A Feasibility Pump heuristic for general Mixed-Integer Problems

Livio Bertacco[*†], Matteo Fischetti[°], Andrea Lodi[§‡]

[*]Department of Pure & Applied Mathematics, University of Padova,
via Belzoni 7 - 35131 Padova (Italy) - livio.bertacco@unipd.it

[†]Dash Optimization Ltd,
64 Trinity Street - Leamington Spa CV32 5YN (UK)

[°]Department of Information Engineering, University of Padova,
via Gradenigo 6/A - 35131 Padova (Italy) - matteo.fischetti@unipd.it

[§] DEIS, University of Bologna,
viale Risorgimento 2, 40136 Bologna (Italy)

‡IBM, T.J. Watson Research Center
PO Box 218, Yorktown Heights, NY 10598, USA - alodi@us.ibm.com

May 31, 2005; Revised November 3, 2005

Abstract

Finding a feasible solution of a given Mixed-Integer Programming (MIP) model is a very important ($\mathcal{NP}$-complete) problem that can be extremely hard in practice. Very recently, Fischetti, Glover and Lodi proposed a heuristic scheme for finding a feasible solution to general MIPs, called *Feasibility Pump* (FP). According to the computational analysis reported by these authors, FP is indeed quite effective in finding feasible solutions of hard 0-1 MIPs. However, MIPs with general-integer variables seem much more difficult to solve by using the FP approach.

In this paper we elaborate on the Fischetti-Glover-Lodi approach and extend it in two main directions, namely (i) handling as effectively as possible MIP problems with both binary and general-integer variables, and (ii) exploiting the FP information to drive a subsequent enumeration phase.

Extensive computational results on large sets of test instances from the literature are reported, showing the effectiveness of our improved FP scheme for finding feasible solutions to hard MIPs with general-integer variables.

Key words: Mixed-integer programming, Heuristics, Feasibility Problem, Computational analysis.

1 Introduction

In this paper we address the problem of finding heuristic solutions of a generic MIP problem of the form

$$(MIP) \quad \min c^T x \tag{1}$$
$$Ax \geq b \tag{2}$$
$$x_j \text{ integer} \quad \forall j \in \mathcal{I} \tag{3}$$

where A is an $m \times n$ matrix and $\mathcal{I}$ is the nonempty index-set of the integer variables. We assume without loss of generality that the MIP constraints $Ax \geq b$ include the variable bounds

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

(possibly $l_j = -\infty$ and/or $u_j = +\infty$ for some j), where $\mathcal{N}$ denotes the set of all (continuous and integer) variables.

Finding any feasible MIP solution is an $\mathcal{NP}$-complete problem that can be extremely hard in practice. As a matter of fact, state-of-the-art MIP solvers may spend a very large computational effort before initializing their incumbent solution. Therefore, heuristic methods aimed at finding (and then refining) any feasible solution for hard MIPs are very important in practice; see [4], [5], [13], [14], [15], [16], [17], [19], [20], [22], [11], [9], and [6] among others.

Very recently, Fischetti, Glover and Lodi [12] proposed a heuristic scheme for finding a feasible solution to general MIPs, called *Feasibility Pump* (`FP`), that works as follows. Let $P := \{x : Ax \geq b\}$ denote the polyhedron associated with the LP relaxation of the given MIP. With a little abuse of notation, we say that a point x is integer if x_j is integer $\forall j \in \mathcal{I}$ (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 $x \in P$ and a given integer $\tilde{x}$ is defined as

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

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 $\Delta(x, \tilde{x})$. For any given integer $\tilde{x}$, the distance function can be written as[1]:

$$\Delta(x, \tilde{x}) := \sum_{j \in \mathcal{I}: \tilde{x}_j = l_j} (x_j - l_j) + \sum_{j \in \mathcal{I}: \tilde{x}_j = u_j} (u_j - x_j) + \sum_{j \in \mathcal{I}: l_j < \tilde{x}_j < u_j} d_j \qquad (4)$$

where variables $d_j (= |x_j - \tilde{x}_j|)$ satisfy constraints

$$d_j \geq x_j - \tilde{x}_j \quad \text{and} \quad d_j \geq \tilde{x}_j - x_j \quad \forall j \in \mathcal{I} : l_j < \tilde{x}_j < u_j \qquad (5)$$

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

$$\min\{\Delta(x, \tilde{x}) : \widetilde{A}x \geq \widetilde{b}\} \qquad (6)$$

where $\widetilde{A}x \geq \widetilde{b}$ is the original system $Ax \geq b$ possibly amended by constraints (5). If $\Delta(x^*, \tilde{x}) = 0$, then $x^*_j \, (= \tilde{x}_j)$ is integer $\forall j \in \mathcal{I}$, so x^* is a feasible MIP solution. Conversely, given a point $x^* \in P$, 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 $x^* \in P$ and $\tilde{x}$ integer, that are iteratively updated with the aim of reducing as much as possible their distance $\Delta(x^*, \tilde{x})$. To be more specific, one starts with any $x^* \in P$, 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 $x^* \in P$ which is as close as possible to $\tilde{x}$. If $\Delta(x^*, \tilde{x}) = 0$, 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 $\Delta(x^*, \tilde{x})$, and the process is iterated.

The `FP` scheme (as stated) tends to stop prematurely due to stalling issues. This happens whenever $\Delta(x^*, \tilde{x}) > 0$ 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 this situation, a few components $\tilde{x}_j$ are heuristically chosen and modified, even if this operation increases the current value of $\Delta(x^*, \tilde{x})$. The reader is referred to [12] for a detailed description of this (and related) anti-stalling mechanisms.

According to the computational analysis reported in [12], `FP` is quite effective in finding feasible solutions of hard 0-1 MIPs. However, as observed in the conclusions of that paper, MIPs with general-integer variables are much more difficult to solve by using the `FP` approach. This can be explained

[1]This expression is slightly different from the one proposed in [12]; both definitions assume an objective function that tends to minimize the value of the d_j variables.

by observing that, for a general integer variable, one has to decide not just the rounding direction (up or down), as for binary variables, but also the new value of the variable; e.g., if a variable x_j is between 0 and 10 and takes value 6.7 (say) in the LP relaxation, the decision of "moving up" its value still leaves four values (7, 8, 9, and 10) to choose from. The same difficulty arises in case of stalling: in the binary case, one only needs to choose the variables to flip (from 0 to 1 or viceversa), whereas for general integer variables one also has to decide their new value.

In this paper we build on the ideas presented in [12] for 0-1 MIPs and extend them in two main directions. The first one is to handle effectively MIP problems with both binary and general integer variables. The second is to exploit the information obtained from the feasibility pump to drive an enumeration stage.

