ÉCOLE POLYTECHNIQUE | INSTITUT POLYTECHNIQUE DE PARIS

Statement of Academic Integrity Regarding Plagiarism

I, the undersigned.....Nigaty Brook.....[family name, given name(s)], hereby certify on my honor that:

1. The results presented in this report are the product of my own work.

2. I am the original creator of this report.

3. I have not used sources or results from third parties without clearly stating thus and referencing them according to the recommended rules for providing bibliographic information.

Declaration to be copied below:

I hereby declare that this work contains no plagiarized material.

I hereby declare that this work contains no plagiarized material.

Date .....18/03/2024.....

Signature Brook

École Polytechnique

BACHELOR THESIS IN COMPUTER SCIENCE

Alternating Projection Algorithms for Finding Feasible Solutions of Large-Scale Optimization Problems

Author:

Brook Nigatu, École Polytechnique

Advisor:

Claudia D'Ambrosio, Laboratoire d'Informatique de l'X

Academic year 2023/2024

Abstract

Feasibility pumps are heuristics to find feasible solutions to mixed-integer programs. The most basic version involves generating two sequences that hopefully both converge to a feasible solution of a given mixed integer program. The main obstacle to the success of such algorithms is cycling: the sequences may get stuck visiting the same succession of points so that there is no convergence to a common limit. Here we introduce a modification of the original algorithm to handle cycling issues and test the performance of this variant on benchmark instances. We shall restrict ourselves to the case of mixed-integer Linear Programs (MILPs) where integral variables are binary.

Contents

1 Introduction

Throughout this manuscript we consider the following formulation of a 0-1 mixed-integer linear program (MILP):

$$\begin{gathered} min\ c^T x \\ A\,x \leq b \\ l \leq x \leq u \\ x_i \in \{0,1\}\ \ \forall i \in \mathcal{I} \subset \{1,\ldots,n\} \end{gathered}$$

where $x \in \mathbb{R}^n$ represents the vector of variables, $A \in M_{m\times n}(\mathbb{R})$, $b \in \mathbb{R}^n$, $c \in \mathbb{R}^n$, $l \in (\{-\infty\} \cup \mathbb{R})^n$, and $u \in (\mathbb{R} \cup \{+\infty\})^n$.

Inequalities are evaluated coordinate-wise. The function $x \mapsto c^T x$, known as the objective function, is to be minimized over the space of vectors satisfying the inequalities and integrality constraints. We refer to a vector satsifying the integrality requirements (but not necessarily the other constraints) as an **integral point**, and we refer to a vector that doesn't satisfy the integrality requirements as a **fractional point**.

The continuous relaxation of such a model is obtained by removing the integrality constraints, or equivalently fixing $\mathcal{I}$ to be the empty set. Unlike the continuous linear programs, for which there exist polynomial-time (as a function of the number of variables and constraints) algorithms to find optimal solutions, general MILPs are NP-hard.

Using cheap heuristics to find **feasible solutions**, i.e solutions that satisfy all the constraints but not necessarily optimal for the objective function, can assist other algorithms in finding optimal solutions more efficiently. Feasible solutions may also be desirable on their own.

One such heuristic, originally proposed in [11], is the feasibility pump. This algorithm generates two sequences of vectors that hopefully converge to a common limit. One is a sequence of solutions to (continuous) linear programs (LPs) obtained by relaxing the integrality constraints in the original problem and changing the objective function, while the other is a sequence of vectors satisfying the integrality constraints, which we refer to as "integral points" for convenience. Therefore, if the two sequences converge to the same limit, the algorithm will have succeeded in finding a feasible solution to the original MILP. Typically, each term of one sequence is generated by minimizing a certain distance with respect to the last term in the other sequence.

In many cases, it is observed that feasibility pumps loop through the same cycle of points over and over again without the desired convergence of the sequences to the same limit. Therefore, the basic algorithm must be modified to have cycle-handling mechanisms.

This paper is organized as follows: we discuss in Section 2 some concepts in Mathematical Programming. In section 3, we study the basic version of the feasibility pump for 0-1 MILPs and the ideas behind it. We then review in Section 4 some of the previous work that has been done regarding feasibility pumps. In Section 5, we present our variant of the algorithm. Finally in Section 6, we look at some computational results on a benchmark set of MIP instances, and conclude in Section 7.

2 Some Concepts in Mathematical Programming

In this section we present some basic theory on mathematical programs, as well as some tools used to approach them. A more complete treatment of these ideas can be found in [10].

Mathematical programs can be formulated as follows:

$$\begin{gathered} minf(x) \\ g_i(x) \leq 0 \ \ \forall i \in \{1, ..., m\} \\ l \leq x \leq u \\ x_j \in \mathbb{Z} \ \ \forall j \in \mathcal{I} \subset \{1, ..., n\} \end{gathered}$$

where $l \in (\{-\infty\} \cup \mathbb{R})^n$, $u \in (\mathbb{R} \cup \{+\infty\})^n$, and the g_i are arbitrary functions from $\mathbb{R}^n$ to $\mathbb{R}$.

The set X of vectors satisfying the constraints is known as the feasible region, and $f : X \to \mathbb{R}$ is known as the objective function. If $X = \emptyset$ the program is said to be *infeasible*, and if $\inf_{x \in X} f(x) = -\infty$, the program is said to be *unbounded*.

It is important to note that the above formulation can be assumed without loss of generality. For instance, a maximization problem can be modeled as minimization problem by negating the the objective function, and $\geq$ constraints can be imposed by negating the corresponding g_i. An equation can also be dealt with by splitting it into two inequalities.

Convex programs are problems where $\mathcal{I} = \emptyset$ and f and the g_i are all convex. In this case, the feasible region is a convex set. A well known property of convex programs is that local optima are always global optima; consequently, algorithms to solve such problems need only find local optima.

A Linear Program (LP) is a convex program where f is linear and the g_i are all affine. $f(x)$ can then be expressed as $c^T x$ for some $c \in \mathbb{R}^n$ and the constraints defined by the g_i can be grouped and expressed as $A\,x \leq b$ for some $A \in M_{m \times n}(\mathbb{R})$ and $b \in \mathbb{R}^n$. The following example of a linear program is depicted in figure 1.

$$\begin{gathered} min -x - 2y \\ x \geq 0 \\ y \geq 0 \\ x + y \leq 10 \\ 2y - x \leq 8 \end{gathered}$$

The shaded area is the feasible region. The optimum is reached at the point $(4, 6)$.

The feasible region of a linear program is a polyhedron, and if an optimum exists, it is guaranteed that at least one vertex of the polyhedron achieves it. Dantzig's simplex method is an iterative algorithm to solve linear programs that exploits this fact. It starts from a vertex of the feasible polyhedron of a linear program, and at each iteration moves to an adjacent vertex with a better objective value. It terminates upon reaching a vertex with no such adjacent vertices.

