RC23532 (W0502-107) February 17, 2005
Mathematics

IBM Research Report

Repairing MIP Infeasibility through Local Branching

Matteo Fischetti
DEI
University of Padova
Via Gradenigo 6A
35131 Padova
Italy

Andrea Lodi
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598
and
DEIS
University of Bologna
Viale Risorgimento 2
40136 Bologna
Italy

Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

Repairing MIP infeasibility through Local Branching

Matteo Fischetti[*], Andrea Lodi[°,†]

[*] DEI, University of Padova, Via Gradenigo 6A - 35131 Padova - Italy

[°]T.J. Watson Research Center, IBM, Yorktown Heights, NY 10598, USA

[†] DEIS, University of Bologna, Viale Risorgimento 2 - 40136 Bologna - Italy

e-mail: {matteo.fischetti@unipd.it, alodi@us.ibm.com}

February 16, 2004

Abstract

Finding a feasible solution to a generic Mixed-Integer Program (MIP) is often a very difficult task. Recently, two heuristic approaches called *Local Branching* and *Feasibility Pump* have been proposed to address this problem. In this paper we introduce and analyze computationally a hybrid method that uses the feasibility pump method to provide, at very low computational cost, an initial (possibly infeasible) solution to the local branching procedure.

1 Introduction

In this paper we consider the problem of finding a feasible solution to a generic Mixed-Integer Program (MIP) with 0-1 variables of the form:

$$
\begin{aligned}
(P) \qquad & \min c^T x && && (1)\\
& Ax \geq b && && (2)\\
& x_j \in \{0,1\} && \forall j \in \mathcal{B} \neq \emptyset && (3)\\
& x_j \geq 0, \text{ integer} && \forall j \in \mathcal{G} && (4)\\
& x_j \geq 0 && \forall j \in \mathcal{C} && (5)
\end{aligned}
$$

Here the variable index set $\mathcal{N} := \{1, \ldots, n\}$ is partitioned into $(\mathcal{B}, \mathcal{G}, \mathcal{C})$, where $\mathcal{B} \neq \emptyset$ is the index set of the 0-1 variables, while the possibly empty sets $\mathcal{G}$ and $\mathcal{C}$ index the general integer and the continuous variables, respectively. Let $\mathcal{I} := \mathcal{B} \cup \mathcal{G}$ denote the index set of all integer-constrained variables.

Heuristics for general-purpose MIPs include [2], [3], [10], [11], [12], [14], [16], [17], [18], [21], [5], [4], [9], and [13] among others. Recently, we proposed in [7] a heuristic approach, called *Local Branching* (`LB`), to improve the quality of a given feasible solution. This method, as well as other refining heuristics such as the recently-proposed RINS approach [5], requires however the availability of a starting feasible solution, which is an issue for several very hard MIPs. This topic was investigated by Fischetti, Glover and Lodi [8], who introduced the so-called *Feasibility Pump* (`FP`) scheme for finding a feasible (or, at least, an "almost feasible") solution to general MIPs through a clever sequence of roundings.

In the present paper we analyze computationally a simple variant of the original `LB` method that allows one to deal with infeasible reference solutions, such as those returned by the `FP` method. Our approach is to start with an "almost feasible" reference solution $\bar{x}$, as available at small computational cost through the `FP` method. We then relax the MIP model by introducing for each violated constraint: (i) an artificial continuous variable in the constraint itself, (ii) a binary (also artificial) variable, and (iii) a constraint stating that, if the artificial variable has to be used to make the constraint satisfied, then the binary variable must be set to 1. Finally, the objective function is replaced, in the spirit of the first phase of the primal simplex algorithm, by the sum of the artificial binary variables. The initial solution turns out now to be feasible for the relaxed model and its value coincides with the number of initial violated constraints. We then apply the standard `LB` framework to reduce the value of the objective function, i.e., the number of infeasibilities and a solution of value 0 turns out to be feasible for the initial problem.

The paper is organized as follows. In Section 2 we review the `LB` and `FP` methods. In Section 3 we describe the `LB` extension we propose to deal with infeasible reference solutions. Computational results are presented in Section 4, where we compare the `LB` performance with that of the commercial software `ILOG-Cplex` on two sets of hard 0-1 MIPs, specifically 44 problems taken from MIPLIB 2003 library [1] and 39 additional instances already considered in [8].

2 Local Branching and Feasibility Pump

We next review the `LB` and `FP` methods; the reader is referred to [7] and [8] for fuller details.

