

# Solving Integer and Disjunctive Programs

by

## Lift-and-Project

(Extended Abstract)

Sebastián Ceria<sup>1</sup> and Gábor Pataki<sup>2</sup> \*

<sup>1</sup> Graduate School of Business and  
Computational Optimization Research Center  
Columbia University, New York, NY 10027  
[sebas@cumparsita.gsb.columbia.edu](mailto:sebas@cumparsita.gsb.columbia.edu),  
<http://www.columbia.edu/~sc244>

<sup>2</sup> Department of Industrial Engineering and Operations Research and  
Computational Optimization Research Center  
Columbia University, New York, NY 10027  
[gabor@ieor.columbia.edu](mailto:gabor@ieor.columbia.edu),  
<http://www.ieor.columbia.edu/~gabor>

**Abstract.** We extend the theoretical foundations of the branch-and-cut method using lift-and-project cuts for a broader class of disjunctive constraints, and also present a new, substantially improved disjunctive cut generator. Employed together with an efficient commercial MIP solver, our code is a robust, general purpose method for solving mixed integer programs. We present extensive computational experience with the most difficult problems in the MIPLIB library.

## 1 Introduction

Disjunctive programming is optimization over a finite union of convex sets. Its foundations were developed, and the term itself coined in the early seventies by Balas [4, 5]; since then it attracted the attention of numerous researchers, including Jeroslow [18, 19], Blair [12], Williams [25], Hooker [15], Beaumont [11], Sherali and Shetty [23], Meyer [21]. Besides having an elegant theory, disjunctive programming provides a way to formulate a wide variety of optimization problems, such as mixed integer programs, linear complementarity, job-shop scheduling, equilibrium problems, and so on.

There is a natural connection between disjunctive programming problems and logic. In fact, a recent paper of Hooker and Osorio [16] proposes to solve discrete optimization problems that can be formulated by using logic and linear

---

\* Both authors were supported by NSF grant DMS 95-27-124

programming. They call this area of mathematical programming MLLP (Mixed Logical Linear Programming). Even though MLLP is, in principle, more general than disjunctive programming, since it allows more flexibility in the representation of logical formulas, every MLLP can be represented as a general disjunctive program.

Recently, Balas, Ceria and Cornuéjols [8,9] proposed the *lift-and-project* method, which is related to the work of Balas on disjunctive programming, the *matrix-cuts* of Lovász and Schrijver [20], the hierarchy of relaxations for mixed-integer programming of Sherali and Adams [22], and the *intersection cuts* of Balas [3]. The implementation of the lift-and-project method in a branch-and-cut framework (see [9]) proved to be very effective when tackling difficult mixed 0–1 programs.

The goal of our work is twofold. First, we show that the lift-and-project method for the 0–1 case can be extended quite naturally for a large class of disjunctions, called *facial* disjunctions. Such disjunctive constraints abound in practice, and our preliminary computational experience shows that – when the disjunctions are carefully chosen – the cuts generated from them outperform 0–1 disjunctive cuts.

Second, we attempt to answer the challenge posed by a new generation of commercial MIP solvers. We implemented a new, much improved lift-and-project cut generator, and we present our computational experience with it. Combining our separator with an efficient commercial code (CPLEX 5.0) in a cut-and-branch framework yields an extremely robust general purpose MIP solver.

In the rest of this section we provide the basic notation and definitions. In Section 2 we give a brief overview of known properties of facial disjunctive programs, and describe how disjunctive cuts generated from facial disjunctions can be lifted. In Section 3 we discuss implementation issues and present new computational results on the problems of MIPLIB 3.0 with a substantially improved version of a lift-and-project cut generator. Finally, in Section 4 we give our conclusions, discuss ongoing work, and future directions.

**Disjunctive sets and disjunctive cuts** A *disjunctive set* is a set of points satisfying a collection of inequalities connected by the logical connectors  $\wedge$  (conjunction, “AND” ) and  $\vee$  (disjunction, “OR”).

