Improved strategies for branching on general disjunctions

Gérard Cornuéjols[1], Leo Liberti[2], Giacomo Nannicini[2]

[1] *Carnegie Mellon University, Pittsburgh, PA, USA*, and *LIF, Faculté de Sciences de Luminy, Marseille, France*
[2] *LIX, École Polytechnique, France*

January 12, 2009

Summary of talk

Mixed-Integer Linear Programs

- A mathematical program with linear objective function, linear constraints and both continuous and integer variables is a *Mixed-Integer Linear Program* (MILP)
- MILPs arise in several real-life situations
- MILP solvers are often used as a tool in the context of solving Mixed-Integer Nonlinear Programs

Mixed-Integer Linear Programs

- Consider the following MILP in standard form:

$$\left.\begin{array}{rrcl} \min & c^\top x & & \\ & Ax & = & b \\ & x & \geq & 0 \\ \forall j \in N_I & x_j & \in & \mathbb{Z}, \end{array}\right\} \mathcal{P}$$

  where $c \in \mathbb{R}^n$, $b \in \mathbb{R}^m$, $A \in \mathbb{R}^{m \times n}$ and $N_I \subset N = \{1, \ldots, n\}$.
- The Linear Program (LP) relaxation of $\mathcal{P}$ is obtained by dropping the integrality constraints, and we denote it by $\bar{\mathcal{P}}$
- If the optimal solution $\bar{x}$ to $\bar{\mathcal{P}}$ is integral, then it is optimal for $\mathcal{P}$

Branch-and-Bound

- The standard method to solve MILPs is with a Branch-and-Bound (BB) algorithm
- There are three basic necessary ingredients in the BB algorithm:
    1. Obtaining lower bounds
    2. Obtaining upper bounds
    3. Dividing a subproblem

Branch-and-Bound

- The standard method to solve MILPs is with a Branch-and-Bound (BB) algorithm
- There are three basic necessary ingredients in the BB algorithm:
  1. Obtaining lower bounds $\leftarrow$ LP relaxation
  2. Obtaining upper bounds
  3. Dividing a subproblem

Branch-and-Bound

- The standard method to solve MILPs is with a Branch-and-Bound (BB) algorithm
- There are three basic necessary ingredients in the BB algorithm:
  1. Obtaining lower bounds $\leftarrow$ LP relaxation
  2. Obtaining upper bounds $\leftarrow$ LP relaxation, heuristics
  3. Dividing a subproblem

Branch-and-Bound

- The standard method to solve MILPs is with a Branch-and-Bound (BB) algorithm
- There are three basic necessary ingredients in the BB algorithm:
  1. Obtaining lower bounds $\leftarrow$ LP relaxation
  2. Obtaining upper bounds $\leftarrow$ LP relaxation, heuristics
  3. Dividing a subproblem $\leftarrow$ our focus

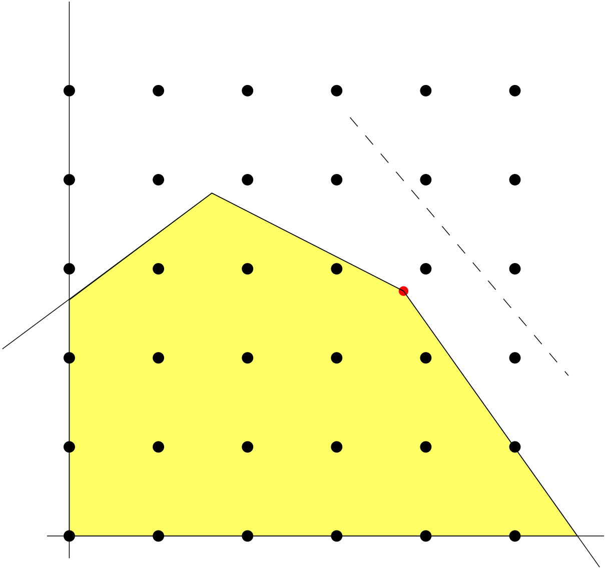
Branching on single variables

- Branching is usually done by changing the bounds of an integer constrained variable:
  - Let $\bar{x}$ be the optimal solution to $\bar{\mathcal{P}}$, and let $i \in N_I$ such that $\bar{x}_i$ is fractional
  - We divide $\mathcal{P}$ into $\mathcal{P}_1$ and $\mathcal{P}_2$ adding the constraints $x_i \leq \lfloor \bar{x}_i \rfloor$ (left branch) and $x_i \geq \lceil \bar{x}_i \rceil$ (right branch) to the two subproblems, respectively
- Very easy and fast approach