Local Branching

The Local Branching approach works as follows. Suppose a feasible *reference solution* $\bar{x}$ of (P) is given, and one aims at finding an improved solution that is "not too far" from $\bar{x}$. Let $\overline{S} := \{j \in \mathcal{B} : \bar{x}_j = 1\}$ denote the binary support of $\bar{x}$. For a given positive integer parameter k, we define the *k-OPT neighborhood* $\mathcal{N}(\bar{x}, k)$ of $\bar{x}$ as the set of the feasible solutions of (P) satisfying the additional *local branching constraint*:

$$\Delta(x, \bar{x}) := \sum_{j \in \overline{S}} (1 - x_j) + \sum_{j \in \mathcal{B} \setminus \overline{S}} x_j \leq k \tag{6}$$

where the two terms in the left-hand side count the number of binary variables flipping their value (with respect to $\bar{x}$) either from 1 to 0 or from 0 to 1, respectively. As its name suggests, the local branching constraint (6) can be used as a branching criterion within an enumerative scheme for (P). Indeed, given the incumbent solution $\bar{x}$, the solution space associated with the current branching node can be partitioned by means of the disjunction

$$\Delta(x, \bar{x}) \leq k \quad \text{(left branch)} \quad \textbf{or} \quad \Delta(x, \bar{x}) \geq k + 1 \quad \text{(right branch)} \tag{7}$$

where the neighborhood-size parameter k is chosen so as make neighborhood $\mathcal{N}(\bar{x}, k)$ "sufficiently small" to be optimized within short computing time, but still "large enough" to likely contain better solutions than $\bar{x}$ (typically, $k = 10$ or $k = 20$).

In [7] we investigated the use of a general-purpose MIP solver as a black-box "tactical" tool to explore effectively suitable solution subspaces defined and controlled at a "strategic" level by a simple external branching framework. The procedure is in the spirit of well-known local search metaheuristics, but the neighborhoods are obtained through the introduction in the MIP model of the local branching constraints (6). This allows one to work within a perfectly general MIP framework, and to take advantage of the impressive research and implementation effort that nowadays is devoted to the design of MIP solvers. The new solution strategy is exact in nature, though it is designed to improve the heuristic behavior of the MIP solver at hand. It alternates high-level strategic branchings to define solution neighborhoods, and low-level tactical branchings (performed within the MIP solver) to explore them. The result can then be viewed as a two-level branching strategy aimed at favoring early updatings of the incumbent solution, hence producing improved solutions at early stages of the computation. The computational results reported in [7] show the effectiveness of the `LB` approach, and are confirmed by the recent works of Hansen, Mladenovíc and Urosevíc

[13] (where LB is used within a Variable Neighborhood Search metaheuristic [20]) and of Fischetti, Polo and Scantamburlo (where MIPs with a special structure are investigated).

Feasibility Pump

Let $P_L := \{x : Ax \geq b\}$ denote the polyhedron associated with the LP relaxation of the given MIP, and assume without loss of generality that system $Ax \geq b$ includes the variable bounds

$$l_j \leq x_j \leq u_j \quad \forall j \in \mathcal{I}$$

where $l_j = 0$ and $u_j = 1$ for all $j \in \mathcal{B}$. With a little abuse of notation, we say that a point x is *integer* if x_j is integer for all $j \in \mathcal{I}$ (no matter the value of the other components). Analogously, the rounding $\tilde{x}$ of a given x is obtained by setting $\tilde{x}_j := [x_j]$ if $j \in \mathcal{I}$ and $\tilde{x}_j := x_j$ otherwise, where $[\cdot]$ represents scalar rounding to the nearest integer. The (L_1-norm) distance between a generic point $x \in P_L$ and a given integer $\tilde{x}$ is defined as

$$\Phi(x, \tilde{x}) = \sum_{j \in \mathcal{I}} |x_j - \tilde{x}_j|$$

Notice that $\tilde{x}$ is assumed to be integer; moreover, the continuous variables x_j with $j \notin \mathcal{I}$, if any, do not contribute to the distance function. For any given integer $\tilde{x}$, the distance function can be written as

$$\Phi(x, \tilde{x}) := \sum_{j \in \mathcal{I}: \tilde{x}_j = l_j} (x_j - l_j) + \sum_{j \in \mathcal{I}: \tilde{x}_j = u_j} (u_j - x_j) + \sum_{j \in \mathcal{I}: l_j < \tilde{x}_j < u_j} (x_j^+ + x_j^-)$$

where the additional variables x_j^+ and x_j^- require the introduction into the MIP model of the additional constraints:

$$x_j = \tilde{x}_j + x_j^+ - x_j^-, \quad x_j^+ \geq 0, \ x_j^- \geq 0, \quad \forall j \in \mathcal{I} : l_j < \tilde{x}_j < u_j \tag{8}$$

It then follows that the closest point $x^* \in P_L$ to $\tilde{x}$ can easily be determined by solving the LP

$$\min\{\Phi(x, \tilde{x}) : Ax \geq b\} \tag{9}$$

If $\Phi(x^*, \tilde{x}) = 0$, then $x_j^* (= \tilde{x}_j)$ is integer for all $j \in \mathcal{I}$, so x^* is a feasible MIP solution. Conversely, given a point $x^* \in P_L$, the integer point $\tilde{x}$ closest to x^* is easily determined by just rounding x^*.

