On Split Cuts from Elementary Disjunctions

Everything You Always Wanted to Know About BUT Were Afraid to Ask Egon

Andrea Lodi
University of Bologna, Italy & ILOG SA
andrea.lodi@unibo.it

Joint work with Matteo Fischetti & Andrea Tramontani

August 2nd, 2007 @ MIP 2007

Notation & Assumptions

- We consider:
$$\min\{c^T x : Ax \geq b, x \text{ integer}\} \tag{1}$$
with bounds on x included in $Ax \geq b$ and x^* as the optimal solution of the continuous relaxation P.

Notation & Assumptions

- We consider:
$$\min\{c^T x : Ax \geq b, x \text{ integer}\} \tag{1}$$
with bounds on x included in $Ax \geq b$ and x^* as the optimal solution of the continuous relaxation P.

- We are also given an elementary disjunction on the form $x_j \leq \pi_0$ OR $x_j \geq \pi_0 + 1$ such that $x_j^* \in\ ]\pi_0, \pi_0 + 1[$.

Notation & Assumptions

- We consider:
$$\min\{c^T x : Ax \geq b, x \text{ integer}\} \tag{1}$$
with bounds on x included in $Ax \geq b$ and x^* as the optimal solution of the continuous relaxation P.

- We are also given an elementary disjunction on the form $x_j \leq \pi_0$ OR $x_j \geq \pi_0 + 1$ such that $x_j^* \in \;]\pi_0, \pi_0 + 1[$.

- The plan is derive the "strongest" cut, $\gamma x \geq \gamma_0$ violated by x^*, by using such a disjunction and doing it by the classical disjunctive approach of Balas:

$$P_0 \qquad\qquad P_1$$

$$\begin{array}{lrcl} (u) & Ax & \geq & b \\ (u_0) & -x_j & \geq & -\pi_0 \end{array} \qquad\qquad \begin{array}{lrcl} (v) & Ax & \geq & b \\ (v_0) & x_j & \geq & \pi_0 + 1 \end{array}$$

which is a valid cutting plane for the union of the two polyhedra P_0 and P_1.

Notation & Assumptions (cont.d)

- Such a cut can be separated by solving the so-called Cut Generating Linear Program:

$$
\begin{array}{rrlrl}
\text{(CGLP)} & \min & \gamma x^* - \gamma_0 & & \\
 & & & & \\
\gamma & = & u^T A & - & u_0 e_j \\
\gamma & = & v^T A & + & v_0 e_j \\
\gamma_0 & = & u^T b & - & u_0 \pi_0 \\
\gamma_0 & = & v^T b & + & v_0(\pi_0 + 1) \\
u, w, & v, z, & u_0, v_0 & \geq 0 &
\end{array}
$$

which is however a cone and must be truncated in order to get a cut.

Notation & Assumptions (cont.d)

- Such a cut can be separated by solving the so-called Cut Generating Linear Program:

$$
\begin{array}{lllll}
\text{(CGLP)} & \min & \gamma x^* - \gamma_0 & & \\
\\
\gamma & = & u^T A & - & u_0 e_j \\
\gamma & = & v^T A & + & v_0 e_j \\
\gamma_0 & = & u^T b & - & u_0 \pi_0 \\
\gamma_0 & = & v^T b & + & v_0(\pi_0 + 1) \\
u, w, & v, z, & u_0, v_0 & \geq 0 &
\end{array}
$$

  which is however a cone and must be truncated in order to get a cut.

- The truncation of such a cone can be obtained in many different ways through a so-called normalization constraint and Balas, Ceria & Cornuéjols (1996) – BCC for short – used

$$\sum u + \sum v + u_0 + v_0 = 1. \qquad (2)$$

Notation & Assumptions (cont.d)

- Such a cut can be separated by solving the so-called Cut Generating Linear Program:

$$
\begin{array}{llll}
\text{(CGLP)} & \min & \gamma x^* - \gamma_0 & \\
 & & & \\
\gamma & = & u^T A & - \quad u_0 e_j \\
\gamma & = & v^T A & + \quad v_0 e_j \\
\gamma_0 & = & u^T b & - \quad u_0 \pi_0 \\
\gamma_0 & = & v^T b & + \quad v_0(\pi_0 + 1) \\
u, w, & v, z, & u_0, v_0 & \geq 0
\end{array}
$$

  which is however a cone and must be truncated in order to get a cut.

- The truncation of such a cone can be obtained in many different ways through a so-called normalization constraint and Balas, Ceria & Cornuéjols (1996) – BCC for short – used

$$\sum u + \sum v + u_0 + v_0 = 1. \qquad (2)$$

- Lately, Balas & Perregaard (2002) developed an elegant and efficient way of solving the CGLP in the space of the original variables which represents a crucial speed-up.

A good exploitation of the elementary disjunction

- Another way of thinking at the procedure of Balas & Perregaard is the following:

A good exploitation of the elementary disjunction

- Another way of thinking at the procedure of Balas & Perregaard is the following:

  1. solve the continuous relaxation

A good exploitation of the elementary disjunction

- Another way of thinking at the procedure of Balas & Perregaard is the following:

  1. solve the continuous relaxation
  2. for every fractional variable

A good exploitation of the elementary disjunction

- Another way of thinking at the procedure of Balas & Perregaard is the following:

  1. solve the continuous relaxation
  2. for every fractional variable

     (a) consider the elementary disjunction associated with the corresponding row

A good exploitation of the elementary disjunction

- Another way of thinking at the procedure of Balas & Perregaard is the following:

  1. solve the continuous relaxation
  2. for every fractional variable
     (a) consider the elementary disjunction associated with the corresponding row
     (b) strengthen the associated Gomory Mixed Integer cut (GMI) by performing a sequence of pivots to (possibly infeasible) alternative basis so as to implicitly solve the CGLP.

A good exploitation of the elementary disjunction

- Another way of thinking at the procedure of Balas & Perregaard is the following:

  1. solve the continuous relaxation
  2. for every fractional variable
     (a) consider the elementary disjunction associated with the corresponding row
     (b) strengthen the associated Gomory Mixed Integer cut (GMI) by performing a sequence of pivots to (possibly infeasible) alternative basis so as to implicitly solve the CGLP.

- The first set of experiments we designed is intended at understanding how and how much one can really gain from such a strengthening and in order to do this we avoided strengthening the cuts *a posteriori* through the Balas & Jeroslow procedure.

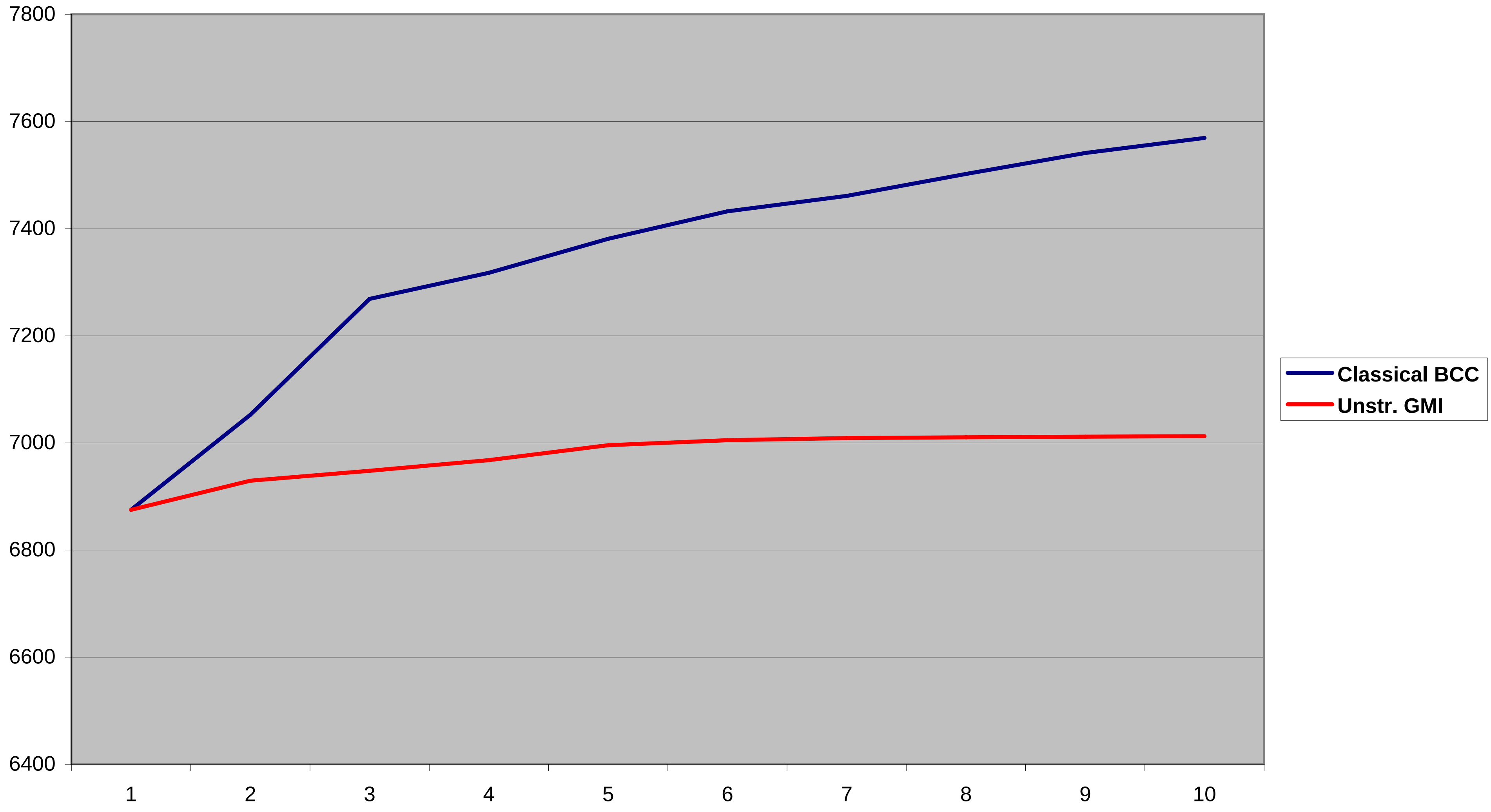
A good exploitation of the elementary disjunction

- Another way of thinking at the procedure of Balas & Perregaard is the following:
  1. solve the continuous relaxation
  2. for every fractional variable
     (a) consider the elementary disjunction associated with the corresponding row
     (b) strengthen the associated Gomory Mixed Integer cut (GMI) by performing a sequence of pivots to (possibly infeasible) alternative basis so as to implicitly solve the CGLP.

