Highlights

A Constraints-Aware Reweighted Feasibility Pump Approach

Antonio Frangioni,Stefania Pan,Emiliano Traversi,Roberto Wolfler Calvo

- We extend the Reweighted Feasibility Pump (RFP) approach for integer feasibility problems
- We define a novel function to be minimised (via the Frank-Wolfe approach) that includes barrier terms together with penalty ones
- Results on hard pure binary feasibility problems show that the modification leads to a more effective and robust method.

A Constraints-Aware Reweighted Feasibility Pump Approach

Antonio Frangioni[a], Stefania Pan[b], Emiliano Traversi[c] and Roberto Wolfler Calvo[c,d,*]

[a]Dipartimento di Informatica, Università di Pisa, Italy
[b]Artelys, 81 Rue Saint-Lazare, 75009 Paris, France
[c]LIPN, Université Sorbonne Paris Nord, France
[d]DIM, Università di Cagliari, Italy

ARTICLE INFO

Keywords:
Feasibility Pump
Binary Feasibility Problem
Frank-Wolfe approach
Barrier function

ABSTRACT

We extend the Reweighted Feasibility Pump (RFP) approach of [9, 8], which is based on the interpretation of the (main step of the) Feasibility Pump as the Frank-Wolfe method applied to an appropriate penalty function that drives the search towards integer solutions. We modify the function that is minimized by incorporating barrier terms that provide information about how much changing a variable is possible due to it being involved in (almost) active constraints. The corresponding Constraints-Aware RWP is experimentally tested on two sets of hard pure binary feasibility problems, and the results show that, all other things being equal, the modification leads to a more effective and robust method w.r.t. the original RFP one.

1. Introduction

In this paper we address the decision problem

$$\exists\, x \in \mathcal{P} = \{\, x \in \{\,0\,,\,1\,\}^n \;:\; Ax \geq b \,\} \;\; ? \tag{1}$$

The set $\mathcal{P}$ is an arbitrary subset of the n-dimensional binary hypercube defined by an arbitrary rational polyhedron described by the $m \times n$ matrix A and the m-vector b. Although not fully as general as the problem of finding feasible integer solutions for Mixed-Integer Linear Programs (MILP), whose huge relevance in practice is too well-known to require discussion, (1) is generally regarded to capture most of the latter's intrinsic hardness. While the basic idea of our approach can be applied to the more general case, it is more easily described in the setting of (1) and therefore we restrict ourselves to that, leaving the discussion to possible extensions for future research.

It is well-known that (1) is, in general, $\mathcal{NP}$-hard in the strong sense. Efficient and effective general approaches for the problem seem therefore not possible in general; yet, the need to develop at least heuristics with good practical performances in many cases is pressing, e.g. for the development of general-purpose MILP solvers. It is not surprising, then, that a huge amount of research has addressed this topic. Most approaches focus on the obvious relaxed set

$$\underline{\mathcal{P}} = \{\, x \in [\,0\,,\,1\,]^n \;:\; Ax \geq b \,\}$$

for which the decision problem is polynomial, trying to leverage on the very well-developed mechanisms——Linear Programming (LP) techniques——available for the latter to (heuristically) solve (1). Roughly speaking, the basic idea of most of these approaches more or less directly revolves around rounding fractional solutions obtained by LP techniques applied to $\underline{\mathcal{P}}$ or its modifications. Yet, doing this in a way that is effective for "all" possible choices of the constraint set is well-known to be highly nontrivial.

A turning point has been reached about a decade ago with the proposal of the *Feasibility Pump* (FP) approach [10], which matches a simple and clean basic idea with surprisingly good performances in very many practical applications. The original approach has therefore been quickly embraced by the community, being extended to general MILPs [2] and improved in several ways, e.g. w.r.t. the quality of the produced solutions when an objective is present (cf. [1, 4] among the others). The clear proof of its practical relevance is that it is now available in all major general-purpose

*Corresponding author
frangio@di.unipi.it (A. Frangioni); traversi@lipn.fr (E. Traversi); wolfler@lipn.fr (R. Wolfler Calvo)
www.di.unipi.it/~frangio (A. Frangioni); www.lipn.fr/~traversi (E. Traversi); www.lipn.fr/~wolfler (R. Wolfler Calvo)
ORCID(s): 0000-0002-5704-3170 (A. Frangioni); 0000-0003-4673-3982 (E. Traversi); 0000-0002-5459-5797 (R. Wolfler Calvo)

MILP solvers, together with others that can be viewed as being related to it [7], and it has been extended to non-linear [5] (and even non-convex [6]) Mixed-Integer programs.