The FP heuristic works with a pair of points $(x^*, \tilde{x})$ with $x^* \in P_L$ and $\tilde{x}$ integer, that are iteratively updated with the aim of reducing as much as possible their distance $\Phi(x^*, \tilde{x})$. To be more specific, one starts with any $x^* \in P_L$, and initializes a (typically infeasible) integer $\tilde{x}$ as the rounding of x^*. At each FP iteration, called *pumping cycle*, $\tilde{x}$ is fixed and one finds through linear programming the point $x^* \in P_L$ which is as close as possible to $\tilde{x}$. If $\Phi(x^*, \tilde{x}) = 0$, then x^* is a MIP feasible solution, and the heuristic stops. Otherwise, $\tilde{x}$ is replaced by the rounding of x^* so as to further reduce $\Phi(x^*, \tilde{x})$, and the process is iterated.

The basic FP scheme above tends to stop prematurely due to stalling issues. This happens whenever $\Phi(x^*, \tilde{x}) > 0$ is not reduced when replacing $\tilde{x}$ by the rounding of x^*, meaning that all the integer-constrained components of $\tilde{x}$ would stay unchanged in this iteration. In the original FP approach [8], this situation is dealt with by choosing heuristically a few components $\tilde{x}_j$ to modified, even if this operation increases the current value of $\Phi(x^*, \tilde{x})$. A different approach, to be elaborated in the next section, is to switch to a different method based on enumeration, in the attempt of exploring a small neighborhood of the current "almost feasible" $\tilde{x}$ (that typically has a very small distance $\Phi(x^*, \tilde{x})$ from P_L).

3 LB with infeasible reference solutions

The basic idea of the method presented in this section is that the LB algorithm does not necessarily need to start with a feasible solution—a partially feasible one can be a valid warm start for the method. Indeed, by relaxing the model in a suitable way it is always possible to consider any infeasible solution, say $\hat{x}$, to be "feasible", and penalize its cost so as the LB heuristic can hopefully drive it to feasibility.

The most natural way to implement this idea is to add a continuous artificial variable for each constraint violated by $\hat{x}$, and then penalize the use of such variables in the objective function by means of a very large cost M. We tested this approach and found it performs reasonably well on most of the problems. However, it has the main drawback that finding a proper value for M may be not easy in practice. Indeed, for a relevant set of problems in the MIPLIB 2003 [1] collection, the value of the objective function is so large that it is really hard to define a value for M that makes any infeasible solution worse than any feasible one. Moreover, the way the the LB method works suggests the use of the following a more combinatorial framework.

Let T be the set of the indices of the constraints $a_i^T x \geq b_i$ that are violated by $\hat{x}$. For each $i \in T$, we relax the original constraint $a_i^T x \geq b_i$ into $a_i^T x + \delta_i y_i \geq b_i$, where $\delta_i := b_i - a_i^T \hat{x}$ is the positive amount of violation computed with respect to $\hat{x}$, and y_i is a binary artificial variable attaining value 1 for each constraint violated by $\hat{x}$. Finally, we replace the original objective function $c^T x$ by $\sum_{i \in T} y_i$, so as to count the number of violated constraints. It has to be noted that the set of binary variables in the relaxed model is $\mathcal{B} \cup \mathcal{Y}$, where $\mathcal{Y} := \{y_i : i \in T\}$, hence the structure of the relaxation turns out to be particularly suited for the LB approach, where the local branching constraint affects precisely the binary variables (including the artificial ones).

An obvious drawback of the method above is that the original objective function is completely disregarded, thus the feasible solution obtained can be arbitrarily bad. A way of avoiding this situation could be to put a term in the artificial objective function that takes the original costs into account. However, a proper balancing of the two contributions (original cost and infeasibility penalty) may be not easy to find. As a matter of fact, the outcome of a preliminary computational study is that a better overall performance is obtained by using the artificial objective function (alone) until feasibility is reached, and then improving the quality of this solution by using a standard LB or RINS approach.

4 Computational Results

In this section we report computational results on the performance of the proposed method by comparing it with both the FP heuristic and the commercial software ILOG-Cplex 9.0.3.

