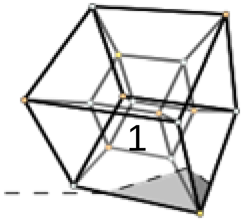
Gomory Reloaded

Matteo Fischetti, DEI, University of Padova

(joint work with Domenico Salvagnin)



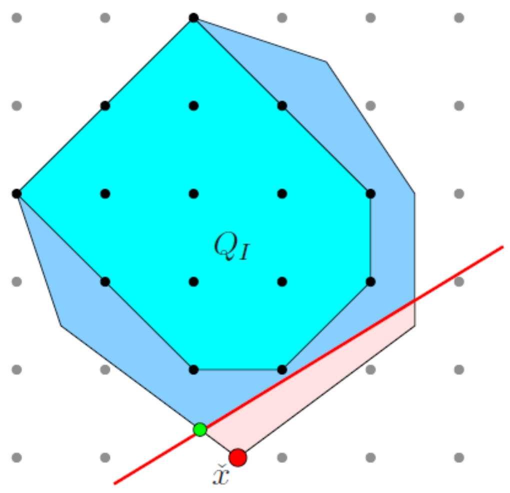
Cutting planes (cuts)

- We consider a general MIPs of the form

  $$\min \{ c\,x : A\,x = b,\ x \geq 0,\ x_j \textit{ integer} \text{ for some } j \}$$

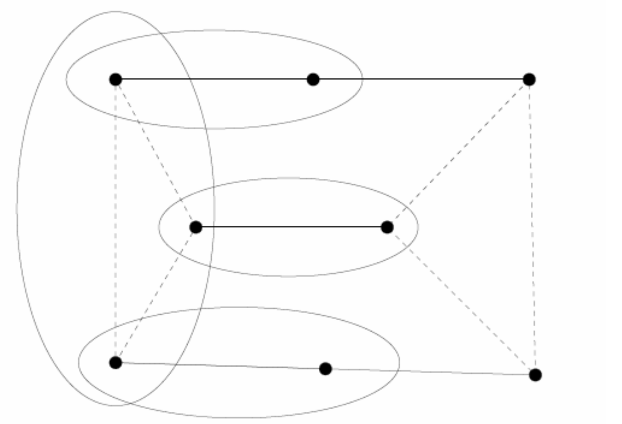
- **Cuts:** linear inequalities valid for the integer hull (but not for the LP relaxation)

- **Questions:**
  - **How to compute?**
  - Are they really useful?
  - If potentially useful, how to better use them?

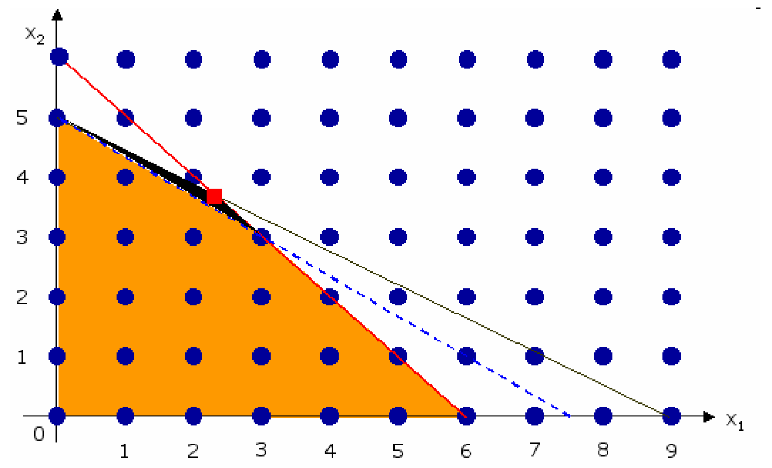


How to compute the cuts?

- **Problem-specific** classes of cuts (with nice theoretical properties)
  - Knapsack: cover inequalities, …
  - TSP: subtour elimination, comb, clique tree, …



- **General** MIP cuts only derived from the input model
  - Cover inequalities
  - Flow-cover inequalities
  - …
  - **Gomory Mixed-Integer Cuts (GMICs):** perhaps the most famous class of MIP cuts…



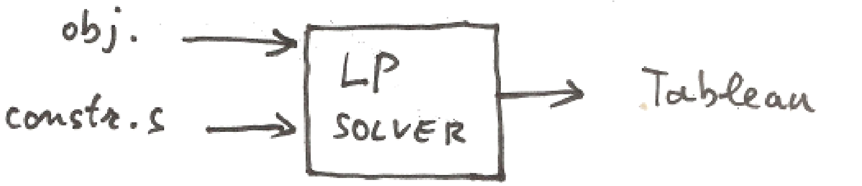
GMICs read from LP tableaux

- GMICs apply a simple formula to the coefficients of a starting equation
  - Q. How to define this **starting equation** (crucial step)?
  - A. The LP (optimal) tableau is plenty of equations, just use them!

		x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8
$-z$	$-\frac{25}{3}$	0	$\frac{4}{3}$	$\frac{19}{6}$	$\frac{9}{2}$	0	0	0	$\frac{7}{6}$
x_5	1	0	1	$-\frac{1}{2}$	$-\frac{3}{2}$	1	0	0	$\frac{3}{2}$
x_1	$\frac{11}{3}$	1	$-\frac{2}{3}$	$-\frac{1}{3}$	0	0	0	0	$\frac{2}{3}$
x_6	$\frac{2}{3}$	0	$\frac{1}{3}$	$\frac{1}{6}$	$-\frac{1}{2}$	0	1	0	$\frac{7}{6}$
x_7	1	0	-3	$\frac{1}{2}$	$\frac{9}{2}$	0	0	1	$-\frac{15}{2}$

