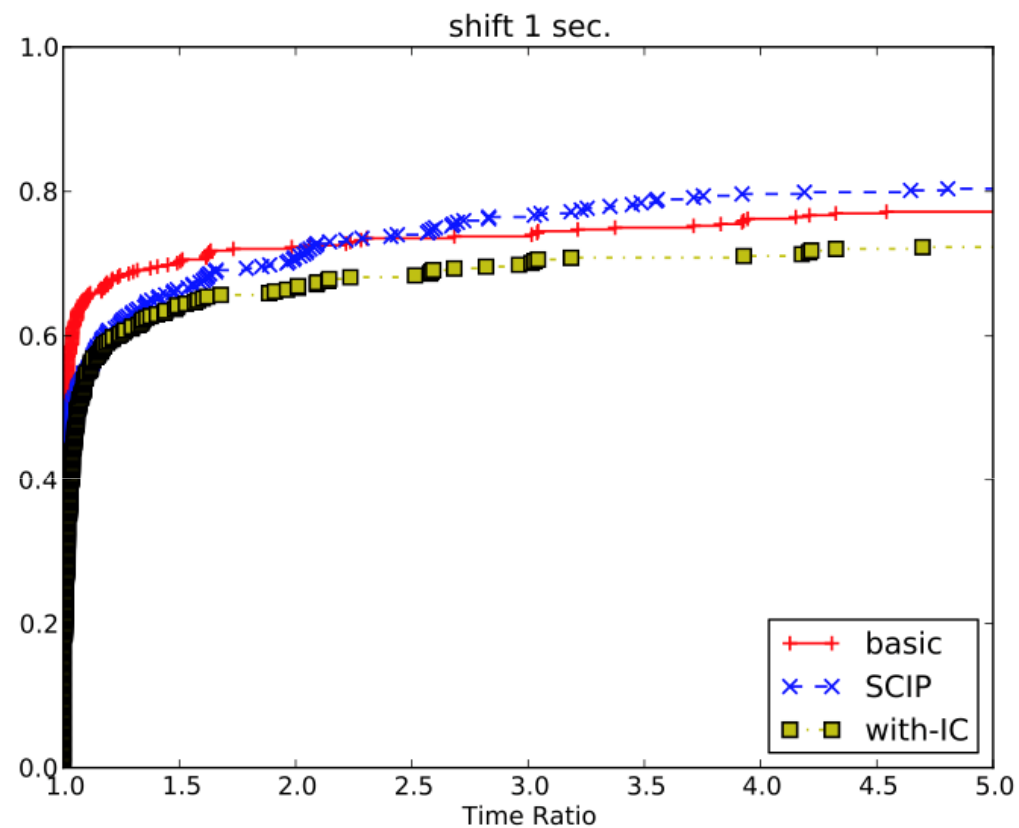


Figure 1: Performance profile comparison of **basic**, **SCIP** and **with-IC**, on the 408 MINLPlib instances that could be solved by at least one method in the 1-hour time limit (time shift of 1 sec.)