In our experiments, we used the "asymmetric" version of the local branching constraint (6), namely

$$\Delta(x, \bar{x}) := \sum_{j \in \overline{S}} (1 - x_j) \tag{10}$$

Indeed, as discussed in [7], this version of the constraint seems to be particularly suited for set covering problems where LB aims at finding solutions with a small binary support—which is precisely the case of interest in our context.

Our testbed is made by 44 0-1 MIP instances collected in MIPLIB[1] 2003 [1] and described in Table 1, plus an additional set of 39 hard 0-1 MIPs described in Table 2 and available, on request, from the second author. The two tables report the instance names and the corresponding number of variables (n), of 0-1 variables ($|\mathcal{I}|$) and of constraints (m).

The framework described in the previous section has been tested by using different starting solutions $\hat{x}$ provided by FP. In particular, we wanted to test the sensitivity of our modified LB algorithm with respect to the degree of infeasibility of the starting solution, as well as its capability of improving it. Thus, we executed the FP code for 0, 10 and 100 iterations[2] and passed to LB the integer (infeasible) solution $\hat{x}$ with minimum distance $\Phi(x^*, \hat{x})$ from P_L. The resulting three versions of the modified LB are called $\texttt{LB}_0$, $\texttt{LB}_{10}$, and $\texttt{LB}_{100}$, respectively.

[1] Another 0-1 MIP included in the library, namely stp3d, was not considered since the computing time required for the first LP relaxation is larger than 1 hour.

[2] The case with 0 iterations actually corresponds to starting from the solution of the continuous relaxation, rounded to the nearest integer.

Name	n	$\|\mathcal{I}\|$	m
10teams	2025	1800	230
A1C1S1	3648	192	3312
aflow30a	842	421	479
aflow40b	2728	1364	1442
air04	8904	8904	823
air05	7195	7195	426
cap6000	6000	6000	2176
dano3mip	13873	552	3202
danoint	521	56	664
ds	67732	67732	656
fast0507	63009	63009	507
fiber	1298	1254	363
fixnet6	878	378	478
glass4	322	302	396
harp2	2993	2993	112
liu	1156	1089	2178
markshare1	62	50	6
markshare2	74	60	7
mas74	151	150	13
mas76	151	150	12
misc07	260	259	212
mkc	5325	5323	3411
mod011	10958	96	4480
modglob	422	98	291
momentum1	5174	2349	42680
net12	14115	1603	14021
nsrand_ipx	6621	6620	735
nw04	87482	87482	36
opt1217	769	768	64
p2756	2756	2756	755
pk1	86	55	45
pp08a	240	64	136
pp08aCUTS	240	64	246
protfold	1835	1835	2112
qiu	840	48	1192
rd-rplusc-21	622	457	125899
set1ch	712	240	492
seymour	1372	1372	4944
sp97ar	14101	14101	1761
swath	6805	6724	884
t1717	73885	73885	551
tr12-30	1080	360	750
van	12481	192	27331
vpm2	378	168	234

Table 1: The 44 0-1 MIP instances collected in MIPLIB 2003 [1]

Name	n	$\|\mathcal{I}\|$	m	source
biella1	7328	6110	1203	[7]
NSR8K	38356	32040	6284	[7]
dc1c	10039	8380	1649	[6]
dc1l	37297	35638	1653	[6]
dolom1	11612	9720	1803	[6]
siena1	13741	11775	2220	[6]
trento1	7687	6415	1265	[6]
rail507	63019	63009	509	[7]
rail2536c	15293	15284	2539	[7]
rail2586c	13226	13215	2589	[7]
rail4284c	21714	21705	4284	[7]
rail4872c	24656	24645	4875	[7]
A2C1S1	3648	192	3312	[7]
B1C1S1	3872	288	3904	[7]
B2C1S1	3872	288	3904	[7]
sp97ic	12497	12497	1033	[7]
sp98ar	15085	15085	1435	[7]
sp98ic	10894	10894	825	[7]
bg512142	792	240	1307	[19]
dg012142	2080	640	6310	[19]
blp-ar98	16021	15806	1128	[17]
blp-ic97	9845	9753	923	[17]
blp-ic98	13640	13550	717	[17]
blp-ir98	6097	6031	486	[17]
CMS750_4	11697	7196	16381	[15]
berlin_5_8_0	1083	794	1532	[15]
railway_8_1_0	1796	1177	2527	[15]
usAbbrv.8.25_70	2312	1681	3291	[15]
manpower1	10565	10564	25199	[22]
manpower2	10009	10008	23881	[22]
manpower3	10009	10008	23915	[22]
manpower3a	10009	10008	23865	[22]
manpower4	10009	10008	23914	[22]
manpower4a	10009	10008	23866	[22]
ljb2	771	681	1482	[5]
ljb7	4163	3920	8133	[5]
ljb9	4721	4460	9231	[5]
ljb10	5496	5196	10742	[5]
ljb12	4913	4633	9596	[5]