The two story characters

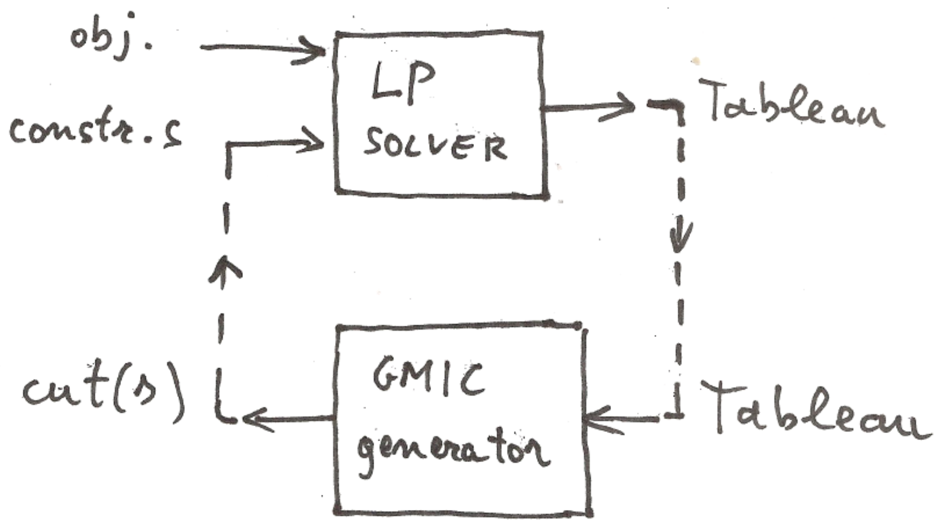
- **The LP solver (beauty?)**
  - Input: a set of linear constraints & objective function
  - Output: an optimal LP tableau (or basis)



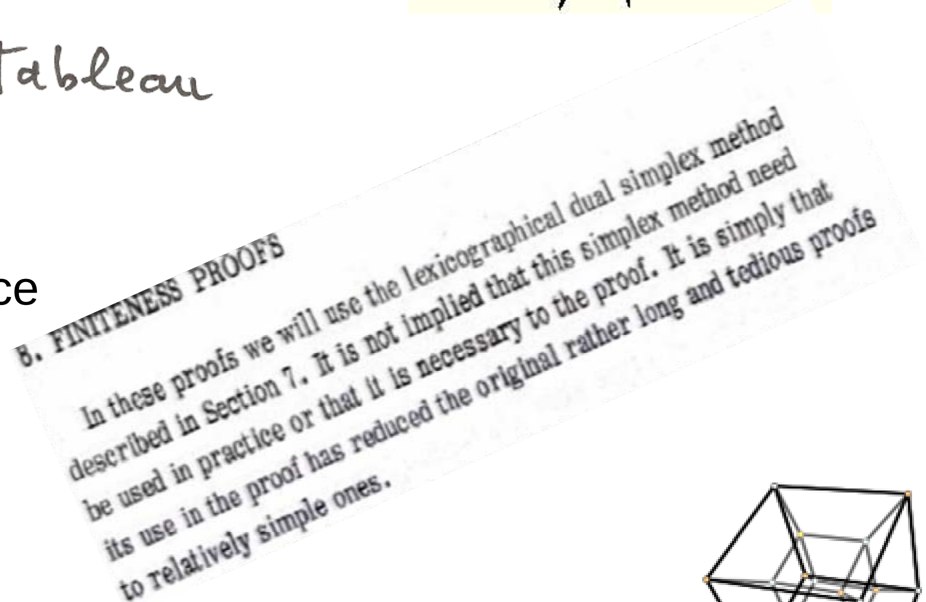
- **The GMIC generator (the beast?)**
  - Input: an LP tableau (or a vertex x* with its associated basis)
  - Output: a *round* of GMICs (potentially, one for each tableau row with fractional right-hand side)

How to combine the two modules?

- A natural (??) **interconnection scheme** (Kelley, 1960):



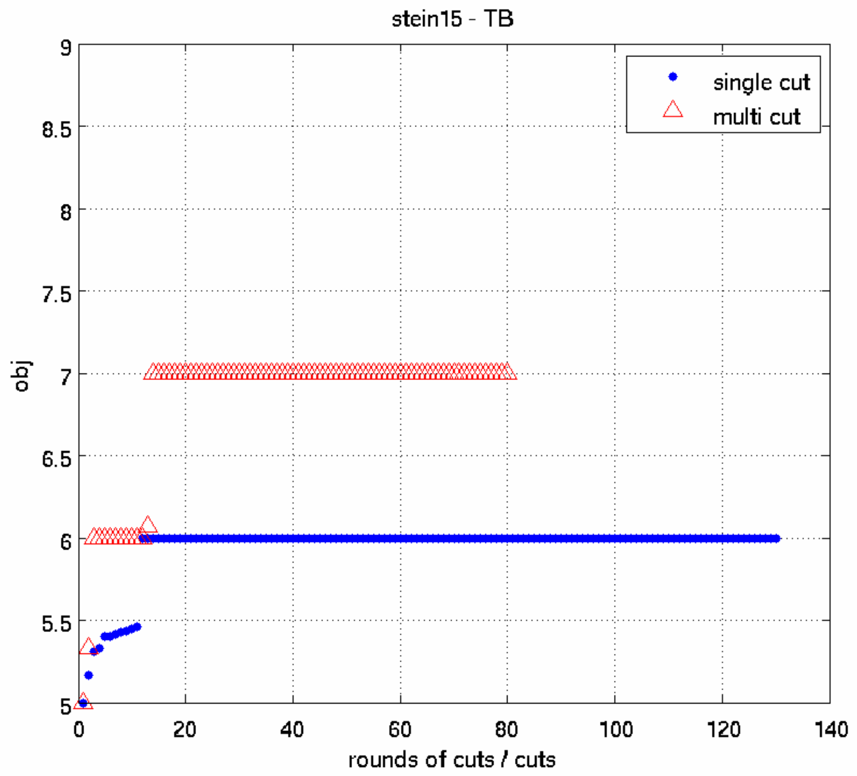
- In theory, this scheme **should** produce a finitely-convergent cutting plane scheme, i.e., an exact solution alg. only based on cuts (no branching)