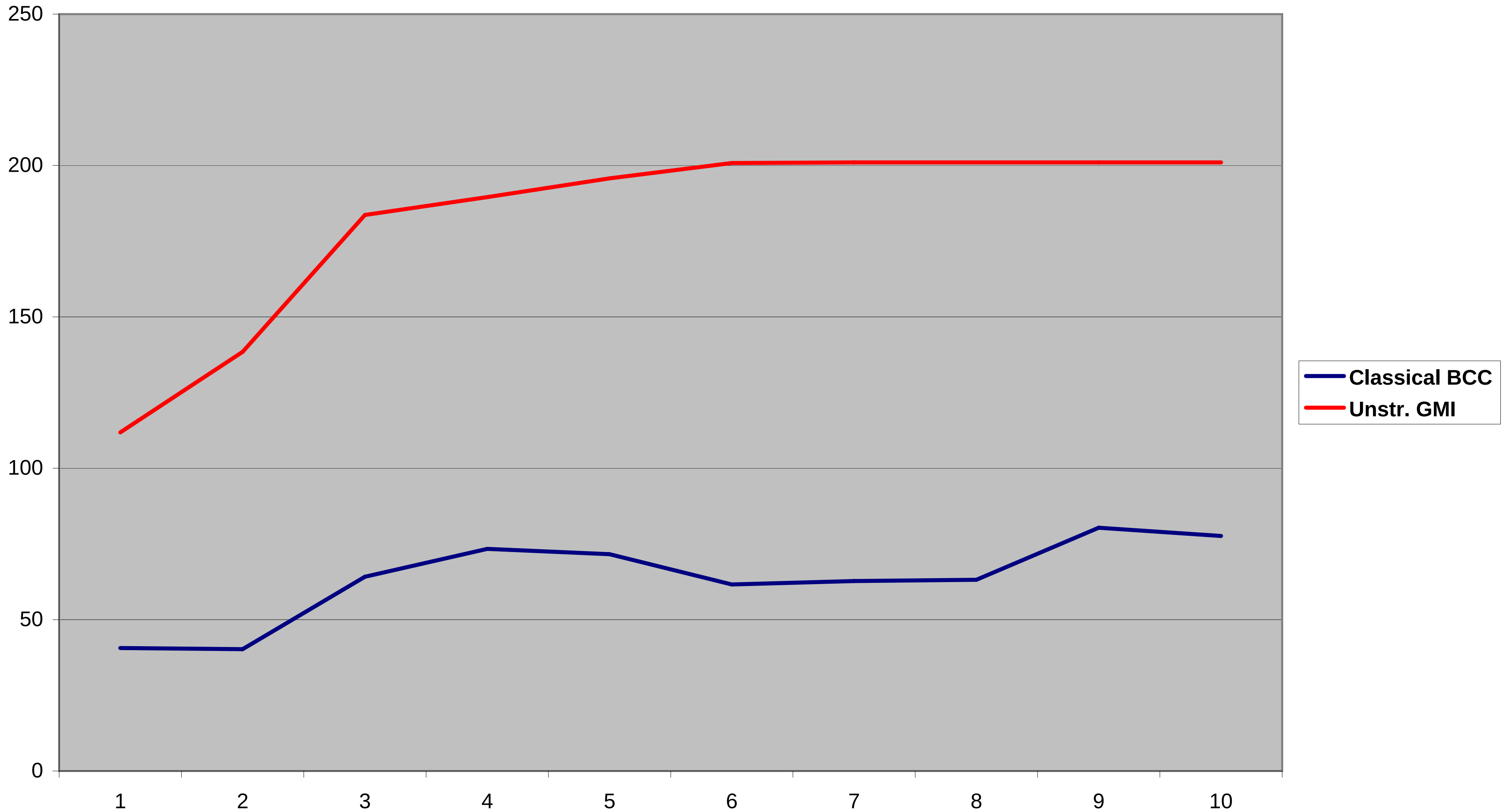
- The first set of experiments we designed is intended at understanding how and how much one can really gain from such a strengthening and in order to do this we avoided strengthening the cuts *a posteriori* through the Balas & Jeroslow procedure.

- Within 10 rounds of cuts, the indicators we report are:
  1. quality of the lower bound
  2. average cuts' density
  3. cuts' rank
  4. average cardinality of (u, v), i.e., how many constraints used on average to generate a cut

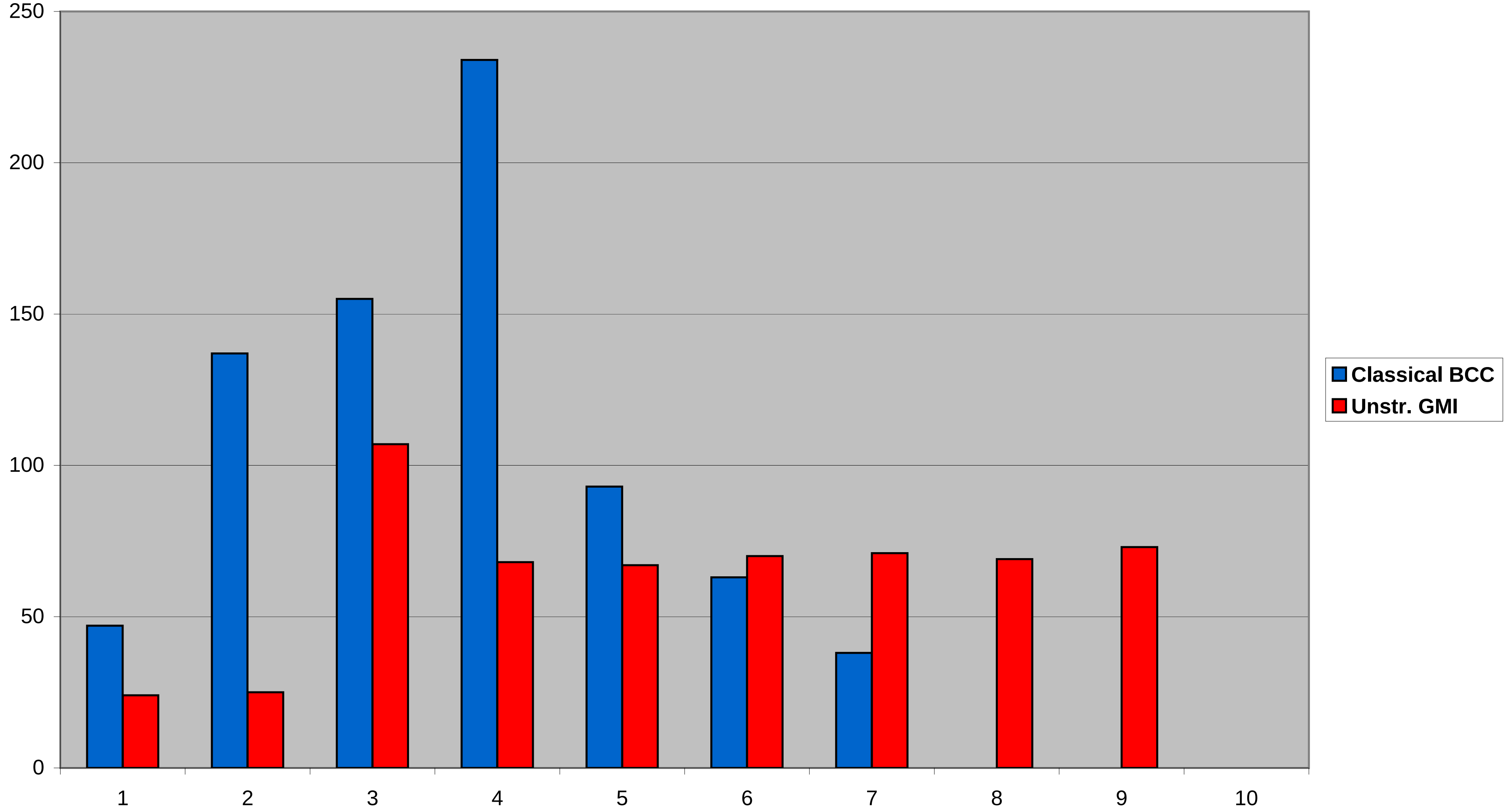
Instance p0201: lower bound



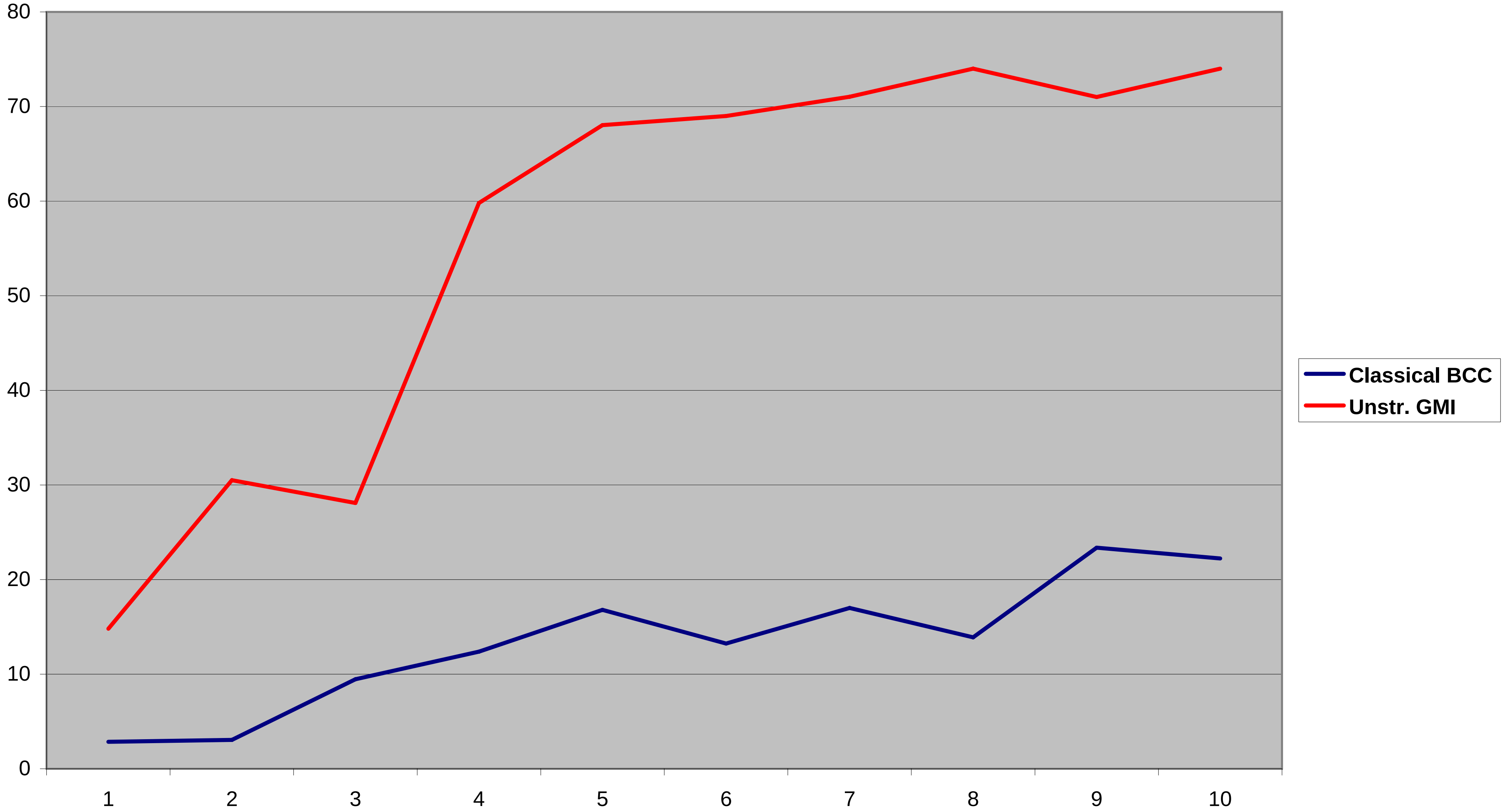
Instance p0201: average cuts' density



Instance p0201: cuts' rank



Instance `p0201`: average cardinality of (u, v)



In Summary

Table 1: 10 iterations of cuts. At each iteration one cut is generated from any fractional variable. No strengthening in the cut computation.