Table 2: The additional set of 39 0-1 MIP instances

In our experiments, we avoided any parameter tuning: `FP` was implemented as in [8], and for the modified `LB` code we used a time limit of 30 CPU seconds for the exploration of each local-branching neighborhood. As to the value of the neighborhood-size parameter k in `LB`, we implemented an adaptive procedure: at each neighborhood exploration we try to half the number of violated constraints in the current solution, i.e., we set $k = \lfloor |T'|/2 \rfloor$, where $|T'|$ is the value of the current solution[3]. The motivation for this choice is that the number of violated constraints in an initial solution can be extremely large, in which case the use of a small value of k would result into a very slow convergence.

All codes are written in ANSI C and use the ILOG-Cplex callable libraries; they are available, on request, from the second author.

The three modified LB codes (LB_0, LB_{10}, and LB_{100}) are compared with FP and ILOG-Cplex 9.0.3 in Table 3 for the MIPLIB-2003 instances, and in Table 4 for the additional set of instances. Computing times are expressed in CPU seconds, and refer to a Pentium M 1.6 GHz notebook with 512 MByte of main memory. A time limit of 1,800 CPU second is provided for each instance to each algorithm and the computation is halted as soon as a first feasible solution is found.

For each instance, in both tables we report: for ILOG-Cplex, the number of nodes (nodes) needed to find an initial solution and the corresponding computing time (time); for FP, the number of iterations (FPit) and its computing time (time); for each of the three variants of LB, the computing time spent in the FP preprocessing phase (FP time), the initial number of violated constraints ($|T|$), the number of LB iterations (LBit), and the overall computing time (time). When one of the algorithms is not able to find a feasible solution in the given time limit, we write (*) in column "nodes" (for ILOG-Cplex) or "FPit" (for FP), or write (μ) in column "LBit" (for LB), where μ is the number of violated constraints in the final solution.

As expected, the degree of infeasibility of the starting solution plays an important role in the LB methods—the better the initial solution, the faster the method. In this view, the FP approach confirms to fit particularly well in our context, in that it is able to provide very good solutions (as far as the degree of infeasibility is concerned) in very short computing times. Among the three LB implementations, LB_0 was at least as fast as the other two in 40 cases, LB_{10} in 47 cases, and LB_{100} in 58 cases. Overall, LB_{100} qualifies as the most effective (and stable) of the three methods.

A comparison of ILOG-Cplex and LB_{100} shows that the latter is strictly faster in 44 cases, while the opposite holds in 21 cases. Moreover, ILOG-Cplex was not able to find any feasible solution (within the 1,800-second time limit) in 4 cases, whereas LB_{100} was unsuccessful 3 times. As expected, the quality of the initial ILOG-Cplex solution (not reported in the tables) is typically better than that provided by the LB methods.

Acknowledgements

Work partially supported by MIUR and CNR, Italy, and by the EU project "ADONET".

References

[1] T. Achterberg, T. Koch, A. Martin. The mixed integer programming library: MIPLIB 2003. http://miplib.zib.de.

[2] E. Balas, S. Ceria, M. Dawande, F. Margot, G. Pataki. OCTANE: A New Heuristic For Pure 0-1 Programs. *Operations Research* 49, 207–225, 2001.

[3] E. Balas, C.H. Martin. Pivot-And-Complement: A Heuristic For 0-1 Programming. *Management Science* 26, 86–96, 1980.

[4] E. Balas, S. Schmieta, C. Wallace. Pivot and Shift-A Mixed Integer Programming Heuristic. *Discrete Optimization* 1, 3–12, 2004.

[5] E. Danna, E. Rothberg, C. Le Paper. Exploring relaxation induced neighborhoods to improve MIP solutions. *Mathematical Programming* 102, 71–90, 2005.

[3] Since the support of the solution also takes into account non-artificial binary variables, when the number of violated constraints becomes less than 20 we fix $k = 10$, i.e., we use the value suggested in [7] for the asymmetric version of the local branching constraint.