8. FINITENESS PROOFS

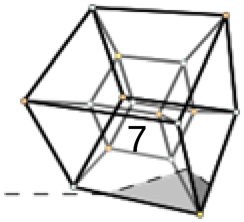
In these proofs we will use the lexicographical dual simplex method described in Section 7. It is not implied that this simplex method need be used in practice or that it is necessary to the proof. It is simply that its use in the proof has reduced the original rather long and tedious proofs to relatively simple ones.

In theory, but … in practice?

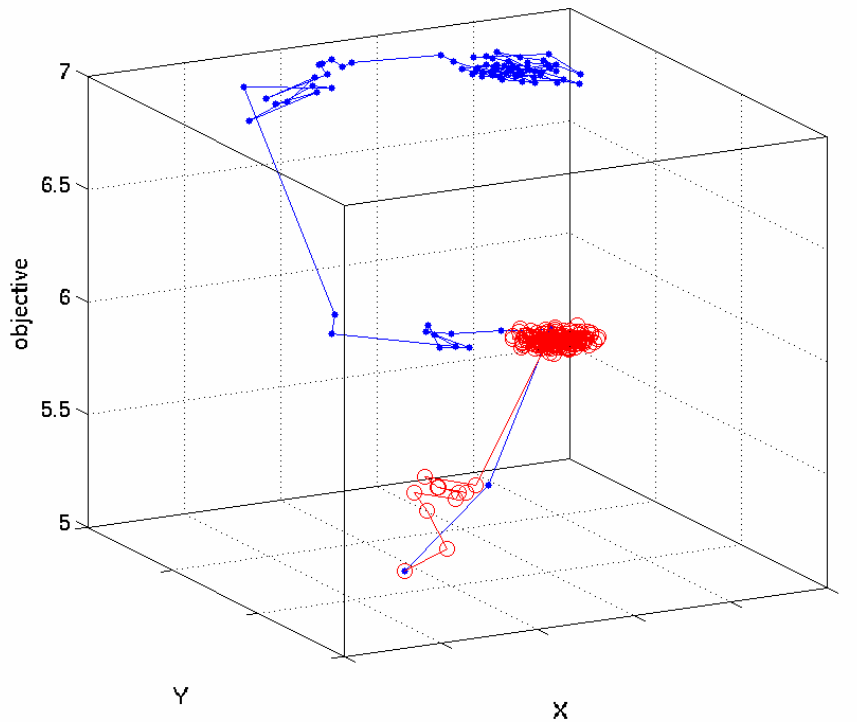


- ***Stein15***: toy set covering instance from MIPLIB
  - LP bound = **5**
  - MIP optimum = **8**

- multi cut generates ***rounds*** of cuts before each LP reopt.