A disjunctive set is in *Disjunctive Normal Form* or *DNF*, if its terms do not contain further disjunctions. For simplicity, we shall be dealing with sets in *DNF* that contain only two terms, i.e. sets of the form

$$K_0 \cup K_1 \tag{1.1}$$

Moreover, we shall assume that there is a polyhedron  $K$  that contains both  $K_0$  and  $K_1$ , and  $K_0$  and  $K_1$  are defined by one additional inequality, that is

$$\begin{aligned} K &= \{x \mid Ax \geq b\} \\ K_j &= \{x \mid Ax \geq b, d^j x \geq g_j\} \quad (j = 0, 1) \end{aligned} \quad (1.2)$$

We shall denote

$$P = \text{cl conv} (K_0 \cup K_1) \quad (1.3)$$

A disjunctive set is in *conjunctive normal form*, or *CNF* if its conjunctions do not contain further conjunctions. E.g. if we are given the set  $K$  as above and

$$K_{ij} = \{x \mid Ax \geq b, d^{ij} x \geq g_{ij}\} \quad (i = 1, \dots, p, j = 0, 1)$$

then the set

$$\{x \mid \bigwedge_{i=1, \dots, p} (x \in K_{i0} \vee x \in K_{i1})\} \quad (1.4)$$

or, equivalently

$$\{x \mid Ax \geq b, \bigwedge_{i=1, \dots, p} (d^{i0} x \geq g_{i0} \vee d^{i1} x \geq g_{i1})\} \quad (1.5)$$

is in conjunctive normal form.

As an example, consider the feasible set of a mixed 0–1 program. In this case  $K$  is the feasible set of the linear programming relaxation, and

$$\begin{aligned} K_{i0} &= \{x \mid Ax \geq b, x_i \leq 0\} \\ K_{i1} &= \{x \mid Ax \geq b, x_i \geq 1\} \end{aligned}$$

for  $(i = 1, \dots, p)$ . Then with this definition of  $K_{i0}$  and  $K_{i1}$  (1.5) is the usual way of expressing the feasible set of the mixed 0–1 program. Moreover, if for some  $i$  we choose

$$K_0 = K_{i0}, \quad K_1 = K_{i1} \quad (1.6)$$

then the set in *DNF* is the strengthening of the LP-relaxation obtained by imposing the 0–1 condition on the variable  $x_i$ .

A *disjunctive program* (DP for short) is an optimization problem with the feasible set being a disjunctive set. Optimizing a linear function over the set in (1.1) can be done by optimizing over  $P$ .

If  $\Pi$  is a polyhedron, then we denote

$$\Pi^* = \{(\alpha, \beta) \mid \alpha x \geq \beta \text{ is a valid inequality for } \Pi\}$$

The next theorem, due to Balas, provides a representation of the set  $P$  in (1.1). This representation will be used in the next section as the basis for disjunctive cutting plane generation.

**Theorem 1.** [ (Balas[4]) ] *Assume that the sets  $K_0$  and  $K_1$  are nonempty. Then  $(\alpha, \beta) \in P^*$  if and only if there exists  $u^0, v_0, u^1, v_1$  such that*

$$\begin{aligned} u^j A + v_j d^j &= \alpha \quad (j = 0, 1) \\ u^j b + v_j g_j &\geq \beta \quad (j = 0, 1) \\ u^j, v_j &\geq 0 \quad (j = 0, 1) \end{aligned} \tag{1.7}$$

In fact, this result holds under a more general regularity condition, than the nonemptiness of all  $K_i$ 's (assuming nonemptiness, the result follows by simply using Farkas' lemma).