Unstrengthened GMI vs. "Classical" BCC approach						
	Unstrengthened GMI			"Classical" BCC		
Instance	n.cuts	gap%	$\#(u,v)$	n.cuts	gap%	$\#(u,v)$
bell3a	137	70.74	59.49	71	70.74	43.72
bell5	202	28.18	31.20	178	94.29	11.75
blend2	156	28.73	11.70	192	30.51	8.10
flugpl	93	15.15	7.57	92	18.36	5.85
gt2	191	98.71	14.52	196	93.46	10.28
lseu	152	32.94	14.34	196	41.33	9.17
*m.share1	68	0.00	1.00	74	0.00	1.39
mod008	104	12.09	10.40	139	17.05	12.41
p0033	103	58.33	5.72	113	67.86	4.81
p0201	574	18.58	56.03	767	93.82	13.43
rout	445	8.52	135.39	434	24.26	68.07
*stein27	235	0.00	19.74	252	0.00	6.53
vpm1	255	36.95	9.03	263	55.84	5.39
vpm2	424	42.08	71.72	403	74.96	17.27

In Summary

Table 1: 10 iterations of cuts. At each iteration one cut is generated from any fractional variable. No strengthening in the cut computation.

Unstrengthened GMI vs. "Classical" BCC approach

	Unstrengthened GMI			"Classical" BCC		
Instance	n.cuts	gap%	$\#(u,v)$	n.cuts	gap%	$\#(u,v)$
bell3a	137	70.74	59.49	71	70.74	43.72
bell5	202	28.18	31.20	178	94.29	11.75
blend2	156	28.73	11.70	192	30.51	8.10
flugpl	93	15.15	7.57	92	18.36	5.85
gt2	191	98.71	14.52	196	93.46	10.28
lseu	152	32.94	14.34	196	41.33	9.17
*m.share1	68	0.00	1.00	74	0.00	1.39
mod008	104	12.09	10.40	139	17.05	12.41
p0033	103	58.33	5.72	113	67.86	4.81
p0201	574	18.58	56.03	767	93.82	13.43
rout	445	8.52	135.39	434	24.26	68.07
*stein27	235	0.00	19.74	252	0.00	6.53
vpm1	255	36.95	9.03	263	55.84	5.39
vpm2	424	42.08	71.72	403	74.96	17.27
avg.	236.333	37.583	35.593	253.667	56.873	17.521

Why does it work so well?

- The normalization (2) used by Balas, Ceria & Cornuéjols has the following very nice properties:

Why does it work so well?

- The normalization (2) used by Balas, Ceria & Cornuéjols has the following very nice properties:

  1. the norm of the separated cuts becomes smaller and smaller (each multiplier < 1), thus using the separated cuts in the derivation of new ones is penalized (sum of multipliers $= 1$)

Why does it work so well?

- The normalization (2) used by Balas, Ceria & Cornuéjols has the following very nice properties:
  1. the norm of the separated cuts becomes smaller and smaller (each multiplier < 1), thus using the separated cuts in the derivation of new ones is penalized (sum of multipliers $= 1$) $\Rightarrow$ low-rank inequalities are separated.

Why does it work so well?

- The normalization (2) used by Balas, Ceria & Cornuéjols has the following very nice properties:

  1. the norm of the separated cuts becomes smaller and smaller (each multiplier < 1), thus using the separated cuts in the derivation of new ones is penalized (sum of multipliers $= 1$) $\Rightarrow$ low-rank inequalities are separated.
  2. since low-rank inequalities are preferred and since the original inequalities (rank-0) are generally sparse, the separated cuts remain sparse (overall (2) makes also sure that a limited number of constraints are used in the derivation).

Why does it work so well?

- The normalization (2) used by Balas, Ceria & Cornuéjols has the following very nice properties:
  1. the norm of the separated cuts becomes smaller and smaller (each multiplier < 1), thus using the separated cuts in the derivation of new ones is penalized (sum of multipliers $= 1$) $\Rightarrow$ low-rank inequalities are separated.
  2. since low-rank inequalities are preferred and since the original inequalities (rank-0) are generally sparse, the separated cuts remain sparse (overall (2) makes also sure that a limited number of constraints are used in the derivation).
  3. since each multiplier tends to be small, the lifting – done afterwards – of the coefficients of the variables outside the support is "safe", i.e., those coefficients remain under control.

Why does it work so well?

- The normalization (2) used by Balas, Ceria & Cornuéjols has the following very nice properties:
  1. the norm of the separated cuts becomes smaller and smaller (each multiplier < 1), thus using the separated cuts in the derivation of new ones is penalized (sum of multipliers $= 1$) $\Rightarrow$ low-rank inequalities are separated.
  2. since low-rank inequalities are preferred and since the original inequalities (rank-0) are generally sparse, the separated cuts remain sparse (overall (2) makes also sure that a limited number of constraints are used in the derivation).
  3. since each multiplier tends to be small, the lifting – done afterwards – of the coefficients of the variables outside the support is "safe", i.e., those coefficients remain under control.

- It can be shown instead that the GMI associated with the same normalization is a basic solution of the CGLP but generally not the optimal one.

Why does it work so well?

- The normalization (2) used by Balas, Ceria & Cornuéjols has the following very nice properties:
  1. the norm of the separated cuts becomes smaller and smaller (each multiplier < 1), thus using the separated cuts in the derivation of new ones is penalized (sum of multipliers $= 1$) $\Rightarrow$ low-rank inequalities are separated.
  2. since low-rank inequalities are preferred and since the original inequalities (rank-0) are generally sparse, the separated cuts remain sparse (overall (2) makes also sure that a limited number of constraints are used in the derivation).
  3. since each multiplier tends to be small, the lifting – done afterwards – of the coefficients of the variables outside the support is "safe", i.e., those coefficients remain under control.

- It can be shown instead that the GMI associated with the same normalization is a basic solution of the CGLP but generally not the optimal one. It is the optimal solution of the CGLP which uses the trivial (and rather bad) normalization:

$$u_0 + v_0 = 1.$$

Why does it work so well?

- The normalization (2) used by Balas, Ceria & Cornuéjols has the following very nice properties:

  1. the norm of the separated cuts becomes smaller and smaller (each multiplier < 1), thus using the separated cuts in the derivation of new ones is penalized (sum of multipliers $= 1$) $\Rightarrow$ low-rank inequalities are separated.
  2. since low-rank inequalities are preferred and since the original inequalities (rank-0) are generally sparse, the separated cuts remain sparse (overall (2) makes also sure that a limited number of constraints are used in the derivation).
  3. since each multiplier tends to be small, the lifting – done afterwards – of the coefficients of the variables outside the support is "safe", i.e., those coefficients remain under control.

- It can be shown instead that the GMI associated with the same normalization is a basic solution of the CGLP but generally not the optimal one. It is the optimal solution of the CGLP which uses the trivial (and rather bad) normalization:

$$u_0 + v_0 = 1. \tag{3}$$

- However, nothing is perfect!

Why does it work so well?

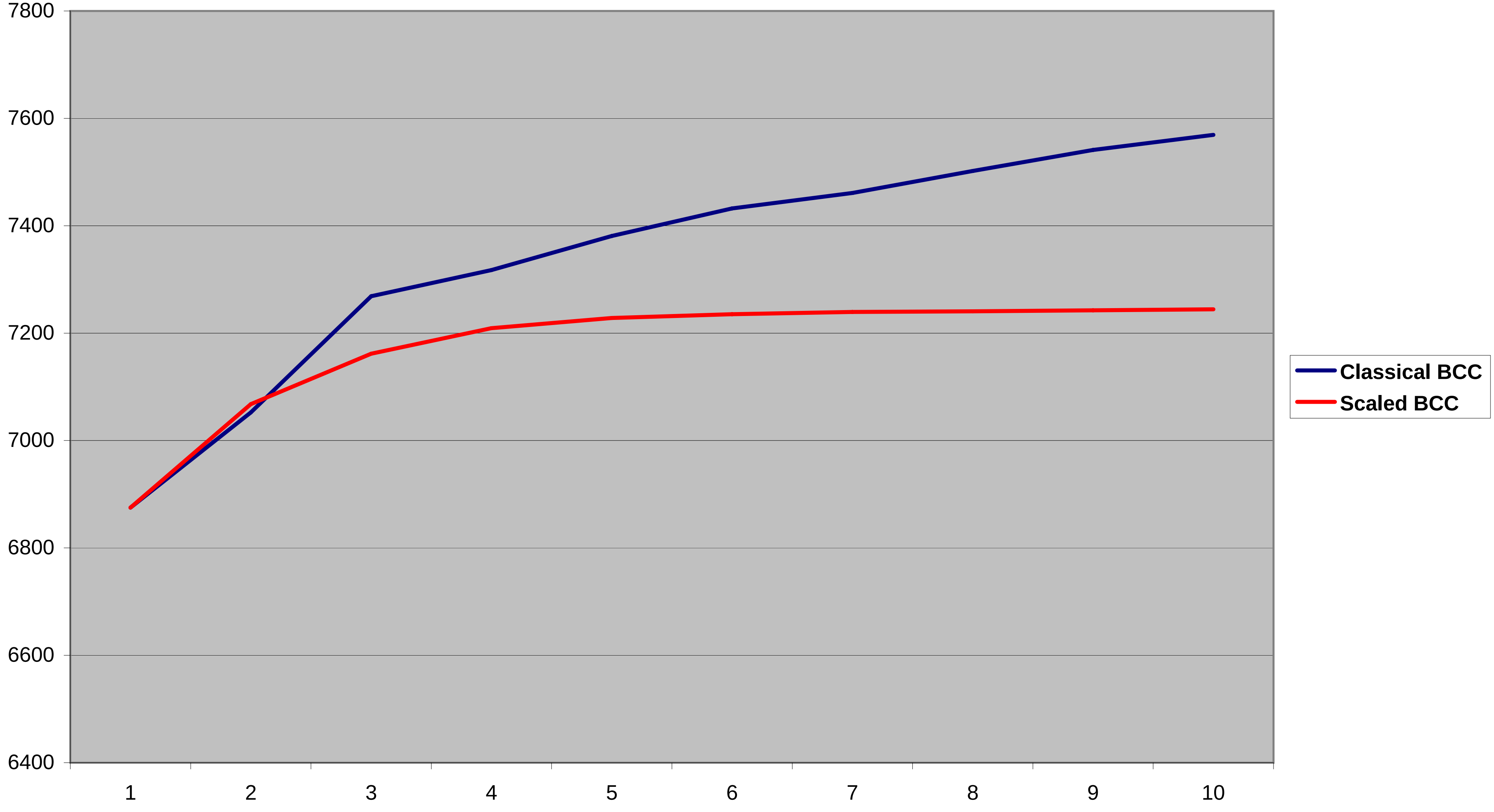
- The normalization (2) used by Balas, Ceria & Cornuéjols has the following very nice properties:
  1. the norm of the separated cuts becomes smaller and smaller (each multiplier < 1), thus using the separated cuts in the derivation of new ones is penalized (sum of multipliers $= 1$) $\Rightarrow$ low-rank inequalities are separated.
  2. since low-rank inequalities are preferred and since the original inequalities (rank-0) are generally sparse, the separated cuts remain sparse (overall (2) makes also sure that a limited number of constraints are used in the derivation).
  3. since each multiplier tends to be small, the lifting – done afterwards – of the coefficients of the variables outside the support is "safe", i.e., those coefficients remain under control.