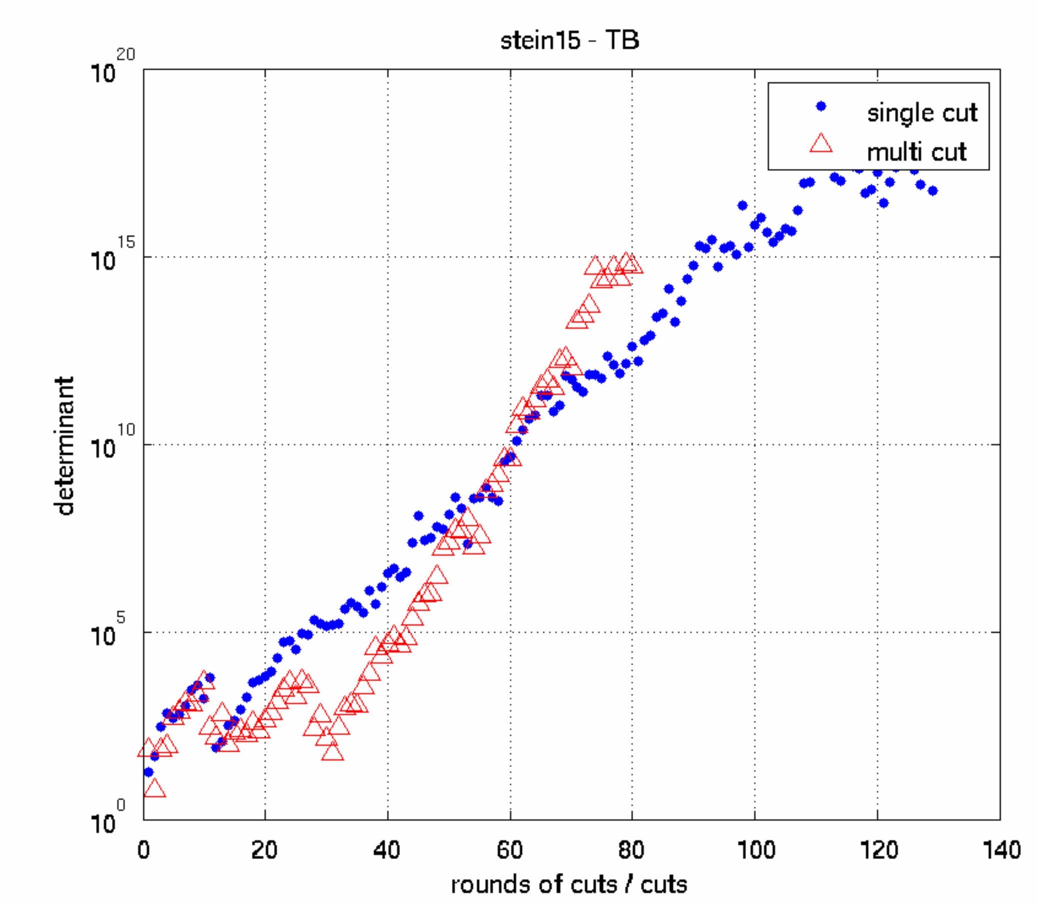
The black-hole effect



(X,Y) = 2D representation of the x-space (multidimensional scaling)

• Plot of the LP-sol. trajectories for **single-cut** (red) and **multi-cut** (blue) versions (multidimensional scaling) → **Both versions collapse after a while: why?**

LP-basis determinant and saturation



Exponential growth → unstable behavior!

Intuition about saturation

- Cuts work reasonably well on the initial LP polyhedron

… however they create **artificial vertices**
… that tend to be very close one to each other
… hence they differ by small quantities and
have “weird entries”
→ very like using a smoothing plane on wood

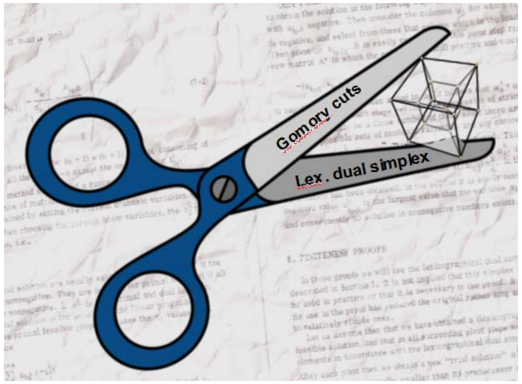
- LP theory tells that small entries in LP basic sol.s x*

… require a **large basis determinant** to be described
… and large determinants amplify the issue and create numerically unstable tableaux

- Kind of **driving a car on ice** with flat tires :
  - Initially you have some grip
  - … but soon wheels warm the ice and start sliding
  - … and the more gas you give the worse!

Gomory's convergent method

- For pure integer problems (with all-integer data) Gomory proved the existence of a finitely-convergent solution method only based on cuts, but one has to follow a **rigid recipe**:
  - use **lexicographic** optimization (a must!)
  - use the **objective function** as a source for GMICs

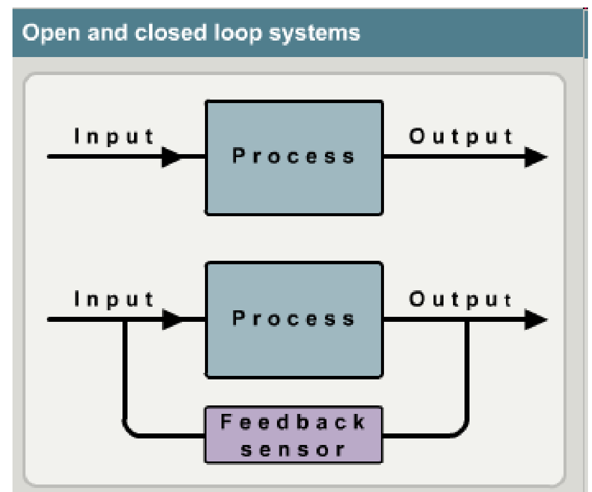


- **Finite convergence**
  guaranteed by an enumeration scheme hidden in **lexicographic** reoptimization: engineers would say that this adds "anti-slip **chains"** to Gomory's wheels (mathematicians would say "polar granularity" instead)