The lift-and-project method is based on the generation of *disjunctive or lift-and-project cuts*,  $\alpha x \geq \beta$  which are valid for  $P$ , and violated by the current LP-solution  $\bar{x}$ , i.e.  $\alpha \bar{x} < \beta$ . Theorem 1.7 provides a way of generating disjunctive cuts for a general disjunctive program through the solution of a linear program, called the *cut-generation LP*, or *CLP* of the form

$$\begin{aligned} \max \quad & \beta - \alpha \bar{x} \\ \text{s.t.} \quad & (\alpha, \beta) \in P^* \\ & (\alpha, \beta) \in S \end{aligned} \tag{1.8}$$

where  $S$  is a normalization set ensuring boundedness of the CLP.

There are several possible choices for the set  $S$  (see [13] for a complete description). In our current implementation we use the following normalization constraint:

$$S = \{(u^0, v^0, u^1, v^1) : \sum_{j=0}^1 (u^j + v^j)^T e \leq 1\}$$

where  $e$  is a vector of all ones of appropriate dimension.

## 2 Facial disjunctions

Our purpose is to use disjunctive cuts for solving general disjunctive programs. We will devote particular attention to a special class of *facial* disjunctive programs. A disjunctive set is called facial if the sets  $K_0$  and  $K_1$  are faces of  $K$ .

Some examples of facial and non-facial disjunctive programs include:

- **Variable upper bound constraints:** Suppose that we wish to model the following situation: if an arc  $i$  in a network is installed, we can send up to  $u_i$  units of flow on it; if it is not installed, we can send none. If we denote by  $y_i$

the amount of flow on the arc, and  $x_i$  is a 0–1 variable indicating whether the arc is installed, or not, then clearly,

$$\begin{aligned} x_i = 1 \vee y_i = 0 \\ 0 \leq y_i \leq u \end{aligned} \tag{2.9}$$

is a correct model.

- **Linear complementarity:** The linear complementarity problem (LCP for short) is finding  $x, z$  satisfying

$$\begin{aligned} x, z \geq 0 \\ Mx + z = q \\ x^T z = 0 \end{aligned} \tag{2.10}$$

Clearly, (2.10) is a facial disjunctive program in CNF, with the disjunctions being  $x_i = 0 \vee z_i = 0$ .

- **Ryan-Foster disjunctions:** This disjunction is used as a very successful *branching rule* in solving set-partitioning problems (SPP's). Precisely, suppose that the feasible set of an SPP is given as

$$\begin{aligned} x_k \in \{0, 1\} \forall k \\ Ax = e \end{aligned}$$

with  $A$  being a matrix of zeros and ones, and  $e$  the vector of all ones. Denote by  $R_i$  the *support* of the  $i^{\text{th}}$  row of  $A$ . Then the disjunction

$$\sum_{k \in R} x_k = 1 \vee \sum_{k \in R} x_k = 0 \tag{2.11}$$

is valid for all the feasible solutions of the SPP, if  $R$  is a subset of *any*  $R_i$ . However, if  $R$  is chosen as the *intersection* of  $R_{i_1}$  and  $R_{i_2}$  for two rows  $i_1$  and  $i_2$ , then the disjunction will perform particularly well as a branching rule, when solving the SPP by branch-and-bound. This disjunction is facial, since the inequalities obtained by replacing '=' by "≤" in (2.11) are valid for all solutions of the SPP. Some computational results using Ryan-Foster disjunctions to generate cutting planes can be found in Chapter 3.

- **Machine scheduling (see [6]):** In the machine scheduling problem we are given a number of operations that need to be performed on different items using a set of machines. The problem can be formulated using disjunctions of the form

$$t_j - t_i \geq d_{ij} \vee t_i - t_j \geq d_{ji} \tag{2.12}$$

where  $t_i$  is the starting time of job  $i$ , and  $d_{ij}$  is the minimum time that must elapse after the start of job  $i$  until the start of job  $j$ , if any. These disjunctions are *not* facial, since the reverse of the inequalities in (2.12) is clearly not valid for all of the feasible schedules.