- It can be shown instead that the GMI associated with the same normalization is a basic solution of the CGLP but generally not the optimal one. It is the optimal solution of the CGLP which uses the trivial (and rather bad) normalization:

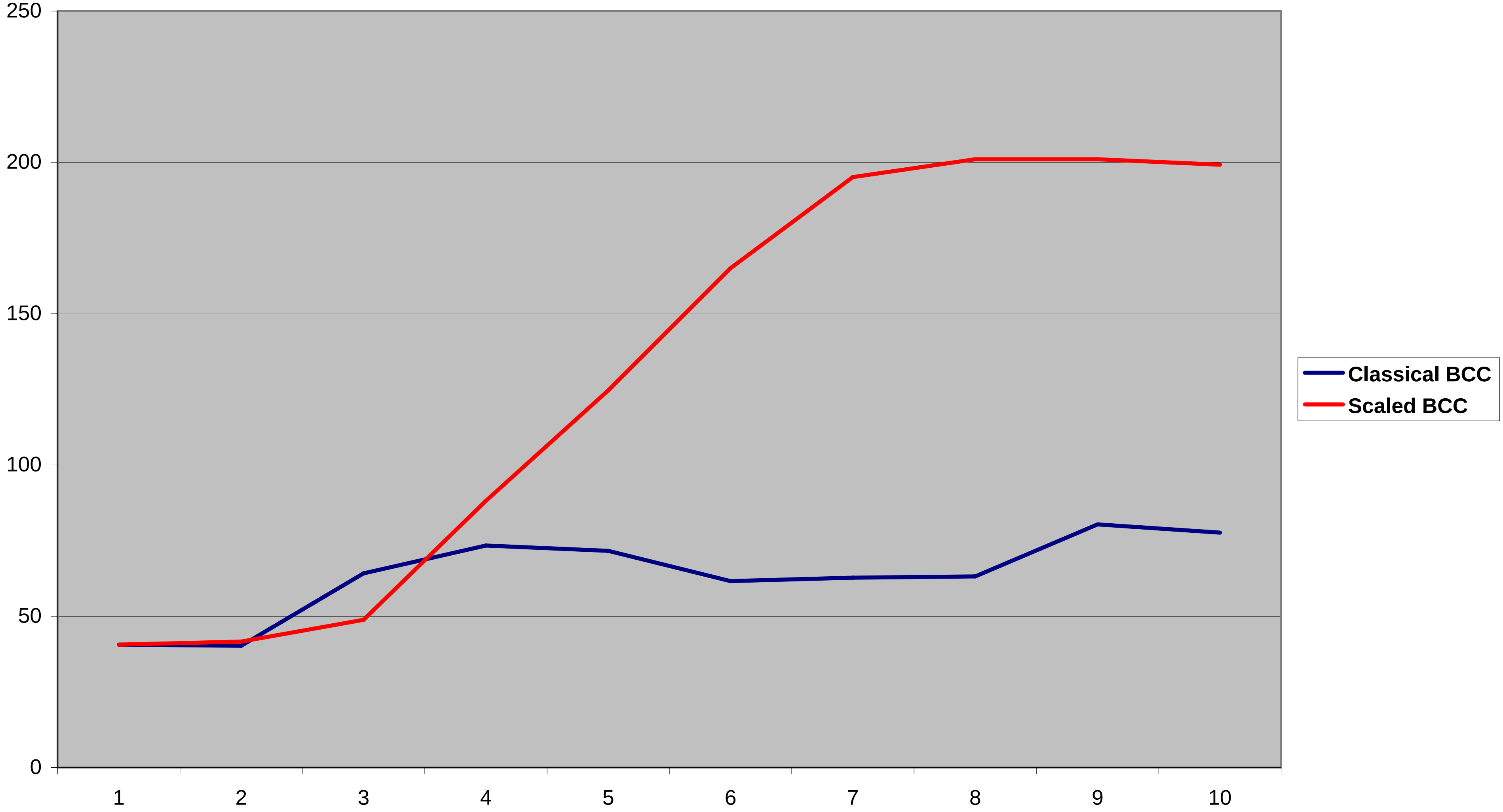
$$u_0 + v_0 = 1. \tag{3}$$

- However, nothing is perfect! Normalization (2) is dependent on the scaling of the constraints. In the second set of experiments we simply multiplied by 1,000 any generated cut before adding it to the current relaxation.

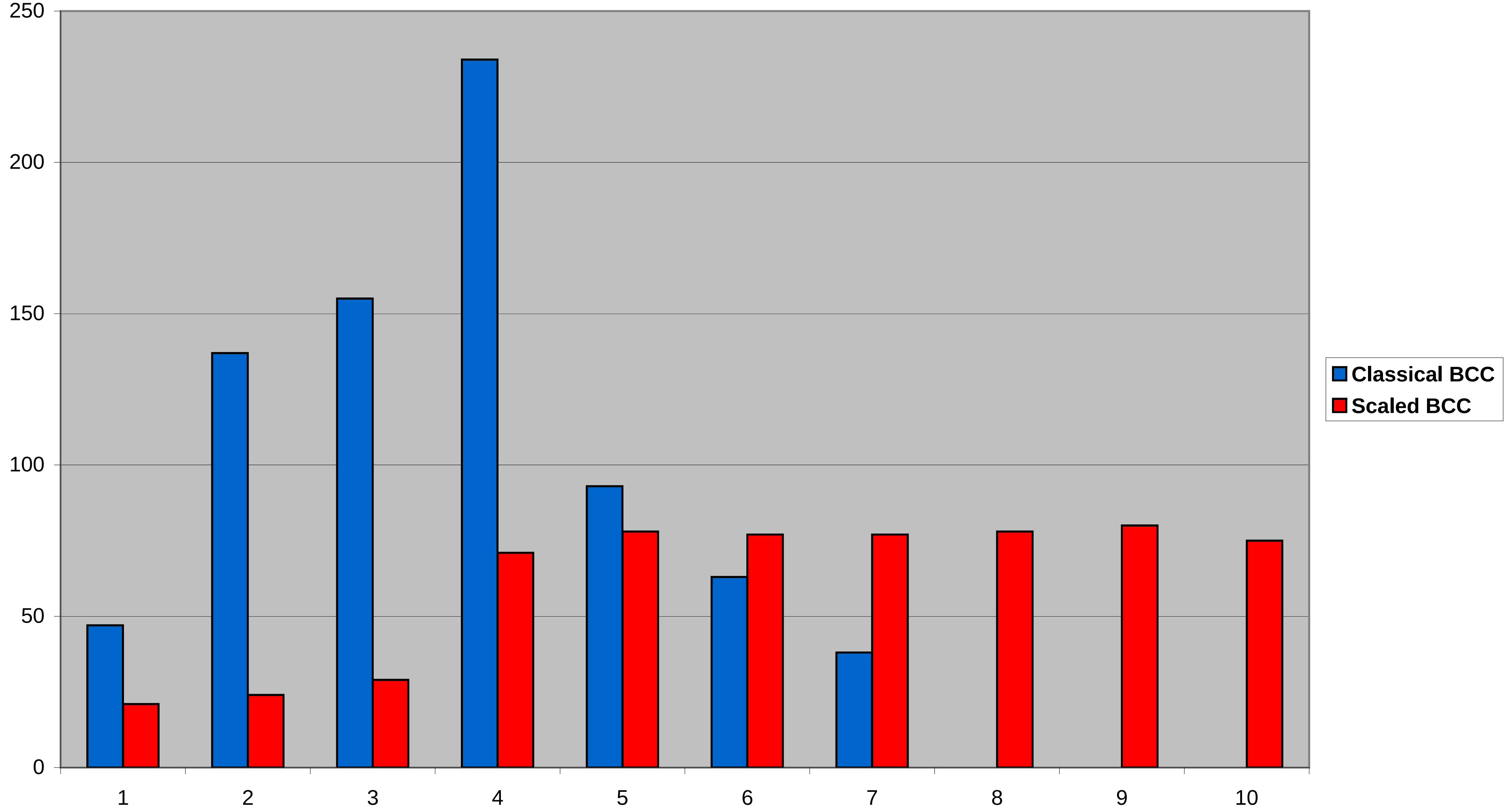
Instance p0201: lower bound



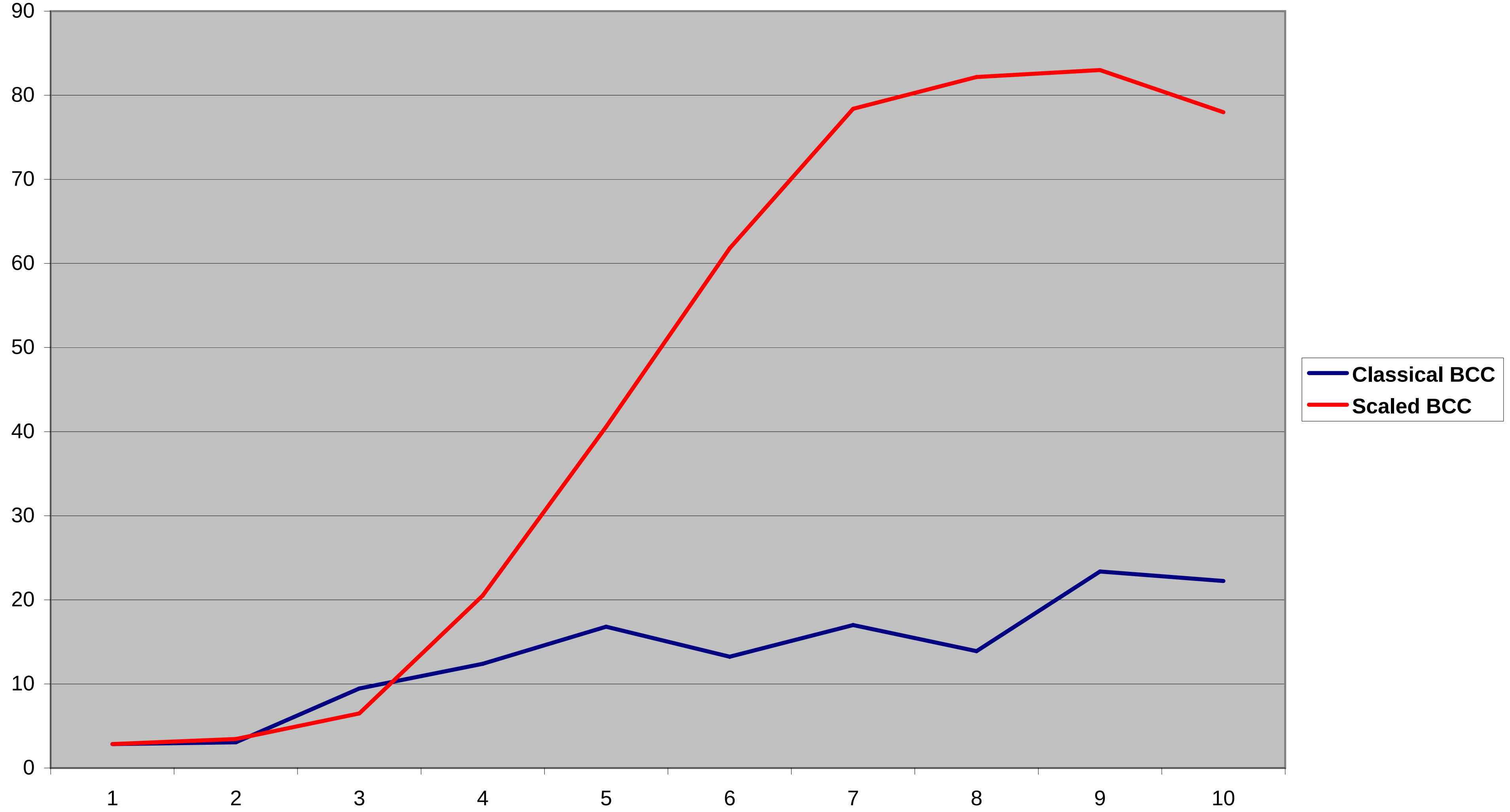
Instance p0201: average cuts' density



Instance p0201: cuts' rank



Instance $\texttt{p0201}$: average cardinality of (u, v)



In Summary

Table 2: 10 iterations of cuts. At each iteration one cut is generated from any fractional variable. No strengthening in the cut computation.

