Pivot and Gomory Cut: a 0-1 mixed integer programming heuristic

Shubhashis Ghosh[1] and Ryan Hayward[1]

Department of Computing Science, University of Alberta
{shubhash,hayward}@cs.ualberta.ca
The research support of NSERC is gratefully acknowledged.

Abstract. We introduce MIP heuristics based on combining Gomory cuts with simplex pivoting and neighbourhood search. Pivot and Gomory Cut (PGC), which integrates Gomory cuts into the bounded variable revised simplex pivoting framework of Balas and Martin, attempts only to find some feasible solution within an allotted time. Neighbourhood Pivot and Gomory Cut (NPGC), an enhancement of PGC obtained by adding a Gomory-cut-guided neighbourhood search subroutine similar to that used by Fischetti and Lodi, attempts to find the best possible feasible solution within the allotted time. Both heuristics are easily implemented in conjunction with a linear programming solver; we implemented them on top of the Gnu Linear Programming Kit. When tested on problems compiled from MIPLib and other sources and compared to similar implementations of other heuristics, PGC and NPGC performed well.

1 Introduction

Mixed integer programming (MIP) problems arise in many contexts. Since they are NP-hard to solve, not only in general [15] but even when restricted to 0-1 programs [21], there is interest in finding heuristic algorithms that, while they may fail to find an optimal or indeed a feasible solution, will execute in a reasonable amount of time.

Although regarded since their introduction in 1960 [14] as theoretically significant, the practical importance of Gomory cuts was not recognized until the 1990s, when Balas, Ceria, Cornuéjols and Natraj used Gomory cuts to strengthen branch and bound IP solvers [1, 7]. Here, we show that Gomory cuts can also be used to strengthen IP heuristic algorithms. In particular, we introduce two 0-1 MIP heuristics which exploit the ease with which Gomory cuts can be combined with simplex pivoting [9].

Pivot and Gomory Cut (PGC) is designed around the bounded variable revised simplex (BVRS) framework as used in the search phase of Pivot and Complement (PC), the classic 0-1 IP heuristic of Balas and Martin in 1980 [3]. Neighbourhood Pivot and Gomory Cut (NPGC) extends PGC by incorporating neighbourhood search as used in the Local Branching (LB) heuristic of Fischetti and Lodi [13]. PGC and NPGC use Gomory cuts in several ways, namely to select pivots, to avoid cycling, to restart whenever a dead end is reached, and (in NPGC) to guide the neighbourhood search.

PGC starts with the usual linear programming relaxation step: the input 0-1 integer program P is transformed into an equivalent program P^+ by adding slack variables to accompany the decision variables of P, and the BVRS method is then used to look for an optimal solution S to the linear programming relaxation $\mathcal{L}(P^+)$ of P^+. If no such solution is found, the original program P is infeasible; otherwise, a feasibility subroutine uses BVRS pivots on the simplex tableau associated with $\mathcal{L}(P^+)$ to search for a P-feasible solution.

The feasibility subroutine has two phases, search and restart. The search phase begins by finding a Gomory cut separating S from the P-polytope. The extent to which S is P-infeasible (with respect to various tableau measures that we will explain later) is decreased by repeated pivoting. During this pivoting, if the current basic feasible solution S' of $\mathcal{L}(P^+)$ satisfies (namely, is on the P-polytope side of) the Gomory cut, then the cut is added to the formulation of $\mathcal{L}(P^+)$ and a new Gomory cut separating S' from the P-polytope is generated. This generation and occasional addition of Gomory cuts to $\mathcal{L}(P^+)$ continues until the search phase either finds a P-feasible solution or reaches a dead end. In the latter case, execution transfers to the restart phase together with the most recently unsatisfied Gomory cut.

The restart phase begins by finding a basic feasible solution of $\mathcal{L}(P^+)$ which crosses the unsatisfied Gomory cut, at the cost of possibly increasing the infeasibility measure. In the same way that cuts are added in the search phase, this cut is added to the formulation of $\mathcal{L}(P^+)$, a feasible solution to the new $\mathcal{L}(P^+)$ is found, and execution transfers back to the search phase.

The operations we have described so far can be easily and efficiently integrated into the BVRS framework. In particular, in the search phase Gomory cuts are added only when the simplex tableau is satisfied by the Gomory cut. Since at that point no further pivoting is required to make the tableau feasible, the only elementary operations that need to be performed in the feasibility subroutine are simplex pivots.

While the only goal of PGC is to find a feasible solution, the goal of NPGC is to find the best possible feasible solution within the allotted time. NPGC is essentially an extension of PGC whose main enhancement is the performance of cut-guided neighbourhood search whenever a promising $\mathcal{L}(P^+)$ feasible solution is found.

In NPGC, we apply the search phase of PGC with a certain stopping criterion. We define a search neighbourhood using a Gomory cut and apply a black box MIP-solver to find the best integer feasible solution within a certain time limit. If an integer feasible solution is obtained, we consider a 0-1 complementation based improvement neighbourhood as defined by Fischetti and Lodi [13] and apply the MIP-solver with a time limit. As long as the MIP-solver gives a new integer feasible solution, we consider an improvement neighbourhood based on the new feasible solution and apply the MIP-solver. When no new integer feasible solution is found in either the search or improvement neighbourhoods, we construct a new integer program which excludes all feasible solutions worse than the current best feasible solution. We then reformulate this as a new version of P, transform P to P^+, reoptimize the resulting $\mathcal{L}(P^+)$, and return to the search phase of PGC. This process continues until the time limit elapses.

Both PGC and NPGC are designed to be easily implemented in conjunction with a linear programming solver. We implemented them on top of the open source Gnu Linear Programming Kit (GLPK) [18]. Additionally, in order to compare NPGC with heuristics which use commercial integer programming solvers as subroutines, we designed NPGC to use such a black box in its neighbourhood search phase; our implementation uses the ILOG Cplex MIP-solver.

We tested PGC against a comparable implementation of PC,[1] the heuristic on which PGC and many other heuristics [4, 17, 5] are based. In our experiment PGC outperformed PC with respect to search success, solution quality, and time.

[1] Although originally designed to solve 0-1 integer programs, the pivoting rules generalize naturally to handle 0-1 mixed integer programs, as described in [4]. We implemented such a generalized version of PC on top of GLPK.

We tested NPGC against the Cplex MIP-solver that it uses as a black box, and against an implementation of the LOCAL BRANCHING (LB) heuristic of Fischetti and Lodi [13]. In our experiment NPGC outperformed these implementations.

2 Previous Work

In 1980 Balas and Martin introduced PC, a 0-1 integer programming heuristic based on simplex pivoting and 0-1 complementation [3]. Since then PC and its generalized MIP heuristic PIVOT AND SHIFT (PS) [4] have become 0-1 (mixed) IP heuristic benchmarks. Løkketangen et al. [17, 16] report (on small multidimensional knapsack problems) improvements over PC but use more memory. Balas et al. [2] report a comparable alternative to PS, but require a sophisticated branch-and-cut framework. Nediak and Eckstein [20] did not compare with PC or PS; however, their heuristic found solutions for only 32 of the 48 0-1 mixed problems of MIPLib 3.0 (1998), whereas PGC and our implementation of PC find solutions for 45 and 37 of these problems.