The paper is organized as follows. In Section 2 we propose an `FP` extension to deal with MIPs with general-integer variables. Computational results are presented in Section 3, where we compare the `FP` performance with that of the commercial solvers `Xpress Optimizer` 16.01.05 and `ILOG Cplex` 9.1 on a set of hard general MIPs taken from MIPLIB 2003 library [3] and other sources. Section 4 considers the important issue of improving the quality of the first solution found by `FP`. Finally, we draw some conclusions in Section 5.

2 The Feasibility Pump for general-integer MIPs

The basic scheme of our `FP` implementation for general MIPs is illustrated in Figure 1. As already stated, the method generates two (hopefully convergent) trajectories of points x^* and $\tilde{x}$ that satisfy feasibility in a complementary but partial way—one satisfies the linear constraints, the other the integer requirement. The current pair $(x^*, \tilde{x})$ is initialized at steps 1-2. The while-do loop at step 4 is executed until the distance $\Delta(x^*, \tilde{x})$ becomes zero (in which case, x^* is a feasible MIP solution), or the current iteration counter `nIter` reaches a given limit (`maxIter`). At each pumping cycle, at step 6 we fix $\tilde{x}$ and re-define x^* as the closest point in P, so as to hopefully reduce the current distance $\Delta(x^*, \tilde{x})$. At step 7 we check whether $\Delta(x^*, \tilde{x}) = 0$, in which case x^* is feasible and we stop. Otherwise, at step 9 we replace $\tilde{x}$ by $[x^*]$ (the rounding of x^*), and repeat. In case the components of the new $\tilde{x}$ indexed by $\mathcal{I}$ would coincide with the previous ones, however, a more involved computation is needed to avoid entering a loop. We first compute, at step 11, a score $\sigma_j = |x^*_j - \tilde{x}_j|$, $j \in \mathcal{I}$, giving the likelihood of component

```
The Feasibility Pump for general MIPs (basic scheme):
 1. initialize x* := argmin{c^T x : Ax ≥ b}
 2. x̃ := [x*] (:= rounding of x*)
 3. nIter := 0
 4. while (Δ(x*, x̃) > 0 and nIter < maxIter) do
 5.   nIter := nIter+1
 6.   x* := argmin{Δ(x, x̃) : Ãx ≥ b̃}
 7.   if Δ(x*, x̃) > 0 then
 8.     if [x*_j] ≠ x̃_j for at least one j ∈ I then
 9.       update x̃ := [x*]
10.     else
11.       for each j ∈ I define the score σ_j := |x*_j − x̃_j|
12.       move the TT=rand(T/2, 3T/2) components x̃_j with largest σ_j
13.       if cycling is detected, perform a restart operation
14.     endif
15.   endif
16. enddo
```

Figure 1: The basic FP scheme for general-integer MIPs

$\tilde{x}_j$ to *move*, i.e., to change its current value from $\tilde{x}_j$ to $\tilde{x}_j + 1$ (if $x^*_j > \tilde{x}_j$) or to $\tilde{x}_j - 1$ (if $x^*_j < \tilde{x}_j$). Then, at step 12 we update $\tilde{x}$ by performing the TT (say) moves with largest score, where TT is generated as a uniformly-random integer in range (T/2, 3T/2), and T is a given parameter.

In order to avoid cycling, at step 13 we check whether one of the following situations occurs:

- the current point $\tilde{x}$ is equal (in its components indexed by $\mathcal{I}$) to the one found in a previous iteration;
- distance $\Delta(x^*, \tilde{x})$ did not decrease by at least 10% in the last KK (say) iterations.

If this is the case, we perform a *restart* operation (to be detailed later), consisting in a random perturbation of some entries of $\tilde{x}$.

As a further step to reduce the likelihood of cycling, we found it useful to also perturb the rounding function used at step 2 of Figure 1. Indeed, the rounded components are typically computed as $[x_j] := \lfloor x_j + \tau \rfloor$ with τ fixed at 0.5. However, in our tests we obtained better results by taking a

random τ defined as follows:

$$\tau(\omega) := \begin{cases} 2\omega(1-\omega) & \text{if } \omega \leq \frac{1}{2} \\ 1 - 2\omega(1-\omega) & \text{if } \omega > \frac{1}{2} \end{cases}$$

where ω is a uniform random variable in $[0, 1)$. Using the definition above, threshold τ can take any value between 0 and 1, but values close to 0.5 are more likely than those near 0 or 1; see Figure 2 for an illustration of the probability distribution and density for $\tau(\omega)$.

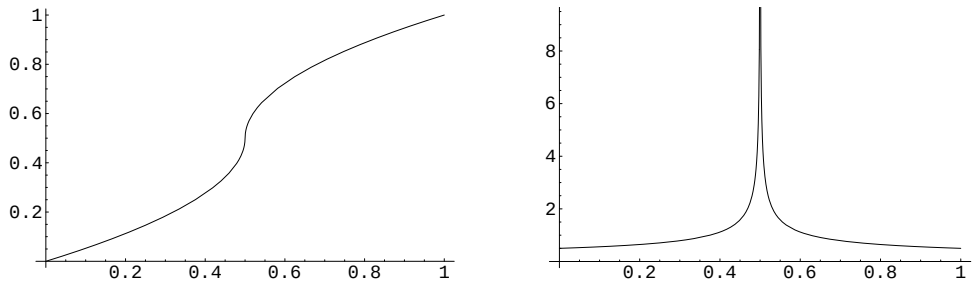

Figure 2: Probability distribution and density of the rounding threshold

2.1 Binary and general-integer stages

Difficult MIPs often involve both binary and general-integer variables playing a quite different role in the model. A commonly-used rule in MIP solvers is to branch first on binary variables, then on general integers. This corresponds to the "smallest domain first" rule in constraint programming: branching on a variable with a large domain (e.g., a general integer variable) will not enforce as powerful constraints as branching on a variable with a small domain (e.g., a binary variable), therefore it is postponed to the bottom of the tree. Following this approach, we found useful to split the `FP` execution in two stages.

At Stage 1 (binary stage), we concentrate on the *binary* variables x_j with $j \in \mathcal{B}$ (say), defined as the integer variables x_j with $u_j - l_j = 1$, and relax the integrality condition on all other x_j, $j \in \mathcal{I} \setminus \mathcal{B}$. This produces an easier MIP, with a distance function $\Delta(x, \tilde{x})$ that does not involve any additional variable d_j. The purpose of the binary stage is to reach as soon as possible a solution that is feasible with respect to the binary variables, with the hope that the general-integer ones are also "almost integer" and only a few of them will require the introduction of the additional variables d_j and of the associated constraints (5).