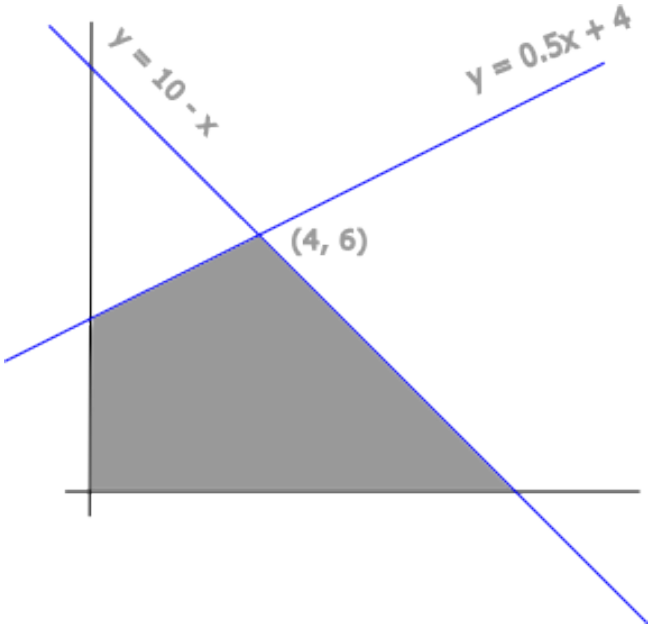

Figure 1: A Linear Program

Mixed-integer linear programs (MILPs), formulated as 1, are not convex. Algorithms to tackle them usually involve solving LPs obtained by relaxing integrality requirements. Additional constraints, a.k.a "cuts", can also be introduced to remove parts of the feasible region of the continuous relaxation of a given MILP in order to guide the search to the optimal solution. As mentioned in the introduction, we refer to a vector satsifying the integrality requirements (but not necessarily the other constraints) as an **integral point**, and we refer to a vector that doesn't satisfy the integrality requirements as a **fractional point**. Figure 2 depicts the following example program.

$$
\begin{aligned}
min -2y - x \\
x \geq 0 \\
y \geq 0 \\
x + y \leq 9 \\
-x + y \leq 2 \\
x \in \mathbb{Z} \\
y \in \mathbb{Z}
\end{aligned}
$$

The feasible points are shown as dots. The optimum is reached at the point $(4, 5)$.

The cutting plane algorithm and Branch-and-Bound are two algorithms that can be used to solve MILPs. The cutting plane algorithm starts with the continuous relaxation of a given MILP and iteratively solves a series of LPs until an integral point is obtained as a solution. Whenever a fractional point is encountered, an additional constraint (also known as a **cut**), which removes part of the feasible region including this fractional point but not any feasible points, is added to the program for the following iterations. An example of a cut for program 2 is illustrated in figure 3. Branch-and-Bound, on the other hand, solves the continuous relaxation, and if a fractional solution is obtained, the problem is split into two sub-problems, to which the same is applied. This branching is done based on the fractional solution x^* by selecting an index $j \in \mathcal{I}$ (the $\mathcal{I}$ in 1) with $x_j^* \notin \mathbb{Z}$—usually the one with the smallest corresponding domain from such indices—and introducing the constraint

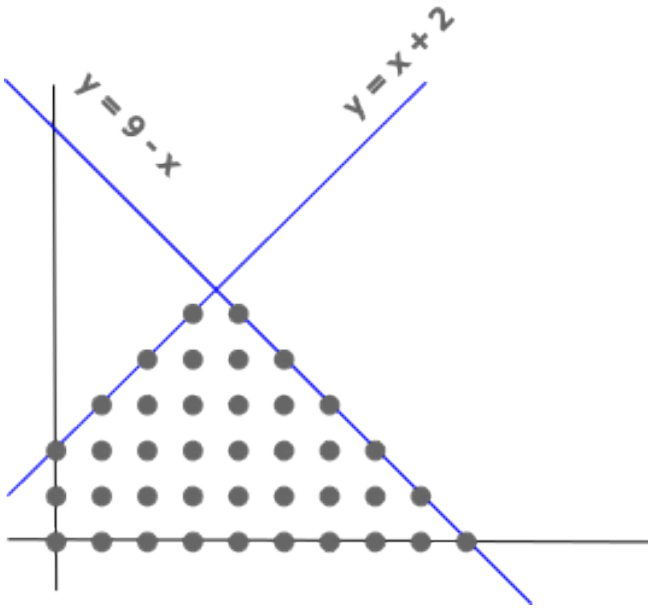

Figure 2: A MILP

$x_j \leq \lfloor x_j^* \rfloor$ to get one sub-problem and the constraint $x_j \geq \lceil x_j^* \rceil$ to get another. This is illustrated in figure 4 for program 2. Objective values of solutions at each step serve as lower bounds on the objective values that can be achieved by solutions from further branching since branches minimize the objective function on smaller subsets. When a branch terminates by finding an integral point, the corresponding objective value is used as an upper-bound to possibly eliminate unexplored sub-problems with larger lower-bounds (determined from the objective value of the solutions to the sub-problems they branch from), i.e the algorithm can eliminate a sub-problem by determining that it won't lead to an optimal solution if it finds a feasible solution with a smaller objective value than that which can be achieved by a solution from this sub-problem. This continues until no sub-problems remain, and the feasible point with the lowest objective value is returned.

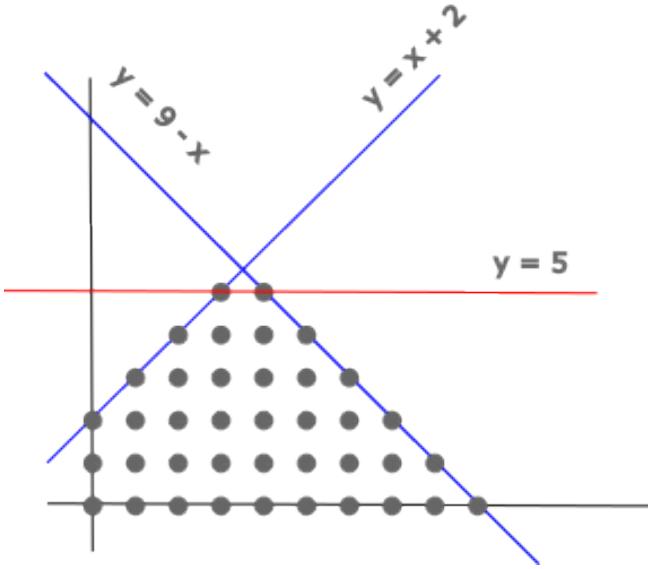

Figure 3: An example of a cut for program 2 The optimum for the relaxation is reached at $(3.5, 5.5)$, which is not feasible. The cut $y \leq 5$ cuts out this point while preserving all feasible solutions, and in fact, the next iteration finds the optimum $(4, 5)$, which solves program 2.

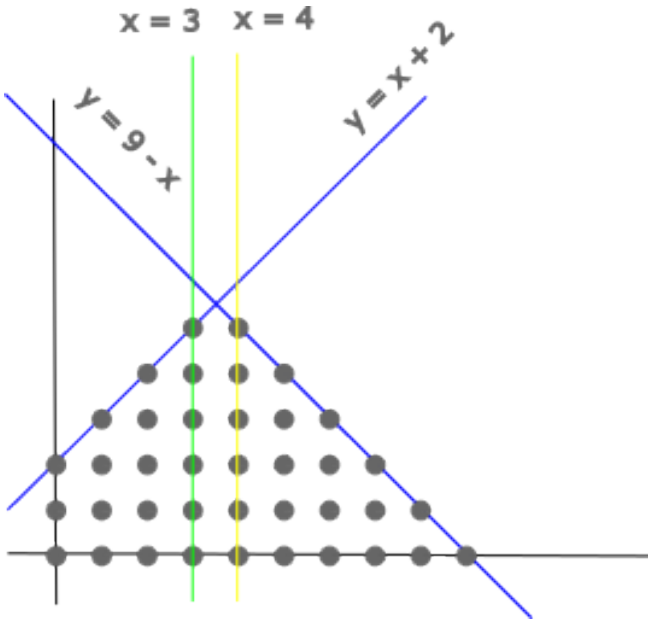

Figure 4: A Branch-and-Bound step. For program 2, the initial continuous relaxation is optimized at the point $(3.5, 5.5)$. This is a fractional (and hence) infeasible point. Branching on the first variable, two sub-problems are created by introducing the constraints $x \leq 3$ for the first one and $x \geq 4$ for the second one. The first sub-problem immediately finds the point $(3, 5)$, which is feasible so it doesn't do further branching. Similarly the second sub-problem finds the point $(4, 5)$. Since $(4, 5)$ has the smaller objective value (for the function in 2), it is returned as the solution.