Recently heuristics have been proposed [13, 5, 8] that aim to find the best possible feasible solution within an allotted time by defining a neighbourhood around a known feasible solution and applying an exact MIP solver to search the neighbourhood for a better feasible solution. We selected LB [13] as a representative of these heuristics to be tested against NPGC.[2]

In 2005 Fischetti et al. [12] introduced FEASIBILITY PUMP (FP), a 0-1 MIP heuristic that finds feasible solutions quickly. Part of their improved performance is due to the use of Cplex as a black box subroutine; the Cplex LP solver is faster than (but more difficult to integrate code with) the GLPK LP solver on which we implemented PGC.

Since the introduction of the MIPLib problem set by Bixby et al. in 1992 [6], MIPLib has become the problem set against which heuristics are most commonly compared. The test suite of 78 problems (shown in Table 1) used in this paper consists of all 0-1 mixed instances from the 2003 version of MIPLib composed by Martin et al. [19], all 0-1 mixed instances from the library maintained by DEIS-Operations Research Group [11], and all other 0-1 mixed instances used in [8].

3 Finding feasible solutions

In this section we discuss the details of PGC, our heuristic whose goal is to find any feasible solution to the given input integer program. Since PGC borrows pivoting rules from PC, and since we also implemented a version of PC to test against PGC, we begin by discussing the details of our implementation of PC.

3.1 Generalized Pivot and Complement

Although introduced originally as a 0-1 integer programming heuristic, the pivoting rules of the 1980 PIVOT AND COMPLEMENT heuristic of Balas and Martin [3] generalize naturally to the mixed integer programming setting, as mentioned in [4]. We now describe the 0-1 mixed integer version of PC that we implemented.

[2] In [8] the authors report comparable performance to LB. The implementation of [5] has not been compared to that of [13] or [8].

We assume that the input program P has the form shown below, where c, x, b, A have dimensions $n, n, m, m \times n$ respectively, $N = \{1, \ldots, n\}$ is the set of variable indices of P, I is the subset of N indexing the binary variables of P, and, for each index j for which x_j is a non-binary variable, l_j and u_j denote the respective lower and upper bounds for x_j. P is transformed into the equivalent

Name	Con	Var	Bin	Opt/Best*	Name	Con	Var	Bin	Opt/Best*
10teams	230	2025	1800	924	sp97ar	1761	14101	14101	662671913.92*
a1c1s1	3312	3648	192	11557.09*	swath	884	6805	6724	467.40*
aflow30a	479	842	421	1158	t1717	551	73885	73885	194487.0*
aflow40b	1442	2728	1364	1168	tr12-30	750	1080	360	130596
air04	823	8904	8904	56137	van	27331	12481	192	4.838*
air05	426	7195	7195	26374	vpm2	234	378	210	13.75
cap6000	2176	6000	6000	−2451377	a2c1s1	3312	3648	192	10889.14*
dano3mip	3202	13873	552	697.857*	b1c1s1	3904	3872	288	24544.25*
danoint	664	521	56	65.6667	b2c1s1	3904	3872	288	25740.15*
ds	656	67732	67732	349.96*	biella1	1203	7328	6110	3065084.57*
disctom	399	10000	10000	−5000	nsr8k	6284	38356	32040	2.0301e+07*
fast0507	507	63009	63009	174	rail507	509	63019	63009	174*
fiber	363	1298	1254	405935	rail2536c	2539	15293	15284	689*
fixnet6	478	878	378	3983	rail2586c	2589	13226	13215	953*
glass4	396	322	302	1571442428.5*	rail4284c	4287	21714	21705	1071*
harp2	112	2993	2993	−73899798	rail4872c	4875	24656	24645	1550*
liu	2178	1156	1089	1222*	sp97ic	1034	12497	12497	429562635.68*
markshare1	6	62	50	1	sp98ar	1436	15085	15085	529814784.70*
markshare2	7	74	60	1	sp98ic	826	10894	10894	449144758.40*
mas74	13	151	150	11801.2	bg512142	1307	792	240	191471.50*
mas76	12	151	150	40005.1	dg012142	6310	2080	640	3038580.54*
misc07	212	260	259	2810	dc1c	1649	10039	8380	1783151.75*
mkc	3411	5325	5323	−563.846	dc1l	1653	37297	35638	1830738.85*
mod011	4480	10958	96	−5.4558e+07	dolom1	1803	11612	9720	20560415.03*
modglob	291	422	98	2.07405e+07	siena1	2220	13741	11775	13360676.13*
momentum1	42680	5174	2249	455740.91*	trento1	1265	7687	6415	5233909.00*
net12	14021	14115	1603	214	CMS750_4	16381	11697	7196	252*
nsrand-ipx	735	6621	6620	51200	berlin_5_8_0	1532	1083	794	62*
nw04	36	87842	87842	16862	railway_8_1_0	2527	1796	1177	400*
opt1217	64	769	768	−16	usAbbrv.8.25_70	3291	2312	1681	122*
p2756	755	2756	2756	3124	blp-ic97	923	9845	9753	4087.10*
pk1	45	86	55	11	blp-ic98	717	13640	13550	4561.52*
pp08a	136	240	64	7350	blp-ar98	1128	16021	15806	6254.87*
pp08aCUTS	246	240	64	7350	blp-ir98	486	6097	6031	2342.31*
protfold	2112	1835	1835	−26*	ljb2	1482	771	681	0.507679*
qiu	1192	840	48	−132.873	ljb7	8133	4163	3920	0.133655*
rd-rplusc-21	125899	622	457	172807.89*	ljb9	9231	4721	4460	0.739*
set1ch	492	712	240	54537.8	ljb10	10742	5496	5196	0.512*
seymor	4945	1372	1372	423	ljb12	9596	4913	4633	0.399*

Table 1. The test bed of 78 0-1 MIP instances used in this paper to benchmark heuristics. Con: constraints. Var: variables. Bin: binary variables. Opt/Best*: best known obj. value as per [8, 12, 13], MIPLib discussion forum [19] and our experiments.

program P^+ by adding the m-dimensional vector y. The scalar elements of x and y are called *decision variables* and *slack variables* respectively. The non-binary decision variables and the slack variables are referred to as *continuous variables*. $\mathcal{L}(P^+)$ is the linear program obtained from P^+ by relaxing the integrality constraints on the binary decision variables. Thus we have

$$
\begin{aligned}
P:&\ \min\{\, c^T x \mid Ax \geq b,\ x_i \in \{0,1\}\ \forall i \in I,\ l_j \leq x_j \leq u_j\ \forall j \in N-I\} \\
P^+:&\ \min\{\, c^T x \mid Ax - y = b,\ x_i \in \{0,1\}\ \forall i \in I,\ l_j \leq x_j \leq u_j\ \forall j \in N-I,\ y \geq 0\,\} \\
\mathcal{L}(P^+):&\ \min\{\, c^T x \mid Ax - y = b,\ 0 \leq x_i \leq 1\ \forall i \in I,\ l_j \leq x_j \leq u_j\ \forall j \in N-I,\ y \geq 0\,\}\,.
\end{aligned}
$$

PC is based on tableau pivot operations of the BVRS algorithm [10] on $\mathcal{L}(P^+)$. The first step is to use the BVRS algorithm to look for an optimal solution to $\mathcal{L}(P^+)$. If no such solution is found, then PC reports that P has no feasible solution; otherwise, an $\mathcal{L}(P^+)$-optimal basic feasible solution $S = x^*$ is passed to the feasibility subroutine.