  → **safe but slow (like driving on a highway with chains…)**

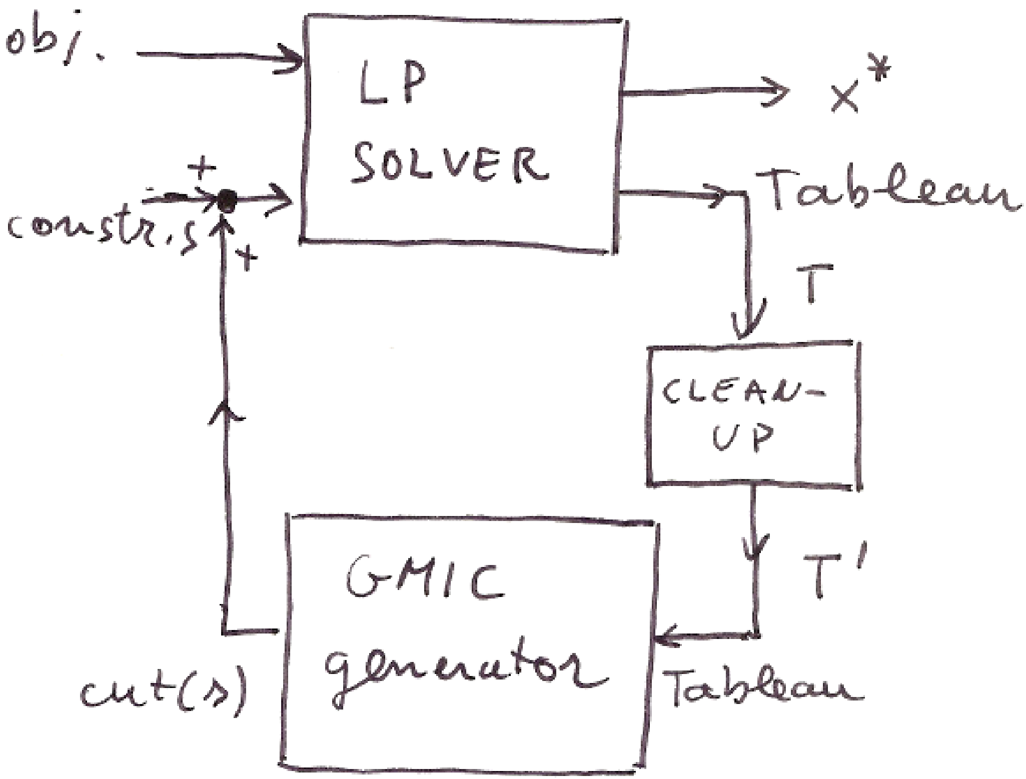
So, what is wrong with GMICs?

- GMICs are not necessarily bad in the long run
- What is problematic is their iterative use in a naïve Kelley's scheme
- A main issue with Kelley is the closed-loop nature of the interconnection scheme
- Closed-loop systems are intrinsically prone to instability…
- … unless a **filter** (like lex-reopt) is used for input-output decoupling



GMIC clean-up

• If you insist on reading GMICs from an LP basis … at least don't use the one provided for free by the LP solver, but keep the freedom to choose!



GMIC clean-up

• Given an optimal LP **vertex** *x** of the “large LP” (original+cuts) and the associated **optimal** basis B*:

- Balas and Perregaard (2003): perform a sequence of pivots leading to a (possibly non-optimal or even infeasible) basis **of the large LP** leading to a deeper cut w.r.t. the given *x**

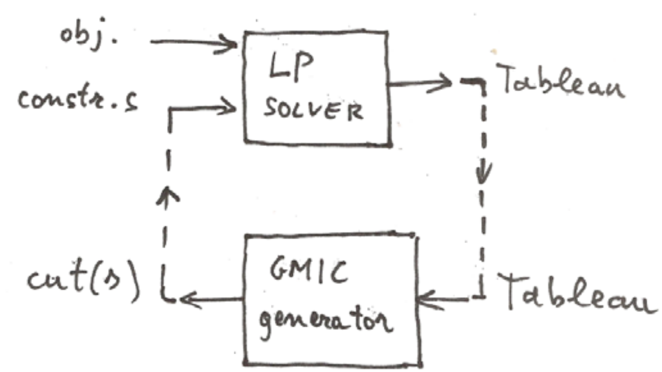
- Dash and Goycoolea (2009): heuristically look for a basis B **of the original LP** that is “close to B*” in the hope of cutting the given *x** with **rank-1** GMICs associated with B

- Cornuéjols and Nannicini (reduce and split)

- … → Tobias’ talk (this afternoon)

Brainstorming about GMICs

- Ok, let's think "laterally" about this cutting plane stuff
- We have a cut-generation module that needs an LP tableau on input



- … but we cannot short-cut it directly onto the LP-solver module (soon the LP determinant burns!)

- **Shall we forget about an extensive use of GMICs …**
- **… or we better design a different scheme to exploit them?**

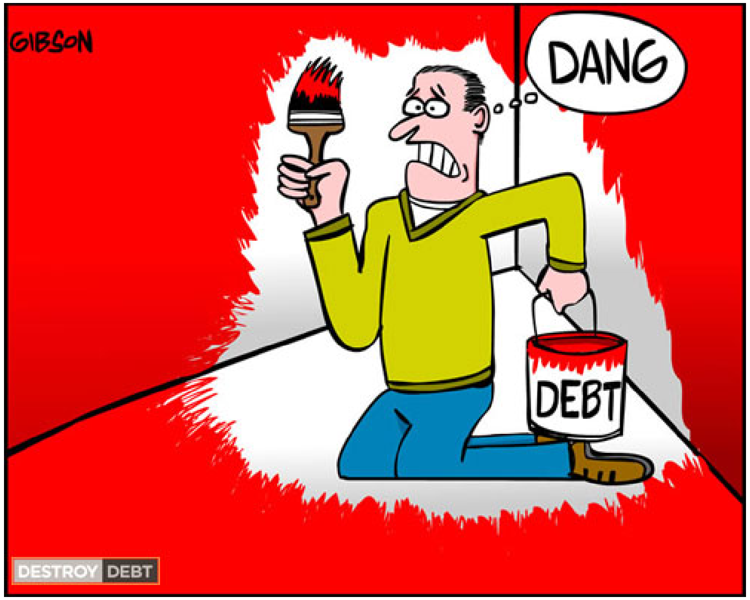
Brainstorming about GMICs

- This sounds like *déjà vu*…

  … we have a **simple module**
  that works well in the beginning

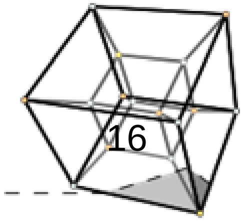
  … but soon it **gets stuck in a corner**

- … Where did I hear this?