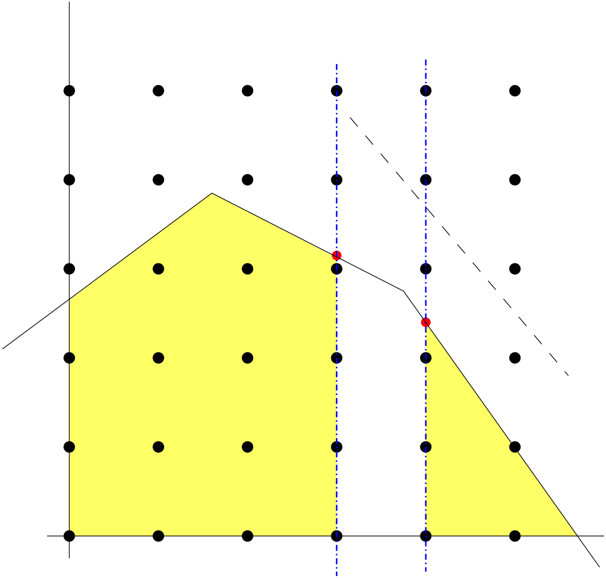
Branching on general disjunctions

- Branching can occur with respect to *any* direction $\pi \in \mathbb{R}^n$ by adding the constraints $\pi^\top x \le \beta_1$ and $\pi^\top x \ge \beta_2$ with $\beta_1 < \beta_2$ to $\mathcal{P}_1$ and $\mathcal{P}_2$ respectively, as long as no integer feasible point is cut off
- Can this be profitable with respect to branching on single variables?

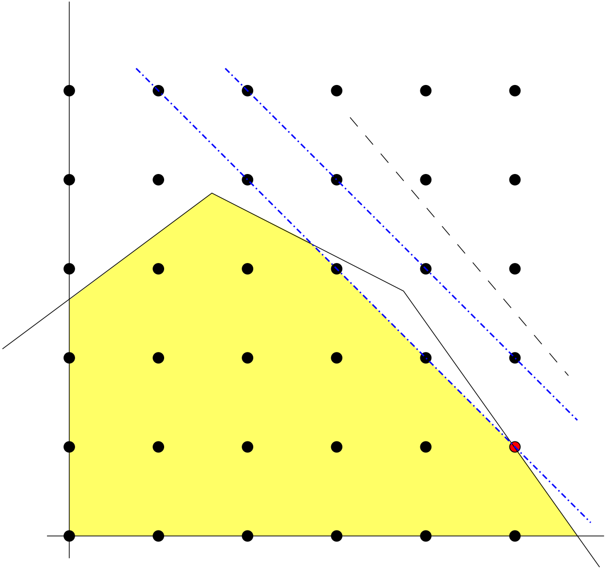
Example



Example



Example



Preliminaries

- Let $D(\pi, \pi_0)$ define the split disjunction $\pi^\top x \leq \pi_0 \vee \pi^\top x \geq \pi_0 + 1$, where $\pi \in \mathbb{Z}^n, \pi_0 \in \mathbb{Z}, \pi_j = 0$ for $i \notin N_I, \pi_0 = \lfloor \pi^\top \bar{x} \rfloor$
- By integrality of (π, π_0), any feasible solution to $\mathcal{P}$ satisfies every split disjunction

Definitions

- Let $B \subset N$ be an optimal basis of $\bar{\mathcal{P}}$, let $J = N \setminus B$ be the set of nonbasic variables
- The corresponding simplex tableau is given by:

$$x_i = \bar{x}_i - \sum_{j \in J} \bar{a}_{ij} x_j \quad \forall i \in B$$

- For $j \in J$, let $r^j \in \mathbb{R}^n$ be the extreme ray (associated with x_j) of the cone $\{x \in \mathbb{R}^n \mid Ax = b \wedge (x_j \geq 0 \; \forall j \in J)\}$ with apex $\bar{x}$
- The r^j's can be read directly from the simplex tableau