The PC feasibility subroutine has two phases, search and restart. In the search phase, the extent to which x^* is P-infeasible is decreased by repeated tableau pivoting, as follows. Each binary decision variable of x^* which is not in the basis will have its value equal to either its upper or lower bound and therefore be integral. Thus, the primary P-infeasibility measure is the number of basic binary decision variables of x^*, and the search phase applies the following pivot whenever possible.

> *PC Pivot Type 1*: The pivot that exchanges a basic binary decision variable with a non-basic continuous variable, maintains $\mathcal{L}(P^+)$-feasibility, and, among all such pivots, maximizes the objective functions.

Once no more such pivots are available, a secondary P-infeasibility measure defined by $I_P(x^*) = \sum_{j\in I} \min\{x_j^*, 1 - x_j^*\}$ is considered.

> *PC Pivot Type 2*: The first pivot found that exchanges a basic continuous variable with a nonbasic continuous variable, or a basic binary decision variable with a nonbasic binary decision variable, maintains $\mathcal{L}(P^+)$-feasibility, and reduces $I_P(x^*)$.

Pivoting continues until either a P-feasible solution is found or a dead end is reached, namely there are no available Type 1 or Type 2 pivots. When such a dead end is encountered, the binary decision variables of x^* are first rounded and then truncated. If either process yields a P-feasible solution, then PC terminates successfully. Otherwise, the restart phase begins.

In the restart phase, the dead end is perturbed by a third kind of pivot into an $\mathcal{L}(P^+)$-infeasible solution whose infeasibility is then repeatedly decreased. The measure of infeasibility used is $I_L(x^*) = \sum_{j\in I} \max\{0, -x_j^*, x_j^* - 1\} + \sum_{j\in N-I} \max\{0, l_j - x_j^*, x_j^* - u_j\} + \sum_{y_j \in y} \max\{0, -y_j^*\}$.

> *PC Pivot Type 3*: The pivot that exchanges a basic binary decision variable with a non-basic continuous variable and leaves the entering continuous variable positive, and, among all such pivots, minimizes $I_L(x^*)$.

At this point, x^* is a basic infeasible solution of $\mathcal{L}(P^+)$. The restart phase next attempts to bring $I_L(x^*)$ to zero by repeatedly complementing one or more nonbasic binary decision variables, namely by changing their values from one bound to the other. If an $\mathcal{L}(P^+)$-feasible solution is found, then it is passed back to the search phase; otherwise, PC aborts.

This completes the description of our implementation of PC. The original versions of PC [3, 4] also have a complementation-based improvement subroutine to improve a P-feasible solution found by the feasibility subroutine. We omit the discussion of this subroutine since in PGC our objective is only to find a feasible solution.

3.2 Pivot and Gomory Cut

Pivot and Gomory Cut (PGC) begins in the same manner as PC, up to the beginning of the feasibility subroutine, which also consists of a search phase and restart phase.

Once execution reaches the search phase, PGC finds a Gomory cut separating S from the P-polytope. The extent to which S is P-infeasible is then decreased by repeated tableau pivoting. During pivoting, whenever the current basic feasible solution S' of $\mathcal{L}(P^+)$ satisfies the Gomory cut, the cut is added to the formulation of $\mathcal{L}(P^+)$ to avoid cycling, and a new Gomory cut separating S' from the P-polytope is generated.

This generation and occasional addition of Gomory cuts to $\mathcal{L}(P^+)$ continues until the search phase either finds a P-feasible solution or reaches a dead end. In the latter case, the fractional binary variables are rounded and then truncated to see whether either process yields a P-feasible solution. If not, execution transfers to the restart phase together with the most recently unsatisfied Gomory cut.

The goal of the restart phase is to find a basic feasible solution of $\mathcal{L}(P^+)$ that satisfies the Gomory cut and stays close to it. The intuition here is that, at the end of the search phase, the current basic feasible solution might be close to a P-feasible solution which in turn might be close to the Gomory cut. To find such a basic feasible solution, the objective function is replaced with the temporary objective function $\max \alpha x$, where $\alpha x \geq \beta$ describes the cut. The BVRS algorithm continues until either the value of the temporary objective function is at least β or the nonbasic x_n chosen as the entering pivot variable has $\max \alpha x$ unbounded. In each case $\alpha x \geq \beta$ is added as a constraint to $\mathcal{L}(P^+)$, and so the slack variable corresponding to this constraint becomes a basic variable x_b. In the latter case the simplex tableau becomes primal infeasible, so a pivot between x_n and x_b is performed to correct this. The original objective function is then restored and execution transfers back to the search phase.

We now describe the Gomory cut used by PGC that separates x^*, the current basic feasible solution of $\mathcal{L}(P^+)$, from the P-polytope. Let B and NB be the respective indices of the basic and non-basic variables of x^* and let j be the index of the basic binary variable of x^* which is most infeasible, namely $\min\{x_j^*, 1 - x_j^*\} = \max_{k \in B \cap I}\{\min\{x_k^*, 1 - x_k^*\}\}$. Let the row corresponding to x_j in the current simplex tableau have form $x_j^* = x_j + \sum_{k \in NB} a_{jk} x_k$, let $f_k = a_{jk} - \lfloor a_{jk} \rfloor$, $f_j = x_j^* - \lfloor x_j^* \rfloor$, and let N_{01},N_* be the indices of the respective binary non-basic and continuous non-basic variables of x^*. Then the cut is

$$\sum_{k \in N_{01} \wedge f_k \leq f_j} f_k x_k + \sum_{k \in N_{01} \wedge f_k > f_j} \frac{f_j(1 - f_k)}{1 - f_j} x_k + \sum_{k \in N_* \wedge a_{jk} \leq 0} \frac{-f_j a_{jk}}{1 - f_j} x_k + \sum_{k \in N_* \wedge a_{jk} > 0} a_{jk} x_k \geq f_j$$

We implemented two variations of the PGC search phase with slightly different pivot selection criteria. The original variation, PGC_0, uses the two pivot rules of the PC search phase (i.e., PC Pivot Type 1 and PC Pivot Type 2). The second variation, PGC_1, alters these two rules and redefines pivot selection based on the distance between extreme points and the Gomory cut.