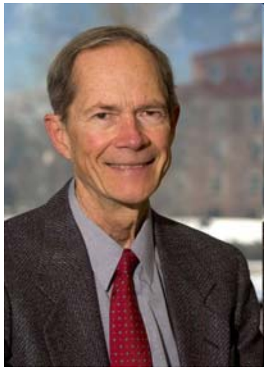
- **Oh yeah! It was about heuristics and metaheuristics…**

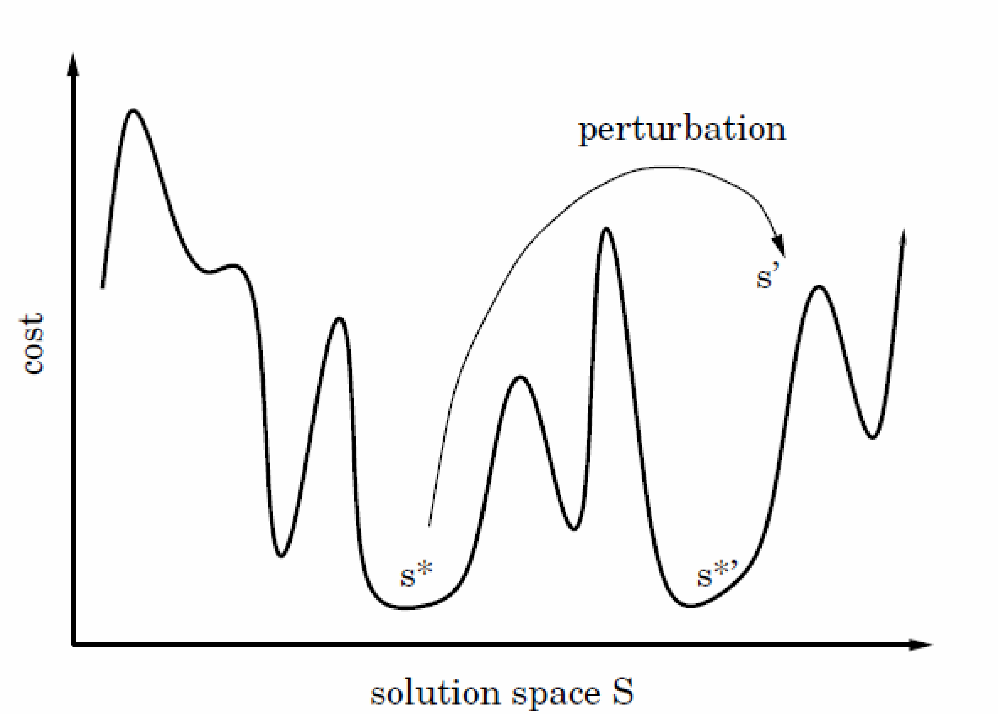
We need a META-SCHEME for cut generation !



Toward a meta-scheme for MIP cuts

- We stick with **simple** cut-generation modules; if we get into trouble…
  … we don't give-up but apply a **diversification step** (isn't this the name, Fred?) to perturb the problem and explore a different "**cut neighborhood**"





A diving meta-scheme for GMICs

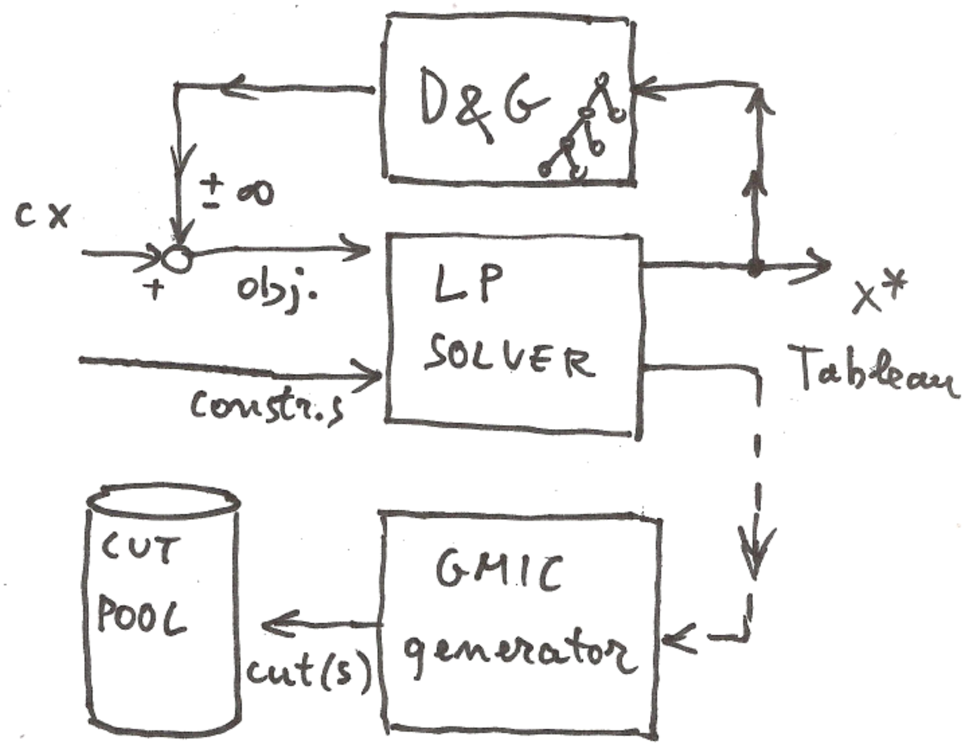
A main source of feedback is the presence of previous GMICs in the LP basis → **avoid modifying the input constr.s, use the obj. func. instead!**