name	ILOG-Cplex 9.0.3		FP		LB$_0$				LB$_{10}$				LB$_{100}$			
	nodes	time	FPit	time	FPtime	$\|T\|$	LBit	time	FPtime	$\|T\|$	LBit	time	FPtime	$\|T\|$	LBit	time
10teams	335	8.4	70	11.7	0.1	75	28	667.7	1.1	18	10	177.4	11.7	–	–	11.7
A1C1S1	150	4.1	8	3.8	0.1	63	4	0.8	3.8	–	–	3.8	3.8	–	–	3.8
aflow30a	0	0.1	18	0.1	0.0	29	3	3.0	0.1	29	3	0.3	0.1	–	–	0.1
aflow40b	370	5.9	6	0.3	0.1	40	4	57.6	0.3	–	–	0.3	0.3	–	–	0.3
air04	40	8.6	6	74.7	3.4	125	23	671.8	74.7	–	–	74.7	74.7	–	–	74.7
air05	70	3.4	25	83.8	0.8	208	11	135.0	22.8	14	2	25.0	83.8	–	–	83.8
cap6000	0	0.2	2	0.2	0.1	1	0	0.2	0.2	–	–	0.2	0.2	–	–	0.2
dano3mip	0	67.7	2	86.3	65.0	946 (105)	30	1,865.0	86.3	–	–	86.3	86.3	–	–	86.3
danoint	40	1.7	23	1.5	0.1	125	5	16.9	0.6	120	4	3.7	1.5	–	–	1.5
ds	0	55.0	133 (*)	1,800.0	54.5	656	15	582.8	229.9	350	7	302.1	1,358.6	133	5	1,385.0
fast0507	0	39.0	3	46.7	43.4	148	0	45.8	46.7	–	–	46.7	46.7	–	–	46.7
fiber	0	0.1	2	0.0	0.0	41	4	0.5	0.0	–	–	0.0	0.0	–	–	0.0
fixnet6	0	0.0	4	0.0	0.0	81	0	0.0	0.0	–	–	0.0	0.0	–	–	0.0
glass4	5389	1.6	124	0.3	0.0	52	4	0.9	0.0	45	3	0.1	0.2	45	3	0.3
harp2	0	0.0	654	5.0	0.0	9	2	0.9	0.1	6	0	0.1	0.8	6	0	0.8
liu	0	0.1	0	0.1	0.1	–	–	0.1	0.1	–	–	0.1	0.1	–	–	0.1
markshare1	0	0.0	4	0.0	0.0	6	1	0.0	0.0	–	–	0.0	0.0	–	–	0.0
markshare2	0	0.0	3	0.0	0.0	6	1	0.0	0.0	–	–	0.0	0.0	–	–	0.0
mas74	0	0.0	1	0.0	0.0	6	1	0.0	0.0	–	–	0.0	0.0	–	–	0.0
mas76	0	0.0	1	0.0	0.0	12	2	0.0	0.0	–	–	0.0	0.0	–	–	0.0
misc07	67	0.2	78	0.4	0.0	135	6	1.7	0.1	81	5	0.6	0.4	–	–	0.4
mkc	0	0.2	2	0.2	0.1	9	2	2.2	0.2	–	–	0.2	0.2	–	–	0.2
mod011	0	0.2	0	0.1	0.1	–	–	0.1	0.1	–	–	0.1	0.1	–	–	0.1
modglob	0	0.0	0	0.0	0.0	–	–	0.0	0.0	–	–	0.0	0.0	–	–	0.0
momentum1	314 (*)	1,800.0	502	1,329.6	1.8	697 (106)	17	1,801.8	42.6	895 (15)	57	1,842.6	178.8	895 (15)	57	1,978.8
net12	203 (*)	1,800.0	1	4.6	1.8	406	13	246.2	12.9	239	6	16.8	21.8	239	6	25.5
nsrand_ipx	0	0.5	1507	225.0	11.3	390	7	14.1	225.0	–	–	225.0	225.0	–	–	225.0
nw04	0	4.9	4	0.9	0.3	6	1	6.8	0.9	–	–	0.9	0.9	–	–	0.9
opt1217	117	0.1	0	0.0	0.0	–	–	0.0	0.0	–	–	0.0	0.0	–	–	0.0
p2756	0	0.1	150023 (*)	1,800.0	0.0	41	5	0.8	0.1	19	0	0.2	1.2	19	0	1.3
pk1	0	0.0	0	0.0	0.0	–	–	0.0	0.0	–	–	0.0	0.0	–	–	0.0
pp08a	0	0.0	2	0.0	0.0	55	0	0.0	0.0	–	–	0.0	0.0	–	–	0.0
pp08aCUTS	0	0.0	2	0.0	0.0	104	0	0.0	0.0	–	–	0.0	0.0	–	–	0.0
protfold	190	640.9	367	502.0	2.7	37 (37)	6	1,802.7	16.1	13 (1)	49	1,816.1	125.6	7 (1)	54	1,925.6
qiu	0	0.2	5	0.3	0.1	132	0	0.2	0.3	–	–	0.3	0.3	–	–	0.3
rd-rplusc-21	10978(*)	1,800.0	401 (*)	1,800.0	3.9	119021 (7094)	22	1,803.9	36.8	119017 (1)	70	1,836.8	449.5	119017 (2)	74	2,249.5
set1ch	0	0.0	2	0.1	0.0	202	0	0.0	0.1	–	–	0.1	0.1	–	–	0.1
seymour	0	3.5	7	3.6	3.0	921	0	3.8	3.6	–	–	3.6	3.6	–	–	3.6
sp97ar	0	3.4	4	4.2	2.9	222	0	3.8	4.2	–	–	4.2	4.2	–	–	4.2
swath	0	0.2	49	2.9	0.1	20	5	124.6	1.0	20	5	70.8	2.9	–	–	2.9
t1717	710	301.0	40	814.8	10.7	445 (50)	24	1,810.7	133.2	108 (5)	34	1,933.2	814.8	–	–	814.8
tr12-30	179	0.9	8	0.1	0.0	348	7	0.6	0.1	–	–	0.1	0.1	–	–	0.1
van	0	872.8	10	300.5	27.4	192 (128)	8	1,827.4	300.5	–	–	300.5	300.5	–	–	300.5
vpm2	0	0.0	3	0.0	0.0	23	3	0.0	0.0	–	–	0.0	0.0	–	–	0.0