In the rules given below, $\alpha x \geq \beta$ is the current cut, x_{cur}^* is the current basic feasible solution, and x_{next}^* is the basic feasible solution resulting from a pivot under consideration.

The PGC_1 pivot selection strategy is based on the fact that a P-feasible solution is somewhere across the current Gomory cut. Thus the goal of the first kind of pivot is to cross the cut while decreasing the primary P-infeasibility measure.

PGC$_1$ *Pivot Type 1*: The first pivot found that exchanges a basic binary decision variable with a non-basic continuous variable, maintains $\mathcal{L}(P^+)$-feasibility, and goes some distance towards the cut (namely, so that $\beta - \alpha x^*_{next} < \beta - \alpha x^*_{cur}$).

Algorithm PGC$_0$/PGC$_1$
INPUT: a 0-1 mixed integer problem P and a time limit T.
OUTPUT: A P-feasible solution S (null in case of failure).

$S \leftarrow$ null, elapsedTime $\leftarrow 0$
construct $\mathcal{L}(P^+)$ from P
find optimal solution S of $\mathcal{L}(P^+)$ using BVRS algorithm
if (S is P-feasible) then return S
repeat
BEGIN SEARCH PHASE
    construct a Gomory Cut $\alpha x \geq \beta$
    atDeadEnd $\leftarrow$ false
    repeat
        while (a *Pivot Type 1* of PGC$_0$/PGC$_1$ exists)
            perform *Pivot Type 1*, $S \leftarrow$ resulting $\mathcal{L}(P^+)$-feasible solution
            if (S is P-feasible) then return S
            if (S satisfies $\alpha x \geq \beta$) then
                add $\alpha x \geq \beta$ in $\mathcal{L}(P^+)$, construct a resulting new Gomory Cut $\alpha x \geq \beta$
        if (a *Pivot Type 2* of PGC$_0$/PGC$_1$ exists) then
            perform the pivot of *Pivot Type 2*, $S \leftarrow$ resulting $\mathcal{L}(P^+)$-feasible solution
            if (S is P-feasible) then return S
            if (S satisfies $\alpha x \geq \beta$) then
                add $\alpha x \geq \beta$ in $\mathcal{L}(P^+)$, construct a resulting new Gomory Cut $\alpha x \geq \beta$
        elsif atDeadEnd $\leftarrow$ true
    until atDeadEnd
    END SEARCH PHASE
    if rounding or truncating binary variables gives P-feasible S then return S
    BEGIN RESTART PHASE
    replace the objective function "min cx" of $\mathcal{L}(P^+)$ with "max αx"
    apply BVRS algorithm until the objective value becomes at least β
    re-establish the objective function to "min cx" and include $\alpha x \geq \beta$ in $\mathcal{L}(P^+)$
    END RESTART PHASE
until (elapsedTime$\geq T$)
return S

Fig. 1. Algorithm PGC$_0$/PGC$_1$

Once there are no more such pivots, the goal becomes to cross the cut without increasing the primary P-infeasibility measure.

PGC$_1$ *Pivot Type 2*: The pivot that exchanges a basic continuous variable with a nonbasic continuous variable, or a basic binary decision variable with a nonbasic binary decision variable, maintains $\mathcal{L}(P^+)$-feasibility, and, among the first (at most) $\lfloor \log n \rfloor$ such pivots, the one which either crosses the cut by the smallest amount or, if no pivot crosses the cut, the one closest to the cut (namely, if $\beta - \alpha x^*_{next} > 0.0$ for all the $\lfloor \log n \rfloor$ pivots, then select the pivot for which $\beta - \alpha x^*_{next}$ is minimum; otherwise, select the pivot for which $\beta - \alpha x^*_{next}$ is maximum over the choices for which $\beta - \alpha x^*_{next} \leq 0.0$).

In the selection of PGC_1 Pivot Type 2, we consider a bounded number of pivots because the first encountered might not lead far towards the cut, whereas considering all (possibly n) pivots might take too long.

4 Finding improved solutions

We tested NPGC against LOCAL BRANCHING of Fischetti and Lodi [13], arguably one of the strongest of the recent mixed integer program heuristics that aim to find the best possible feasible solution within an allotted time. As NPGC uses neighbourhood search in a similar manner to that of LB, we briefly discuss LB before describing NPGC.

4.1 Local Branching

LB explores a neighbourhood around a known feasible solution in the hopes of finding an improved feasible solution. The *improvement neighbourhood* [13] of a feasible solution $\overline{x}$ is defined by limiting at some integer p the number of variables currently at 0 or 1 that can switch bounds. This is achieved by adding to the input program the inequality $D(x,\overline{x}) \leq p$, where

$$D(x,\overline{x}) := \sum_{j\in V_0} x_j + \sum_{j\in V_1} (1-x_j) \ ,$$

with V_0 and V_1 being the index sets of the binary variables that are at 0 and 1 respectively in $\overline{x}$.

This inequality splits the original P-polytope into the improvement neighbourhood defined by $D(x,\overline{x}) \leq p$ and the remainder polytope given by $D(x,\overline{x}) \geq p+1$. Using the black box MIP-solver, the improvement neighbourhood is explored first with a specified time limit (the 'node time limit'). Based on whether the neighbourhood was completely explored and whether an improved feasible solution was found, the search continues.

4.2 Neighbourhood Pivot and Gomory Cut (NPGC)

In contrast to the LB neighbourhood search [13], which starts with a feasible solution and employs diversification in the improvement neighbourhood, NPGC (which defines both search and improvement neighbourhoods) achieves diversification in successive steps implicitly from the process of generating the search neighbourhood.

For many problems, the initial search phase of PGC runs out of time before finding a feasible solution or reaching a dead end. This motivates the addition of search neighbourhood subroutines to the PGC framework. NPGC applies the search phase of PGC, forcing it to terminate as soon as (i) a specified time limit t_{pivot} elapsed, or (ii) an integer feasible solution is obtained, or (iii) a dead end is encountered. If the current basic feasible solution is not integer-feasible, NPGC generates a Gomory cut (as in PGC, but now expressed only in terms of decision variables) separating the current basic feasible solution from the P-polytope. Otherwise, it separates the current basic feasible solution from the P-polytope with the so-called "vertex cut" [20],

$$\sum_{i\in V_0} x_i + \sum_{i\in V_1} (1-x_i) \geq 1$$

where V_0, V_1 denote the sets of 0-1 variables at 0,1 in the current basic feasible solution x^* of $\mathcal{L}(P^+)$. Since $x_i = 0$ for all $i \in V_0$ and $x_i = 1$ for all $i \in V_1$, this cut forces at least one binary decision variable to change value.

For a Gomory or vertex cut $\alpha x \geq \beta$ separating x^* from the P-polytope, define a search neighbourhood of x^* by adding to P the inequalities

$$\beta \leq \alpha x \leq \beta + d,$$

where d is defined so that the orthogonal distance between $\alpha x = \beta$ and $\alpha x = \beta + d$ is $\frac{\sqrt{n_c}}{k}$, where k is an execution parameter and n_c is the number of variables in the cut $\alpha x \geq \beta$. We choose $k > 1$ since, for 0-1 integer programs, setting $k \leq 1$ causes the search neighbourhood polytope to include the complete feasible region of the P-polytope.