One of the most important theoretical properties of 0–1 disjunctive cuts is the ease with which they can be *lifted* ([8,9]). For clarity, recall that

$$\begin{aligned} K &= \{x \mid Ax \geq b\} \\ K_j &= \{x \mid x \in K, d^j x \geq g_j\} \quad (j = 0, 1) \\ P &= \text{cl conv} (K_0 \cup K_1) \end{aligned} \tag{2.13}$$

Let  $K'$  be a face of  $K$ , and

$$\begin{aligned} K'_j &= \{x \mid x \in K', d^j x \geq g_j\} \quad (j = 0, 1) \\ P' &= \text{cl conv} (K'_0 \cup K'_1) \end{aligned} \tag{2.14}$$

Suppose that we are given a disjunctive cut  $(\alpha', \beta')$  valid for  $P'$  and violated by  $\bar{x} \in K'$ . Is it possible to quickly compute a cut  $(\alpha, \beta)$ , which is valid for  $P$ , and violated by  $\bar{x}$ ?

The answer is yes, if  $(\alpha', \beta')$  is a cut obtained from a 0–1 disjunction ([8,9]), and  $K'$  is obtained from  $K$  by setting several variables to their bounds. Moreover, it is not only sufficient to solve the CLP with the constraints representing  $K'$  in place of  $K$ , its size can be reduced by removing those columns from  $A$  which correspond to the variables at bounds. In practice these variables are the ones fixed in branch-and-cut, plus the ones that happen to be at their bounds in the optimal LP-solution at the current node. Cut lifting is vital for the viability of the lift-and-project method within branch-and-cut; if the cut generation LP's are solved with putting all columns of  $A$  into the CLP, the time spent on cut generation is an order of magnitude larger, while the cuts obtained are rarely better ([9]). The main result of this section is :

**Theorem 2.** *Let  $K'$  be an arbitrary face of  $K$  and  $K'_j$  and  $P'$  as above. Let  $(\alpha', \beta') \in (P')^*$  with the corresponding multipliers given. Then we can compute a cut  $(\alpha, \beta)$  that is valid for  $P$  and for all  $x \in K'$*

$$\alpha x - \beta = \alpha' x - \beta' \tag{2.15}$$

*Proof.* Somewhat surprisingly, our proof is even simpler, than the original for the 0–1 case. Let  $K'$  be represented as

$$K' = \{x \mid A_- x = b_-, A_+ x \geq b_+\}$$

where the systems  $A_- x \geq b_-$  and  $A_+ x \geq b_+$  form a partition of  $Ax \geq b$ . Since  $(\alpha', \beta') \in (P')^*$ , we have

$$\begin{aligned} \alpha &= u_-^0 A_- + u_+^0 A_+ + v_0 d^0 \\ &= u_-^1 A_- + u_+^1 A_+ + v_1 d^1 \\ \beta &\leq u_-^0 b_- + u_+^0 b_+ + v_0 g_0 \\ \beta &\leq u_-^1 b_- + u_+^1 b_+ + v_1 g_1 \end{aligned} \tag{2.16}$$

with  $u_+^0 \geq 0$ ,  $u_+^1 \geq 0$ ,  $v_0 \geq 0$ ,  $v_1 \geq 0$ , and  $u_-^0$  and  $u_-^1$  unconstrained. Then if we replace the negative components of  $u_-^0$  and  $u_-^1$  by 0, and compute the corresponding  $(\alpha, \beta)$  it will clearly be valid for  $P$  and satisfy (2.15).  $\square$

It is interesting to note that the above proof also implies that if the CLP is solved with a normalization that is imposed only on  $\beta$ , such as  $\beta = \pm 1$ , or  $|\beta| \leq 1$ , then the lifted cut will also be optimal. Hence the above theorem also generalizes the cut-lifting theorem in [8].