At Stage 2, instead, the integrality condition on all (binary and non-binary) variables x_j, $j \in \mathcal{I}$, is restored, and the FP method continues by taking into account all the integrality requirements (this requires the introduction, at each iteration, of the additional variables d_j needed to express the distance function with respect to the current point $\tilde{x}$).

During Stage 1, the restart operation at step 13 is only performed in case of cycling (i.e., when the same $\tilde{x}$ is found in different iterations), and consists in a random flip of the binary variables that did not change in the last iteration, with probability $|x_j^* - [x_j^*]| + \rho$, where $\rho = 0.03$ is an hardwired parameter.

The algorithm exits Stage 1 and moves to Stage 2 when: (a) a "feasible" (with respect to the binary variables only) solution x^* has been found, or (b) the incumbent minimum $\Delta(x^*, \tilde{x})$ has not been updated in the last KK $= 70$ iterations, or (c) an iteration limit has been reached. The point $\tilde{x}$ that produced the smallest $\Delta(x^*, \tilde{x})$ during Stage 1 is stored and passed to Stage 2 as the initial $\tilde{x}$.

2.2 Enumeration stage

For some difficult instances, FP (as stated) turns out to be unable to find a feasible solution within acceptable computing time. In this case, instead of insisting with the classical FP scheme one can think of resorting to a sort of "enumeration stage" based on the information provided by the previous FP execution. Following this idea, we have implemented the following simple scheme.

Let x^B (B for best) be the LP point x^* computed at step 6 of the algorithm of Figure 1 which is as close as possible to its rounding $\tilde{x}$. Even in case x^B is not feasible, we typically have that the infeasibility measure $\Delta(x^B, \tilde{x})$ is small. Therefore it seems reasonable to concentrate on $\tilde{x}$ and use a (possibly heuristic) enumerative MIP method in the attempt of finding a feasible integer solution which is close to $\tilde{x}$. In our implementation, this is obtained by applying a general-purpose (truncated) MIP solver to the original problem (1), after having replaced the original objective function $c^T x$ by the distance function (4), where $\tilde{x} := [x^B]$ is the "almost feasible" solution available after Stage 2. The idea here is to exploit the full power of the MIP solver, but with an objective function that penalizes the solutions that are far from the available "almost feasible" FP solution $\tilde{x}$.

As the enumeration phase above is applied at the end of Step 2, it will be referred to as the Stage 3 of the overall FP method.

3 Computational experiments

In this section we report computational results comparing the performance of the proposed FP method with that of the two state-of-the-art commercial solvers, namely, Xpress Optimizer 16.01.05 [8] and ILOG Cplex 9.1 [18]. Of course, the heuristic performance of a MIP solver depends heavily on the branching rule (as discussed, e.g., in [7]), on the tree-exploration strategy [9], and on the tuning of specific parameters of the MIP solver at hand. In our experiments, however, we decided to use as much as possible the default parameter values of the codes under comparison. To be specific, in our FP implementation we used the following parameters:

- iteration limit set to 10000 and 2000 for Stage 1 and 2, respectively;
- parameter TT set to 20;
- parameter KK set to 70 for stage 1 and 600 for Stage 2;
- in the perturbation phase of steps 11-12 of Figure 1, we consider only variables with fractional value (defined as $|x_j - [x_j]|$) greater than 0.02, and always leave the other variables unmodified.

In order to have a fair comparison, the LP/MIP functions used within FP are the same used by the method under comparison. To be more specific, we ran Xpress Optimizer against an FP implementation based on Xpress Optimizer procedures (called FP-xpress in the sequel), and ILOG Cplex against an FP implementation based on ILOG Cplex procedures (called FP-cplex in the sequel). Within our FP code, we used the ILOG Cplex function CPXoptimize and Xpress Optimizer function XPRSminim to solve the initial LP (thus leaving to the solver the choice of the actual LP algorithm to invoke), with the default parameter setting except for disabling the presolver. For the subsequent LPs with the modified objective function, we forced the use of the primal simplex algorithm. The reason for this choice is that, from iteration to iteration, the feasibility of the current basis is always preserved during stage 1 and, quite often, also during stage 2 (according to our computational tests, using the primal simplex yields to better computing times on 31 out of 35 instances and, on average, to a 25% performance improvement). Finally, within the enumeration Stage 3 we set the ILOG Cplex parameter MIP emphasis to 4 (i.e., *Emphasize hidden feasible solutions*, so as to activate the RINS heuristic [9]), in order to bias the search towards feasibility rather than optimality. All other solver parameters were left at their default values.

The MIP solvers compared against FP have been run in their default settings (with presolver enabled), except the ILOG Cplex parameter MIP mphasis (set to 1, i.e., *Emphasize integer feasibility*) and the Xpress Optimizer parameter XPRScutstrategy (set to 3, i.e., *Aggressive cut strategy*). According to our computational experience, these settings gave the best average results, for the respective solvers, on the instances we considered.

Our testbed is made by general-integer MIP instances drawn from four sources:

1. instances from MIPLIB 2003 [3];
2. instances from MILPlib [21];
3. the periodic scheduling instances described in [23];
4. the network design and multicommodity routing instances described in [9].

Pure 0-1 instances from all sets have been excluded from the comparison, as they have been addressed in [12].

Table 1 reports the instance name, the corresponding number of variables (n), of 0-1 variables ($|\mathcal{B}|$), of general-integer variables ($|\mathcal{G}| = |\mathcal{I}| - |\mathcal{B}|$) and of constraints ($m$).

The results of our experiments are reported in Table 2 (Xpress Optimizer vs FP-xpress) and in Table 3 (ILOG Cplex vs FP-cplex). The focus was to evaluate the capability of the compared methods to converge to an initial feasible solution, hence all methods were stopped as soon as the first feasible solution was found. For the MIP solvers, the tables report the computing time to get the first feasible solution (*time*) and the corresponding number of branching nodes (*nodes*).

As to FP, we report the computing time to get the first feasible solution (*time*), the stage where this solution was found (*stage*), the overall number of FP iterations (*iter*) and of restarts (*restarts*) in stages 1 and 2. The last column of the tables gives the computing time speedup of FP over the compared methods (a value greater than 1 meaning that FP was faster). Finally, the last rows of the tables report the total computing time needed to process the whole testbed, and the average speedup of FP over the compared method.

Computing times are expressed in CPU seconds, and refer to an Intel Pentium IV 2.4GHz personal computer with 512 Mbyte of main memory. A time limit of 1 hour of CPU time was imposed for all methods.