NPGC applies a black-box MIP-solver on the search neighbourhood with a specified time limit t_{search}. When the solver terminates, it removes the inequalities $\beta \leq \alpha x \leq \beta + d$ from P and then, based on the termination statuses, modifies P. If the neighbourhood was completely explored (an optimal solution was found or the neighbourhood was integer infeasible), it adds the inequality $\alpha x \geq \beta + d$. If time expired during the search, it adds only the cut $\alpha x \geq \beta$. When the solver returns a P-feasible solution x^*_{new}, NPGC also puts an upper bound on the objective value by adding $c^T x \leq c^T x^*_{new}$.

If the solver terminates from a search neighbourhood with a P-feasible solution, NPGC generates an improvement neighbourhood by adding as in [13] the inequality

$$1 \leq \sum_{i \in V_0} x_i + \sum_{i \in V_1} (1 - x_i) \leq p,$$

where V_0, V_1, and p are as mentioned earlier. NPGC then applies the solver on the improvement neighbourhood with a time limit $t_{improve}$ and modifies P according to the termination statuses of the solver in the same way when terminating from the search neighbourhood (in this case we consider $\beta = 1$, αx to be the $\sum_{i \in V_0} x_i + \sum_{i \in V_1} (1 - x_i)$ and $d = p$).

As long as the solver terminates from improvement neighbourhood with a P-feasible solution, NPGC regenerates the improvement neighbourhood based on the new integer-feasible solution and applies the solver on that neighbourhood.

When the solver terminates from a search or improvement neighbourhood without generating a P-feasible solution, NPGC transforms P to P^+, re-optimizes the resulting $\mathcal{L}(P^+)$, and restarts execution at the initial PGC search phase.

If t_{search} is set to ∞, the search neighbourhoods will be mutually disjoint and, since $n_c \geq 1$, will contain at least a portion of the P-polytope for a certain value of k. It can be shown that this implies that, if P is bounded, NPGC will eventually explore all of P-polytope, and so functions as an exact solver in this case.

Algorithm NPGC

INPUT: a 0-1 mixed integer problem P, a search time limit t_{search}, an improvement time limit $t_{improve}$, a pivoting time limit t_{pivot}, a global time limit T and the parameters k_{start} and p.

OUTPUT: A P-feasible solution S_{MIP} (null in case of failure).

```
S_MIP ← null, elapsedTime ← 0, k ← k_start
repeat
    construct L(P^+) from P
    find optimal solution S_LP of L(P^+) using BVRS algorithm
    isFeas ← (S_LP is P-feasible)
    if (not isFeas) then
        construct a Gomory Cut αx ≥ β
        atDeadEnd ← false, elapsedPivotTime ← 0
        repeat
            while (a Pivot Type 1 of PGC_1 exists)
                perform Pivot Type 1, S_LP ← resulting L(P^+)-feasible solution
                isFeas ← (S_LP is P-feasible)
                if (not isFeas and S_LP satisfies αx ≥ β) then
                    add αx ≥ β in L(P^+), construct a resulting new Gomory Cut αx ≥ β
            if (not isFeas and a Pivot Type 2 of PGC_1 exists) then
                perform the pivot of Pivot Type 2, S_LP ← resulting L(P^+)-feasible solution
                isFeas ← (S_LP is P-feasible)
                if (not isFeas and S_LP satisfies αx ≥ β) then
                    add αx ≥ β in L(P^+), construct a resulting new Gomory Cut αx ≥ β
            else atDeadEnd ← true
        until (atDeadEnd or isFeas or (elapsedPivotTime ≥ t_pivot))
    if (not isFeas) then construct a Gomory cut αx ≥ β based on the current L(P^+)
    else S_MIP ← S_LP, construct the Vertex cut αx ≥ β
    determine the parameter d using k
    construct a search neighbourhood by adding αx ≥ β and αx ≤ β + d to P
    apply the black box MIP-solver on the search neighbourhood with time limit t_search
    modify P and k (as described in §5.3) based on MIP-solver termination statuses
    if (a feasible solution found by the MIP-solver) then
        repeat
            update S_MIP
            construct improvement neighbourhood using p
            apply the black box MIP-solver on improvement neighbourhood with time limit t_improve
            modify P according to MIP-solver termination statuses
        until (no update to S_MIP)
until (elapsedTime ≥ T)
return S_MIP
```

Fig. 2. Algorithm NPGC

5 Experimental Data and Analysis

5.1 Experimental tools

We implemented (generalized) PC, PGC_0, PGC_1, NPGC, and LB, in the C programming language. All but LB were implemented on top of GLPK 4.7. All experiments were run on an 2403 MHz AMD Athlon processor with 128 MByte of memory under Redhat Linux 9.0.

We used ILOG Cplex 8.1 as the MIP-solver subroutine in NPGC and LB. This Cplex version allows three settings for weighting the importance of feasible versus (feasible and) optimal solutions: optimal, feasible, and a default balanced setting (Cplex-D). Preliminary testing showed Cplex-D to be the most effective setting in both NPGC and LB, so that is the setting used in experiments reported in this paper.

5.2 Performance of PGC

Using the 78 0-1 MIP instance test bed shown in Table 1, we compared the two variants PGC_0 and PGC_1 of our "finds a feasible solution"-solver PGC against PC. These results are shown in Tables 2 and 3.

For each instance, we allotted a time limit of one CPU-hour for each solution attempt. For this time limit, PC, PGC_0 and PGC_1 failed in 43, 18, and 13 instances respectively. In the 35 instances in which PC succeeded, PGC_0 and PGC_1 found an equal or better solution in 29 and 19 instances and took equal or less time in 32 and 25 instances. Thus, in this experiment PGC_0 and PGC_1 outperformed PC with respect to search success, solution quality, and time.

Also shown by this experiment is the advantage of the restart design of PGC, which is designed to guarantee restart success; on the other hand, the PC restart phase failed in 31 of the 78 instances.

Notice that PGC_1 slightly outperformed PGC_0, finding solutions for 65 versus 60 instances, and finding better solutions in 32 versus 27 instances, taking less time in 34 versus 24 instances. This suggests that the pivot rules of PGC_1, which selects a pivot making progress towards the Gomory cut, are more effective than the PC-pivot rules of PGC_0.

Our experiment revealed weaknesses in the PGC search phase. It often reached a dead end without having improved feasibility (in instances 10teams, ds, net12, protfold, t1717, swath) and often continued pivoting without reaching a dead end (in dano3mip, momentum1, van, nsr8k, rail4284c, rail4872c and siena1). In the latter respect, PGC_1 was somewhat stronger than PC and PGC_0, occasionally reaching a dead end leading to a feasible solution via rounding or truncation when PC and PGC_0 failed to reach any dead end (in cms750-4, ljb7, ljb9, ljb10 and ljb12). Possible remedies (which we incorporated into NPGC) of these weaknesses include termination of the search phase at the first occurrence of either reaching a dead end or elapsing a time limit.