3 Basic Feasibility Pump for 0-1 MILPs

As introduced earlier, feasibility pumps involve producing two sequences of vectors that shall in a successful run both converge to the same feasible point. In this section, we discuss a basic version from the original work on feasibility pumps by M. Fischetti, F. Glover, and A. Lodi [11]. The pseudo-code (adapted from the one given in the paper), assuming the formulation of 0-1 MILPs 1, reads as follows:

Data: a 0-1 MILP , T , TL
$nIT \leftarrow 0$;
$x^* \leftarrow argmin\{c^T x : Ax \leq b,\ l \leq x \leq u\}$;
$\tilde{x} \leftarrow round(x^*)$ (point-wise rounding of coordinates in $\mathcal{I}$);
while $time < TL$ **do**
  $nIT \leftarrow nIT + 1$;
  $x^* \leftarrow argmin\{\Delta(\tilde{x}, x) : Ax \leq b,\ l \leq x \leq u\}$;
  if $\forall j \in \mathcal{I}$, $x^*_j \in \{0, 1\}$ **then**
    $return\ x^*$
  end
  if $round(x^*) \neq \tilde{x}$ **then**
    $\tilde{x} \leftarrow round(x^*)$
  else
    $TT \leftarrow rand(\frac{T}{2}, 3 * \frac{T}{2})$;
    flip the TT entries $\tilde{x}_j$ $(j \in \mathcal{I})$ with highest $|x^*_j - \tilde{x}_j|$
  end
end

Algorithm 1: The Feasibility Pump (basic version)

The algorithm generates the two sequences $(x^*_k)_{k\geq 0}$ and $(\tilde{x}_k)_{k\geq 0}$. x^*_0 is computed by solving the continuous relaxation of the MILP. For each $k \geq 0$, $\tilde{x}_k$ is obtained by rounding the coordinates of x^*_k corresponding to indices in $\mathcal{I}$ in what is called the **rounding step**, and x^*_{k+1} is obtained by minimizing a "distance" from $\tilde{x}_k$ over the feasible region of the continuous relaxation in the **projection step**.

The function used in the paper is given by:

$$\Delta(\tilde{x}, x) = \sum_{j\in\mathcal{I}} |x_j - \tilde{x}_j| = \sum_{j\in\mathcal{I},\, \tilde{x}_j=0} x_j \ + \sum_{j\in\mathcal{I},\, \tilde{x}_j=0} (1 - x_j) \tag{1}$$

where $\tilde{x}$ is an integral point. Ignoring the constant term, this is a linear function of x. Hence the projection step amounts to solving a linear program.

Also note that the rounding step minimizes the same distance over the space of integral points, i.e $\tilde{x}_k \in argmin\,\{\Delta(x, x^*_k) : \forall j \in \mathcal{I}\,,\, x_j \in \{0, 1\}\}$ for each $k \geq 0$. Therefore, the sequences grow closer and closer to each other with respect to this distance.

The algorithm returns a feasible solution to the original MILP if and only if the projection step yields an integer point at some iteration. We remark that if the rounding step yields a feasible solution $\tilde{x}_k$, the next projection step will find also find a feasible solution. Indeed, since $\tilde{x}_k$ is in the feasible region of the continuous relaxation of the MILP with $\Delta(\tilde{x}_k, \tilde{x}_k) = 0$, x^*_{k+1} satisfies $\Delta(x^*_{k+1}, \tilde{x}_k) = 0$ as a solution of the minimization problem in the next projection step. This implies that x^*_{k+1} is an integral point and thus a feasible solution.

The heuristic described above can be seen as an alternating projection algorithm aimed at finding $x \in W \cap Z$,

where $W = \{w \in \mathbb{R}^n \ : \ Aw \leq b, \ l \leq w \leq u\}$, and $Z = \{z \in \mathbb{R}^n \ : \ \forall j \in \mathcal{I}, \ z_j \in \{0, 1\}\}$. At each iteration k it finds a pair of points $(w_k, z_k) \in W \times Z$ using projections with respect to Δ from a point in one space into the other space.

The algorithm is vulnerable to the issue of cycling. It may start revisiting the same points over and over again. One way this could happen, for instance is if for some $k \geq 1$, $round(x_k^*) = \tilde{x}_{k-1}$. Since each iteration only depends on the previous one, further iterations would then revisit these same points until the time limit is reached. Similarly, if either the projection step or the rounding step finds a point encountered at any previous iteration, a cycle is created and the algorithm is stuck. We refer to the length of a cycle as the number of iterations it consists of.

Algorithm 1 uses a randomized approach to escape from cycles of length 1. As soon as a rounding is found to be the same one obtained in the previous iteration, a number TT is randomly sampled based on the parameter T and TT coordinates corresponding to indices in $\mathcal{I}$ are flipped. The coordinates to be flipped are chosen so as to minimize the distance of the resulting integer point from the vector obtained in the projection step. This is done by sorting indices in $\mathcal{I}$ in decreasing order of the distance of the corresponding values to their binary roundings (their "fractionality") and flipping entries in the integral point indexed by the first TT indices.

The algorithm described so far is still susceptible to cycles of length 2 or more. In order to handle these, the authors also utilized a randomized approach they described as *"for each $j \in \mathcal{I}$ we generate a uniformly random $\rho_j \in [-0.3, 0.7]$ and flip $\tilde{x}_j$ in case $|x_j^* - \tilde{x}_j| + max\{0, \rho_j\} > 0.5$"*.

4 Literature Review

The Feasibility Pump has been subject to numerous works of research since its first introduction in 2005 [11]. Many publications on the algorithm exist due to efforts to improve certain aspects of the heuristic, extend it to larger problem classes, or analyse it theoretically. T. Berthold et al. [6] conducted a survey of much of the literature regarding the topic, some of which is discussed below.

L. Bertacco et al. [5] extended the algorithm to handle models with general integer variables. In this extended version, the rounding step remains the same. However, (**??**) has to be modified to accommodate general integer variables since integral values need not lie on one of the bounds. It takes on the following form:

$$\sum_{j \in \mathcal{I}\ \tilde{x}_j = l_j} x_j - l_j \ + \sum_{j \in \mathcal{I}\ \tilde{x}_j = u_j} u_j - x_j \ + \sum_{j \in \mathcal{I}\ l_j < \tilde{x}_j < u_j} d_j$$

where for each $j \in \mathcal{I}\ l_j < \tilde{x}_j < u_j$, an extra variable d_j and two extra constraints $d_j \geq \tilde{x}_j - x_j,\ d_j \geq x_j - \tilde{x}_j$ are introduced so that when the above expression is minimized, $d_j = |\tilde{x}_j - x_j|$. Furthermore, a different perturbation scheme is used to escape cycles since the idea of flipping variables no longer makes sense in the context of non-binary integer variables.

Extensions also exist for non-linear and non-convex mixed integer programs [7, 8, 9, 13, 14], as well as multi-objective mixed-integer programs [3, 4].