Intersection cuts

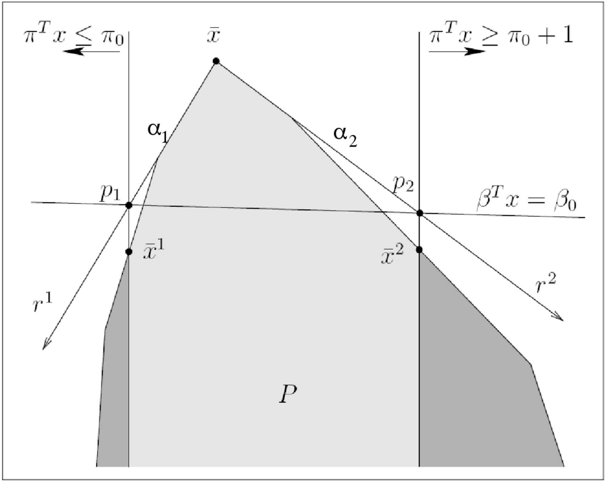
- Let $\epsilon(\pi, \pi_0) = \pi^\top \bar{x} - \pi_0$
- Assume that the disjunction $D(\pi, \pi_0)$ is violated by $\bar{x}$, i.e. $0 < \epsilon(\pi, \pi_0) < 1$
- The intersection cut associated with a basis B and a split disjunction $D(\pi, \pi_0)$ is

$$\sum_{j \in J} \frac{x_j}{\alpha_j(\pi, \pi_0)} \geq 1,$$

where $\forall j \in J$ we define

$$\alpha_j(\pi, \pi_0) = \begin{cases} -\frac{\epsilon(\pi,\pi_0)}{\pi^\top r^j} & \text{if } \pi^\top r^j < 0 \\ \frac{1-\epsilon(\pi,\pi_0)}{\pi^\top r^j} & \text{if } \pi^\top r^j > 0 \\ +\infty & \text{otherwise} \end{cases}$$

Intersection cuts and branching



Mixed-Integer Gomory Cuts

- Mixed-Integer Gomory Cuts can be seen as intersection cuts
- The split disjunction $D(\pi^i, \pi^i_0)$ that defines the MIGC associated to a row $\bar{a}_i$ of the simplex tableau where x_i is basic can be read directly from the simplex tableau
- The corresponding $\alpha_j(\pi, \pi_0)$ is

$$\alpha_j(\pi, \pi_0) = \begin{cases} \max\left(\frac{\bar{x}_i - \lfloor \bar{x}_i \rfloor}{\bar{a}_{ij} - \lfloor \bar{a}_{ij} \rfloor}, \frac{\lceil \bar{x}_i \rceil - \bar{x}_i}{\lceil \bar{a}_{ij} \rceil - \bar{a}_{ij}}\right) & \text{if } j \in J \cap N_I \\ \max\left(\frac{\bar{x}_i - \lfloor \bar{x}_i \rfloor}{\bar{a}_{ij}}, \frac{\lceil \bar{x}_i \rceil - \bar{x}_i}{-\bar{a}_{ij}}\right) & \text{if } j \in J \setminus N_I \end{cases}$$

- By convention, $\alpha_j(\pi, \pi_0)$ is equal to $+\infty$ when one of the denominators is zero

Improving the disjunctions

- We seek to find disjunctions that increase the α_j's
- We consider linear combinations with integer coefficients of the rows of the simplex tableau, so as to obtain new rows that give rise to "stronger" disjunctions
- Recall the formula: both terms of the fraction are nonlinear for $j \in J \cap N_I$, while only the numerator is nonlinear for $j \in J \setminus N_I$
- Thus, we focus on $j \in J \setminus N_I$: the numerator is nonlinear, but we can aim to obtain a denominator as close to zero as possible