As far as we are aware, the only heuristic which outperforms PGC is the Feasibility Pump introduced very recently by Fischetti, Glover and Lodi. Part of the improved performance of this heuristic is it uses Cplex as its solver, whereas PGC uses GLPK. As shown in Table 4, the Cplex LP solver is faster than (but more difficult to integrate code with) the GLPK LP solver.

Name	PC			PGC$_0$			PGC$_1$		
	Solution	Time	R	Solution	Time	R	Solution	Time	R
10teams	$-$	$+$	1	$-$	$+$	514	$-$	$+$	256
a1c1s1	$-$	$+$	1	19303.32	127.4	7	17560.70	277.2	15
aflow30a	$-$	442.1	1	1790	1.2	2	2534	0.3	2
aflow40b	$-$	$+$	1	2673	15.7	8	3226	7.1	2
air04	$-$	$+$	1	58376	97.3	2	57528	96.1	5
air05	$-$	$+$	1	30291	35.9	4	31020	18.4	2
cap6000	-2395777	4.3	0	-2395777	4.3	0	-2442801	0.4	0
dano3mip	$-$	$+$	0	$-$	$+$	0	$-$	$+$	0
danoint	$-$	18.5	1	66.5	12.2	1	66.5	190.5	1
ds	$-$	$+$	1	$-$	$+$	49	1907.48	3585.3	41
disctom	$-$	$+$	1	-5000	179.3	19	-5000	678.2	39
fast0507	234	95.1	0	232	95.1	0	211	142.7	1
fiber	1326556.84	1.3	0	1326556.84	1.3	0	2269195.25	0.3	0
fixnet6	12169	0.9	0	12146	0.9	0	15810	1.3	10
glass4	$-$	119.7	1	12000171916.66	0.7	17	3000029600.00	3.1	11
harp2	$-$	$+$	1	-67089096	0.9	14	-62765126	0.7	1
liu	2516	21.0	0	2566	18.9	0	6022	0.6	0
markshare1	368	0.0	0	368	0.0	0	413	0.0	0
markshare2	494	0.0	0	494	0.0	0	166	0.0	0
mas74	19197.46	0.0	0	19197.46	0.0	0	25211.22	0.0	0
mas76	44839.82	0.0	0	44839.82	0.0	0	48515.28	0.0	0
misc07	$-$	128.9	1	4365	2.9	104	4315	1.8	60
mkc	94.85	18.7	0	29.54	18.7	0	54.15	5.3	1
mod011	-42783986.66	74.3	0	-42902314.08	79.5	0	-42546420.05	28.6	0
modglob	20784597.91	0.1	0	20784597.91	0.1	0	27542712.43	0.1	0
momentum1	$-$	$+$	0	$-$	$+$	0	$-$	$+$	1
net12	$-$	$+$	1	$-$	$+$	1	$-$	$+$	99
nsrand-ipx	246400	0.7	0	246400	0.7	0	241440	3.6	3
nw04	19882	5.2	0	19882	5.7	0	17652	5.1	0
opt1217	-16	0.0	0	-16	0.0	0	-16	0.0	0
p2756	$-$	$+$	1	45983	3.5	24	38618	2.8	0
pk1	51	0.0	0	51	0.0	0	37	0.0	0

Table 2. Experimental results of PC, PGC$_0$ and PGC$_1$ on the first 32 of the 78 test bed instances. $+$: time limit exceeded (1 CPU-hour). $-$: heuristic reported failure. Sol'n: obj. value of found sol'n. Time: seconds to find sol'n or report failure. R: no. of times restarted.

Name	PC			PGC_0			PGC_1		
	Solution	Time	Restart	Solution	Time	Restart	Solution	Time	Restart
pp08a	9240	0.1	0	9200	0.1	0	10130	0.1	2
pp08aCUTS	8930	0.3	0	9070	0.2	0	10560	0.1	0
protfold	–	+	1	–	+	361	–	+	88
qiu	–	8.9	1	891.51	151.4	59	706.34	119.1	27
rd-rplusc-21	–	+	1	–	+	22	–	+	17
set1ch	69226	2.5	0	83967	2.6	0	108616.25	3.1	2
seymor	579	130.4	0	576	123.1	0	499	77.8	14
sp97ar	911456446.18	16.9	0	887138830.9	16.8	0	848063756.64	23.2	1
swath	–	+	1	1180.69	310.8	125	–	+	224
t1717	–	+	1	–	+	11	–	+	5
tr12-30	–	838.1	1	141531	10.6	3	211939	16.1	4
van	–	280.7	1	–	+	1	–	+	1
vpm2	21	0.1	0	21	0.1	1	23.5	0.2	4
a2c1s1	–	+	1	19418.19	83.6	1	17852.57	132.1	20
b1c1s1	–	+	1	57065.05	246.2	5	77286.35	418.9	23
b2c1s1	54955.67	448.9	0	54955.67	448.9	0	74917.38	446.5	22
biella1	–	+	1	74718802.4	187.2	2	54743498.12	113.8	12
nsr8k	–	+	0	–	+	0	–	+	0
rail 507	248	103.7	0	230	93.4	0	230	125.0	1
rail2536c	995	412.5	0	1005	402.3	0	867	179.5	1
rail2586c	1551	1118.7	0	1479	1087.5	0	1245	662.7	8
rail4284c	–	+	0	–	+	0	–	+	0
rail4872c	–	+	0	–	+	0	–	+	0
sp97ic	576581599.04	3.9	0	576581599.04	3.9	0	600124906.72	5.2	0
sp98ar	828039478.24	13.1	0	793574259.2	14.1	0	748227449.92	18.5	1
sp98ic	624212225.92	4.0	0	611732774.08	3.7	0	607993402.88	5.0	1
bg512142	11463704.5	26.2	0	13904130.5	24.0	0	9836438.0	18.5	0
dg012142	21837419.2	533.9	0	45417612.0	407.7	0	55482373.0	478.9	0
dc1c	–	+	1	134568591.1	453.7	2	100961709.52	670.0	26
dc1l	–	+	1	146923054.01	1687.1	6	87943999.57	2728.2	13
dolom1	–	+	1	1853291861.1	921.7	4	1.000e+10	847.26	14
siena1	–	+	0	–	+	0	–	+	0
trento1	–	+	1	2231236508.0	328.8	2	497671343.00	250.3	12
cms750-4	–	+	0	–	+	0	1000	239.9	0
berlin-5-8-0	100	20.4	0	100	20.4	0	100	3.6	0
railway-8-1-0	500	81.7	0	500	81.7	0	500	4.6	0
usabbrv-8-25-70	200	257.4	0	200	257.4	0	200	8.9	0
blp-ic97	–	+	1	5846.22	11.6	9	5993.52	29.7	19
blp-ic98	–	+	1	9934.38	17.0	11	8440.41	12.6	4
blp-ar98	–	+	1	10698.27	53.9	15	11833.15	60.4	5
blp-ir98	–	+	1	3717.96	4.3	3	4337.81	4.7	5
ljb2	0.85	21.0	0	0.85	21.0	0	7.23	1.4	0
ljb7	–	+	0	–	+	0	8.61	68.9	0
ljb9	–	+	0	–	+	0	9.47	256.2	0
ljb10	–	+	0	–	+	0	7.31	223.9	0
ljb12	–	+	0	–	+	0	6.19	178.9	0

Table 3. Experimental results of PC, PGC_0 and PGC_1 on the remaining 46 test bed instances. See Table 2 for the remaining caption.