As heuristics to find feasible solutions, early versions of the feasibility pump pay no attention to the quality of the solutions found, i.e. how close their objective values are to the optimal ones. In fact, the initial objective function is only used once. T. Achterberg and T. Berthold [2] proposed a variant where the objective function used in the projection step is modified to penalize larger objective values. More specifically, a convex combination of the distance function and the original objective is used.

In addition to extensions and improvements to the feasibility pump, there has also been some research investigating theoretical properties of the heuristic. Some analyses consider the feasibility pump with no randomized perturbation mechanisms—the so-called *idealized feasibility pump*—and interpret it as a special case of other well-known algorithms. For example, B. Geißler et al. [12] interpret the idealized feasiblity pump as an alternating direction method. An interesting property shown in *Lemma 3* of the paper is that the idealized feasibility pump can not enter cycles of length 2 or more under the assumption that either the rounding step or the projection step always has a unique solution for the function used to represent distances.

5 An Alternative Flipping Mechanism

Data: a 0-1 MILP , TL
$nIT \leftarrow 0$;
$x^* \leftarrow argmin\{c^T x \,:\, Ax \leq b,\ l \leq x \leq u\}$;
$\tilde{x} \leftarrow round(x^*)$ (point-wise rounding of coordinates in $\mathcal{I}$);
while $time < TL$ **do**
  $nIT \leftarrow nIT + 1$;
  $x^* \leftarrow argmin\{\Delta(\tilde{x}, x) \,:\, Ax \leq b,\ l \leq x \leq u\}$;
  if $\forall j \in \mathcal{I}\,,\, x^*_j \in \{0,1\}$ **then**
    $return\ x^*$
  end
  if $round(x^*) \neq \tilde{x}$ **then**
    $\tilde{x} \leftarrow round(x^*)$
  else
    $k \leftarrow ceil\,(\Delta(\tilde{x}, x^*))$;
    flip the k entries $\tilde{x}_j$ $(j \in \mathcal{I})$ with highest $|x^*_j - \tilde{x}_j|$;
    add constraint $\Delta(\tilde{x}, x) \geq k$
  end
end

Algorithm 2: The Feasibility Pump (modified version)

We propose a variant of feasibility pump largely similar to 1 except that we replace the randomized procedure to handle 1-cycles. The modification relies on the following observation.

Assume that a 1-cycle is detected with a fractional point x^* and an integer point $\tilde{x}$, i.e x^* rounds to $\tilde{x}$ and $\tilde{x}$ projects to x^* w.r.t Δ (defined as in 1).

Let $\Delta(\tilde{x},\, x^*) = \beta$. Then for any feasible solution $\tilde{y}$, we have

$$\Delta(\tilde{x}, \tilde{y}) \geq \lceil \beta \rceil$$

Indeed, since x^* minimizes Δ over the feasible region of the continuous relaxation of the MILP, which contains y, $\Delta(\tilde{x}, y) \geq \beta$. Furthermore, since Δ only involves indices in $\mathcal{I}$ and both $\tilde{x}$ and y satisfy integrality constraints $\Delta(\tilde{x}, y) \in \mathbb{N}$.

To escape the cycle, the modified algorithm adds the constraint

$$\Delta(\tilde{x}, x) \geq \lceil \beta \rceil \tag{2}$$

and selects a new integer point y such that

$$\tilde{y} \in argmin\,\{\Delta(y, x^*) \mid \Delta(\tilde{x}, y) = \lceil \beta \rceil,\ \forall j \in \mathcal{I}\ \tilde{y}_j \in \{0,1\}\}$$

i.e. $\tilde{y}$ minimizes the distance to x^* from all integer points with distance $\lceil \beta \rceil$ from $\tilde{x}$. Similar to what was described for the randomized flipping procedure in 1, this point can be obtained by flipping the $\lceil \beta \rceil$ most fractional entries from those indexed by indices in $\mathcal{I}$. The algorithm then proceeds to the projection step in the next iteration using this integer point and escapes the cycle.. Moreover, since x^* has been cut out of the feasible region, the same cycle can't be entered again even if another fractional point rounds to $\tilde{x}$ later.

The new integer point is chosen this way to obtain a possible feasible solution while minimally increasing the distance between the two sequences.

6 Computational Results, Observations, and Further Modifications

In this section we compare computational results of some feasibility pump variants, including Algorithm 2, on a set of 161 0-1 MILP instances from the MIPLIB 2017 benchmark set [1] (the benchmark set was filtered to only contain models with binary and continuous variables). Also tested were 2 variants of the algorithm designed to address observed weaknesses of Algorithm 2

The following table shows the names, number of variables, number of constraints, and number of binary variables of instances used for testing.

Table 1: Instance Details

name	nvars	nconstrs	$\lvert\mathcal{I}\rvert$	name	nvars	nconstrs	$\lvert\mathcal{I}\rvert$
CMS750_4	11697	16381	7196	neos-4413714-turia	190402	2303	190201
academictimetablesmall	28926	23294	28926	neos-4532248-waihi	86842	167322	86841
air05	7195	426	7195	neos-4647030-tutaki	12600	8382	7000
app1-1	2480	4926	1225	neos-4763324-toguru	53593	106954	53592
app1-2	26871	53467	13300	neos-4954672-berkel	1533	1848	630
assign1-5-8	156	161	130	neos-5049753-cuanza	242736	322248	8304
b1c1s1	3872	3904	288	neos-5052403-cygnet	32868	38268	32868
bab2	147912	17245	147912	neos-5093327-huahum	40640	51840	64
bab6	114240	29904	114240	neos-5104907-jarama	345856	489818	9520
beasleyC3	2500	1750	1250	neos-5107597-kakapo	3114	6498	2976
binkar10_1	2298	1026	170	neos-5114902-kasavu	710164	961170	14560
blp-ar98	16021	1128	15806	neos-5188808-nattai	14544	29452	288
blp-ic98	13640	717	13550	neos-5195221-niemur	14546	42256	9792
bnatt400	3600	5614	3600	neos-631710	167056	169576	167056
bnatt500	4500	7029	4500	neos-787933	236376	1897	236376
bppc4-08	1456	111	1454	neos-827175	32504	14187	21350
cbs-cta	24793	10112	2467	neos-848589	550539	1484	747
chromaticindex1024-7	73728	67583	73728	neos-860300	1385	850	1384
chromaticindex512-7	36864	33791	36864	neos-873061	175288	93360	87644
cmflsp50-24-8-8	16392	3520	1392	neos-911970	888	107	840
co-100	48417	2187	48417	neos-933966	31762	12047	27982
cod105	1024	1024	1024	neos-957323	57756	3757	57756
cost266-UUE	4161	1446	171	neos-960392	59376	4744	59376
csched007	1758	351	1457	neos17	535	486	300
csched008	1536	351	1284	neos5	63	63	53
cvs16r128-89	3472	4633	3472	net12	14115	14021	1603
dano3_3	13873	3202	69	netdiversion	129180	119589	129180
dano3_5	13873	3202	115	nexp-150-20-8-5	20115	4620	17880
drayage-100-23	11090	4630	11025	ns1116954	12648	131991	7482
drayage-25-23	11090	4630	11025	ns1208400	2883	4289	2880
dws008-01	11096	6064	6608	ns1644855	40200	40698	10000
eil33-2	4516	32	4516	ns1760995	17956	615388	17822
eilA101-2	65832	100	65832	ns1830653	1629	2932	1458
ex10	17680	69608	17680	nw04	87482	36	87482
ex9	10404	40962	10404	opm2-z10-s4	6250	160633	6250
exp-1-500-5-5	990	550	250	p200x1188c	2376	1388	1188