Some more notation

- Let $B_I = B \cap N_I$ (basic integer variables), $J_C = J \setminus N_I$ (nonbasic continuous variables)
- Define the matrix $D \in \mathbb{R}^{|B_I| \times |J_C|}$ as the submatrix of A which contains the coefficients on the nonbasic continuous variables of the rows where an integer variable is basic
- The denominators of $\alpha_j(\pi, \pi_0)\, \forall j \in J \setminus N_I$ for the intersection cut associated with a row of the simplex tableau are exactly the elements of the corresponding row of D $\Rightarrow$ a reduction of $\|d_i\|$ should yield an increase in the α_j's
- We seek linear combinations (with integer coefficients) of the rows of D that minimize the norm of the resulting row

A quadratic optimization approach

- Idea: for each row d_k of D, choose a subset of the rows $R_k \subset B_I$, and reduce $\|d_k\|$ as much as possible with a linear combination of d_k and $d_j \; \forall j \in R_k$
- Quadratic convex minimization problem:

$$\min_{\lambda^k \in \mathbb{R}^{|R_k|}} \|d_k + \sum_{j \in R_k} \lambda_j^k d_j\|$$

- Can be solved via an $|R_k + 1| \times |R_k + 1|$ linear system
- We must round the coefficients λ_j^k to the nearest integer $\lfloor \lambda_j^k \rceil$ in order to get valid disjunctions

Final step

- Once the linear system is solved and we have the optimal coefficients $\lambda^k \in \mathbb{R}^{|R_k|}$, we do the following:
  1. Round them to the nearest integer
  2. Consider the norm of $d_k + \sum_{j \in R_k} \lfloor \lambda_j^k \rceil d_j$
  3. If $\|d_k + \sum_{j \in R_k} \lfloor \lambda_j^k \rceil d_j\| < \|d_k\|$, then we use the row $a_k + \sum_{j \in R_k} \lfloor \lambda_j^k \rceil \bar{a}_j$ instead of row $\bar{a}_k$ to compute the split disjunction
  4. Consider the possibly improved disjunction for branching, otherwise use the original row
- The norm reduction step can be applied to all rows of the simplex tableau in which an integer variable is basic or only to a subset

Choosing R_k

- The choice of $R_k \subset B_I \ \forall k$ is important
- We propose this approach to choose R_k associated with row $\bar{a}_k$:
  1. Fix a maximum number of rows $M_{|R_k|}$
  2. Pick the $M_{|R_k|}$ rows which have the smallest number of nonzero coefficients on the nonbasic integer variables on which $\bar{a}_k$ is zero
- Reason: we would like the coefficients on the variables $\in J \cap N_I$ that are zero in row $\bar{a}_k$ to be left unmodified when we compute $\bar{a}_k + \sum_{j \in R_k} \left\lfloor \lambda_j^k \right\rceil \bar{a}_j$

Computational experiments: tested methods

- Implementation within Cplex 11.0
- Comparison of the following branching methods:
  - branching on single variables (Simple Disjunctions, SD)
  - branching on split disjunctions after the reduction step that we propose (Improved General Disjunctions, IGD)
  - branching on the split disjunctions that define the MIGCs at the current basis (General Disjunctions, GD)
  - a combination of the SD and IGD method (Combined General Disjunction, CGD)

Choosing the branching direction

- We applied strong branching
- Best branching decision:
  - Generates the smallest number of feasible children, or, in the case of a tie,
  - Closes more gap, computed as $\min\{c^\top \bar{x}^1, c^\top \bar{x}^2\}$ where $\bar{x}^1, \bar{x}^2$ are the optimal solutions of the LP relaxations of the children nodes

Dataset and parameters

- Our testbed is the union of `miplib2.0`, `miplib3` and `miplib2003`, after the removal of all instances that can be solved to optimality in less than 10 nodes[1], and the instances where one node cannot be processed in less than 30 minutes[2]
- In total, the set consists in 96 heterogeneous instances
- The node selection strategy was set to *best bound*
- The value of the optimal solution was given as a cutoff value for all those instances where the optimum is known
- No heuristics, no cutting planes