An interesting trait of FP is that, despite its apparent simplicity, it can be re-interpreted in a number of different ways that shed light on its properties, and help in getting a sense of its (somewhat surprising) practical efficiency. Indeed, FP can be seen to be related e.g. to the successive projection method [6], the proximal point algorithm [3], and the Frank-Wolfe approach [9, 8], with each of these interpretations suggesting different ways to improve its performances. Our approaches hinges on, and extends, the last of these re-interpretations and modifications of FP, which we will now quickly summarise.

A succinct description of the FP scheme is provided by Algorithm 1:

Algorithm 1 The Feasibility Pump algorithm (basic scheme):

1: find some $x^* \in \underline{\mathcal{P}}$; $\tilde{x} = [\, x^* \,]$; // rounding of x^*
2: **for** ($i = 0$; (x^* fractional) && ($i < maxIter$) ; $++i$) **do**
3: $x^* = argmin\{\ \Delta(\, x\, ;\, \tilde{x}\,)\ :\ x^* \in \underline{\mathcal{P}}\ \}$;
4: **if** $\Delta(x^*, \tilde{x}) > 0$ **then**
5: **if** $[\, x^* \,] \neq \tilde{x}$ **then**
6: $\tilde{x} = [\, x^* \,]$;
7: **else**
8: perform a random perturbation of $\tilde{x}$;
9: if cycling is detected, perform a random restart operation;
10: **end if**
11: **end if**
12: **end for**

The details of the random perturbation, cycling detection and the restart phase are not relevant for our discussion and we omit them, referring the interested reader to [2] and other papers dealing with the actual implementation of the scheme in general-purpose solvers. The crucial object for our discussion is the objective function $\Delta(\, x\, ;\, \tilde{x}\,)$, parametric in the current (integer but unfeasible) solution $\tilde{x}$, that it is used to solve the LP at step 3. This function is chosen to try to force the new (feasible but likely not-integer) solution x^* to be "close" to the previous integer one $\tilde{x}$, in the hope that the rounding of x^* will be closer to the feasible region, and ultimately will belong to it (feasibility is iteratively "pumped" from x^* to $\tilde{x}$). The standard choice for Δ is

$$\Delta(\, x\, ;\, \tilde{x}\,) = \sum_{i\,:\,\tilde{x}_i = 0} x_i - \sum_{i\,:\,\tilde{x}_i = 1} x_i$$

which, intuitively, "pushes x^*_i towards the same binary value of $\tilde{x}_i$". The observation of [9, 8] is that, since $\tilde{x}$ is a (very simple) function of the previous continuous solution x^*, Δ can be seen as being directly parametric in the latter:

$$\Delta(\, x\, ;\, x^*\,) = \sum_{i\,:\,x^*_i < 1/2} x_i - \sum_{i\,:\,x^*_i \geq 1/2} x_i \quad .$$

This is again intuitively clear: already integer components of the old x^*_i "push" the corresponding new component to remain at the same (0 or 1) value, while fractional components are "pushed" towards the closest of the two integer values in the new iterate. This leads to the interpretation of the basic step of the FP (the solution of the LP) as one step of the standard Frank-Wolfe approach applied to the solution of

$$\min \left\{\ \psi(x) = \sum_{i=1}^{n} \phi(x_i)\ :\ x^* \in \underline{\mathcal{P}}\ \right\} \tag{2}$$

where $\phi : \mathbb{R} \to \mathbb{R}$ is the concave nondifferentiable function given by

$$\phi(t) = \min\{\, t\,,\ 1 - t\,\} \quad . \tag{3}$$

Indeed, it is easy to see that the i-th component of the (super)gradient of $\psi(x)$ is precisely 1 if $x_i < 1/2$ and -1 if $x_i \geq 1/2$. One is therefore minimizing the (concave) *penalty function* $\psi(x)$, which attains minimal value 0 in all and only the integer points of $\underline{\mathcal{P}}$ (if any), in the hope that doing so will guide the search towards one such point. Clearly, (3) is perhaps the simplest, but certainly not the only, penalty function which may drive the search towards integral points.