Name	n	$\lvert\mathcal{B}\rvert$	$\lvert\mathcal{G}\rvert$	m	source
arki001	1388	415	123	1048	[3]
atlanta-ip	48738	46667	106	21732	[3]
gesa2	1224	240	168	1392	[3]
gesa2-o	1224	384	336	1248	[3]
manna81	3321	18	3303	6480	[3]
momentum2	3732	1808	1	24237	[3]
momentum3	13532	6598	1	56822	[3]
msc98-ip	21143	20237	53	15850	[3]
mzzv11	10240	9989	251	9499	[3]
mzzv42z	11717	11482	235	10460	[3]
noswot	128	75	25	182	[3]
roll3000	1166	246	492	2295	[3]
rout	556	300	15	291	[3]
timtab1	397	64	107	171	[3]
timtab2	675	113	181	294	[3]
neos10	23489	23484	5	46793	[21]
neos16	377	336	41	1018	[21]
neos20	1165	937	30	2446	[21]
neos7	1556	434	20	1994	[21]
neos8	23228	23224	4	46324	[21]
ic97_potential	728	450	73	1046	[23]
ic97_tension	703	176	4	319	[23]
icir97_potential	2112	1235	422	3314	[23]
icir97_tension	2494	262	573	1203	[23]
rococoB10-011000	4456	4320	136	1667	[9]
rococoB10-011001	4456	4320	136	1677	[9]
rococoB11-010000	12376	12210	166	3792	[9]
rococoB11-110001	12431	12265	166	8148	[9]
rococoB12-111111	9109	8910	199	8978	[9]
rococoC10-001000	3117	2993	124	1293	[9]
rococoC10-100001	5864	5740	124	7596	[9]
rococoC11-010100	12321	12155	166	4010	[9]
rococoC11-011100	6491	6325	166	2367	[9]
rococoC12-100000	17299	17112	187	21550	[9]
rococoC12-111100	8619	8432	187	10842	[9]

Table 1: Our test bed of MIPs with general integer variables

	Xpress Optimizer		FP-xpress				
name	time	nodes	time	stage	iter	restarts	speedup
arki001	7.03	1	66.70	3	1132	100	0.11
atlanta-ip	962.36	220	191.83	1	53	12	5.02
gesa2	0.05	1	0.06	2	5	0	0.75
gesa2-o	0.07	1	0.14	2	25	5	0.50
manna81	0.16	1	3.78	2	3	0	0.04
momentum2	1996.14	295	> 3600.00	3	442	123	< 0.55
momentum3	> 3600.00	1	1479.75	3	350	128	> 2.43
msc98-ip	303.23	334	23.91	1	30	6	12.68
mzzv11	251.56	194	26.66	1	1	0	9.44
mzzv42z	8.45	1	19.52	1	2	0	0.43
noswot	0.02	1	0.00	2	4	0	5.21
roll3000	12.45	72	0.84	2	7	0	14.80
rout	0.06	1	0.05	1	25	9	1.33
timtab1	3.75	1819	0.77	2	293	31	4.90
timtab2	124.58	65387	6.97	3	806	60	17.88
neos10	19.41	1	13.31	1	2	0	1.46
neos16	> 3600.00	1154567	> 3600.00	3	726	70	1.00
neos20	12.11	634	9.95	3	685	82	1.22
neos7	0.20	1	0.17	2	3	0	1.21
neos8	19.30	1	45.08	1	1	0	0.43
ic97_potential	0.05	1	4.75	3	991	35	0.01
ic97_tension	2.92	1325	2.13	2	659	47	1.38
icir97_potential	> 3600.00	99765	13.75	3	767	17	> 261.82
icir97_tension	10.20	714	22.74	3	775	116	0.45
rococoB10-011000	0.69	1	1.13	1	18	1	0.61
rococoB10-011001	0.66	1	0.83	1	27	2	0.79
rococoB11-010000	2.03	1	2.33	1	25	1	0.87
rococoB11-110001	5.47	1	4.95	1	14	0	1.10
rococoB12-111111	1520.30	2376	> 3600.00	3	736	102	< 0.42
rococoC10-001000	0.20	1	0.75	1	63	13	0.27
rococoC10-100001	0.95	1	3.44	1	63	10	0.28
rococoC11-010100	2.08	1	2.45	1	19	1	0.85
rococoC11-011100	1.03	1	1.82	1	20	1	0.57
rococoC12-100000	8.39	1	8.08	1	14	0	1.04
rococoC12-111100	3.06	1	2.02	1	13	0	1.52
Total times	16078.96		12760.64	Geometric mean			1.14

Table 2: Convergence to a first feasible solution using Xpress Optimizer

	ILOG Cplex		FP-cplex				
name	time	nodes	time	stage	iter	restarts	speedup
arki001	2.83	474	46.53	3	937	74	0.06
atlanta-ip	1562.58	230	113.64	1	5	0	13.75
gesa2	0.05	0	0.02	2	4	0	3.00
gesa2-o	0.25	90	0.03	2	6	0	8.00
manna81	0.22	0	0.34	2	3	0	0.64
momentum2	> 3600.00	0	> 3600.00	3	585	131	1.00
momentum3	> 3600.00	0	1248.13	3	393	125	> 2.88
msc98-ip	1330.23	120	97.09	1	37	4	13.70
mzzv11	243.34	80	214.83	1	3	0	1.13
mzzv42z	46.58	50	68.56	1	2	0	0.68
noswot	0.00	0	0.00	2	3	0	1.00
roll3000	7.05	300	0.83	2	6	0	8.51
rout	0.34	90	0.05	1	29	5	7.33
timtab1	0.88	752	0.08	2	37	3	11.20
timtab2	129.31	49264	2.14	2	631	64	60.41
neos10	6.88	0	8.28	1	2	0	0.83
neos16	1272.05	400000	1660.88	3	755	99	0.77
neos20	2.17	194	7.41	3	696	93	0.29
neos7	0.64	50	1.84	3	296	139	0.35
neos8	6.80	0	5.00	1	1	0	1.36
ic97_potential	0.52	40	2.98	3	775	18	0.17
ic97_tension	5.11	4730	2.67	3	1110	99	1.91
icir97_potential	3.48	120	61.09	3	787	7	0.06
icir97_tension	2380.35	464527	4.38	2	344	54	544.08
rococoB10-011000	1.14	0	1.41	1	23	1	0.81
rococoB10-011001	8.06	70	0.89	1	23	1	9.05
rococoB11-010000	1.86	0	3.20	1	22	0	0.58
rococoB11-110001	5.75	0	7.80	1	22	0	0.74
rococoB12-111111	1808.09	3590	718.55	3	899	101	2.52
rococoC10-001000	0.28	0	0.50	1	53	11	0.56
rococoC10-100001	558.73	1520	2.03	1	58	8	275.07
rococoC11-010100	1.48	0	3.34	1	27	1	0.44
rococoC11-011100	2.13	0	2.39	1	26	1	0.89
rococoC12-100000	51.72	20	7.13	1	21	0	7.26
rococoC12-111100	2.00	0	3.30	1	13	0	0.61
Total times	16642.90		7897.33	Geometric mean			2.00