[1] `air01, air02, air03, air06, misc04, stein09`

[2] `atlanta-ip, ds, momentum1, momentum2, momentum3, msc98-ip, mzzv11, mzzv42z, net12, rd-rplusc-21, stp3d`

Branching for 1000 nodes

- At each node, the most promising 10 branching directions are selected, then we apply strong branching
- For SD, we picked the 10 variables with largest fractionary part (i.e. closer to 0.5)
- For GD and IGD, we picked the 10 disjunctions associated with the MIGCs with largest distance cut off (closed form formula)
- We solved up to 1000 nodes, and compared closed integrality gap (if unsolved) or number of nodes (if solved)
- For IGD, we sum up at most 50 rows, i.e. $M_{|R_k|} = 49$
- For this experiment only, we had to exclude 7 instances[3], as they took too much time
- All averages in the following are geometric

[3] `dano3mip`, `fast0507`, `manna81`, `mitre`, `protfold`, `sp97ar`, `t1717`

Comparison: branching for 1000 nodes

Number of solved instances	
Simple disjunctions (SD):	35
General disjunctions (GD):	42
Improved general disjunctions (IGD):	42

Number of instances with largest closed gap	
Simple disjunctions (SD):	55
General disjunctions (GD):	56
Improved general disjunctions (IGD):	63

Comparison: branching for 1000 nodes

Average gap closed on instances not solved by any method	
Simple disjunctions (SD):	9.36%
General disjunctions (GD):	13.78%
Improved general disjunctions (IGD):	14.15%

Average number of nodes on instances solved by all methods	
Simple disjunctions (SD):	92.7
General disjunctions (GD):	52.9
Improved general disjunctions (IGD):	43.2

Analysis of the results

- Results suggest that IGD is capable of closing more gap per node on a large number of instances
- A more detailed analysis shows that there are a few instances where branching on general disjunctions is not profitable (2 instances are solved by SD but not by GD or IGD)
- This may also happen in zero gap instances, where the enumeration of nodes with SD is usually more effective
- We decided to combine both the SD and the IGD method into a single branching algorithm which tries to decide, for each instance, if it is more effective to branch on simple disjunctions or on general disjunctions

Practical considerations

- Branching on general disjunctions is slower than using simple disjunctions
- Branching on general disjunctions should be used only if it is truly profitable $\Rightarrow$ we used the amount of closed gap as a measure of profit
- As the polyhedron may change, general disjunctions should be periodically tested even when disabled

A combined branching algorithm

- At each node, branching on GDs can be active or not
  - If it is active, we test 10 possible branching decisions: 7 GDs, and 3 SDs
  - GDs are picked only if they generate a smaller amount of children nodes, or (in case of a tie) if the amount of closed gap is at least 50% larger
- At the beginning, branching on GDs is active for 3 nodes (increased effort at root node: 20 SDs, 20 GDs); whenever a GD is chosen, branching on GDs is activated for the following 10 nodes
- When it is deactivated, it is reactivated for one node after 100 nodes; permanently disabled after 10 consecutive unfruitful activations

Some more computational experiments

- Full test set (96 instances)
- Heuristics are disabled, but cutting planes are enabled (with default parameters)
- We run for 2 hours SD and CGD
- We compare number of nodes and closed integrality gap after 2 hours (unsolved instances), or number of nodes and solution time (solved instances)
- We measure gap closed only by branching, not by cuts

Comparison: branching for 2 hours

Number of solved instances	
Simple disjunctions (SD):	67
Combined general disjunctions (CGD):	70

Average number of nodes on instances solved by both methods	
Simple disjunctions (SD):	195.1
Combined general disjunctions (CGD):	98.0

Average CPU time [sec] on instances solved by both methods	
Simple disjunctions (SD):	3.03
Combined general disjunctions (CGD):	3.35

Easy instances

- Among examples that were solved by both algorithms:
  - `bell3a` required 15955 nodes using SD vs 20 using CGD
  - `bell5` required 773432 nodes using SD vs 24 using CGD
  - `gesa2` required 38539 nodes using SD vs 140 using CGD
- There is also an improvement in computing time by several orders of magnitude on these three instances