In [9] several variants are tested, all having the same separable form as ψ for different basic functions ϕ starting from the obvious $\phi(t) = t(1-t)$. This is used to develop the *Reweighted Feasibility Pump* (RFP) heuristic where

$$\Delta(\, x\,;\, x^*\,) = \sum_{i:\, x_i^* < 1/2} w(x_i^*) x_i - \sum_{i:\, x_i^* \geq 1/2} w(x_i^*) x_i \tag{4}$$

and the *positive* weights $w(x_i^*)$ depend on the (derivative of the) chosen ϕ, i.e.,

$$w(x) = |\, \phi'(x)\,| \quad . \tag{5}$$

This exactly reproduces the standard FP with (3) but is different with other ϕ, and it is experimentally found that the logistic function

$$\phi(t) = \min\left\{ \frac{1}{1+e^{-t}}, \frac{1}{1+e^{t-1}} \right\} \tag{6}$$

appears to provide the best practical results [8].

2. Constraints-Aware Reweighted Feasiblity Pump

Building on the idea of [9], we examine possible ways to modify the objective function $\psi(x)$. Our intuition is that while all the previously proposed penalty function are arguably appropriate to "push" the components towards what *appears* to be the most promising of the two possible integer values, they also fail to take into account the role of the constraints. An obvious example of why this might be an issue is given by the simple polyhedron defined by $1/3 \leq x_1 \leq x_2$ with initial point $x_1^* = x_2^* = 1/3$, and hence $\tilde{x}_1 = \tilde{x}_2 = 0$. This leads any (R)FP to choose positive coefficients for x_1 and x_2 and therefore not changing the fractional point, leading to a restart; choosing the opposite sign would instead immediately lead to the integer feasible point $x_1 = x_2 = 1$. The crux here is the constraint $1/3 \leq x_1$, which is active in x^*: no amount of "push" from the objective could ever lead x_1 to reach the (apparently, closest) integer value 0. While this example is obviously over-simplifying matters for illustrative purposes, it confirms the intuition that constraints have a fundamental role to play. Among them, we argue that *active* constraints (satisfied as equalities) are likely to be the most relevant ones, since they restrict independent changes in the value of the involved variables (at least in one direction, and in absence of coordinated changes of the others). We therefore consider the following conceptual modification of (2):

$$\min\{\, \Phi(\, x\,,\, s\,) \;:\; s = Ax - b\,,\, s \geq 0\,,\, x \in [\,0\,,\,1\,]^n \,\} \quad . \tag{7}$$

The idea is that the (nonconvex) objective function guiding the search for the integer solution should not only depend on the values of x, but also on the *slacks* of the constraints. This allows it to incorporate information about variables being more or less involved into constraints that are active, or close to being so, which may help the process. As a first concrete implementation we consider the restricted, but still possibly large, class of functions of the form

$$\Phi(\, x\,,\, s\,) = \sum_{i=1}^{n} \left(\psi(x_i) \frac{1}{|C(i)|} \sum_{j \in C(i)} \beta(s_j) \right) \tag{8}$$

where $C(i)$ is the set of constraints in which variable x_i appears, ψ is a "standard" penalty function whose role is to push x towards either 0 or 1 (think $\psi(x) = x(1-x)$ or any other ones from [9]), while β is a barrier-type function meant to grow when slacks are (close to) 0, i.e., constraints are (almost) active. An obvious choice is

$$\beta(s) = (\, s + \epsilon\,)^{-1} \quad , \tag{9}$$

with a "small" $\epsilon > 0$ due to ensure that the function is still well-defined on the boundary of the feasible region (unlike, say, the logarithmic barrier function at the heart of most interior-point methods). Of course these choices are largely arbitrary, and different ones could be considered; in particular, there is nothing preventing from using different ψ and/or β for different variables, or even more involved form of Φ, but it is only reasonable to start experimenting with the simplest and most uniform choices. It is also appropriate to remark that one could treat the bound constraints $0 \leq x_i \leq 1$ like "ordinary" ones and add the corresponding slack variables, but our intuition (and our preliminary

experiments) discourage this choice because of the obviously different role of these constraints, as well as to the fact that they are in some sense already taken care of by the ψ component.