Table 3: Convergence to a first feasible solution using ILOG Cplex

According to the tables, FP compares favorably with both MIP solvers. Indeed, both FP-xpress and Xpress Optimizer found a feasible solution for all but 3 instances, but FP-xpress was 14% (in geometric mean) faster than Xpress Optimizer in finding its first solution. As to the ILOG Cplex implementation, FP-cplex found a feasible solution to all but one instance, thus solving one instance more than ILOG Cplex and was 2.00 times (geometric mean) faster than ILOG Cplex. Also to be noted is that 25 out of the 35 instances have been solved by FP-cplex either in Stage 1 or 2, i.e., without the enumeration of Stage 3.

To test the effectiveness of the binary stage, we also ran FP-cplex with its Stage 1 disabled. The results are reported in Table 4 and show that the binary stage has a really big impact on the overall performance: without Stage 1, 4 more instances could not be solved by FP-cplex, whose computing time was on average 9 times worse due to the increased number of iterations and of auxiliary variables (the latter reported in column *aux var*) required.

Table 5 reports the total time and percent time spent by FP-cplex in each individual stage.

Finally, in order to validate the effectiveness of our approach we compared these results with the performance of the original FP algorithm [12]. Since this method can only handle 0-1 MIPs, we converted each model in our testbed to a 0-1 problem by replacing each general-integer variable with a set of binary variables representing the binary encoding of the integer values. More precisely, we replaced each general-integer variable x_i, where $0 \leq x_i \leq u_i$, with $n_i := \lceil \log_2(u_i + 1) \rceil$ binary variables x_{ik} such that $x_i = \sum_{k=0}^{n_i-1} 2^k x_{ik}$. The original FP applied to the resulting 0-1 MIPs turned out to be faster in reaching its first feasible solution on just 3 instances (namely, arki001, neos10, and rococoC11-011100), whereas, on all other instances, it took much longer or could not find any solution at all.

4 Improving feasible solutions

As already mentioned, in the previous experiments our main attention was on the computing time spent to find a first feasible solution. In this respect, the FP results were very satisfactory. However, the quality of the solution delivered by FP is often considerably worse than that computed (in a typically longer time) by ILOG Cplex or Xpress Optimizer. This can be explained by noting that the FP method uses the original objective function only at step 1, when the solution of the LP relaxation is used to initialize x^*, while the original costs are completely disregarded during the next iterations. As

	FP with binary stage			FP without binary stage					
Name	time	iter	aux vars	time	ratio	iter	iter diff	aux vars	var diff
arki001	46.53	937	96	30.33	0.652	685	-252	95	-1
atlanta-ip	113.64	5	0	168.47	1.482	223	218	68	68
gesa2	0.02	4	35	0.09	6.000	13	9	26	-9
gesa2-o	0.03	6	25	0.08	2.500	11	5	27	2
manna81	0.34	3	2497	0.44	1.273	3	0	2504	7
momentum2	> 3600.00	585	1	> 3600.00	1.000	616	31	1	0
momentum3	1248.13	393	1	> 3600.00	2.884	441	48	1	0
msc98-ip	97.09	37	0	105.50	1.087	72	35	49	49
mzzv11	214.83	3	0	873.75	4.067	638	635	131	131
mzzv42z	68.56	2	0	488.70	7.128	662	660	141	141
noswot	0.00	3	4	0.02		19	16	12	8
roll3000	0.83	6	27	64.20	77.528	900	894	466	439
rout	0.05	29	0	0.11	2.333	41	12	7	7
timtab1	0.08	37	88	0.17	2.200	67	30	91	3
timtab2	2.14	631	163	1.72	0.803	421	-210	160	-3
neos10	8.28	2	0	7.31	0.883	1	-1	0	0
neos16	1660.88	755	41	874.41	0.526	978	223	41	0
neos20	7.41	696	30	9.67	1.306	978	282	30	0
neos7	1.84	296	20	1.91	1.034	197	-99	20	0
neos8	5.00	1	0	5.66	1.131	1	0	0	0
ic97_potential	2.98	775	68	4.81	1.613	1183	408	73	5
ic97_tension	2.67	1110	4	1.95	0.731	938	-172	4	0
icir97_potential	61.09	787	291	96.58	1.581	713	-74	292	1
icir97_tension	4.38	344	556	11.31	2.586	431	87	573	17
rococoB10-011000	1.41	23	0	629.59	447.711	633	610	134	134
rococoB10-011001	0.89	23	0	91.72	102.982	632	609	134	134
rococoB11-010000	3.20	22	0	2146.19	670.029	632	610	166	166
rococoB11-110001	7.80	22	0	> 3600.00	461.723	636	614	166	166
rococoB12-111111	718.55	899	173	> 3600.00	5.010	612	-287	193	20
rococoC10-001000	0.50	53	0	22.59	45.188	456	403	124	124
rococoC10-100001	2.03	58	0	2012.59	990.815	416	358	122	122
rococoC11-010100	3.34	27	0	1234.38	369.159	524	497	165	165
rococoC11-011100	2.39	26	0	527.63	220.706	621	595	163	163
rococoC12-100000	7.13	21	0	> 3600.00	505.263	574	553	187	187
rococoC12-111100	3.30	13	0	> 3600.00	1091.943	518	505	186	186
				mean	8.986		224		69