Table 3: Convergence to a first feasible solution on the MIPLIB-2003 instances

name	ILOG-Cplex 9.0.3		FP		LB$_0$				LB$_{10}$				LB$_{100}$			
	nodes	time	FPit	time	FPtime	$\|T\|$	LBit	time	FPtime	$\|T\|$	LBit	time	FPtime	$\|T\|$	LBit	time
biella1	594	108.4	4	2.8	2.3	1193	8	18.2	2.8	–	–	2.8	2.8	–	–	2.8
NSR8K	5 (*)	1,800.0	3	195.5	185.8	5488 (5488)	0	1,985.8	195.5	–	–	195.5	195.5	–	–	195.5
dc1c	4749	474.0	2	12.7	11.6	1483	10	81.6	12.7	–	–	12.7	12.7	–	–	12.7
dc1l	0	80.8	2	16.2	14.0	1567	0	14.8	16.2	–	–	16.2	16.2	–	–	16.2
dolom1	367	504.4	22	22.6	11.9	1410	11	277.1	17.7	632	7	49.4	22.6	–	–	22.6
siena1	600	1,371.5	3	43.7	40.6	1750	11	271.2	43.7	–	–	43.7	43.7	–	–	43.7
trento1	340	276.8	7	11.0	9.3	603	7	22.6	11.0	–	–	11.0	11.0	–	–	11.0
rail507	0	32.8	2	8.7	6.5	218	0	7.4	8.7	–	–	8.7	8.7	–	–	8.7
rail2536c	0	16.8	1	15.2	14.3	2008	0	14.9	15.2	–	–	15.2	15.2	–	–	15.2
rail2586c	0	63.9	1	8.3	7.6	1871	0	7.9	8.3	–	–	8.3	8.3	–	–	8.3
rail4284c	0	204.9	2	56.7	53.5	3305	0	54.2	56.7	–	–	56.7	56.7	–	–	56.7
rail4872c	0	186.4	2	19.3	17.5	3254	0	18.3	19.3	–	–	19.3	19.3	–	–	19.3
A2C1S1	0	0.1	5	4.7	0.1	60	0	0.2	4.7	–	–	4.7	4.7	–	–	4.7
B1C1S1	0	0.1	6	5.0	0.1	208	0	0.2	5.0	–	–	5.0	5.0	–	–	5.0
B2C1S1	0	0.1	7	4.7	0.1	217	0	0.3	4.7	–	–	4.7	4.7	–	–	4.7
sp97ic	0	2.4	3	3.1	1.7	173	0	2.4	3.1	–	–	3.1	3.1	–	–	3.1
sp98ar	0	3.8	3	5.2	3.5	260	6	23.6	5.2	–	–	5.2	5.2	–	–	5.2
sp98ic	0	2.1	2	2.6	1.8	147	5	6.0	2.6	–	–	2.6	2.6	–	–	2.6
blp-ar98	8300	158.3	835	122.9	0.5	212	7	31.7	2.5	204	6	16.4	15.7	205	6	25.9
blp-ic97	1120	16.2	8	1.3	0.3	59	4	5.5	1.3	–	–	1.3	1.3	–	–	1.3
blp-ic98	1570	33.6	3	1.5	0.9	76	4	5.0	1.5	–	–	1.5	1.5	–	–	1.5
blp-ir98	1230	8.1	4	0.4	0.1	37	3	1.3	0.4	–	–	0.4	0.4	–	–	0.4
CMS750_4	940	27.2	16	6.5	0.7	2446	0	9.2	3.3	2441	0	11.7	6.5	–	–	6.5
berlin_5_8_0	152	0.4	13	0.2	0.0	170	19	275.4	0.1	167	0	0.2	0.2	–	–	0.2
railway_8_1_0	350	1.3	12	0.3	0.1	374	16	358.4	0.2	373	0	0.5	0.3	–	–	0.3
usAbbrv.8.25_70	274581	1,371.5	31	0.7	0.1	400	0	0.6	0.3	376	0	0.8	0.7	–	–	0.7
bg512142	0	0.3	0	0.2	0.2	–	–	0.2	0.2	–	–	0.2	0.2	–	–	0.2
dg012142	0	1.0	0	0.8	0.8	–	–	0.8	0.8	–	–	0.8	0.8	–	–	0.8
manpower1	154	1,800.0	30	18.8	8.4	1142	14	108.7	13.4	336	8	52.0	18.8	–	–	18.8
manpower2	150	364.6	92	137.5	39.5	1181	29	774.2	73.6	309	12	394.8	137.5	–	–	137.5
manpower3	181	326.9	42	76.2	27.1	1160	22	534.7	55.5	427	16	363.3	76.2	–	–	76.2
manpower3a	181	925.1	293	294.1	30.6	1327 (7)	56	1,830.6	53.3	369	17	491.2	114.8	9	1	372.1
manpower4	185	671.0	208	138.9	14.3	1105	33	1,010.8	41.8	604	18	427.7	80.5	40	7	383.8
manpower4a	194	1,039.9	308	289.2	36.4	1226	36	814.8	69.1	483	17	440.0	159.3	7	3	206.2
ljb2	30	0.2	0	0.0	0.0	–	–	0.0	0.0	–	–	0.0	0.0	–	–	0.0
ljb7	100	3.8	0	0.6	0.6	–	–	0.6	0.6	–	–	0.6	0.6	–	–	0.6
ljb9	180	7.0	0	0.8	0.8	–	–	0.8	0.8	–	–	0.8	0.8	–	–	0.8
ljb10	90	5.9	0	1.1	1.1	–	–	1.1	1.1	–	–	1.1	1.1	–	–	1.1
ljb12	110	5.8	0	0.7	0.7	–	–	0.7	0.7	–	–	0.7	0.7	–	–	0.7