With these definitions, we consider the simple variant of the RFP using (4) where the weights are defined by the modified form of (5) given by

$$w_i = \left| \frac{\partial}{\partial x_i} \Phi(x^* , Ax^* - b) \right| \quad . \tag{10}$$

The first goal of the objective function using (10) is to "push" the solution toward the closest integer point, which is the same purpose of the original version based on (5). However, which variable should be "pushed harder" depends on two factors: the proximity to the rounded point, as in the original RFP, and the additional term (9) which measures the distance from the constraints through the slack variables. To see this, let $A(i)$ be the set of *active* constraints (w.r.t., x^*) where the variable x_i appears: for each $j \in A(i)$ one has $s_j = 0$, i.e., $\beta(s_j) = 1/\epsilon = M$. Computing the derivative of (8) (with (9)) and some tedious algebra shows that as $\epsilon \to 0$

$$(\partial/\partial x_i)\Phi(x^* , Ax^* - b) \approx \sum_{h=1}^{n} \sum_{j \in A(h) \cap A(i)} A_{ji} \psi(x^*_h) \quad .$$

Therefore, while with the original RFP w_i only depends on the value of the single corresponding variable x_i, in the new approach the value that is used to set the "pushing priorities" is much more nuanced: it depends on all variables sharing active constraints with x_i, taking into account how far from integrality each of these is and also the corresponding coefficient in the constraint. This confirms that (10) uses more information on the current x^* than RFP does to guide its main decisions. We remark that, due to our choice (8), the partial derivative has a closed-formula expression depending on ψ and β, which allowed us to easily code all the different variants of ψ of [9]. Our results confirmed that (6) is a competitive choice for our approach, too, and therefore we only discuss it. We also tested the simpler (and cheaper) form of "barrier" function $\beta(s)$ that evaluates to 1 if $s = 0$, and to 0 otherwise; that is, β just counts if the given constraint (containing variable x_i) is active. Besides not being differentiable (but then again, neither the original (3) is), this gave clearly worse results than (9), indicating that not only the active/inactive status, but also a measure of how close a constraint is of being active, is useful to the approach; a result that did not seem surprising to us.

We call Algorithm 1 in which $\Delta(x ; x^*)$ is implemented by means of (5) and (10) the *Constraints-Aware Reweighted Feasibility Pump* (CARFP). In the following we will discuss some computational experiments aimed at assessing whether the idea may have practical usefulness.

3. Computational experiments and conclusions

To make our analysis fair we implemented our approach starting from the very same code used in [9], that the authors kindly shared for us; we therefore do not need to discuss in details the (nontrivial) aspects of randomization and restarts, since they will always be the same for all the variants tested. We remark that that code was, in turn, adapted by the one developed for [2, 8, 10], and therefore we can reasonably assume the choices made for these aspects to have been well thought-of. Anyway, our aim is to evaluate whether CARFP could, without any other change, improve on the performance of the RFP. In order to do that, we kept the same penalty function ψ (in particular (6), which was found to be the best one for RFP), so that the only difference is related to the presence, or not, of the barrier terms.

To test the effectiveness of our approach we concentrated on the pure binary feasibility problem. Although FP and RFP have been successfully extended to consider an objective, and this could be done for CARFP exactly along the same lines, we believe that the results have a simpler and clearer interpretation if one just has to measure the ability of the algorithm to find feasible solutions. For this to be interesting, however, these should arguably be instances where the task is "hard". We therefore devised two specific test sets:

- `3sat`: this set consists of instances of the standard MILP formulation of the Boolean Satisfiability Problem (SAT), generated using the generator available at `https://toughsat.appspot.com`. To obtain the benchmark, we started by generating random 3-SAT instances with clause-to-variables ratio in {3.8, 3.9, 4.0, 4.1, 4.2} and a number of variables between 41 and 200. To restrict our analysis to the most challenging instances, we kept only those for which `CPLEX` could not identify a feasible solution in less than 200 seconds; on the other hand, we discarded the infeasible ones. We ended up collecting 56 instances; each instance has a name of the form `3sat_x_y`, where `x` is the number of variables and `y` is the number of clauses of the instance.