fast0507	63009	507	63009	peg-solitaire-a3	4552	4587	4552
fastxgemm-n2r6s0t2	784	5998	48	pg	2700	125	100
fhnw-binpack4-4	520	620	481	pg5_34	2600	225	100
fhnw-binpack4-48	3710	4480	3605	physiciansched3-3	79555	266227	72141
glass-sc	214	6119	214	physiciansched6-2	111827	168336	109346
glass4	322	396	302	pk1	86	45	55
gmu-35-40	1205	424	1200	qap10	4150	1820	4150
gmu-35-50	1919	435	1914	rail01	117527	46843	117527
graph20-20-1rand	2183	5587	2183	rail02	270869	95791	270869
h80x6320d	12640	6558	6320	rail507	63019	509	63009
irish-electricity	61728	104259	9888	ran14x18-disj-8	504	447	252
irp	20315	39	20315	rd-rplusc-21	622	125899	457
istanbul-no-cutoff	5282	20346	30	reblock115	1150	4735	1150
leo1	6731	593	6730	rmatr100-p10	7359	7260	100
leo2	11100	593	11099	rmatr200-p5	37816	37617	200
lotsize	2985	1920	1195	rocII-5-11	11523	26897	11341
mad	220	51	200	roi2alpha3n4	6816	1251	6642
map10	164547	328818	146	roi5alpha10n8	106150	4665	105950
map16715-04	164547	328818	146	s100	364417	14733	364417
markshare2	74	7	60	s250r10	273142	10962	273139
markshare_4_0	34	4	30	satellites2-40	35378	20916	34324
mas74	151	13	150	satellites2-60-fs	35378	16516	34324
mas76	151	12	150	savsched1	328575	295989	252731
mc11	3040	1920	1520	seymour	1372	4944	1372
milo-v12-6-r2-40-1	2688	5628	840	seymour1	1372	4944	451
momentum1	5174	42680	2349	sing326	55156	50781	40010
n2seq36q	22480	2565	22480	sing44	59708	54745	43524
n3div36	22120	4484	22120	sorrell3	1024	169162	1024
neos-1122047	5100	57791	100	sp150x300d	600	450	300
neos-1171448	4914	13206	2457	sp97ar	14101	1761	14101
neos-1171737	2340	4179	1170	sp98ar	15085	1435	15085
neos-1445765	20617	2147	2150	supportcase10	14770	165684	14770
neos-2075418-temuka	122304	349602	122304	supportcase18	13410	240	13410
neos-2978193-inde	20800	396	64	supportcase22	7129	260602	7129
neos-2987310-joes	27837	29015	3051	supportcase26	436	870	396
neos-3216931-puriri	3555	5989	3268	supportcase40	16440	38192	2000
neos-3402294-bobin	2904	591076	2616	swath1	6805	884	2306
neos-3402454-bohle	2904	2897380	2616	swath3	6805	884	2706
neos-3555904-turama	37461	146493	37461	tbfp-network	72747	2436	72747
neos-3627168-kasai	1462	1655	535	tr12-30	1080	750	360
neos-3754480-nidda	253	402	50	trento1	7687	1265	6415
neos-3988577-wolgan	25870	44662	25870	uccase12	62529	121161	9072
neos-4300652-rahue	33003	76992	20900	uct-subprob	2256	1973	379
neos-4387871-tavua	4004	4554	2000	unitcal_7	25755	48939	2856
var-smallemery-m6j6	5608	13416	5606				

To solve linear programs (obtained as continuous relaxations), the Python API of Gurobi Optimizer (version 11.0) was used, and tests were run on a server with the specifications in 6. The solver was largely treated as a black-box with little modifications to the default parameters. The only alteration made was to limit the number of threads to 1. This lead to faster computation overall, possibly due to reduced competition for CPU resources between threads resulting from the program being run on different instances in parallel; with the number of threads set to 1, each run of the algorithm can be isolated to 1 CPU. The code for the implementations of the algorithms can be found in the appendix A.

6.1 Initial Tests and Results

To establish a baseline for our results, the original feasibility pump from [11] was implemented with the additional procedure to handle cycles of length ≥ 2 described at the end of section 3. To detect such cycles, a fixed-size queue of the most recently visited points is maintained to check whether or not points found in each iteration match any others from the previous few iterations. The T parameter was set to 20 and the queue size was set to 100 as in [11]. We label this "config1".

Two additional configurations ("config2" and "config3") based on config1 were also tested. In config2, the queue length was set to 3 instead of 100. In config3, the following constraint was added after the first iteration

$$c^T x \leq c^T x_1^* + 50.5 * \left| c^T x_1^* \right| \tag{3}$$

This was a suggestion from A. Lodi (one of the authors of [11]) based on improvements that were observed with this additional constraint.

For the first round of testing, these 3 configurations, along with Algorithm 2 ("config4") were run on the 161 0-1 MILP instances listed in Table 1. The time limit was set to 1800 seconds for each run. Results are presented in tables 3, 4, and 5.

Our feasibility pump variant (config4) found feasible solutions for 105 of the 161 instances, which was less than the number of successful runs of config1 (115) and config2 (109) but higher than that of config3 (103). It performs significantly less iterations due to the fact that the extra constraints introduced to escape 1-cycles add to the complexity of the linear program to be solved at each iteration, which can be seen from the average LP solving times. The special constraint (3) used in config3 speeds up the algorithm considerably by cutting out areas of the feasibility regions, however, it leads to less successes as it may also cut out feasible solutions.

Descriptions of the different configurations used can be found in Table 2. Summaries of the results for each configuration can be found in Table 3. Results for each configuration on each instance can be found in Table 4 and Table 5. In these two tables the columns "ObjVal" correspond to the objective values (w.r.t. the original objective function) of feasible solutions if they were found. "N/A" means that a feasible solution wasn't found. "nIT" denotes the number of iterations performed.

6.2 Observations from Failed Instances

The logs from the tests were studied to spot any weaknesses of Algorithm 2 (config4).

One common pattern amongst the failed instances was the development of 2-cycles (and a 3-cycle in one instance) after some iterations. The algorithm handles 1-cycles, but unlike configs 1-3, it doesn't have an explicit procedure to deal with longer cycles. This shows that without the guarantee that either the rounding step or projection step always has a unique solution, *Lemma 3* of [12] no longer holds. To see why, let us consider

the following example of a 2-cycle.

$$\tilde{x}_1 \xrightarrow{project} x_1^* \xrightarrow{round} \tilde{x}_2 \xrightarrow{project} x_2^* \xrightarrow{round} \tilde{x}_1 \xrightarrow{project} x_1^* \tag{4}$$

Observe that since Δ (1) is minimized at each projection and rounding step, we have

$$\Delta(\tilde{x}_1, x_1^*) \leq \Delta(\tilde{x}_2, x_1^*) \leq \Delta(\tilde{x}_2, x_2^*) \leq \Delta(\tilde{x}_1, x_2^*) \leq \Delta(\tilde{x}_1, x_1^*)$$

Since $\Delta(\tilde{x}_1, x_1^*)$ is at both ends of the chain of inequalities, we deduce that

$$\Delta(\tilde{x}_1, x_1^*) = \Delta(\tilde{x}_2, x_1^*) = \Delta(\tilde{x}_2, x_2^*) = \Delta(\tilde{x}_1, x_2^*) = \Delta(\tilde{x}_1, x_1^*)$$

Hence $\tilde{x}_1$ and $\tilde{x}_2$ are both valid roundings of x_1^* and x_2^*. In addition, x_1^* and x_2^* are both solutions of the linear programs representing the projection steps from the integer points $\tilde{x}_1$ and $\tilde{x}_2$. If rounding had unique solutions x_1^* would round back to $\tilde{x}_1$, and instead of a 2-cycle, we would have a 1-cycle, which the algorithm can handle. Similarly if LP solutions were unique, $\tilde{x}_2$ would project back to x_1^* and we would also end up with a 1-cycle.