Results without small instances

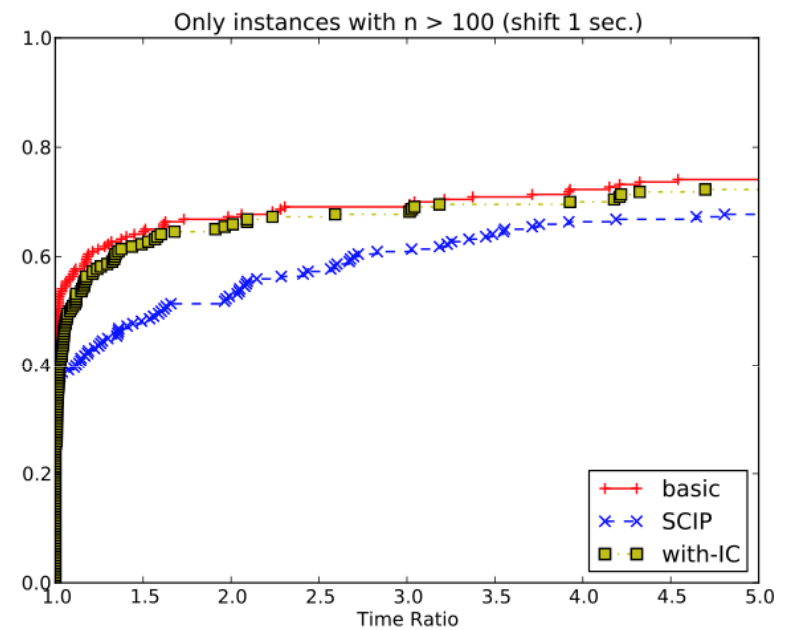
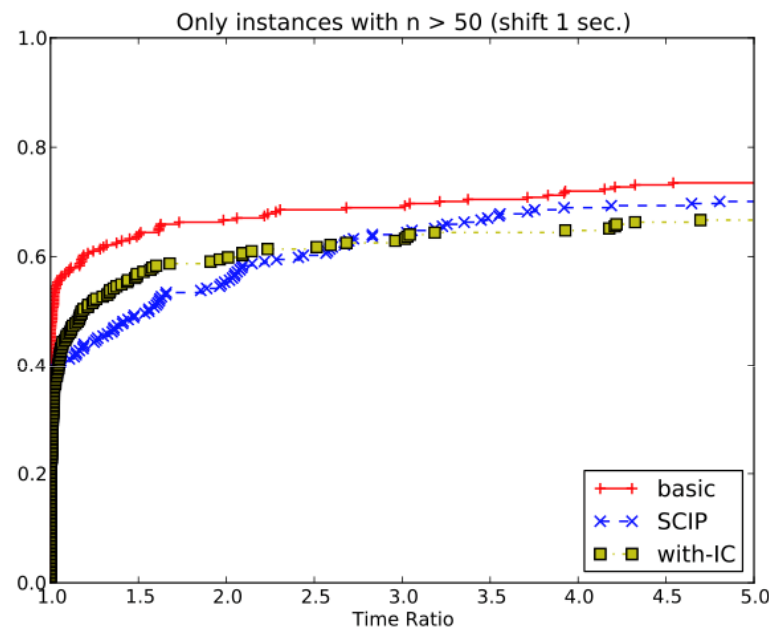


Figure 3: Performance profile comparison of basic, SCIP and with-IC as in Figure 1, when small instances are removed (time shift of 1 sec.)

ICs can make a difference

Instance	SCIP	basic	with-IC
blend531	234.21	3600.00	31.05
crudeoil_lee4_09	89.12	9.83	2.21
portfol_classical050_1	57.03	54.37	33.26
powerflow0009r	3600.00	3600.00	969.12
powerflow0014r	3600.00	3600.00	302.77
sporttournament14	3600.00	182.41	125.50
sqfl015-080	3600.00	238.53	137.32
sqfl025-030	3600.00	44.46	18.72
turkey	61.19	3600.00	0.11

Table 4: Selected instances for which adding intersection cuts is highly beneficial.

Thanks for your attention!

Paper available at <http://www.dei.unipd.it/~fisch/papers/>

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

Intersection cuts from bilinear disjunctions

Matteo Fischetti¹ and Michele Monaci²

¹ DEI, Università di Padova, via Gradenigo 6/A, 35100 Padova (Italy)

² DEI “Guglielmo Marconi”, Università di Bologna, Viale Risorgimento 2, 40136 Bologna (Italy)

e-mail: *matteo.fischetti@unipd.it*, *michele.monaci@unibo.it*

February 22, 2019; Revised, 27 June 2019

Abstract

In this paper we consider the Mixed-Integer Bilinear Programming problem, a widely-used reformulation of the classical mixed-integer quadratic programming problem. For this problem we describe a branch-and-cut algorithm for its exact solution, based on a new family of intersection cuts derived from bilinear-specific disjunctions. We computationally analyze the behavior of the proposed algorithm on a large set of mixed-integer quadratic instances from the MINLPlib problem library. Our results show that our method, even without intersection cuts, is competitive with a state-of-the-art mixed-integer nonlinear solver. As to intersection cuts, their extensive use at each branching node tends to slow down the solver for most problems in our test bed, but they are extremely effective for some specific instances.

Keywords: (O) mixed-integer quadratic programming, bilinear programming, branch-and-cut algorithms, intersection cuts, computational experiments.