"Classical" BCC approach vs. "Scaled" BCC approach						
	"Classical" BCC			"Scaled" BCC		
Instance	n.cuts	gap%	$\#(u,v)$	n.cuts	gap%	$\#(u,v)$
bell3a	71	70.74	43.72	69	70.74	44.32
bell5	178	94.29	11.75	214	88.83	17.47
blend2	192	30.51	8.10	166	28.91	11.71
flugpl	92	18.36	5.85	90	15.40	7.40
gt2	196	93.46	10.28	184	93.42	17.22
lseu	196	41.33	9.17	137	38.58	10.88
*m.share1	74	0.00	1.39	206	0.00	14.60
mod008	139	17.05	12.41	104	3.90	10.21
p0033	113	67.86	4.81	94	57.09	6.40
p0201	767	93.82	13.43	610	49.91	45.72
rout	434	24.26	68.07	435	13.03	152.66
*stein27	252	0.00	6.53	248	0.00	22.39
vpm1	263	55.84	5.39	244	47.59	8.50
vpm2	403	74.96	17.27	420	54.39	22.27

In Summary

Table 2: 10 iterations of cuts. At each iteration one cut is generated from any fractional variable. No strengthening in the cut computation.

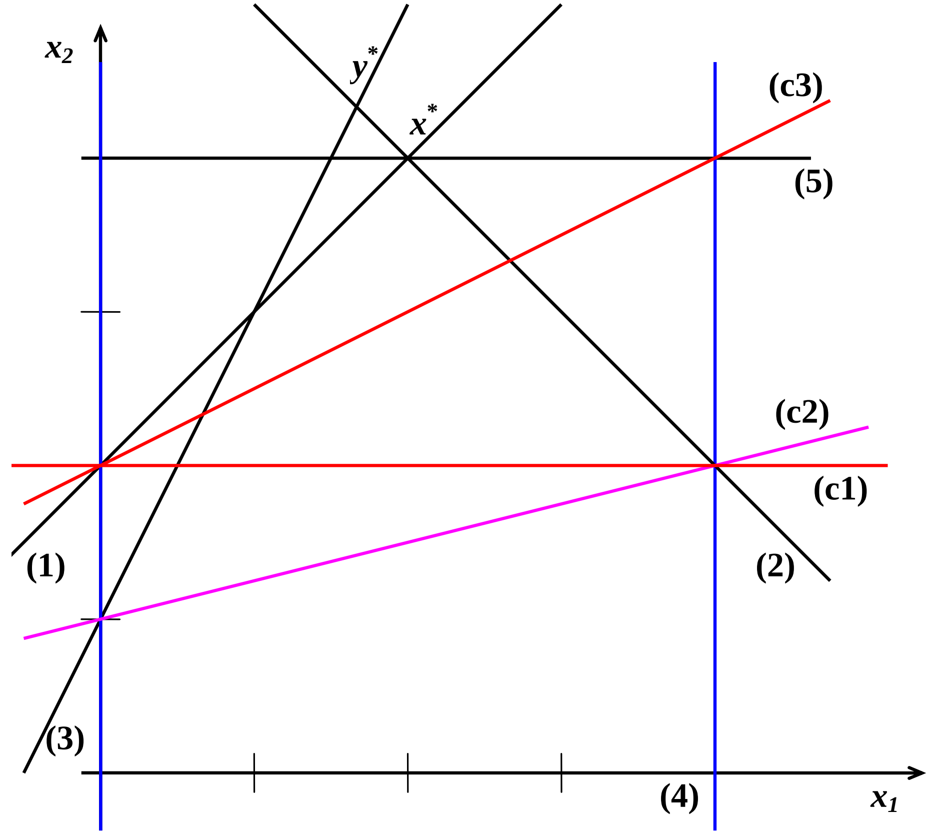
“Classical” BCC approach vs. “Scaled” BCC approach						
	“Classical” BCC			“Scaled” BCC		
Instance	n.cuts	gap%	$\#(u,v)$	n.cuts	gap%	$\#(u,v)$
bell3a	71	70.74	43.72	69	70.74	44.32
bell5	178	94.29	11.75	214	88.83	17.47
blend2	192	30.51	8.10	166	28.91	11.71
flugpl	92	18.36	5.85	90	15.40	7.40
gt2	196	93.46	10.28	184	93.42	17.22
lseu	196	41.33	9.17	137	38.58	10.88
*m.share1	74	0.00	1.39	206	0.00	14.60
mod008	139	17.05	12.41	104	3.90	10.21
p0033	113	67.86	4.81	94	57.09	6.40
p0201	767	93.82	13.43	610	49.91	45.72
rout	434	24.26	68.07	435	13.03	152.66
*stein27	252	0.00	6.53	248	0.00	22.39
vpm1	263	55.84	5.39	244	47.59	8.50
vpm2	403	74.96	17.27	420	54.39	22.27
avg.	253.667	56.873	17.521	230.583	46.816	29.563

Nothing is perfect: Example 1

$$
\begin{array}{lrrcl}
\min & -x_1 & -2x_2 & & \\
(1) & 4x_1 & -4x_2 & \geq & -2 \\
(2) & -2x_1 & -2x_2 & \geq & -3 \\
(3) & 8x_1 & -4x_2 & \geq & -1 \\
(4) & -x_1 & & \geq & -1 \\
(5) & & -k\ x_2 & \geq & -k \quad (k > 0) \\
 & x_1, & x_2 & \geq & 0
\end{array}
$$

Cuts from the disjunction $x_1 \leq 0$ OR $x_1 \geq 1$:

$$
\begin{array}{lrrcl}
(c1) & & 2x_2 & \leq & 1 \\
(c2) & -x_1 & +4x_2 & \leq & 1 \\
(c3) & -x_1 & +2x_2 & \leq & 1
\end{array}
$$



$(c1)$: corresponds to the basic solution of the CGLP (u_1, v_2, u_0, v_0), of value $z_1 = -\frac{2}{11}$, optimal for $k \leq 8$

$(c2)$: corresponds to the basic solution of the CGLP (u_3, v_2, u_0, v_0), of value $z_2 = -\frac{1}{6}$, never optimal

$(c3)$: corresponds to the basic solution of the CGLP (u_1, v_5, u_0, v_0), of value $z_3 = -\frac{k}{4+5k}$, optimal for $k \geq 8$

Nothing is perfect: Example 1 (cont.d)

- By using PORTA we can get a nice picture of what happens:

Nothing is perfect: Example 1 (cont.d)

- By using PORTA we can get a nice picture of what happens:

  1. in the space $(\gamma, \gamma_0, u, v, u_0, v_0)$ the cone has 117 extreme rays which results into 117 vertices once normalization (2) is applied.

Nothing is perfect: Example 1 (cont.d)

- By using PORTA we can get a nice picture of what happens:

  1. in the space $(\gamma, \gamma_0, u, v, u_0, v_0)$ the cone has 117 extreme rays which results into 117 vertices once normalization (2) is applied.

  2. only 6 of these vertices correspond to violated constraints and 3 are the ones shown in the previous slide.

Nothing is perfect: Example 1 (cont.d)

- By using PORTA we can get a nice picture of what happens:

  1. in the space $(\gamma, \gamma_0, u, v, u_0, v_0)$ the cone has 117 extreme rays which results into 117 vertices once normalization (2) is applied.

  2. only 6 of these vertices correspond to violated constraints and 3 are the ones shown in the previous slide.

  3. in the space (γ, γ_0), instead, the cone has only 4 extreme rays corresponding to the facets of the union of P_0 and P_1.

Nothing is perfect: Example 1 (cont.d)

- By using PORTA we can get a nice picture of what happens:

  1. in the space $(\gamma, \gamma_0, u, v, u_0, v_0)$ the cone has 117 extreme rays which results into 117 vertices once normalization (2) is applied.

  2. only 6 of these vertices correspond to violated constraints and 3 are the ones shown in the previous slide.

  3. in the space (γ, γ_0), instead, the cone has only 4 extreme rays corresponding to the facets of the union of P_0 and P_1.

  4. in other words, working on the extended space $(\gamma, \gamma_0, u, v, u_0, v_0)$ makes points in the interior of the polyhedron become vertices but this is independent of the normalization itself.

Nothing is perfect: Example 1 (cont.d)

- By using PORTA we can get a nice picture of what happens:

  1. in the space $(\gamma, \gamma_0, u, v, u_0, v_0)$ the cone has 117 extreme rays which results into 117 vertices once normalization (2) is applied.

  2. only 6 of these vertices correspond to violated constraints and 3 are the ones shown in the previous slide.

  3. in the space (γ, γ_0), instead, the cone has only 4 extreme rays corresponding to the facets of the union of P_0 and P_1.

  4. in other words, working on the extended space $(\gamma, \gamma_0, u, v, u_0, v_0)$ makes points in the interior of the polyhedron become vertices but this is independent of the normalization itself.