Table 4: Comparison of FP with and without binary stage

	times			percentages		
Name	stage 1	stage 2	stage 3	stage 1	stage 2	stage 3
arki001	0.17	39.30	8.31	0.36%	82.24%	17.40%
atlanta-ip	8.73			100.00%		
gesa2	0.00	0.02		0.00%	100.00%	
gesa2-o	0.00	0.03		0.00%	100.00%	
manna81	0.00	0.27		0.00%	100.00%	
momentum2	175.77	224.83	3199.41	4.88%	6.25%	88.87%
momentum3	160.08	432.44	160.77	21.25%	57.41%	21.34%
msc98-ip	7.81			100.00%		
mzzv11	2.13			100.00%		
mzzv42z	1.09			100.00%		
noswot	0.00	0.00				
roll3000	0.52	0.06		89.19%	10.81%	
rout	0.06			100.00%		
timtab1	0.02	0.08		16.67%	83.33%	
timtab2	0.14	2.02		6.52%	93.48%	
neos10	4.72			100.00%		
neos16	0.55	1.94	1713.63	0.03%	0.11%	99.86%
neos20	0.45	4.17	2.95	5.98%	55.05%	38.97%
neos7	0.28	1.52	0.08	15.00%	80.83%	4.17%
neos8	1.67			100.00%		
ic97_potential	0.52	2.47	0.13	16.58%	79.40%	4.02%
ic97_tension	0.19	2.00	0.52	6.94%	73.99%	19.08%
icir97_potential	1.73	8.98	52.67	2.74%	14.17%	83.09%
icir97_tension	0.95	3.47		21.55%	78.45%	
rococoB10-011000	0.30			100.00%		
rococoB10-011001	0.33			100.00%		
rococoB11-010000	0.94			100.00%		
rococoB11-110001	1.28			100.00%		
rococoB12-111111	45.13	79.66	608.11	6.16%	10.87%	82.97%
rococoC10-001000	0.39			100.00%		
rococoC10-100001	1.42			100.00%		
rococoC11-010100	1.02			100.00%		
rococoC11-011100	0.66			100.00%		
rococoC12-100000	1.55			100.00%		
rococoC12-111100	0.53			100.00%		
mean over all instances				54.68%	29.33%	13.14%
mean over instances performing the stage				54.68%	57.02%	93.31%

Table 5: Time spent in each stage

a consequence, the quality of x^* and $\tilde{x}$ tends to deteriorate rapidly with the number of iterations and of restarts performed. This explains why the same behavior is much less pronounced in the binary case studied in [12], where driving the pair $(x^*, \tilde{x})$ towards feasibility turns out to be much easier than in the general-integer case and requires a considerably smaller number of iterations and of restarts.

In this section we investigate three simple `FP` strategies aimed at improving the quality of the solutions found by the method.

The first strategy is based on the idea of adding an artificial upper bound constraint $c^T x \leq UB$ to the LP solved at step 6, where UB is updated dynamically each time an improved feasible solution is found. To be more specific, right after step 1 we initialize $z^*_{LP} = c^T x^*$ (= LP relaxation value) and $UB = +\infty$. Each time an improved feasible MIP solution x^* of value $z^H = c^T x^*$ is found at step 6, we update $UB = \alpha z^*_{LP} + (1 - \alpha) z^H$ for a certain $\alpha \in (0, 1)$, and continue the while-do loop. We observed that, due to the additional constraint $c^T x \leq UB$, it is often the case that the integer components of $\tilde{x}$ computed at step 9 define a feasible point of the *original* system $Ax \geq b$, but not of the current one. In order to improve the chances of updating the incumbent solution, we therefore apply (right after step 9) a simple post-processing of $\tilde{x}$ consisting in solving the LP $\min\{c^T x : Ax \geq b,\ x_j = \tilde{x}_j\ \forall j \in \mathcal{I}\}$ and comparing the corresponding solution $\overline{x}$ (if any) with the incumbent one–solution $\overline{x}$ being guaranteed to be feasible for the original problem, as all the integer-constrained variables have been fixed at their corresponding value in $\tilde{x}$.

In the other two strategies, we stop `FP` as soon as it finds a feasible solution, and pass this solution either to a Local Branching heuristic [11], or to a MIP solver using RINS strategy [9].

Table 6 compares the quality of the best solution returned by `ILOG Cplex` with that of the solution found (within a 3600-second time limit) by `FP-cplex` and then improved by means of one of the three strategies above. In the table, the first four columns report the instance name (*name*), the value of the LP relaxation (*LP relax*) and of best feasible solutions found within the 3600-second time limit by `ILOG Cplex` (*Cplex*) with two different settings of its `MIP emphasis` parameter, namely "`emp=1`" for *integer feasibility* and "`emp=4`" for *hidden feasible solutions* (i.e., RINS heuristic). As to `FP-cplex` with the artificial upper bound, it was run with the same settings described earlier, by requiring a 20% (respectively, 30%) improvement at each main iteration (i.e., with $\alpha \in \{0.2, 0.3\}$); see columns *FP-XX%*. Tests with $\alpha = 0.1$ and $\alpha = 0.4$ led to slightly worse solutions (with an average quality ratio of about 1.26) and are not shown in the table.

name	(Ref) Cplex emp=1	Cplex emp=4	FP-20%	FP-30%	FP-lb	FP-rins
arki001	7.581E+06	1.0000	1.0007	1.0006	0.9999 *	1.0000
atlanta-ip	1.000E+02	N/A	0.9600	0.9800	0.9600	0.9500 *
gesa2	2.578E+07 *	1.0000 *	1.0004	1.0004	1.0000	1.0000 *
gesa2-o	2.578E+07 *	1.0000 *	1.0011	1.0013	1.0000 *	1.0000 *
manna81	-1.316E+04 *	1.0000 *	1.0000 *	1.0005	1.0000 *	1.0000 *
msc98-ip	2.250E+07	N/A	0.8993	0.8984 *	0.9529	0.9699
mzzv11	-2.172E+04 *	1.0000 *	1.2209	1.0950	1.1144	1.0018
mzzv42z	-2.054E+04 *	1.0000	1.0235	1.0188	1.0118	1.0000
noswot	-4.100E+01 *	1.0000 *	1.0000 *	1.0000 *	1.0000 *	1.0000 *
roll3000	1.343E+04	0.9596 *	1.0708	1.1188	0.9800	0.9657
rout	1.078E+03	1.0000	1.0151	1.0061	1.0000 *	1.0000 *
timtab1	7.927E+05	0.9647 *	1.3123	1.1528	1.0034	1.6313
timtab2	1.232E+06	0.8990 *	1.3245	1.1675	1.0224	0.9648
neos10	-1.135E+03 *	1.0000 *	4.4862	2.9481	1.0000 *	1.0000 *
neos16	4.510E+02	N/A	1.0067	1.0067	1.0067	0.9978 *
neos20	-4.340E+02 *	1.0000 *	4.1731	4.1731	1.0383	1.0000
neos7	7.219E+05 *	1.0000	1.0582	1.0028	1.0000	1.0000
neos8	-3.719E+03	1.0000	1.0005	3.1570	1.0000 *	1.0000 *
ic97_potential	3.961E+03	0.9965 *	1.0106	1.0155	0.9970	0.9965 *
ic97_tension	3.942E+03 *	1.0003	1.0018	1.0032	1.0000 *	1.0000 *
icir97_potential	6.410E+03	0.9964	1.0264	1.0434	1.0034	0.9945 *
icir97_tension	6.418E+03	0.9949	0.9948	0.9996	0.9956	0.9938 *
rococoB10-011000	1.951E+04	0.9967 *	1.1947	1.0873	1.0365	1.0593
rococoB10-011001	2.131E+04 *	1.0501	1.2451	1.2443	1.0037	1.1349
rococoB11-010000	3.348E+04	0.9901 *	1.1968	1.0978	1.0138	1.1833
rococoB11-110001	4.947E+04	0.9738 *	1.5647	1.2136	1.0573	1.2941
rococoB12-111111	4.623E+04	0.8589 *	2.0923	2.0923	1.0035	1.0372
rococoC10-001000	1.146E+04 *	1.0004	1.1645	1.0883	1.0013	1.0013
rococoC10-100001	1.943E+04	0.9336 *	1.5803	1.7790	0.9377	1.0649
rococoC11-010100	2.163E+04 *	1.0189	1.1680	1.0668	1.0389	1.3361
rococoC11-011100	2.192E+04	0.9561 *	1.1306	1.2290	1.0410	1.1887
rococoC12-100000	3.753E+04 *	1.0177	1.6447	1.4960	1.0742	1.0775
rococoC12-111100	4.097E+04	0.9138 *	1.0858	1.0176	0.9794	0.9448
Geometric means		0.9833 (+)	1.2352	1.2292	1.0078	1.0469