- A kick-off (very simple) scheme:

  Dive & Gomory

  Idea: Simulate enumeration by adding/subtracting a bigM to the **cost** of some var.s and apply a classical GMIC generator to each LP

  … but **don't add the cuts to the LP** (just store them in a **cut pool** for future use…)



D&G results

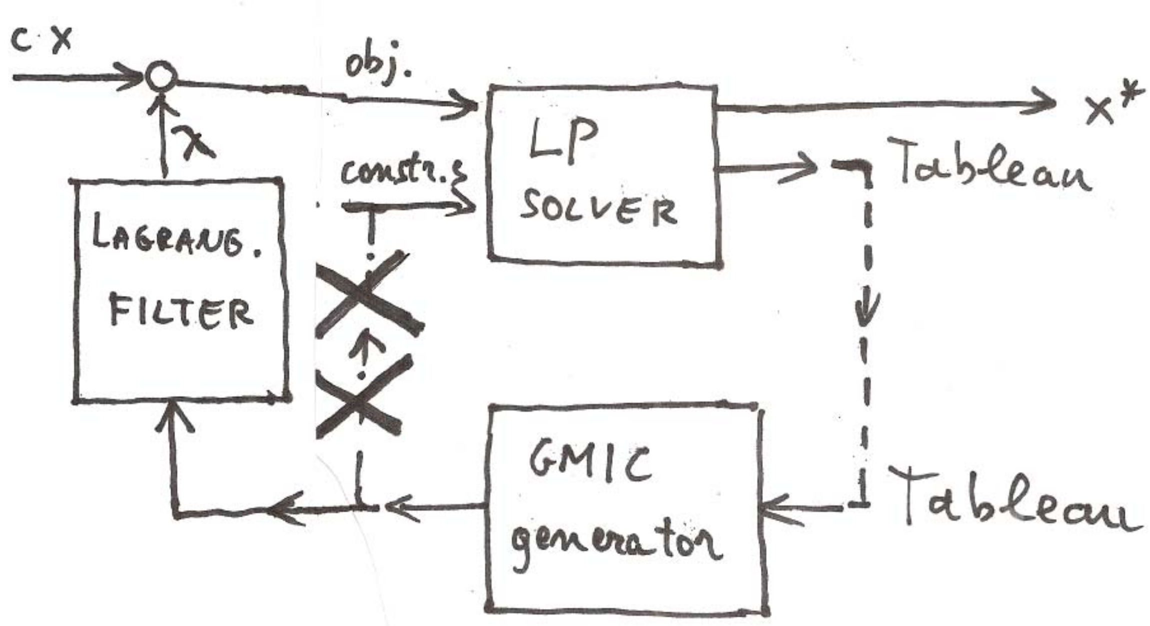
	MIPLIB 2003	
method	cl.gap	time (s)
1gmi	18.3%	0.54
Lift&Project	30.7%	95.23
Dive&Gomory	31.5%	7.45

- ***cl.gap*** *:* root node integrality gap (MIP opt. – LP opt.) closed
- ***1gmi*** *:* 1 round of GMICs from the initial LP tableau
- ***Lift&Project*** : Balas & Bonami lift-and-project scheme following Balas & Perregaard recipe

A Lagrangian filter for GMICs

• As in Dive&Gomory, diversification can be obtained by changing the objective function passed to the LP-solver module so as to produce LP tableaux that are only **weakly correlated** with the LP optimal solution x* that we want to cut

• A promising framework is ***relax-and-cut*** where GMICs are not added to the LP but immediately relaxed in a **Lagrangian** fashion



Back to Lagrange

- Forget about Kelley: optimizing over the first GMIC closure actually reads

$$\min c^T x$$
$$x \, \varepsilon \, P$$

< all rank-1 GMI cuts >

- Dualize (in a Lagrangian way) the GMICs, i.e. …

- … solve a sequence of Lagrangian subproblems

$$\min \{ c(\boldsymbol{\lambda})^T x : x \, \varepsilon \, P \}$$

on the **original LP** but using the Lagrangian cost vector $c(\boldsymbol{\lambda})$

- **Subgradient** $\boldsymbol{s}$ *at* $\boldsymbol{\lambda}$ *:* s_i = violation of the i-th GMIC w.r.t.

$$x^*(\lambda) := \text{argmin} \{ c(\lambda)^T x : x \, \varepsilon \, P \}$$

Back to Lagrange

- During the Lagrangian dual optimization process, a large number of **bases of the original** LP is traced → round of rank-1 GMICs can easily be generated "on the fly" and stored (just a heuristic policy)

- Use of a **cut pool** to explicitly store the generated cuts, needed to compute (approx.) **subgradients** used by Lagrangian optimization

- **Warning**: new GMICs added on the fly → possible convergence issues due to the imperfect nature of the computed "subgradients"