Table 4: Convergence to a first feasible solution on the additional set of 0-1 MIP instances

[6] Double-Click sas. Personal communication, 2001.

[7] M. Fischetti, A. Lodi. Local Branching. *Mathematical Programming* 98, 23–47, 2003.

[8] M. Fischetti, F. Glover, A. Lodi. The Feasibility Pump. *Mathematical Programming* to appear, 2005.

[9] M. Fischetti, C. Polo, M. Scantamburlo. A Local Branching Heuristic for Mixed-Integer Programs with 2-Level Variables. *Networs* 44, 2, 61–72, 2004.

[10] F. Glover, M. Laguna. General Purpose Heuristics For Integer Programming: Part I. *Journal of Heuristics* 2, 343–358, 1997.

[11] F. Glover, M. Laguna. General Purpose Heuristics For Integer Programming: Part II. *Journal of Heuristics* 3, 161–179, 1997.

[12] F. Glover, M. Laguna. *Tabu Search.* Kluwer Academic Publisher, Boston, Dordrecht, London, 1997.

[13] P. Hansen, N. Mladenovíc, D. Urosevíc. Variable Neighborhood Search and Local Branching. *Technical Report* G-2004-54, Les Chaiers du GERAD, June 2004.

[14] F. S. Hillier. Effcient Heuristic Procedures For Integer Linear Programming With An Interior. *Operations Research* 17, 600–637, 1969.

[15] G.W. Klau. Personal communication, 2002.

[16] T. Ibaraki, T. Ohashi, H. Mine. A Heuristic Algorithm For Mixed-Integer Programming Problems. *Mathematical Programming Study* 2, 115–136, 1974.

[17] A. Løkketangen. Heuristics for 0-1 Mixed-Integer Programming. In P.M. Pardalos and M.G.C. Resende (ed.s) *Handbook of Applied Optimization*, Oxford University Press, 474–477, 2002.

[18] A. Løkketangen, F. Glover. Solving Zero/One Mixed Integer Programming Problems Using Tabu Search. *European Journal of Operational Research* 106, 624-658, 1998.

[19] A.J. Miller. Personal communication, 2003.

[20] N. Mladenovíc, P. Hansen. Variable Neighborhood Search. *Computers & Operations Research* 24, 1097–1100, 1997.

[21] M. Nediak, J. Eckstein. Pivot, Cut, and Dive: A Heuristic for 0-1 Mixed Integer Programming *Research Report* RRR 53-2001, RUTCOR, Rutgers University, October 2001.

[22] E. Rothberg. Personal communication, 2002.