(+) not counting the 3 cases of failure

Table 6: Solution quality with respect to ILOG Cplex (emp=1); 3600-second time limit

Column *FP-lb* refers to the Local Branching implementation available in `ILOG Cplex` 9.1 by activating its *local branching* flag, whereas column *FP-rins* refers to `ILOG Cplex` with MIP emphasis 4 (that activates the internal RINS improvement heuristics). For both *FP-lb* and *FP-rins*, the incumbent solution is initialized, via an MST file, by taking the first `FP-cplex` solution.

For all strategies, the table gives the solution ratio with respect to the best solution found by `ILOG Cplex` (emp=1). Ratios were computed as the value of the best solution found by the various methods over the value of the solution found by `ILOG Cplex` (emp=1); if the values were negative, the problem was viewed as a maximization one and the ratio was inverted, hence a ratio smaller than 1.0 always indicates an improvement over `ILOG Cplex`. In the last line of the table, the average ratio (geometric mean) is reported; the average does not take into account the instances where `FP` succeeded in finding a solution, while `ILOG Cplex` did not. For each instance, we marked with an asterisk the method that produced the best feasible solution.

According to the table, all the `FP` methods are able to improve significantly the quality of their incumbent solution. The most effective `FP` strategies seem to be *FP-lb* and *FP-rins*, that produced the best solutions in 8 and 13 cases, respectively.

`ILOG Cplex` (emp=1) ranked first 14 times. As to `ILOG Cplex` (emp=4), it produced the best solution in 18 cases but failed in 3 cases to find any solution within the 3600-second time limit. Moreover, pure `ILOG Cplex` methods seem to be particularly suited for exploiting the structure of *rococo** instances–if these 11 instances were removed from the testbed, *FP-rins* would have ranked first 13 times, thus outperforming both `ILOG Cplex` (emp=1, that ranked first 10 times) and `ILOG Cplex` (emp=4, first 11 times but with 3 failures).

Among the compared *FP-XX%* methods, the one requiring 30% improvement at each main iteration is the more effective one, though its performance is still inferior to the one of the LB/RINS local search methods.

Finally, Figures 3, 4 and 5 plot the value of the best feasible solution over time, for the three instances atlanta-ip, msc98-ip and icir97_tension.

5 Conclusions

In this paper we addressed the problem of finding a feasible solution of a given MIP model, which is a very important ($\mathcal{NP}$-complete) problem that can be extremely hard in practice.

We elaborated the Feasibility Pump (`FP`) heuristic presented in [12], and

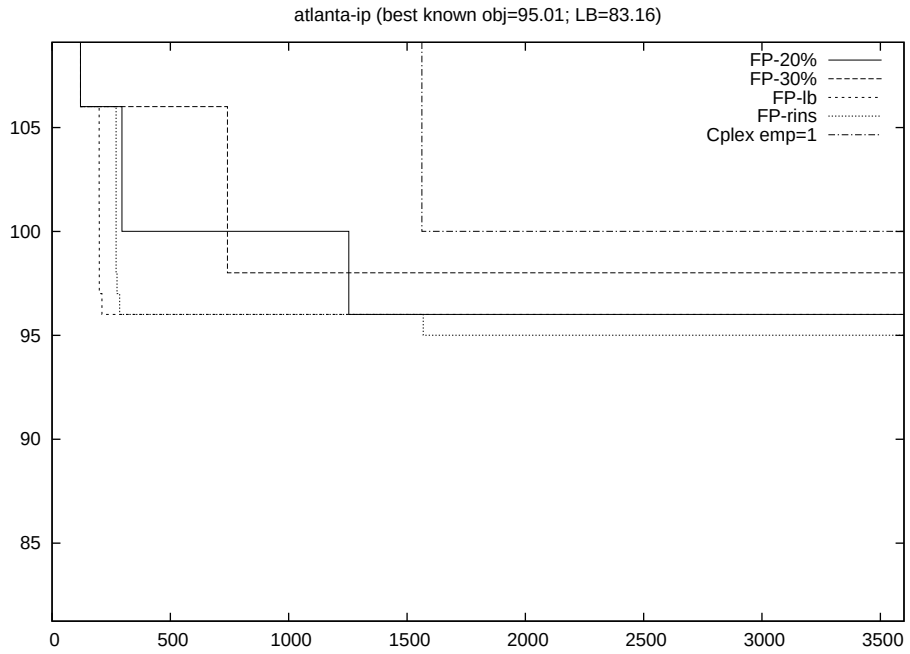

Figure 3: Incumbent solution over time (instance atlanta-ip)

extended it in two main directions, namely (i) handling as effectively as possible MIP problems with both binary and general-integer variables, and (ii) exploiting the FP information to drive an effective enumeration phase.

We presented extensive computational results on large sets of test instances from the literature, showing the effectiveness of our improved FP scheme for finding feasible solutions to hard MIPs with general-integer variables.

As to the solution quality, it appears to be rather poor when the very first feasible solution is found, but it can be improved considerably by integrating FP with improvement tools such as Local Branching or RINS.