Suppose that after possibly complementing variables, multiplying rows by a scalar, adding rows, and permuting columns  $A_-$  and  $b_-$  can be brought into the form

$$A_- = [I, 0], b_- = 0$$

This condition is satisfied for most facial disjunctions of importance. Then just as in the 0–1 case, we can remove the columns from  $A$  that correspond to the columns of  $I$ , and solve the cut generation LP in the smaller space.

The consequence of these results is that facial disjunctions can be used for branching in a branch-and-cut algorithm, in place of the usual 0–1 branching. At any given node of the tree, the LP-relaxation is always a system that arises from the system defining  $K$  by imposing equality in some valid inequalities. In other words, the LP-relaxation at any given node defines a face of the LP-relaxation at the root. Therefore, if we also wish to generate disjunctive *cuts*, this can always be done using the LP-relaxation at the current node of the branch-and-cut tree, then lifting the resulting cut to be globally valid. Notice, that for this scheme to work, we only require the disjunctions for *branching* to be facial; the disjunctions for *cutting* can be arbitrary.

### 3 Computations

#### 3.1 The implementation

The computational issues that need to be addressed when generating lift-and-project cuts were thoroughly studied in [9]. Their experience can be briefly summarized as:

- It is better to generate cuts
  - in large rounds, before adding them to the linear programming relaxation, and reoptimizing.
  - in the space of the variables which are strictly between their upper and lower bounds, then to lift the cut to the full space.
- The distance of the current fractional point from the cut hyperplane is a reliable measure of cut quality.

We adopted most of their choices, and based on our own experience, we added several new features to our code, namely,

- In every round of cutting, we choose 50, (or less, if fewer are available) 0–1 disjunctions for cut generation. In an analogous way, we also choose a set of general integer disjunctions of the form  $x_i \leq \lfloor \bar{x}_i \rfloor \vee x_i \geq \lceil \bar{x}_i \rceil$ .
- The 50 0–1 disjunctions are chosen from a candidate set of 150. It is rather conceivable that a disjunction will give rise to a strong cut if and only if it would perform well when used for branching; i.e. the improvement of the objective function on the two branches would be substantial. Therefore, we use “branching” information to choose the variables from the candidate set. In the current implementation we used the **strongbranch** routine of CPLEX 5.0, which returns an estimate of the improvements on both generated branches. Our strategy is testing the 150 candidate variables (or fewer if less are available) then picking those 50 which maximize some function (currently we use the harmonic mean) of the two estimates. We call this procedure the *strong choice* of cutting variables. We then repeat this process for the general integer variables, if any. We are in the process of testing other rules commonly used for selecting branching variables, like pseudo-costs and integer estimates.
- We pay particular attention to the *accuracy* of the cuts. If a CLP turns out to be numerically unstable, we resolve it with a stricter tolerance setting.
- We use a normalization constraint (the set  $S$  in CLP) that bounds the sum of all multipliers  $(u^i, v^i)$ .
- In joint work with Avella and Rossi [1], we have chosen to generate more than one cut from one disjunction using a simple heuristic. After solving the CLP, we fix a nonzero multiplier to zero, then resolve. We repeat this procedure several times always checking whether the consecutive cuts are close to being parallel; if so, one of them is discarded.

### 3.2 The test-bed and the comparison

As a benchmark for comparison, we used the commercial MIP solver CPLEX 5.0, with the default parameters.

As the testbed, we used problems from MIPLIB 3.0, a collection of publicly available mixed integer programming problems. We excluded those problems which were too easy for CPLEX, namely the ones that could be solved within 100 nodes, and also the *fast0507* problem, since it is too large. We divided the remaining problems into two groups.

- “Hard” problems; the ones that cannot be solved within one thousand seconds by CPLEX with the default setting.
- “Medium” problems. All the rest.

Finally, since the enumeration code of MIPO was written 4 years ago, currently it is not competitive with the best commercial solvers. Therefore, we tested our cut-generator in a cut-and-branch framework. We generated 2 and 5 rounds of 50-100 cuts, after every round adding them to the LP formulation, reoptimizing, and dropping inactive constraints. After the fifth round we fed the strengthened formulation to the CPLEX 5.0 MIP solver with the above setting. Also, all the cut-generation LP's were solved using the CPLEX dual simplex code.