- `no-good`: this set of instances consists of hypercubes where the majority of the vertices are cut by the so-called *no-good cuts*: given a vertex $\tilde{x}$ of the n-dimensional hypercube, the corresponding no-good cut is simply the linear inequality $\sum_{i:\tilde{x}_i=0} x_i + \sum_{i:\tilde{x}_i=1}(1-x_i) \geq \frac{1}{2}$. Each no-good cut makes it infeasible one and only one of the vertices ($\tilde{x}$), and is not active in any. Each instance has a name of the form `ng_x_y`, where `x` is the number of variables (n) and `y` is the number of constraints. Since the number of feasible solutions is $2^x - y$ and we wanted "hard" instances we chose y close to 2^x, which meant that the size of the instances grows exponentially in n, preventing us to test any $n > 12$. However, we believe that the 42 generated instances are still interesting in that they should become more and more challenging as y is closer to 2^x, which can allow to spot interesting trends.

We tested the two versions of Algorithm 1 corresponding to RFP and CARFP. In both cases the initial feasible point (step 1 of Algorithm 1) is found with the Cplex barrier optimizer, without crossover, which yields a point "well in the interior" of the continuous feasible region; however, we also experimented with using the Chebishev centre, which can be computed by any LP algorithm, with no significant differences. The parameter *maxIter* was set to 300000. The random restart of line 9 of Algorithm 1 can be performed several time until a value of $\tilde{x}$ that has never been generated before is obtained, for this reason the maximum number of restarts is set to 10×*maxIter*. For each instance we ran the algorithm 10 times, with different random seeds, so as to have statistically significant results.

In Table 1 and Table 2 we report the results on both classes of instances. Both tables have identical structure. The first column shows the name of the instance; the second and the third columns show the number of runs (out of 10) where the algorithms found a feasible solution; the forth and the fifth columns show the average (among all runs) iterations needed to find a feasible solution, and the sixth and the seventh columns show the average number of restarts.

CARFP finds a feasible solution significantly more often for both sets of instances. In Table 1 we find that CARFP finds a feasible solution 547 times over 560 runs (97% of the total), while RFP only 426 times over 560 runs (76%). Moreover, for 6 instances RFP is unable to find a solution, while CARFP always finds a feasible solution with at least 6 different seeds. A similar picture can be observed in Table 2; furthermore, there we can notice that CARPF invariably outperforms RFP, often significantly so, on the "hardest" instances where $y = 2^x - 1$.

Another relevant observation is that restarts are one order of magnitude more frequent for RFP than for CARFP. This shows that while for RFP the random components constitute a fundamental aspect of the algorithm, CARFP is able to perform significantly more iterations before running into a dead-end and requiring a restart to continue the search. A case in point is the instance `3sat_3_68_272`: on average RFP requires 500 iterations to find a feasible solution and 277 restarts, which means that about on every other iteration the algorithm is stuck and the only alternative is a random restart. In the same instance, while CARFP requires significantly more iterations (1646), it restarts only once in about 30 iterations.

To conclude, the results show that, for the instances considered, CARFP is more effective and more resilient than the standard RFP. This seems to indicate that, indeed, explicitly considering the constraints of the problem in the Frank-Wolfe objective function provides a better guide towards a feasible solution, thus requiring significantly fewer restarts. This indicates that CARFP could be considered for the implementation of more effective heuristics within general-purpose MILP solvers.

References

[1] T. Achterberg and T. Berthold. Improving the feasibility pump. *Discrete Optimization*, 4:77–86, 2007.

[2] L. Bertacco, M. Fischetti, and A. Lodi. A feasibility pump heuristic for general mixed-integer problems. *Discrete Optimization*, 4(1):63–76, 2007.

[3] N.L. Boland, A.C. Eberhard, F.G. Engineer, and A. Tsoukalas. A new approach to the Feasibility Pump in Mixed Integer Programming. *SIAM Journal on Optimization*, 22:831—-861, 2013.

[4] N.L. Boland, A.C. Eberhard, F.G. Engineer, M. Fischetti, M.W.P. Savelsbergh, and A. Tsoukalas. Boosting the feasibility pump. *Mathematical Programming Computation*, 6:55—-279, 2014.

[5] P. Bonami, G. Cornuéjols, A. Lodi, and F. Margot. A feasibility pump for mixed integer nonlinear programs. *Mathematical Programming*, 119:331–352, 2009.

[6] C. d'Ambrosio, A. Frangioni, L. Liberti, and A. Lodi. A storm of feasibility pumps for nonconvex minlp. *Mathematical Programming*, 36(2): 375–402, 2012.