Future directions of work include extending the FP idea by using a nonlinear (quadratic) distance function, to be applied to linear and (even more interestingly) to nonlinear problems with integer variables. Also interesting is the incorporation of the original objective function (through an adaptive scaling multiplier) in the definition of the FP distance function; interesting results in this directions have been recently reported by Achterberg and Berthold [2] and used in the non-commercial MIP solver SCIP [1].

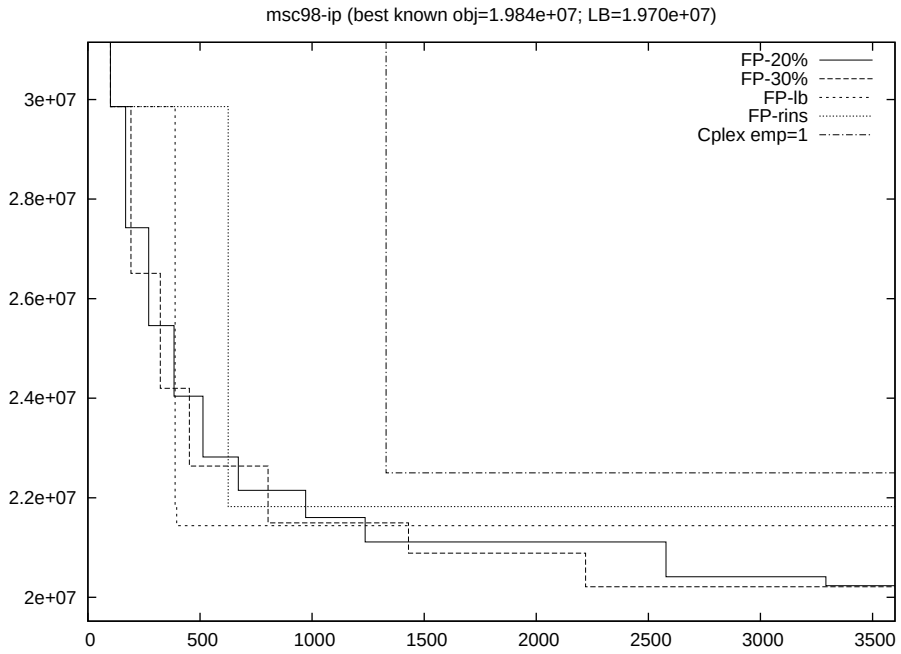

Figure 4: Incumbent solution over time (instance msc98-ip)

Finally, a topic to be investigated is the integration of `FP` within an overall enumerative solution scheme. In this context, the `FP` heuristic can of course be applied at the root node, so as to hopefully initialize the incumbent solution. But one can also think of running `FP` (possibly without its time-consuming stage 3) from the LP relaxation of different nodes in the branch-and-cut tree, thus increasing the chances of finding improved feasible solutions.

Acknowledgements

Work supported by MIUR, Italy, and by the EU project ADONET. Thanks are due to two anonymous referees for their helpful comments.

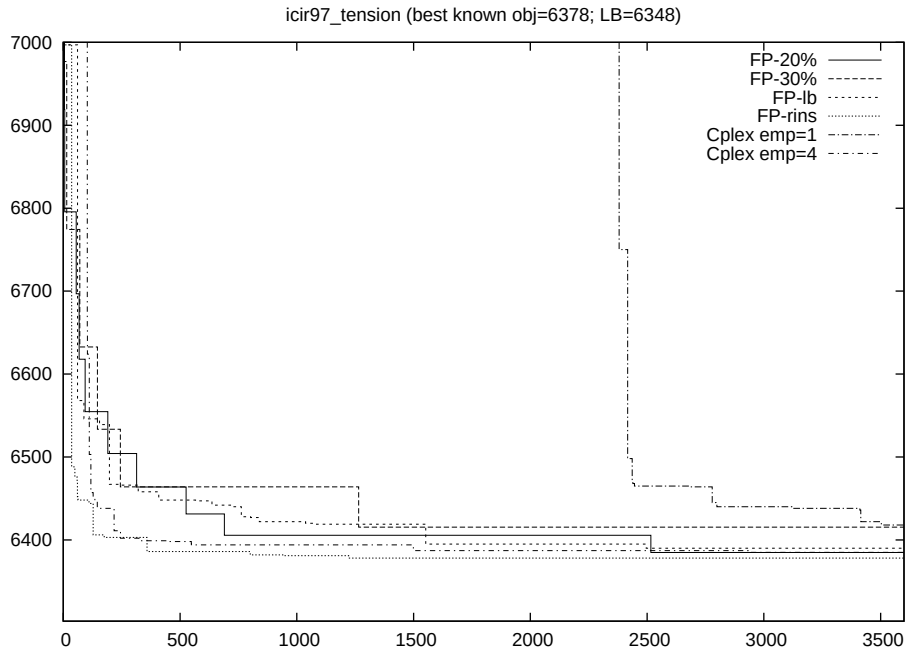

Figure 5: Incumbent solution over time (instance icir97_tension)

References

[1] T. Achterberg. SCIP - a framework to integrate Constraint and Mixed Integer Programming, Technical Report 04-19, Zuse Institute Berlin, 2004 (available at `http://www.zib.de/Publications/abstracts/ZR-04-19/`).

[2] T. Achterberg, T. Berthold, Improving the feasibility pump, Technical Report Zuse Institute Berlin, September 2005.

[3] T. Achterberg, T. Koch, A. Martin. MIPLIB 2003. Technical Report 05-28, Zuse Institute Berlin, 2005 (available at `http://www.zib.de/PaperWeb/abstracts/ZR-05-28/`).

[4] 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.

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

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

[7] J.W. Chinneck, J. Patel. Faster MIP Solutions Through Better Variable Ordering, ISMP 2003, Copenhagen, August 2003.

[8] Dash Xpress-Optimizer 16.01.05: Getting Started and Reference Manual, Dash Optimization Ltd, `http://www.dashoptimization.com/`, 2004.

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

[10] DIMACS Second Challenge. `http://mat.gsia.cmu.edu/challenge.html`.

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

[12] M. Fischetti, F. Glover, A. Lodi, The Feasibility Pump. *Mathematical Programming* 104, 91–104, 2005.

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

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

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

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

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

[18] ILOG Cplex 9.1: User's Manual and Reference Manuals, ILOG, S.A., `http://www.ilog.com/`, 2004.

[19] 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.

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

[21] H. D. Mittelmann. Benchmarks for Optimization Software: Testcases. `http://plato.asu.edu/topics/testcases.html`.

[22] 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.

[23] L. Peeters. Cyclic Railway Timetable Optimization. ERIM PhD Series, Erasmus University Rotterdam, June, 2003.