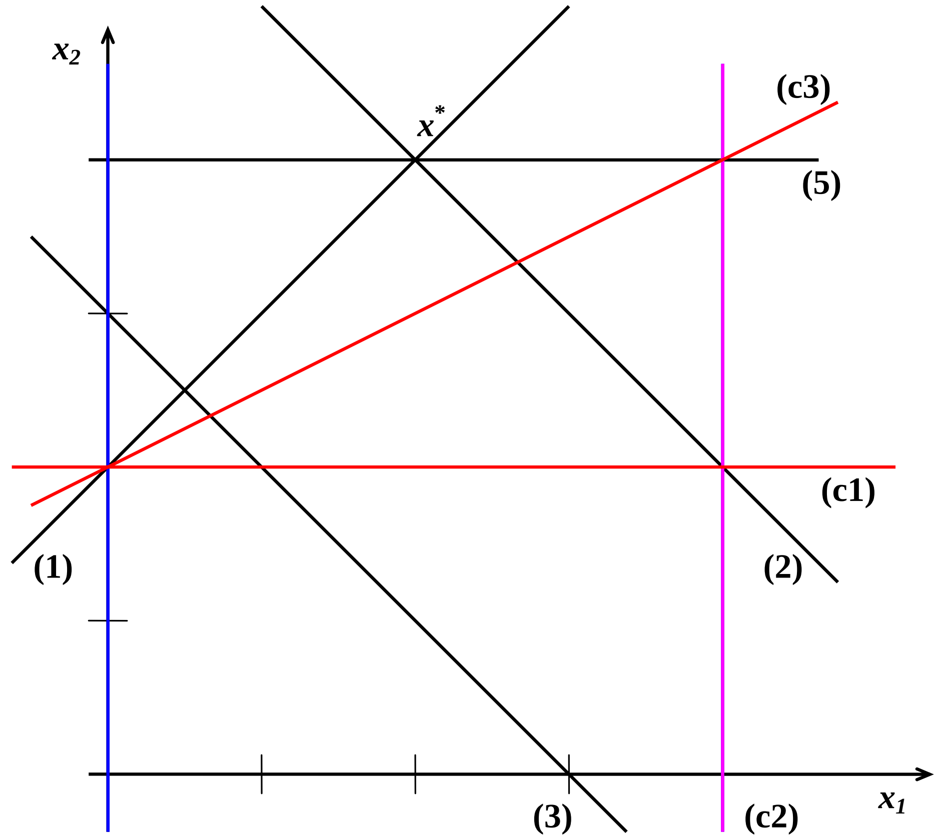
  5. however, the normalization changes the ranking of these vertices in terms of violation and this can result in very bad choices in terms of separated cuts.

Nothing is perfect: Example 2

$$\begin{array}{lrrcr} \min & -x_1 & -2x_2 & & \\ (1) & 2x_1 & -2x_2 & \geq & -1 \\ (2) & -2x_1 & -2x_2 & \geq & -3 \\ (3) & 4x_1 & +4x_2 & \geq & 3 \\ (4) & -x_1 & & \geq & -1 \\ (5) & & -x_2 & \geq & -1 \\ & x_1, & x_2 & \geq & 0 \end{array}$$

Cuts from the disjunction $x_1 \leq 0$ OR $x_1 \geq 1$:

$$\begin{array}{lrrcr} (c1) & & 2x_2 & \leq & 1 \\ (c2) & x_1 & & \geq & 1 \\ (c3) & -x_1 & +2x_2 & \leq & 1 \end{array}$$



$(c1)$: corresponds to the basic solution of the CGLP (u_1, v_2, u_0, v_0), of value $z_1 = -\frac{1}{6}$

$(c2)$: corresponds to the basic solution of the CGLP (u_1, u_3, u_0, v_0), of value $z_2 = -\frac{1}{22}$

$(c3)$: corresponds to the basic solution of the CGLP (u_1, v_5, u_0, v_0), of value $z_3 = -\frac{1}{10}$

$P_0 = \emptyset \Rightarrow x_1 \geq 1$ is a valid cut, but is not the best one for the CGLP

BCC eliminating redundant constraints

- Redundant constraints hurt!

BCC eliminating redundant constraints

- Redundant constraints hurt!
- Geometrically, they forbid the intersection cut to go as deep as possible and generally speaking the separated cuts can be NON supporting (as in the examples), i.e., slack with respect to P_0 and/or P_1.

BCC eliminating redundant constraints

- Redundant constraints hurt!
- Geometrically, they forbid the intersection cut to go as deep as possible and generally speaking the separated cuts can be NON supporting (as in the examples), i.e., slack with respect to P_0 and/or P_1.
- From a mathematical viewpoint, a redundant constraint can be obtained by conic combination of other constraints.

BCC eliminating redundant constraints

- Redundant constraints hurt!
- Geometrically, they forbid the intersection cut to go as deep as possible and generally speaking the separated cuts can be NON supporting (as in the examples), i.e., slack with respect to P_0 and/or P_1.
- From a mathematical viewpoint, a redundant constraint can be obtained by conic combination of other constraints. If the sum of the multipliers used to obtain it is > 1, then using a redundant constraint is cheaper (wrt normalization (2)) than using the constraints that generate it and it is a way of cheating wrt the normalization.

BCC eliminating redundant constraints

- Redundant constraints hurt!
- Geometrically, they forbid the intersection cut to go as deep as possible and generally speaking the separated cuts can be NON supporting (as in the examples), i.e., slack with respect to P_0 and/or P_1.
- From a mathematical viewpoint, a redundant constraint can be obtained by conic combination of other constraints. If the sum of the multipliers used to obtain it is > 1, then using a redundant constraint is cheaper (wrt normalization (2)) than using the constraints that generate it and it is a way of cheating wrt the normalization.
- Redundant constraints do not introduce new cuts but scaled copies of already existent cuts, i.e., additional vertices that, due to the cheating in the normalization, have a higher objective function (violation) and are then selected.

BCC eliminating redundant constraints

- Redundant constraints hurt!
- Geometrically, they forbid the intersection cut to go as deep as possible and generally speaking the separated cuts can be NON supporting (as in the examples), i.e., slack with respect to P_0 and/or P_1.
- From a mathematical viewpoint, a redundant constraint can be obtained by conic combination of other constraints. If the sum of the multipliers used to obtain it is > 1, then using a redundant constraint is cheaper (wrt normalization (2)) than using the constraints that generate it and it is a way of cheating wrt the normalization.
- Redundant constraints do not introduce new cuts but scaled copies of already existent cuts, i.e., additional vertices that, due to the cheating in the normalization, have a higher objective function (violation) and are then selected.
- The effect can be mitigated by getting rid of redundant constraints in the derivation of the cut.

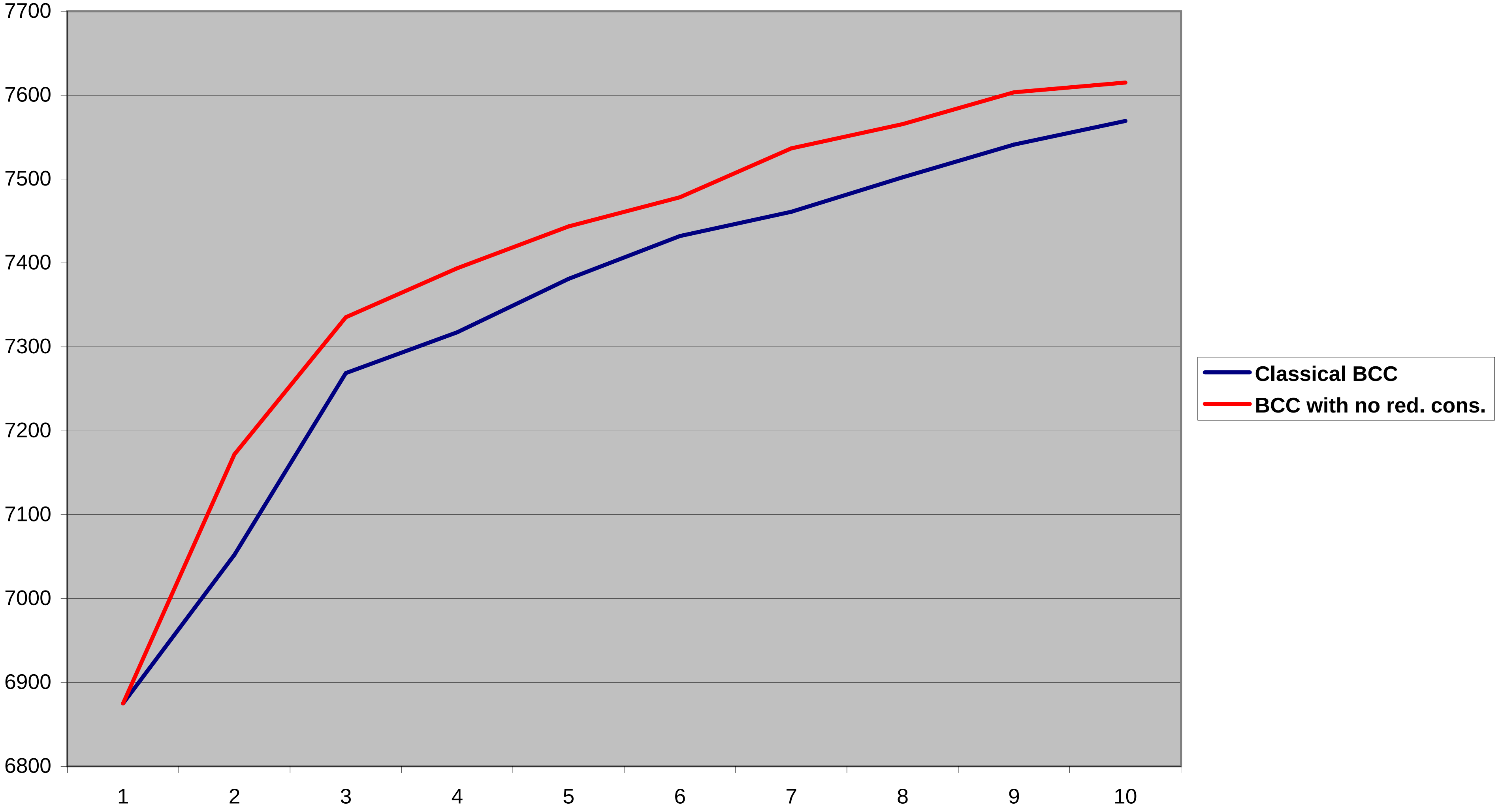
BCC eliminating redundant constraints

- Redundant constraints hurt!
- Geometrically, they forbid the intersection cut to go as deep as possible and generally speaking the separated cuts can be NON supporting (as in the examples), i.e., slack with respect to P_0 and/or P_1.
- From a mathematical viewpoint, a redundant constraint can be obtained by conic combination of other constraints. If the sum of the multipliers used to obtain it is > 1, then using a redundant constraint is cheaper (wrt normalization (2)) than using the constraints that generate it and it is a way of cheating wrt the normalization.
- Redundant constraints do not introduce new cuts but scaled copies of already existent cuts, i.e., additional vertices that, due to the cheating in the normalization, have a higher objective function (violation) and are then selected.
- The effect can be mitigated by getting rid of redundant constraints in the derivation of the cut.
- In Example 1 with PORTA, the CGLP without redundant constraints has only 9 extreme rays and 9 vertices. Only 1 corresponds to a violated constraint: $(c2) : -x_1 + 4x_2 \leq 1$.