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

[8] M. De Santis, S. Lucidi, and F. Rinaldi. A new class of functions for measuring solution integrality in the Feasibility Pump approach. *SIAM Journal on Optimization*, 23:1575—-1606, 2013.

instance	Found		Avg. iterations		Avg. restarts	
	CARFP	RFP	CARFP	RFP	CARFP	RFP
3sat_3_41_172	10	10	1175.0	1024.4	58.9	349.8
3sat_3_64_243	10	10	5992.9	4672.4	211.3	5194.6
3sat_3_68_272	10	10	1645.8	500.0	60.5	276.8
3sat_3_69_289	10	10	1263.4	239.8	44.5	211.6
3sat_3_77_323	10	10	4007.8	1824.6	183.0	368.9
3sat_3_81_332	10	10	989.9	2040.9	42.7	1543.5
3sat_3_83_332	10	10	1127.4	4341.8	48.3	2923.5
3sat_3_84_352	10	10	434.1	124.8	18.7	9.7
3sat_3_86_344	10	4	12522.5	204409.1	523.3	188370.3
3sat_3_90_378	10	10	3485.5	635.3	170.7	172.6
3sat_3_92_368	10	10	8266.0	188.4	385.8	24.8
3sat_3_94_366	10	10	4719.2	522.7	208.8	348.1
3sat_3_95_380	10	10	14906.7	3029.9	713.1	1094.6
3sat_3_95_399	10	8	15898.8	151083.7	739.4	94824.6
3sat_3_98_392	10	10	14133.4	12327.8	666.9	8692.4
3sat_3_108_453	10	11	1813.3	3095.9	93.2	777.7
3sat_3_111_455	10	0	6795.5	300000.0	370.8	180357.1
3sat_3_112_436	10	7	2327.7	161152.0	131.0	155355.5
3sat_3_115_437	10	10	1709.3	3966.7	90.8	1555.8
3sat_3_115_471	10	10	520.1	163.8	29.7	7.2
3sat_3_115_483	10	10	2411.0	2645.3	143.1	1315.0
3sat_3_117_491	10	10	4888.0	747.6	273.9	81.1
3sat_3_123_492	10	10	5560.9	639.7	330.1	85.2
3sat_3_128_524	10	10	22072.5	31185.4	1325.0	17706.4
3sat_3_129_516	7	0	148905.7	300000.0	8966.8	243678.1
3sat_3_129_541	10	10	659.7	854.6	39.8	113.0
3sat_3_131_550	10	10	3937.5	331.2	242.0	56.3
3sat_3_137_561	10	10	1917.9	67.3	110.8	3.5
3sat_3_138_565	6	7	214401.6	177245.3	13260.2	155837.5
3sat_3_140_588	10	0	28985.3	300000.0	1703.5	203897.6
3sat_3_142_553	10	10	10733.2	70327.2	656.6	36171.0
3sat_3_144_576	10	0	20968.8	300000.0	1316.3	186162.7
3sat_3_146_613	8	0	152929.5	300000.0	10568.7	268188.6
3sat_3_147_588	10	1	74465.1	285583.8	4700.3	161194.6
3sat_3_148_592	10	8	17745.6	88154.5	1137.2	47822.6
3sat_3_152_608	10	10	1578.1	6236.1	101.7	1229.3
3sat_3_153_612	10	10	1802.0	1023.7	114.6	119.9
3sat_3_154_631	10	8	4887.7	65480.2	315.8	49854.5
3sat_3_156_624	10	10	5033.4	8724.0	369.9	1339.7
3sat_3_159_636	10	9	53796.0	35122.1	3744.2	20869.9
3sat_3_164_656	10	6	2071.8	172537.6	144.4	156724.0
3sat_3_166_664	10	7	62880.4	138286.6	4606.9	36590.1
3sat_3_169_692	10	0	139427.5	300000.0	9821.1	140503.2
3sat_3_172_688	10	2	18254.6	247304.6	1443.4	236769.4
3sat_3_172_705	10	10	12913.2	16651.3	807.0	3692.9
3sat_3_173_692	10	10	4347.7	13730.6	306.9	2712.9
3sat_3_174_713	6	4	196530.1	219645.0	12756.5	168713.5
3sat_3_174_730	10	10	2423.3	4973.2	189.0	1293.8
3sat_3_176_721	10	10	8212.1	7536.5	561.4	2159.4
3sat_3_179_733	10	10	2652.8	41442.6	188.8	26384.4
3sat_3_185_777	10	1	20754.7	274387.5	1786.3	161451.2
3sat_3_187_785	10	4	31886.2	224951.8	2225.9	116858.8
3sat_3_190_741	10	9	4963.8	96510.7	372.7	43948.6
3sat_3_193_791	10	5	1588.0	152717.1	111.5	82180.6
3sat_3_196_784	10	10	7338.9	12086.1	529.3	2626.6
3sat_3_200_840	10	6	7089.1	163105.4	533.7	120520.4