Comparison: branching for 2 hours

Average number of nodes on instances not solved by either method	
Simple disjunctions (SD):	35796.0
Combined general disjunctions (CGD):	15075.7

Average gap closed on instances not solved by either method	
Simple disjunctions (SD):	5.35%
Combined general disjunctions (CGD):	7.03%

Difficult instances

Instance	SD algorithm: Closed Gap Abs.	SD algorithm: Closed Gap Rel.	SD algorithm: Nodes	CGD algorithm: Closed Gap Abs.	CGD algorithm: Closed Gap Rel.	CGD algorithm: Nodes	Gap Closed By Cuts
10teams*	0	0%	11775	2	28.5%	398	71.3%
a1c1s1	337.58	3.21%	5340	371.423	3.54%	2578	62.29%
aflow40b	36.854	22.7%	20398	25.8243	15.9%	5477	57.3%
arki001	88.0556	6.83%	3612	580.27	45%	4000	28.27%
dano3mip	0.322586	-	8	0.374207	-	6	0%
danoint	0.310476	10.2%	5547	0.286139	9.44%	4790	2%
fast0507	0.262111	14.1%	587	0.0561795	3.03%	96	0%
gesa2_o*	84644.7	27.9%	195797	147352	48.5%	13181	51.4%
glass4	3293.85	0%	84369	3104.73	0%	79050	0%
harp2	199205	43.9%	74255	215937	47.5%	12565	32.6%
liu	214	-	108162	214	-	100347	0%
markshare1	0	0%	11027872	0	0%	2540405	0%
markshare2	0	0%	8606987	0	0%	2431791	0%
mas74	859.296	65.2%	2405902	641.509	48.7%	800207	4.6%
mkc	2.92749	6.1%	14486	6.52824	13.6%	8663	5.7%
noswot	0	0%	3192040	0	0%	1598812	0%
nsrand-ipx	158.293	6.82%	3932	222.726	9.6%	1431	49.08%
opt1217	0	0%	409010	1.33599	33.2%	316821	17%
protfold	2.32009	21.2%	140	2.14092	19.5%	150	3.6%
roll3000	127.615	7.12%	3083	293.192	16.4%	1406	40.68%
rout*	55.1337	57.6%	189312	94.9211	99.2%	28137	0.8%
set1ch	977.236	4.34%	120033	1355.82	6.02%	41034	86.06%
seymour	1.44368	7.54%	1251	1.09335	5.71%	688	41.66%
sp97ar	1.48955e+06	-	4231	1.41919e+06	-	318	0%
swath	28.3223	21.3%	20831	15.7973	11.9%	4724	34.9%
t1717	785.581	-	76	695.249	-	31	0%
timtab1	108754	14.8%	130014	103832	14.1%	35760	62.2%
timtab2	531157	-	50595	530454	-	12461	0%
tr12-30	183.374	0.158%	17852	691.388	0.594%	6883	99.142%

A notorious instance: arki001

- The arki001 instance has been first solved only recently by [Balas and Saxena, 2008]:
  - A large computational effort is invested to generate rank-1 split cuts that close 83.05% of the integrality gap
  - The remaining gap (16.95%) is closed by Cplex's BB algorithm in 643425 nodes
- If we run CGD on arki001 without time limits:
  - 28.27% of the integrality gap is closed by Cplex's cutting planes with default parameters
  - the remaining 71.73% is closed by our branching algorithm in 925738 nodes
- Note that Balas and Saxena used the preprocessed problem as input, while we always work with the original instances (i.e. without preprocessing)

Conclusions and future research

- In our experiments the combination between simple disjunctions and general disjunctions seems clearly superior to the traditional branching strategy
- The implementation of CGD could be made faster because Cplex's callable library is not optimized for branching on general disjunctions
- There is great potential in branching on general disjunctions, and it is useful for difficult instances: studying different methods to obtain good branching directions is promising for research
- Investigating the relationship between using split disjunctions for branching and to generate cutting planes is also interesting from a theoretical and computational point of view

...and that's all

Thank you!

Bibliography

Balas, E. and Saxena, A. (2008).
Optimizing over the split closure.
Mathematical Programming, 113(2):219–240.