Problem	LP value	IP value
10teams	897.00	904.00
air04	55,264.43	55,866.00
air05	25,877.60	26,374.00
arki001	7,009,391.43	7,010,963.84
gesa2	25,476,489.68	25,781,982.72
gesa2_o	25,476,489.68	25,781,982.72
harp2	-74,325,169.35	-73,893,948.00
misc07	1415.00	2,810.00
mod011	-62,121,982.55	-54,558,535.01
modglob	20,430,947.62	20,740,51
p6000	-2,350,838.33	-2,349,787.00
pk1	0.00	11.00
pp08a	2748.35	7350.00
pp08aCUTS	5480.61	7350.00
qiu	-931.64	-132.87
rout	-1393.38	-1297.69
set1ch	30426.61	49,846.25
vpm2	9.89	13.75

**Table 1.** Problem description

All of our tests were performed on a Sun Enterprise 4000 with 8-167MHz CPU; we set a memory limit of 200 MB, and ran all our tests using one processor only.

### 3.3 The Computational Results

The results for those “hard” problems which could be solved with, or without cuts, are summarized in Table 2. There are 17 such problems. Their description,

and also of *set1ch* is included in Table 1. The problems not solved by any of the two methods are: *danoimt*, *dano3mip*, *noswot*, *set1ch*, *seymour*. Nevertheless, on the last two problems lift-and-project cuts perform quite well; *set1ch* can be solved with 10 rounds, and on *seymour* we were able to get the best bound known to date (see the next section).

Also, the medium problems were run, and solved by CPLEX and cut-and-branch as well. The comparisons for these problems are not presented here, but cut-and-branch was roughly twice as fast if we aggregate all the results.

Problem	CPLEX 5.0		Cut-and-Branch <sub>2</sub>		Cut-and-Branch <sub>5</sub>	
	Time	Nodes	Time	Nodes	Time	Nodes
10teams	5404	2265	1274	306	5747	1034
air04	2401	146	1536	110	5084	120
air05	1728	326	1411	141	4099	213
gesa2	9919	86522	3407	22601	1721	6464
gesa2_o	12495	111264	4123	28241	668	4739
modglob	+++	+++	10033	267015	435	5623
pp08a	+++	+++	1924	47275	178	1470
pp08aCUTS	50791	1517658	277	3801	134	607
vpm2	8138	481972	1911	63282	974	18267
harp2	14804	57350	10686	28477	13377	31342
misc07	2950	15378	2910	12910	4133	14880
p6000	1115	2911	1213	2896	805	1254
qiu	35290	27458	15389	10280	27691	15239
arki001	6994	21814	18440	68476	13642	12536
mod011	22344	18935	63481	24090	+++	+++
pk1	3903	130413	5094	122728	6960	150243
rout	19467	133075	26542	155478	40902	190531

**Table 2.** Computational results for cut-and-branch

The following preliminary conclusions can be drawn.

- (1) In 9 problems out of the 17, either 2, or 5 rounds (in most cases 5) of lift-and-project cuts substantially improve the solution time. In 5 problems the difference is “make-or-break”; between solving, or not solving a problem, or improving the solution time by orders of magnitude.
- (2) On 4 problems, our cuts do not make much difference in the solution time.

- (3) On 4 problems, adding our cuts is actually detrimental. It is important to note, that the deterioration in the computing time is *not* due to the time spent on generating the cuts, rather to the fact, that they make the linear programming relaxation harder to solve. In fact, it is most likely possible to catch this effect by monitoring, e.g. the density of the cuts, and the deterioration of the LP relaxation's condition number.