There was also another type of cycle that was observed in some failed instances. It was caused by 1-cycles where the distances between the integer and fractional points were too close to their ceilings. Since the solver used to solve LPs operates under a tolerance (10^{-6}) for constraint satisfaction, the additional constraint (2) to handle the cycle effectively does nothing. This is because β is too close (or exactly equal) to $\lceil \beta \rceil$, and the point x^* satisfies the constraint up to the tolerance. For example if $\Delta(\tilde{x}, x^*) = 4$, the algorithm adds the constraint $\Delta(\tilde{x}, x) \geq 4$, which doesn't exclude x^* from the feasible region.

Since some variables are flipped, the algorithm still escapes the 1-cycle; however, if x^* is revisited before any other 1-cycles are reached, a longer cycle will form as flipping is done deterministically and subsequent iterations will visit the same points without changes to the feasible region which would introduced by handling other 1-cycles between the iterations that find x^*.

Two other interesting instances were *ex9* and *ex10* (see Table 1). In both of these instances, LP solution times become very large after the first constraint is added, and the algorithm only does a few iterations. Large LP solution times are also observed for the other configurations, so this is not necessarily caused by the additional constraints.

6.3 Additional Configurations and Final Results

To try and overcome the issues described in the previous subsection, we added two other variants ("config5" and "config6").

Whenever a 1-cycle is detected with $\beta = \Delta(\tilde{x}, x^*)$ within 10^{-6} of $\lceil \beta \rceil$, the algorithms corresponding to these configurations use $\lceil \beta \rceil + 1$ for the additional constraint (2) and for the number of variables to be flipped. This guarantees that x^* will be cut out of the feasibility region of the continuous relaxation, but it comes at a cost of potentially removing feasible solutions since it's possible to have a feasible solution y with $\Delta(\tilde{x}, y) = \lceil \beta \rceil$.

The difference between config5 and config6 is in how they deal with the issue of longer cycles. Config5 utilizes the randomized approaches used by config1 with 100 for the queue size and 20 for the parameter T, whereas config6 forces the algorithm into a 1-cycle whenever distances don't drop from a projection step to a rounding step. When this happens the rounding step is forced to round the fractional point to the integer point it was rounded from, and the resulting 1-cycle is handled. In the 2-cycle example (4), this means that x_1^* is forced to round to $\tilde{x}_1$ since $\Delta(\tilde{x}_1, x_1^*) = \Delta(\tilde{x}_2, x_1^*)$.

The results for these two configurations are better than those for config4 in terms of the number of successes, but they perform the fewest iterations out of all configurations due to the the more aggressive cutting strategies, which lengthen LP solving times and possibly remove some feasible solutions.

Table 2: Configuration Descriptions

config1	Algorithm 1 with additional randomized procedure (described in section 3 to handle cycles of length up to 100. T param is set to 20
config2	Algorithm 1 with additional randomized procedure (described in section 3 to handle cycles of length up to 3. T param is set to 20
config3	Same as config1 but with the additional constraint 3 added after the first iteration
config4	Algorithm 2
config5	Algorithm 2 with $\lceil\beta\rceil + 1$ used instead of $\lceil\beta\rceil$ in 2 and for the number of variables flipped whenever a 1-cycle is detected with $\lceil\beta\rceil - \beta < 10^{-6}$. In addition, the same randomized procedure used in config1 is used to handle longer cycles.
config6	Same as config5 but forcing 1-cycles is used to prevent longer cycles from developing.

Table 3: Configuration Summaries

Config	Successes	Avg. LP sol time (sec)	Avg. n of Iters	Avg. total time
config1	115	0.11	3927.84	569.02
config2	109	0.07	6539.34	634.94
config3	103	0.02	35443.07	704.75
config4	105	1.08	587.25	676.44
config5	110	2.07	283.65	620.41
config6	111	2.2	270.02	625.09

Table 4: Results for Configs 1-3

	config1			config2			config3		
name	ObjVal	nIt	Time	ObjVal	nIt	Time	ObjVal	nIt	Time
CMS750_4	562.0	58	23	554.0	66	25	565.0	70	28
academictimetablesmall	N/A	2450	1800	N/A	2323	1800	N/A	2443	1800
air05	44905.0	8	1	42992.0	16	2	45633.0	11	1
app1-1	-2.0	8	0	-2.0	7	0	-1.0	39	2
app1-2	-23.0	17	80	-23.0	24	113	-23.0	42	197
assign1-5-8	520.0	2	0	520.0	2	0	520.0	2	0
b1c1s1	67020.85	9	1	62533.35	7	0	53609.91	7	0
bab2	N/A	330	1800	N/A	321	1800	N/A	305	1800
bab6	N/A	447	1800	N/A	473	1800	N/A	380	1801
beasleyC3	1033.0	13	0	962.0	18	0	930.0	13	0
binkar10_1	1010881.95	14	0	10293.83	25	0	341815.18	11	0
blp-ar98	22647.64	165	59	N/A	5041	1800	22087.68	24	7
blp-ic98	14905.14	4	0	13503.61	4	0	20157.9	12	2
bnatt400	N/A	21513	1800	N/A	18708	1800	N/A	20091	1800
bnatt500	N/A	19694	1800	N/A	20337	1800	N/A	19028	1800
bppc4-08	132.0	2	0	132.0	2	0	2656.89	2	0
cbs-cta	33486825271.26	2	0	33486825271.26	2	1	0.0	14	7
chromaticindex1024-7	4.0	3	6	4.0	3	7	4.0	3	6
chromaticindex512-7	4.0	3	4	4.0	3	4	4.0	3	4
cmflsp50-24-8-8	N/A	2259	1800	N/A	2812	1800	N/A	2357	1800
co-100	N/A	975	1800	N/A	1027	1800	N/A	605	1801
cod105	0.0	2	2	0.0	2	2	0.0	2	3
cost266-UUE	42682286.46	6	0	43110970.14	6	0	44084034.97	5	0
csched007	N/A	107460	1800	N/A	101465	1800	N/A	117949	1800
csched008	188.0	40213	867	N/A	82851	1800	8806.5	24199	506
cvs16r128-89	-2.0	2	0	-2.0	2	0	-2.0	2	0
dano3_3	1000.0	28	26	1000.0	44	36	29675.93	174	126
dano3_5	1000.0	12	20	1000.0	68	59	29675.93	27	31
drayage-100-23	N/A	10236	1800	856845.77	2866	541	N/A	8906	1800
drayage-25-23	N/A	9368	1800	N/A	10013	1800	N/A	8569	1800
dws008-01	N/A	12000	1800	N/A	18201	1800	N/A	17485	1800