Table 1
3sat instances

[9] M. De Santis, S. Lucidi, and F. Rinaldi. Feasibility Pump-like heuristics for mixed integer problems. *Discrete Applied Mathematics*, 165: 152–167, 2014.

[10] M. Fischetti, F. Glover, and A. Lodi. The feasibility pump. *Mathematical Programming*, 104:91–104, 2005.

instance	Found		Avg. iterations		Avg. restarts	
	CARFP	RFP	CARFP	RFP	CARFP	RFP
no-good_5_30	8	7	60012.5	90009.4	600070.9	900082.6
no-good_5_31	6	5	120011.2	150006.2	1200056.6	1500060.7
no-good_6_60	10	10	15.5	15.1	52.5	129.2
no-good_6_62	10	3	22.9	210007.7	128.9	2100082.6
no-good_6_63	6	5	120011.4	150012.3	1200035.4	1500137.8
no-good_7_121	9	10	30020.9	13.6	300065.6	74.4
no-good_7_125	10	10	19.0	8.3	75.6	39.4
no-good_7_126	10	9	49.5	30017.3	262.9	300159.0
no-good_7_127	6	3	120021.6	210014.9	1200103.4	2100146.7
no-good_8_243	10	10	15.3	14.0	36.9	62.7
no-good_8_250	10	10	44.0	39.5	130.2	242.3
no-good_8_253	8	8	60069.3	60053.7	600267.3	600491.2
no-good_8_254	9	9	30057.8	30041.4	300228.7	300296.3
no-good_8_255	8	5	60031.0	150007.8	600110.4	1500046.2
no-good_9_486	10	10	26.1	17.4	63.7	73.2
no-good_9_501	10	10	52.9	34.1	152.4	168.0
no-good_9_506	10	10	83.5	56.4	253.1	293.5
no-good_9_509	10	8	115.8	60066.1	392.7	600369.2
no-good_9_510	7	8	90111.2	60076.8	900426.0	600520.6
no-good_9_511	5	4	150058.4	180025.3	1500198.0	1800138.7
no-good_10_1003	10	10	41.3	38.6	89.8	160.6
no-good_10_1013	10	10	57.8	126.1	131.3	640.9
no-good_10_1018	10	10	208.5	159.0	585.1	802.5
no-good_10_1021	10	8	275.0	60110.6	830.8	600554.9
no-good_10_1022	8	5	60208.8	150118.4	600571.1	1500611.5
no-good_10_1023	5	3	150191.4	210025.5	1500671.8	2100109.8
no-good_10_972	10	10	18.2	23.4	37.8	84.8
no-good_11_1945	10	10	22.9	32.2	46.6	120.2
no-good_11_2007	10	10	25.0	61.1	43.2	247.4
no-good_11_2027	10	10	89.3	90.4	195.6	374.5
no-good_11_2037	10	10	222.5	296.0	586.6	1468.9
no-good_11_2043	10	10	335.0	396.9	891.5	2007.0
no-good_11_2045	10	10	1.0	1.0	0.0	0.0
no-good_11_2047	7	2	90329.3	240033.9	900875.4	2400132.2
no-good_12_3891	10	10	2.0	20.1	0.0	66.8
no-good_12_4014	10	10	51.1	60.4	91.2	210.7
no-good_12_4055	10	10	109.1	83.7	198.9	291.0
no-good_12_4075	10	10	207.8	81.2	398.1	285.8
no-good_12_4087	10	10	466.8	443.1	966.8	1824.1
no-good_12_4091	10	10	627.9	722.2	1365.8	2891.2
no-good_12_4095	7	2	90911.2	240323.6	902199.4	2401477.4

Table 2
`no-good` instances