**Lift-and-project cuts on two very difficult problems** There were 5 problems that neither CPLEX alone, nor our code (with at most 5 rounds of cuts) was able to solve. These are : *danoimt*, *dano3mip*, *noswot*, *set1ch* and *seymour*. All of them are notoriously hard, and currently unsolvable by general purpose MIP-solvers within a reasonable time. In fact, *dano3mip*, *noswot*, and *seymour* have never been solved to optimality (although an optimal value for *noswot* is reported in MIPLIB 3.0, we could not find anyone to confirm the existence of such a solution).

Our cuts do not perform well on the first 3 problems; *danoimt* and *dano3mip* are network design problems with a combinatorial structure, already containing many special purpose cuts, and *noswot* is highly symmetric. However, disjunctive cuts perform strikingly well on the last two instances.

Until now, *set1ch* could be solved to optimality only by using special purpose *path-inequalities* [26]. After exhausting the memory limits, CPLEX could only find a solution within 15.86 % of the optimum. We ran our cutting plane generator for 10 rounds, raising the lower bound to within 1.4 % of the integer optimum. CPLEX was then able to solve the strengthened formulation in 28 seconds by enumerating 474 nodes. It is important to note that no numerical difficulties were encountered during the cutting phase, (even if we generated 15 rounds, although this proved unnecessary) and the found optimal solution precisely agrees with the one reported in MIPLIB (the objective coefficients are one-fourth integral).

The problem *seymour* is an extremely difficult setcovering instance; it was donated to MIPLIB by Paul Seymour, and its purpose is to find a minimal "irreducible configuration" in the proof of the four-colour conjecture. It has not been solved to optimality. The value of the LP-relaxation is 403.84, and an integer solution of 423.0 is known. The best previously known *lower* bound of 412.76 [2] was obtained by running CPLEX 4.0 on an HP SPP2000 with 16 processors, each processor having 180 MHz frequency, and 720 Mflops peak performance, for the total of approximately 58 hours and using approx. 1360 Mbytes of memory.

Due to the difficulty of the problem, we ran our cutting plane algorithm on this problem with a rather generous setting. We generated 10 rounds of cuts, in each round choosing the 50 cutting variables picked by our strong choice from among *all* fractional variables with the iteration limit set to 1000. The total time spent on generating the 10 rounds was approximately 10.5 hours, and the lower bound was raised to 413.16. The memory useage was below 50Mbytes. Running

CPLEX 4.0 on the strengthened formulation for approximately 10 more hours raised the lower bound to 414.20 - a bound that currently seems unattainable without using our cuts.

**Computational results with other disjunctions** We ran our cut-generator on two of the most difficult set-partitioning problems in MIPLIB, namely *air04* and *air05*, by using the Ryan-Foster disjunctions described in the previous section. The results with two rounds of cuts are summarized in Table 3.

Problem	CPLEX 5.0	CPLEX 5.0	C&B	C&B
	Time	Nodes	Time	Nodes
air04	2401	146	1300	115
air05	1728	326	1150	123

**Table 3.** Results with the Ryan-Foster disjunctions

## 4 Conclusions and Future Directions

In the near future we plan to explore the following topics:

- Making our computational results with cut-and-branch more consistent. The key here is, finding the right amount of cutting, that sufficiently strengthens the LP-relaxation, but does not make it too difficult to solve.
- We are currently implementing a branch-and-cut method that uses disjunctive cuts and allows branching on facial disjunctions, using cut lifting based on Theorem 2. We will use the disjunctions which are given as part of the formulation, and in some other cases, we will use the structure of the problem to generate other valid disjunctions. Our goal is to treat disjunctions in a way similarly to inequalities (that is, to maintain a set of “active” disjunctions, and to keep the rest in a “pool”), and handle them efficiently throughout the code.
- We are in the process of testing our cutting plane generator with other commercial LP and MIP solvers (XPRESS-MP). This program allows for the generation of cutting planes within the enumeration tree without the need of programming our own enumeration, and hence improving on the efficiency.