5.3 Performance of NPGC

Using the 56 instances of our 78 instance test bed for which Cplex cannot provide proof of optimality in one CPU-hour, we compared our "finds-best-possible-solution"-solver NPGC against LB and the three feasible/optimal emphasis settings of Cplex. In 55 of these instances, GLPK solves an LP relaxation in under 10 minutes; in the remaining instance 'nsr8k', this takes almost 50 minutes. For this reason, for all instances except nsr8k, we allotted each heuristic one CPU-hour; for nsr8k, we allotted two CPU-hours.

NPGC uses PGC_1 pivot rules. Important parameters in NPGC are t_{search}, which limits the time spent in a search neighbourhoods and $t_{improve}$, which limits spent in improvement neighbourhoods. Experimentation showed a useful $t_{search}/t_{improve}$ ratio to be three to one. In order be successful, the black box MIP-solver must solve at least one LP relaxation. Thus, the time taken by the MIP-solver to find the first LP relaxation solution is recorded and used to set t_{search}. After experimenting, we chose the following scheme: set t_{search} to 360, 540, 720, 1800 seconds if LP solving time was less than 10 seconds, between 10 and 50 seconds, between 50 and 200 seconds, at least 200 seconds respectively. We defined t_{pivot} to be fraction of t_{search} that allows enough pivoting to get close to some feasible solution; after experimentation, we set t_{pivot} by $t_{search}/12$. For p, we used the value p reported in LB [13].

Name	LP solution time (secs)		Name	LP solution time (secs)		Name	LP solution time (secs)	
	GLPK 4.7	Cplex 8.1		GLPK 4.7	Cplex 8.1		GLPK 4.7	Cplex 8.1
10teams	0.18	0.10	net12	4.69	6.35	rail2586c	116.91	48.29
a1c1s1	1.21	0.04	nsrand-ipx	0.29	0.40	rail4284c	496.63	153.31
aflow30a	0.03	0.01	nw04	3.17	1.22	rail4872c	527.94	189.99
aflow40b	0.29	0.11	opt1217	0.01	0.01	sp97ic	3.31	1.04
air04	14.42	2.47	p2756	0.01	0.00	sp98ar	7.98	1.59
air05	2.48	0.52	pk1	0.00	0.00	sp98ic	2.99	0.90
cap6000	0.43	0.37	pp08a	0.01	0.00	bg512142	0.23	0.10
dano3mip	55.14	52.97	pp08aCUTS	0.01	0.01	dg012142	1.63	0.51
danoint	0.15	0.06	protfold	3.08	0.83	dc1c	21.00	10.41
ds	83.97	46.29	qiu	0.26	0.07	dc1l	145.78	45.48
disctom	15.63	14.80	rd-rplusc-21	136.26	2.75	dolom1	39.88	18.40
fast0507	48.01	18.13	set1ch	0.01	0.01	siena1	129.09	46.78
fiber	0.02	0.00	seymor	5.21	2.43	trento1	32.18	11.34
fixnet6	0.02	0.00	sp97ar	8.70	1.54	cms750-4	57.16	0.52
glass4	0.00	0.00	swath	0.09	0.07	berlin-5-8-0	0.27	0.03
harp2	0.04	0.03	t1717	84.09	7.42	railway-8-1-0	0.93	0.04
liu	0.21	0.02	tr12-30	0.12	0.00	usabbrv-8-25-70	1.49	0.07
markshare1	0.00	0.00	van	120.00	19.62	blp-ic97	0.45	0.27
markshare2	0.00	0.00	vpm2	0.01	0.00	blp-ic98	0.66	0.45
mas74	0.00	0.00	a2c1s1	1.16	0.06	blp-ar98	1.41	0.31
mas76	0.00	0.00	b1c1s1	1.96	0.06	blp-ir98	0.18	0.11
misc07	0.02	0.00	b2c1s1	2.04	0.08	ljb2	0.21	0.03
mkc	0.24	0.16	biella1	8.54	4.97	ljb7	9.09	0.25
mod011	0.96	0.11	nsr8k	2903.14	1001.56	ljb9	12.80	0.31
modglob	0.01	0.01	rail507	46.78	20.90	ljb10	17.05	0.45
momentum1	49.43	3.40	rail2536c	101.57	16.43	ljb12	13.30	0.35

Table 4. Comparative LP solution time of the instances by GLPK 4.7 and Cplex 8.1.

In NPGC, the value of k ranges from $k_{start}/4$ to $4k_{start}$ according to a doubling flag and termination statuses from the search neighbourhood. Each time the solver finds a new solution, k is set to k_{start} and the flag is cleared. When the neighbourhood is integer infeasible, the flag is set, and k is halved (unless already at its minimum). If time expires with no feasible solution found and the flag is set then k is halved (unless at its minimum) and doubled (unless at its maximum) otherwise; then the flag is flipped if k is at an extremum. For k, after experimentation we set k_{start} to 10.

In the Fischetti and Lodi [13] implementation of LB, local branching was applied after the first feasible solution was found by Cplex. In our implementation of LB, we applied local branching from the best feasible solution obtained after running Cplex for t_{search} time. This variation makes our implementation of LB comparable with NPGC in that NPGC also explores the search neighbourhood for t_{search} time. Our LB implementation achieves better results than an implementation of the original LB (which starts from the first feasible solution) for all the instances except the instances for which Cplex-D cannot find a feasible solution in t_{search} time (namely net12 and rd-rplusc-21). For these two instances we ran LB from the first feasible solution obtained by Cplex-D. As in NPGC, the node time limit for LB is set to $t_{improve}$.