eil33-2	3459.24	58	4	2296.64	108	9	1771.59	14	1	
eilA101-2	N/A	958	1800	N/A	910	1800	N/A	877	1800	
ex10	N/A	3	1800	N/A	3	1800	N/A	3	1800	
ex9	N/A	11	1800	N/A	14	1801	81.0	8	1064	
exp-1-500-5-5	666788.0	2	0	666788.0	2	0	131727.0	4	0	
fast0507	185.0	6	4	185.0	6	4	183.0	4	4	
fastxgemm-n2r6s0t2	6472.72	18	1	6472.72	26	2	855.01	34	4	
fhnw-binpack4-4	N/A	212685	1800	N/A	233406	1800	N/A	208253	1800	
fhnw-binpack4-48	N/A	44532	1800	N/A	41300	1800	N/A	40598	1800	
glass-sc	40.0	4	0	30.0	3	0	46.0	4	0	
glass4	4250041950.0	447	2	N/A	302105	1800	41200123600.0	450	2	
gmu-35-40	-1861854.52	30	0	-2076273.51	10	0	-1960177.38	28	0	
gmu-35-50	-2205354.31	44	1	-2136797.64	57	2	-2469836.98	5	0	
graph20-20-1rand	-3.0	21	1	-5.0	23	1	-6.0	13	0	
h80x6320d	13125.56	8	1	13938.76	7	2	11479.47	10	3	
irish-electricity	N/A	342	1800	N/A	367	1800	N/A	250	1800	
irp	13544.63	13	3	12623.07	9	1	15571.79	47	14	
istanbul-no-cutoff	290.01	2	6	290.01	2	5	290.01	2	6	
leo1	910690607.04	4	0	872876747.2	4	0	809417916.64	4	0	
leo2	912288774.84	5	1	959170767.8	5	1	939502602.54	5	1	
lotsize	12599278.0	23	0	13028036.0	12	0	11702173.0	59	1	
mad	2.47	2	0	2.47	2	0	N/A	430386	1800	
map10	0.0	2	11	0.0	2	10	0.0	2	10	
map16715-04	0.0	2	10	0.0	2	11	0.0	2	9	
markshare2	1950.0	3	0	1946.0	3	0	N/A	1079749	1800	
markshare_4_0	91.0	2	0	91.0	2	0	N/A	2544667	1800	
mas74	19142.39	2	0	19142.39	2	0	539863.96	2	0	
mas76	48566.14	2	0	48566.14	2	0	2003036.04	2	0	
mc11	36000.0	22	0	34177.0	29	0	N/A	84428	1800	
milo-v12-6-r2-40-1	835603.74	22	2	837415.8	20	2	836936.53	22	2	
momentum1	455626.85	298	1179	N/A	361	1800	462018.32	290	1459	
n2seq36q	3143800.0	18	8	N/A	3314	1800	62800.0	3	1	
n3div36	597600.0	5	1	604000.0	5	1	268400.0	5	1	
neos-1122047	191.0	28	222	163.0	228	1182	212.0	25	235	
neos-1171448	0.0	2	0	0.0	2	0	0.0	2	0	

neos-1171737	0.0	2	0	0.0	2	0	0.0	2	0
neos-1445765	N/A	22812	1800	N/A	20940	1800	N/A	24125	1800
neos-2075418-temuka	N/A	2	1800	N/A	2	1800	N/A	2	1800
neos-2978193-inde	44.53	2	0	44.53	2	0	47.2	2	0
neos-2987310-joes	-443066262.18	9	8	-603133662.02	16	14	-471037569.3	18	16
neos-3216931-puriri	N/A	9518	1800	N/A	10085	1800	N/A	9012	1800
neos-3402294-bobin	0.63	153	596	N/A	287	1800	N/A	263	1800
neos-3402454-bohle	N/A	34	1800	N/A	32	1800	N/A	6	1800
neos-3555904-turama	N/A	616	1800	N/A	567	1800	N/A	599	1800
neos-3627168-kasai	1021049.11	5	0	1019352.73	5	0	1025430.44	6	0
neos-3754480-nidda	29227.37	2	0	29227.37	2	0	29599.02	2	0
neos-3988577-wolgan	N/A	480	1800	N/A	478	1800	N/A	518	1800
neos-4300652-rahue	N/A	349	1800	N/A	333	1800	N/A	393	1800
neos-4387871-tavua	N/A	35717	1800	N/A	39142	1800	N/A	32616	1800
neos-4413714-turia	466.18	11	58	466.18	10	52	2238.94	9	51
neos-4532248-waihi	N/A	686	1800	N/A	974	1800	N/A	59	1800
neos-4647030-tutaki	39102.98	8	33	38031.41	9	38	38092.86	9	37
neos-4763324-toguru	5626.97	2	12	5626.97	2	10	57730.78	7	91
neos-4954672-berkel	11182960.0	2	0	11182960.0	2	0	11182960.0	2	0
neos-5049753-cuanza	N/A	89	1800	N/A	121	1800	N/A	132	1800
neos-5052403-cygnet	198.0	4	386	198.0	4	400	200.0	6	451
neos-5093327-huahum	11298.0	882	1308	9876.0	1003	1422	9936.0	560	865
neos-5104907-jarama	N/A	47	1800	N/A	60	1800	N/A	71	1800
neos-5107597-kakapo	105534.0	2097	243	57438.0	4466	486	N/A	15996	1800
neos-5114902-kasavu	N/A	29	1800	N/A	28	1800	N/A	42	1800
neos-5188808-nattai	6.14	6	3	6.14	6	3	N/A	2410	1800
neos-5195221-niemur	N/A	835	1800	0.2	160	312	N/A	2415	1800
neos-631710	N/A	571	1800	N/A	580	1800	N/A	543	1801
neos-787933	316.0	18	72	316.0	17	65	N/A	852	1801
neos-827175	121.0	4	2	121.0	4	2	121.0	4	2
neos-848589	29648194.7	2	5	29648194.7	2	5	N/A	886	1801
neos-860300	6292.0	16	3	7128.0	6	1	5230.0	5	1
neos-873061	270.15	9	66	240.35	7	61	262.78	7	68
neos-911970	207.8	2	0	207.8	2	0	298.26	14	0
neos-933966	77297.0	2	0	77297.0	2	0	16377.0	10	4

neos-957323	-228.77	2	1	-228.77	2	1	-228.77	2	1
neos-960392	-198.0	10	8	-204.0	14	11	-204.0	10	8
neos17	0.92	3	0	0.92	3	0	N/A	77087	1800
neos5	26.0	2	0	26.0	2	0	26.0	2	0
net12	337.0	10	1	337.0	20	2	337.0	13	2
netdiversion	493.0	35	452	4020165.0	289	1151	N/A	208	1800
nexp-150-20-8-5	591.0	6	1	617.0	6	1	580.0	10	2
ns1116954	N/A	860	1800	N/A	1025	1800	N/A	808	1800
ns1208400	N/A	11529	1800	N/A	17515	1800	N/A	13641	1800
ns1644855	-500.0	3	225	-500.0	3	212	-500.0	3	226
ns1760995	N/A	124	1800	N/A	87	1801	N/A	70	1800
ns1830653	N/A	23860	1800	N/A	27061	1800	N/A	25149	1800
nw04	19792.0	2	2	19792.0	2	2	19882.0	2	2
opm2-z10-s4	-1874.0	2	37	-1874.0	2	37	-1874.0	2	37
p200x1188c	94934.0	3	0	47026.0	3	0	83779.0	3	0
peg-solitaire-a3	N/A	13964	1800	N/A	14779	1800	N/A	14098	1800
pg	-4329.66	4	0	-3393.39	4	0	-4056.99	4	0
pg5_34	-10840.58	2	0	-10840.58	2	0	-10840.58	2	0
physiciansched3-3	N/A	14	1800	N/A	14	1800	N/A	3	1800
physiciansched6-2	N/A	535	1800	N/A	512	1800	N/A	474	1800
pk1	109.0	2	0	109.0	2	0	N/A	850580	1800
qap10	472.0	3	12	524.0	3	10	470.0	3	22
rail01	-14.46	70	414	N/A	533	1800	-14.45	286	967
rail02	-139.58	2	838	-139.58	2	859	-139.58	2	861
rail507	179.0	3	3	179.0	3	3	179.0	3	4
ran14x18-disj-8	8755.0	5	0	8897.0	5	0	6615.0	6	0
rd-rplusc-21	N/A	2117	1800	N/A	2498	1801	N/A	1122	1800
reblock115	-16885689.13	64	3	N/A	43719	1800	-1539863.93	40	2
rmatr100-p10	755.0	2	0	755.0	2	0	691.0	2	0
rmatr200-p5	6425.0	2	12	6425.0	2	13	4717.0	206	1015
rocII-5-11	N/A	1639	1800	N/A	1866	1800	N/A	1762	1800
roi2alpha3n4	0.0	577	117	0.0	39	8	0.0	36	8
roi5alpha10n8	N/A	967	1802	N/A	1054	1800	N/A	1018	1800
s100	N/A	26	1800	N/A	95	1802	N/A	14	1800
s250r10	2.84	5	153	2.84	5	150	2.84	5	150