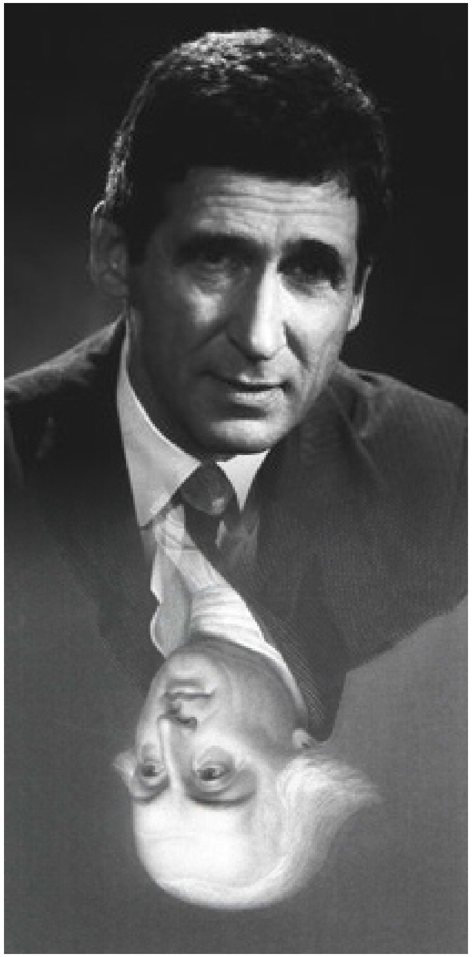
- … as the separation oracle does not return the list of **all** violated GMICs, hence the subgradient is **truncated** somehow …

Lagrange + Gomory = LaGromory

- A **Relax&Cut** scheme (Lucena and Escudero, Guignard & Malik): Generate cuts and immediately dualize them (don't wait they become wild!)

- **No fractional point x* to cut:** separation used to find (approx.) subgradients

- The method can add rank-1 GMICs on top of any other class of cuts (Cplex cuts etc.), including branching conditions → just dualize them!



Experiments with LaGromory cuts

- Four possible implementations (more details in the paper):
  - **subg**: naïve Held-Karp subgradient opt. scheme (10,000 iter.s)
  - **hybr:** as before, but every 1,000 subgr. iter.s we solve a “large LP” just to recompute the optimal Lagrangian multipliers for **all** the cuts in the current pool
  - **fast:** as before but tweaked for speed: only **10 large LPs** solved, each followed by just **100 subgradient** iterations with **very small step size** → 10 short walks around the Lagrangian dual optimum to collect bases of the original LP, each made by 100 small steps to perturb Lagrangian costs
  - **faster**: same as fast, but with 50 sugradient iter.s instead of 100

Computational results

Testbed: 72 instances from MIPLIB 3.0 and 2003

CPU seconds on a standard PC (2.4 Ghz, 1GB RAM allowed)

Rank 1 GMI cuts (root node only)

	MIPLIB 3.0		MIPLIB 2003	
method	cl.gap	time (s)	cl.gap	time (s)
1gmi	26.9%	0.02	18.3%	0.54
faster	57.9%	1.34	43.3%	33.33
fast	59.6%	2.25	45.5%	58.40
hybr	60.8%	15.12	48.6%	314.73
subg	56.0%	25.16	43.5%	290.82
dgDef	61.6%	20.05	39.7%	853.85

- ***1gmi*** : 1 round of GMICs from the initial LP tableau
- ***dgDef*** *:* Dash & Goycoolea heuristic for rank-1 GMICs

Higher rank GMICs

Safe (too conservative?) approach: (1) Stick with rank-1 GMICs on "sampling phase" (2) When diversification is required, feed the pool with a round of (higher-rank) GMICs read from the "large LP"

		MIPLIB 3.0		MIPLIB 2003	
method	rank	cl.gap	time (s)	cl.gap	time (s)
gmi	1	26.9%	0.02	18.3%	0.54
faster	1	57.9%	1.34	43.3%	33.33
fast	1	59.6%	2.25	45.5%	58.40
gmi	2	36.0%	0.03	24.0%	0.88
faster	2	62.1%	2.75	47.2%	58.37
fast	2	64.1%	5.12	48.5%	106.76
gmi	5	47.8%	0.07	30.3%	2.17
faster	5	65.6%	5.47	49.9%	126.65
fast	5	67.2%	10.09	51.1%	238.33
L&P	10	57.0%	3.50	30.7%	95.23

- ***gmi*** : multiple rounds of GMICs (root node)
- ***L&P*** : Balas & Bonami lift-and-project code (root node)

GMICs on top of other cuts

	MIPLIB 3.0		MIPLIB 2003	
method	cl.gap	time (s)	cl.gap	time (s)
cpx	54.7%	0.15	49.0%	6.57
cpx+fast	69.5%	1.90	58.2%	56.03
cpx2	64.7%	0.50	52.9%	22.48
cpx2+fast	71.6%	1.76	59.6%	59.75

- Root node only
- ***cpx*** *:* IBM ILOG Cplex 12.0 (default setting)
- ***cpx2*** *:* IBM ILOG Cplex 12.0 (aggressive cuts)

LaGromory in an enumerative scheme

method	solved	time	nodes	fin.gap
cpx	1	9,780	253,090	21.1%
cpx2	2	9,183	52,605	22.8%
cpx+fast	3	6,723	104,406	16.2%

- A collection of 18 hard MIPLIB 2003 instances
- **Cut & Branch** : root node + preprocessing + cuts, then Cplex 11.2 (no further cuts) on the resulting enhanced formulation
- Time limit of 10,000 CPU sec.s for each run
- ***cpx*** : IBM ILOG Cplex 12.0 (default setting)
- ***cpx2*** : IBM ILOG Cplex 12.0 (aggressive cuts)

Thank you for your attention
… and of course for not sleeping!