Name	PD/AD*			Name	PD/AD*		
	Cplex	LB	NPGC		Cplex	LB	NPGC
a1c1s1	1.475	**0.878**	1.725	rail507	0.574	0.574	**0.000**
aflow40b	5.907	**2.996**	5.136	rail2536c	**0.000**	**0.000**	**0.000**
dano3mip	1.576	1.937	**1.215**	rail2586c	1.259	1.154	**0.629**
danoint	**0.000**	**0.000**	**0.000**	rail4284c	0.933	0.373	**0.186**
ds	22.036	1.765	**0.000**	rail4872c	0.774	0.193	**0.064**
fast0507	0.574	**0.000**	0.574	sp97ic	1.473	**0.498**	2.494
glass4	3.939	1.818	**0.000**	sp98ar	0.986	**0.286**	0.741
harp2	0.010	**0.000**	**0.000**	sp98ic	0.901	**0.652**	1.370
liu	13.256	3.927	**0.000**	bg512142	6.967	0.280	**0.000**
markshare1	**3***	12*	7*	dg012142	1.640	8.453	**0.000**
markshare2	**12***	20*	15*	dc1c	0.648	0.008	**0.000**
mkc	14.003	**0.818**	0.857	dc1l	340.784	**0.000**	0.598
momentum1	–	–	–	dolom1 (poff)	179.721	**0.000**	21.493
net12	**0.000**	19.158	19.158	siena1 (poff)	154.334	**0.000**	89.851
nsrand-ipx	0.625	0.625	**0.312**	trento1	**0.000**	3.064	1.011
protfold	11.538	**0.000**	**0.000**	cms750-4	1.190	**0.000**	**0.000**
rd-rplusc-21 (poff)	**0.000**	4.618	–	berlin-5-8-0	**0.000**	**0.000**	**0.000**
seymor	0.709	**0.236**	0.472	railway-8-1-0	**0.000**	**0.000**	**0.000**
sp97ar	0.646	0.806	**0.627**	usabbrv-8-25-70	4.918	**0.000**	**0.000**
swath	1.007	**0.000**	15.832	blp-ic97	**0.000**	0.339	5.198
t1717	**0.000**	5.476	5.144	blp-ic98	**0.000**	0.763	0.603
tr12-30	**0.002**	0.013	0.003	blp-ar98	1.325	**0.000**	3.567
van (poff)	**0.000**	10.045	5.477	blp-ir98	**0.000**	0.038	0.375
a2c1s1	2.899	**1.224**	1.801	ljb2	11.476	**4.332**	16.158
b1c1s1	2.118	3.286	**1.818**	ljb7	115.951	35.421	**34.863**
b2c1s1	3.844	**1.676**	4.575	ljb9	**81.150**	93.680	89.377
biella1	0.255	0.333	**0.017**	ljb10	66.406	37.812	**19.199**
nsr8k (poff)	1525.880	0.019	**0.000**	ljb12	125.087	225.614	**110.000**

Table 5. Experimental results of Cplex, LB, and, NPGC. –: reported failure. Name: suffix '(poff)' added for instances using Cplex-D with no presolver; in these instances Cplex-D with presolver failed to find a solution in one CPU-hour. PD/AD*: PD = $|100 \times (b-s)/b|$, where b is obj. value of best known sol'n, s is obj. value of found sol'n; AD (used when $b = 1$) = $|b-s|$. LB: our implementation of LB. Cplex: obj. value of best sol'n found by three Cplex settings. **bold font** best method (among Cplex, LB, or NPGC) for this instance.

Comparing NPGC, LB, and the three settings of Cplex on the 56 test bed instances, the best solution after time expired was found by NPGC, LB, and one of the three Cplex variations respectively in 26 (strictly best in 19), 24 (strictly best in 17), and 16 (strictly best in 12) instances. Compared only to Cplex, NPGC found better solutions in 35 (strictly better in 30) instances; compared only to LB, NPGC found better solutions in 35 (strictly better in 27) instances. Thus, in this experiment, NPGC performed the best of the five methods.

Recall that the search neighbourhood in NPGC is defined either by a Gomory cut or a vertex cut. Over the 56 test bed instances, the latter case occurred only three times (in markshare1, markshare2 and bg512142, and in each instance only in the first iteration of the heuristic). This provides experimental evidence for the effectiveness of Gomory cuts in defining the search neighbourhood.

6 Conclusions

We have introduced PGC and NPGC, two competitive IP heuristics based on Gomory cuts, simplex pivoting, and neighbourhood search. In particular, PGC and NPGC demonstrate the effectiveness of using Gomory cuts in such heuristics. Although our implementations are restricted to 0-1 mixed programs, the ideas used generalize easily to general mixed programs.

Acknowledgements We thank Bill Cook for bringing various references to our attention.

References

1. E. Balas, S. Ceria, G. Cornuéjols, and N. Natraj. Gomory cuts revisited. *Operations Research Letters*, 19:1–9, 1996.
2. E. Balas, S. Ceria, M. Dawande, F. Margot, and G. Pataki. Octane: a new heuristic for pure 0-1 programs. *Operations Research*, 49(2):207–225, 2001.
3. E. Balas and C.H. Martin. Pivot and complement – a heuristic for 0-1 programming. *Management Science*, 26(1):86–96, 1980.
4. E. Balas and C.H. Martin. Pivot and shift – a heuristic for mixed integer programming. Technical report, GSIA, Carnegie Mellon University, 1986.
5. E. Balas, S. Schmieta, and C. Wallace. Pivot and shift – a mixed integer programming heuristic. *Discrete Optimization*, 1:3–12, 2004.
6. Robert E. Bixby, E.Ãndrew Boyd, and Ronni R. Indovina. MIPLIB: A test set of mixed integer programming problems. *SIAM News*, 25(20):16, March 1992.
7. G. Cornuéjols. Revival of the gomory cuts in the 1990's. to be appeared on State of the Art and Recenet Advances in Integer Programming.
8. E. Danna, E. Rothberg, and C.L. Pape. Exploring relaxation induced neighborhhods to improve mip solutions. *Mathematical Programming*, 102:71–90, 2005.
9. G.B̃. Dantzig. Maximization of a linear function of variables subject to linear inequalities. In Tj. C. Koopmans, editor, *Activity Analysis of Production and Allocation*, pages 339–347. Wiley, New York, 1951.
10. G.B̃. Dantzig. *Linear Programming and Extensions*. Princeton University Press, Princeton, 1963.
11. DEIS. Library of instances. http://www.or.deis.unibo.it/research_pages/ORinstances/MIPs.html.
12. M. Fischetti, F. Glover, and A. Lodi. The feasibilty pump. to be appeared on Mathematical Programming.
13. M. Fischetti and A. Lodi. Local branching. *Mathematical Programming B*, 98:23–49, 2003.
14. R.E. Gomory. An algorithm for the mixed integer problem. Technical Report RM-2597, The Rand Corporation, Santa Monica, CA, 1960.
15. R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, New York, 1972.
16. A. Løkketangen and F. Glover. Solving zero/one mixed integer programming problems using tabu search. *European Journal of Operational Research*, 106:624–658, 1998.
17. A. Løkketangen, K. Jörnsten, and S. Storøy. Tabu search within a pivot and complement framework. *International Transactions in Operational Research*, 1(3):305–317, 1994.

18. Andrew Makhorin. Gnu linear programming kit, 2000-2003. http://www.gnu.org/software/glpk/.
19. A. Martin, T. Achterberg, and T. Koch. Miplib 2003. http://miplib.zib.de.
20. M. Nediak and J. Eckstein. Pivot, cut, and dive: A heuristic for mixed 0-1 integer programming. *RUTCOR Research Report*, RRR 53-2001, 2001.
21. S. Sahni. Computationally related problems. *SIAM J. Comput.*, 3:262–269, 1974.