## References

1. P. Avella and F. Rossi, Private communication.

2. G. Astfalk and R. Bixby, Private communication.
3. E. Balas, Intersection cuts – A new type of cutting planes for integer programming, *Operations Research* 19 (1971) 19-39.
4. E. Balas, Disjunctive programming: facets of the convex hull of feasible points, Technical Report No. 348, GSIA, Carnegie Mellon University (1974).
5. E. Balas, Disjunctive programming, *Annals of Discrete Mathematics* 5 (1979) 3-51.
6. E. Balas, Disjunctive programming and a hierarchy of relaxations for discrete optimization problems, *SIAM J. Alg. Disc. Meth.* 6 (1985) 466-486.
7. E. Balas, Enhancements of lift-and-project, Technical Report, GSIA, Carnegie Mellon University, (1997).
8. E. Balas, S. Ceria and G. Cornuéjols, A lift-and-project cutting plane algorithm for mixed 0–1 programs, *Mathematical Programming* 58 (1993) 295-324.
9. E. Balas, S. Ceria and G. Cornuéjols, Mixed 0–1 Programming by lift-and-project in a branch-and-cut framework, *Management Science* 42 (1996) 1229-1246.
10. E. Balas, S. Ceria, G. Cornuéjols and G. Pataki, Polyhedral Methods for the Maximum Clique Problem, *AMS, Dimacs series on Discrete Mathematics and Computer Science* 26 11-28 (1996).
11. N. Beaumont, An algorithm for disjunctive programs, *European Journal of Operations Research* 48 (1990) 362-371.
12. C. Blair, Two rules for deducing valid inequalities for 0-1 problems, *SIAM Journal of Applied Mathematics* 31 (1976) 614-617.
13. S. Ceria and J. Soares, Disjunctive cuts for mixed 0–1 programming: duality and lifting, Working paper, Graduate School of Business, Columbia University (1997).
14. S. Ceria and J. Soares, Convex programming for disjunctive optimization, Working paper, Graduate School of Business, Columbia University (1997).
15. J. Hooker, Logic based methods for optimization, in A. Borning, ed., *Principles and practice of constraint programming, Lecture Notes in Computer Science* 626 (1992) 184-200.
16. J. Hooker, M. Osorio, Mixed logical/linear programming, Technical Report, GSIA, Carnegie Mellon University (1997).
17. J. Hooker, H. Yan, I. Grossman and R. Raman, Logic cuts for processing networks with fixed charges, *Computers and Operations Research* 21 (1994) 265-279.
18. R. Jeroslow, Representability in mixed-integer programming I: Characterization results, *Discrete Applied Mathematics* 17 (1987), 223-243.
19. R. Jeroslow, Logic based decision support: mixed-integer model formulation, *Annals of Discrete Mathematics* 40, (1989) North Holland, Amsterdam.
20. L. Lovász and A. Schrijver, Cones of matrices and set-functions and 0–1 optimization, *SIAM J. Optimization* 1 (1991) 166-190.
21. R. Meyer, Integer and mixed-integer programming models: general properties, *Journal of Optimization Theory and Applications* 16 (1975) 191-206.
22. H. Sherali and W. Adams, A hierarchy of relaxations between the continuous and convex hull representations for zero-one programming problems, *SIAM J. Disc. Math.* 3 (1990) 411-430.
23. H. Sherali and C. Shetty, Optimization with disjunctive constraints, In M. Beckman and H. Kunzi, editors, *Lecture notes in Economics and Mathematical Systems* 181, Springer-Verlag, (1980).
24. J. Soares, *Disjunctive methods for discrete optimization problems*, Ph.D. Thesis (in preparation), Graduate School of Business, Columbia University (1997).
25. H.P. Williams, An alternative explanation of disjunctive formulations, *European Journal of Operations Research* 72 (1994) 200-203.
26. L. Wolsey, Private communication.