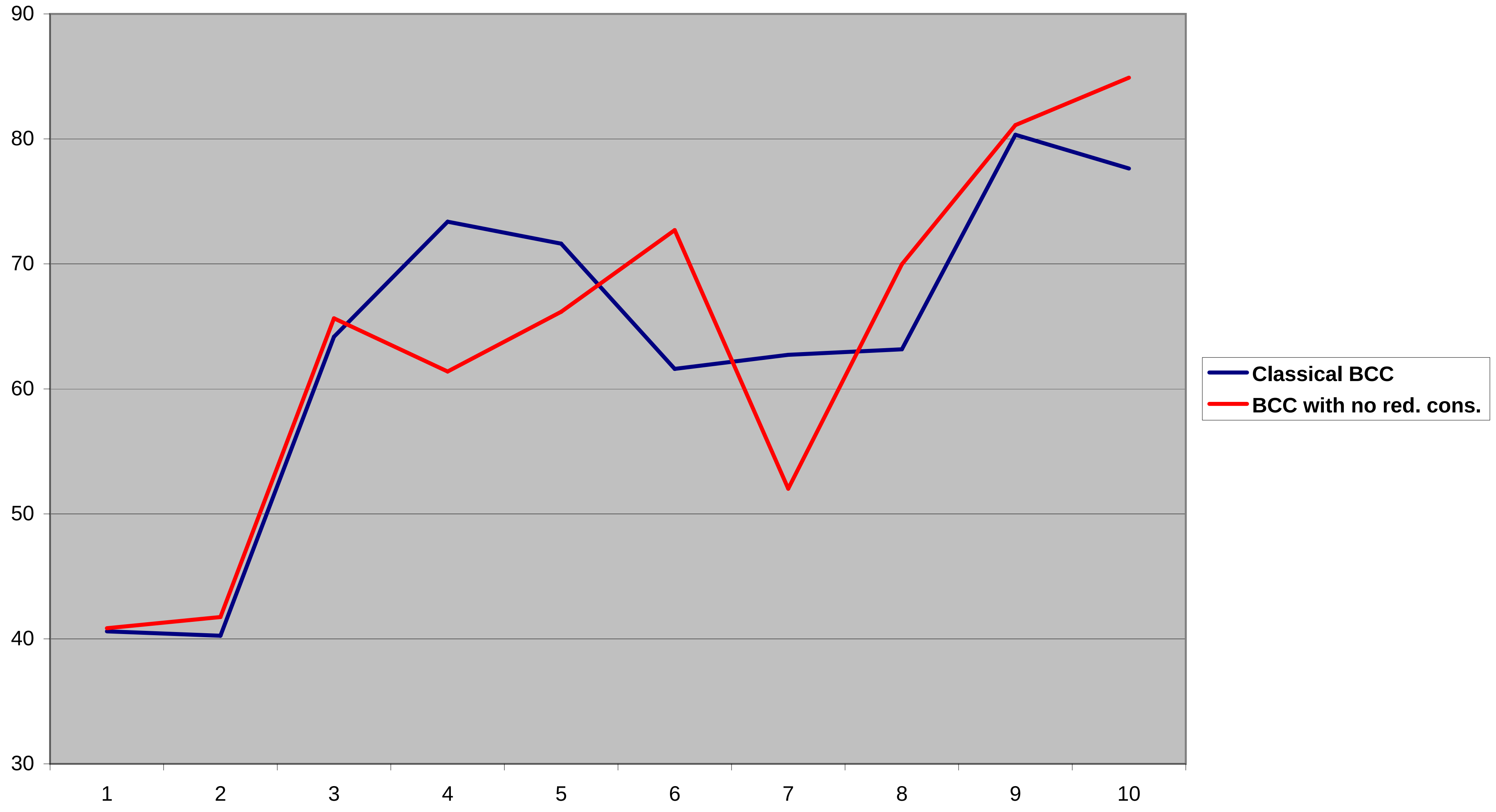
BCC eliminating redundant constraints

- Redundant constraints hurt!
- Geometrically, they forbid the intersection cut to go as deep as possible and generally speaking the separated cuts can be NON supporting (as in the examples), i.e., slack with respect to P_0 and/or P_1.
- From a mathematical viewpoint, a redundant constraint can be obtained by conic combination of other constraints. If the sum of the multipliers used to obtain it is > 1, then using a redundant constraint is cheaper (wrt normalization (2)) than using the constraints that generate it and it is a way of cheating wrt the normalization.
- Redundant constraints do not introduce new cuts but scaled copies of already existent cuts, i.e., additional vertices that, due to the cheating in the normalization, have a higher objective function (violation) and are then selected.
- The effect can be mitigated by getting rid of redundant constraints in the derivation of the cut.
- In Example 1 with PORTA, the CGLP without redundant constraints has only 9 extreme rays and 9 vertices. Only 1 corresponds to a violated constraint: $(c2) : -x_1 + 4x_2 \leq 1$.
- In the third set of experiments we eliminated redundant constraints in a trivial way (i.e., by solving LPs) before solving the CGLP. To get a full picture, we did not project the separation problem on the support of x^* (to be discussed later).

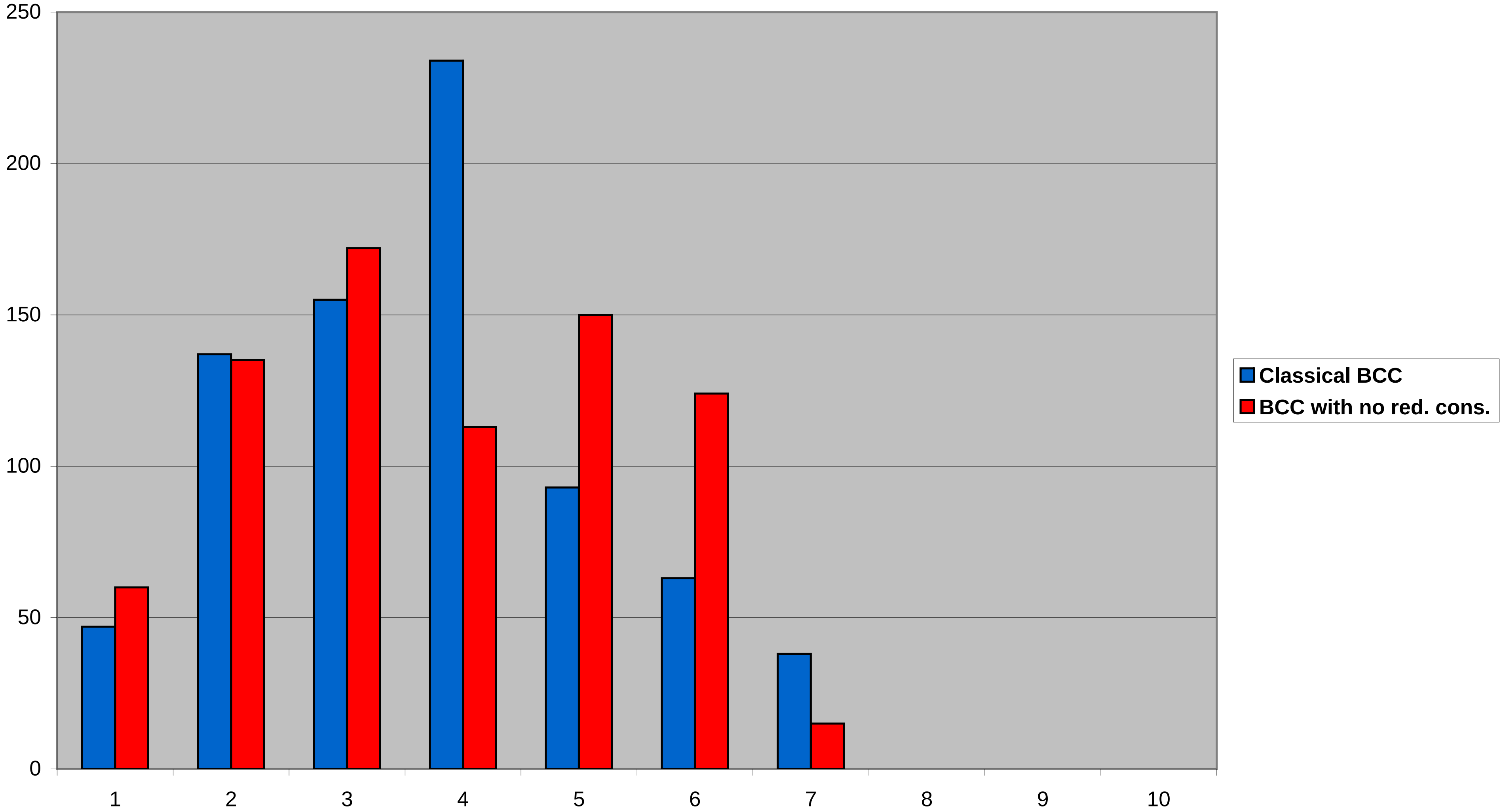
Instance p0201: lower bound



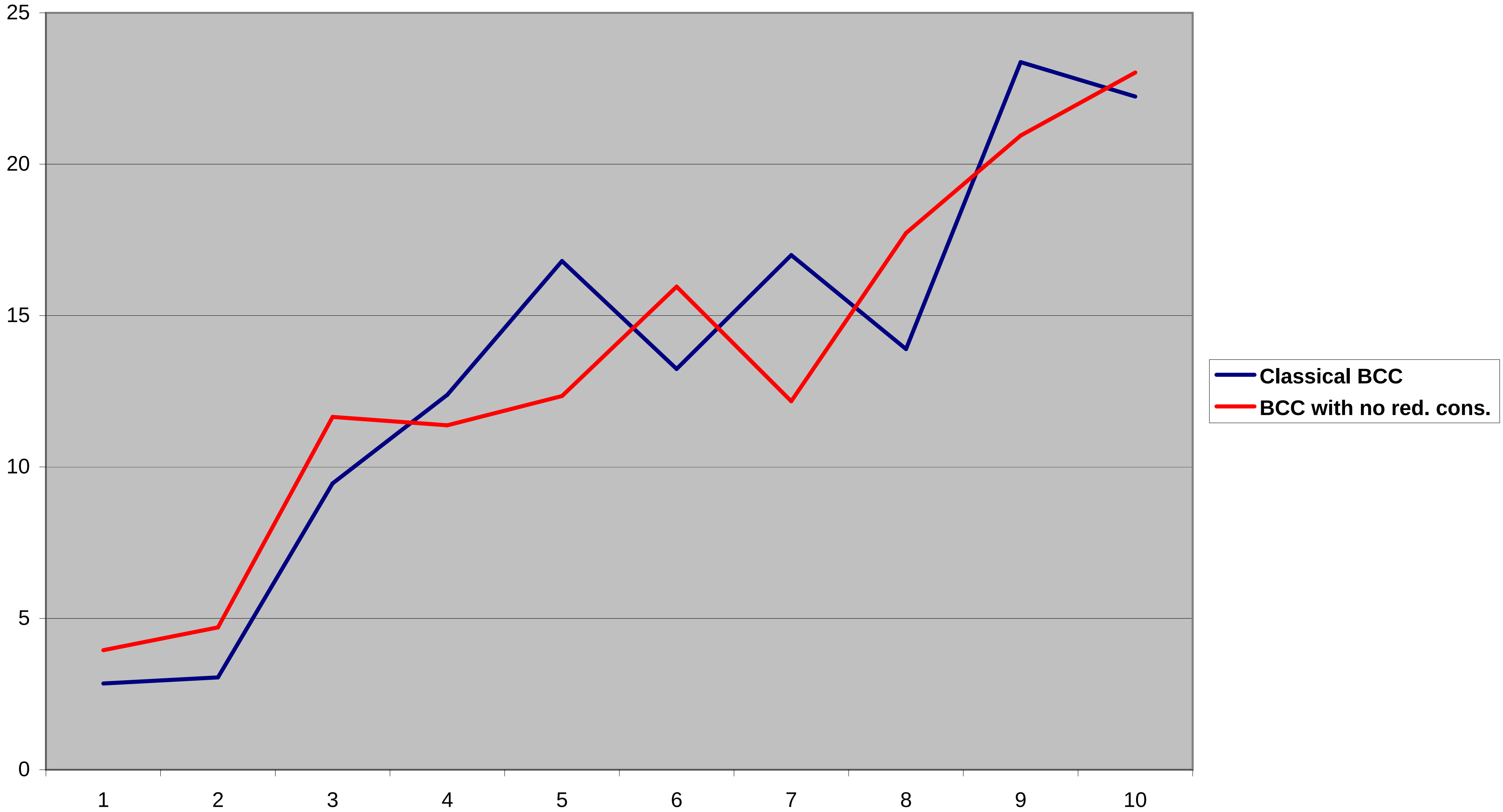
Instance p0201: average cuts' density



Instance p0201: cuts' rank



Instance p0201: average cardinality of (u, v)



In Summary

Table 3: 10 iterations of cuts. At each iteration one cut is generated from any fractional variable. No strengthening in the cut computation.

“Classical” BCC approach vs. “No redundancy” BCC approach with no projection						
	“Classical” BCC			“No redundancy” BCC		
Instance	n.cuts	gap%	$\#(u,v)$	n.cuts	gap%	$\#(u,v)$
bell3a	71	70.74	64.65	54	70.74	66.19
bell5	188	94.12	16.83	189	93.54	15.80
blend2	197	30.49	71.42	212	30.63	119.90
flugpl	93	18.34	6.45	90	18.83	6.48
gt2	218	94.13	58.11	167	93.68	63.16
lseu	171	42.46	23.86	184	45.10	30.96
*m.share1	77	0.00	55.99	77	0.00	56.00
mod008	107	15.46	304.18	107	15.48	304.19
p0033	116	57.25	8.75	126	70.32	10.99
p0201	692	92.53	23.40	757	98.31	37.44
rout	349	29.46	189.07	384	31.93	202.18
*stein27	251	0.00	7.29	249	0.00	6.46
vpm1	267	50.62	11.13	282	54.55	11.10
vpm2	390	74.73	24.23	376	76.47	22.82

In Summary

Table 3: 10 iterations of cuts. At each iteration one cut is generated from any fractional variable. No strengthening in the cut computation.

"Classical" BCC approach vs. "No redundancy" BCC approach with no projection						
	"Classical" BCC			"No redundancy" BCC		
Instance	n.cuts	gap%	$\#(u,v)$	n.cuts	gap%	$\#(u,v)$
bell3a	71	70.74	64.65	54	70.74	66.19
bell5	188	94.12	16.83	189	93.54	15.80
blend2	197	30.49	71.42	212	30.63	119.90
flugpl	93	18.34	6.45	90	18.83	6.48
gt2	218	94.13	58.11	167	93.68	63.16
lseu	171	42.46	23.86	184	45.10	30.96
*m.share1	77	0.00	55.99	77	0.00	56.00
mod008	107	15.46	304.18	107	15.48	304.19
p0033	116	57.25	8.75	126	70.32	10.99
p0201	692	92.53	23.40	757	98.31	37.44
rout	349	29.46	189.07	384	31.93	202.18
*stein27	251	0.00	7.29	249	0.00	6.46
vpm1	267	50.62	11.13	282	54.55	11.10
vpm2	390	74.73	24.23	376	76.47	22.82
avg.	238.250	55.861	66.840	244.000	58.298	74.267

Working on the support

- Projecting the separation problem into the support of x^* has of course the advantage of dealing with a problem of smaller size. However, the effect of the elimination of the redundant constraints can get lost.

Working on the support

- Projecting the separation problem into the support of x^* has of course the advantage of dealing with a problem of smaller size. However, the effect of the elimination of the redundant constraints can get lost.

- Consider a variable x_k such that $x^*_k = 0$. Then, one can project it out, not considering explicitly the constraints:
$$\gamma_k = u^T A_k = v^T A_k \tag{4}$$
and derive the coefficient γ_j afterwards by lifting it.

Working on the support

- Projecting the separation problem into the support of x^* has of course the advantage of dealing with a problem of smaller size. However, the effect of the elimination of the redundant constraints can get lost.

- Consider a variable x_k such that $x_k^* = 0$. Then, one can project it out, not considering explicitly the constraints:
$$\gamma_k = u^T A_k = v^T A_k \qquad (4)$$
and derive the coefficient γ_j afterwards by lifting it.

- However, if the constraint $x_k \geq 0$ is redundant, it can be very useful to explicitly write (4) above so as to be able to impose $u_{m+k} = 0$ and/or $v_{m+k} = 0$ (the multipliers associated with $x_k \geq 0$).

Working on the support

- Projecting the separation problem into the support of x^* has of course the advantage of dealing with a problem of smaller size. However, the effect of the elimination of the redundant constraints can get lost.

- Consider a variable x_k such that $x_k^* = 0$. Then, one can project it out, not considering explicitly the constraints:
$$\gamma_k = u^T A_k = v^T A_k \tag{4}$$
and derive the coefficient γ_j afterwards by lifting it.

- However, if the constraint $x_k \geq 0$ is redundant, it can be very useful to explicitly write (4) above so as to be able to impose $u_{m+k} = 0$ and/or $v_{m+k} = 0$ (the multipliers associated with $x_k \geq 0$).

- In other words, not stating explicitly (4), i.e., projecting, implies allowing the use of the constraint $x_k \geq 0$ in the separation of the cut which can be a very bad idea.

Working on the support

- Projecting the separation problem into the support of x^* has of course the advantage of dealing with a problem of smaller size. However, the effect of the elimination of the redundant constraints can get lost.

- Consider a variable x_k such that $x^*_k = 0$. Then, one can project it out, not considering explicitly the constraints:
$$\gamma_k = u^T A_k = v^T A_k \tag{4}$$
and derive the coefficient γ_j afterwards by lifting it.

- However, if the constraint $x_k \geq 0$ is redundant, it can be very useful to explicitly write (4) above so as to be able to impose $u_{m+k} = 0$ and/or $v_{m+k} = 0$ (the multipliers associated with $x_k \geq 0$).

- In other words, not stating explicitly (4), i.e., projecting, implies allowing the use of the constraint $x_k \geq 0$ in the separation of the cut which can be a very bad idea.

- This seems to be particularly crucial for the variable bounds and we defined an extended support of x^* by avoiding projecting out variables at the bound whose bound constraints are in turn redundant.

Working on the support: computation

Table 4: 10 iterations of cuts. At each iteration one cut is generated from any fractional variable. No strengthening in the cut computation.

"Classical" BCC approach vs. "No redundancy" BCC approach with cuts separated projected on the support

	"Classical" BCC				"No redundancy" support				"No redundancy" extended support			
Instance	n.cuts	gap%	supp%	$\#(u,v)$	n.cuts	gap%	supp%	$\#(u,v)$	n.cuts	gap%	supp%	$\#(u,v)$
bell3a	71	70.74	69.25	43.72	88	70.74	69.32	44.82	54	70.74	65.61	44.60
bell5	178	94.29	72.69	11.75	207	94.62	72.88	13.32	180	94.29	71.64	11.99
blend2	192	30.51	53.06	8.10	200	30.99	53.54	10.84	193	30.53	53.99	8.34
flugpl	92	18.36	86.11	5.85	93	18.94	86.11	5.89	93	18.86	86.29	5.95
gt2	196	93.46	18.30	10.28	191	94.13	18.14	10.58	187	93.88	20.00	13.10
lseu	196	41.33	29.44	9.17	191	40.16	27.08	12.28	178	43.45	29.41	9.08
*m.share1	74	0.00	11.94	1.39	130	0.00	13.39	2.56	77	0.00	12.59	1.69
mod008	139	17.05	4.51	12.41	136	17.70	4.42	12.17	157	19.13	5.85	14.43
p0033	113	67.86	55.76	4.81	106	70.32	55.76	5.74	146	70.29	58.84	5.89
p0201	767	93.82	45.02	13.43	873	81.59	43.43	25.83	769	100.00	48.93	13.39
rout	434	24.26	42.19	68.07	355	6.56	38.11	58.23	353	30.88	69.46	140.29
*stein27	252	0.00	93.70	6.53	252	0.00	93.70	6.68	251	0.00	93.61	7.13
vpm1	263	55.84	62.14	5.39	275	50.18	62.25	6.30	259	57.63	65.18	6.60
vpm2	403	74.96	64.74	17.27	377	75.30	65.08	18.10	373	75.84	67.15	17.71

Working on the support: computation

Table 4: 10 iterations of cuts. At each iteration one cut is generated from any fractional variable. No strengthening in the cut computation.

"Classical" BCC approach vs. "No redundancy" BCC approach with cuts separated projected on the support												
	"Classical" BCC				"No redundancy" support				"No redundancy" extended support			
Instance	n.cuts	gap%	supp%	$\#(u,v)$	n.cuts	gap%	supp%	$\#(u,v)$	n.cuts	gap%	supp%	$\#(u,v)$
bell3a	71	70.74	69.25	43.72	88	70.74	69.32	44.82	54	70.74	65.61	44.60
bell5	178	94.29	72.69	11.75	207	94.62	72.88	13.32	180	94.29	71.64	11.99
blend2	192	30.51	53.06	8.10	200	30.99	53.54	10.84	193	30.53	53.99	8.34
flugpl	92	18.36	86.11	5.85	93	18.94	86.11	5.89	93	18.86	86.29	5.95
gt2	196	93.46	18.30	10.28	191	94.13	18.14	10.58	187	93.88	20.00	13.10
lseu	196	41.33	29.44	9.17	191	40.16	27.08	12.28	178	43.45	29.41	9.08
*m.share1	74	0.00	11.94	1.39	130	0.00	13.39	2.56	77	0.00	12.59	1.69
mod008	139	17.05	4.51	12.41	136	17.70	4.42	12.17	157	19.13	5.85	14.43
p0033	113	67.86	55.76	4.81	106	70.32	55.76	5.74	146	70.29	58.84	5.89
p0201	767	93.82	45.02	13.43	873	81.59	43.43	25.83	769	100.00	48.93	13.39
rout	434	24.26	42.19	68.07	355	6.56	38.11	58.23	353	30.88	69.46	140.29
*stein27	252	0.00	93.70	6.53	252	0.00	93.70	6.68	251	0.00	93.61	7.13
vpm1	263	55.84	62.14	5.39	275	50.18	62.25	6.30	259	57.63	65.18	6.60
vpm2	403	74.96	64.74	17.27	377	75.30	65.08	18.10	373	75.84	67.15	17.71
avg.	253.667	56.873	50.268	17.521	257.667	54.269	49.677	18.675	245.167	58.793	53.529	24.281

Conclusions and Future Work

- We got some insights about the use of normalizations in the separation of disjunctive cuts.
- We have shown that such normalizations – even the good ones – are not fully safe.
- We have shown that redundant constraints hurt in the separation of disjunctive cuts.

Conclusions and Future Work

- We got some insights about the use of normalizations in the separation of disjunctive cuts.
- We have shown that such normalizations – even the good ones – are not fully safe.
- We have shown that redundant constraints hurt in the separation of disjunctive cuts.

- Even after the elimination of redundant constraints one might separate non supporting cuts. Can we do better?
- Can we come up with a better normalization (equivalently, a different objective function) such that the cheating effect of redundant constraints can be mitigated?
- Can we remove redundant constraints efficiently, e.g., in the framework of Balas & Perregaard?
- Can we separate directly on the (γ, γ_0) space?