satellites2-40	92.0	53	46	92.0	112	114	92.0	145	147	
satellites2-60-fs	87.0	63	50	47.0	124	129	87.0	99	86	
savsched1	15038.0	215	899	15416.4	199	820	N/A	235	1800	
seymour	476.0	6	0	476.0	6	0	475.0	5	0	
seymour1	1163.0	2	0	1163.0	2	0	1163.0	2	0	
sing326	79143558.84	2	6	79143558.84	2	5	7353910 4.01	2	6	
sing44	89002621.68	2	5	89002621.68	2	4	81406679.01	2	5	
sorrell3	-3.0	52	43	-2.0	55	48	-1.0	57	48	
sp150x300d	70.0	5	0	88.0	7	0	76.0	7	0	
sp97ar	1416984682.56	7	3	1418589197.76	7	3	1745313155.84	5	2	
sp98ar	966933789.28	6	3	986612819.04	6	3	9428789205.76	14	8	
supportcase10	N/A	34	1800	N/A	36	1800	N/A	33	1800	
supportcase18	N/A	8496	1800	N/A	9725	1800	N/A	8760	1800	
supportcase22	N/A	262	1800	N/A	295	1800	N/A	2219	1800	
supportcase26	2201.88	2	0	2201.88	2	0	2201.88	2	0	
supportcase40	82002.6	2	0	82002.6	2	0	1160628.89	2	1	
swath1	1303.07	3	0	2499.86	3	0	2357.21	4	0	
swath3	1136.19	4	0	1474.53	6	0	2359.78	6	0	
tbfp-network	156.13	18	111	181.88	21	121	89.79	12	103	
tr12-30	258884.0	467	5	274083.0	103	1	274960.0	56	0	
trento1	10000000000.0	2	5	10000000000.0	2	5	266912080.67	3	5	
uccase12	1008620.62	11	12	1008494.55	11	11	592625.19	3	16	
uct-subprob	2245.0	2	0	2245.0	2	0	2205.0	2	0	
unitcal_7	N/A	4259	1800	N/A	2479	1800	N/A	3341	1800	
var-smallemery-m6j6	-97.44	2	2	-97.44	2	2	7724.79	2	2	

Table 5: Results for Configs 4-6

	config4			config5			config6		
name	ObjVal	nIt	Time	ObjVal	nIt	Time	ObjVal	nIt	Time
CMS750_4	482.0	59	143	482.0	59	142	482.0	59	141
academictimetablesmall	N/A	956	1800	N/A	858	1800	N/A	485	1800
air05	30822.0	17	2	30822.0	17	2	30822.0	14	1
app1-1	-3.0	229	44	N/A	1699	1800	N/A	1779	1800
app1-2	N/A	277	1801	N/A	272	1800	N/A	273	1800
assign1-5-8	520.0	2	0	520.0	2	0	520.0	2	0
b1c1s1	63530.42	42	6	63530.42	42	7	63530.42	42	6
bab2	N/A	149	1801	N/A	144	1800	N/A	97	1800
bab6	N/A	152	1800	N/A	151	1800	N/A	140	1800
beasleyC3	857.0	111	6	857.0	111	6	857.0	111	6
binkar10_1	1010547.59	6	0	1010547.59	6	0	1010547.59	6	0
blp-ar98	N/A	776	1801	14664.89	186	224	15830.02	414	1119
blp-ic98	11200.34	9	2	11200.34	9	2	11200.34	9	2
bnatt400	N/A	1826	1800	N/A	1856	1800	N/A	1745	1800
bnatt500	N/A	1494	1800	N/A	1562	1800	N/A	1513	1800
bppc4-08	132.0	2	0	132.0	2	0	132.0	2	0
cbs-cta	33486825271.26	2	0	33486825271.26	2	1	33486825271.26	2	0
chromaticindex1024-7	4.0	3	5	4.0	3	6	4.0	3	6
chromaticindex512-7	4.0	3	4	4.0	3	4	4.0	3	4
cmflsp50-24-8-8	64895828.7	61	85	64895828.7	61	87	64904751.16	62	84
co-100	N/A	410	1801	N/A	318	1801	N/A	280	1801
cod105	0.0	2	2	0.0	2	3	0.0	2	2
cost266-UUE	39599897.68	18	1	39599897.68	18	1	39599897.68	18	1
csched007	N/A	3915	1800	N/A	2857	1800	N/A	2594	1800
csched008	189.0	450	60	189.0	450	61	186.0	2082	1382
cvs16r128-89	-2.0	2	0	-2.0	2	0	-2.0	2	0
dano3_3	1000.0	25	24	1000.0	9	15	1000.0	9	16
dano3_5	1000.0	4	14	1000.0	4	16	1000.0	4	15
drayage-100-23	N/A	792	1800	N/A	793	1800	N/A	774	1800
drayage-25-23	N/A	771	1800	N/A	791	1801	N/A	774	1800
dws008-01	N/A	1400	1800	N/A	1199	1800	N/A	1228	1800

eil33-2	1539.21	64	8	1539.21	64	7	1364.33	60	7
eil51A101-2	N/A	361	1800	1744.73	216	1046	1751.42	227	1176
ex10	N/A	5	1800	N/A	3	1800	N/A	3	1800
ex9	N/A	3	1800	N/A	3	1800	N/A	3	1800
exp-1-500-5-5	666788.0	2	0	666788.0	2	0	666788.0	2	0
fast0507	185.0	6	4	185.0	6	4	183.0	6	4
fastxgemm-n2r6s0t2	6472.72	13	1	6472.72	13	0	6472.72	13	1
fhnw-binpack4-4	N/A	5827	1800	N/A	3263	1800	N/A	3029	1800
fhnw-binpack4-48	N/A	1331	1800	N/A	1305	1800	N/A	1351	1800
glass-sc	29.0	5	0	29.0	5	0	29.0	5	0
glass4	N/A	8338	1800	N/A	5156	1800	N/A	4597	1800
gmu-35-40	-2020035.47	113	4	-2020035.47	113	4	-2020035.47	113	3
gmu-35-50	-2057442.02	23	0	-2057442.02	23	0	-2057442.02	23	0
graph20-20-1rand	-7.0	18	1	-7.0	18	1	-6.0	14	0
h80x6320d	N/A	2024	1801	13828.59	58	31	13828.59	58	30
irish-electricity	N/A	214	1800	N/A	212	1800	N/A	223	1800
irp	12185.87	5	1	12185.87	5	1	12185.87	5	1
istanbul-no-cutoff	290.01	2	6	290.01	2	6	290.01	2	5
leo1	668440248.96	8	1	668440248.96	8	1	668440248.96	8	1
leo2	625513308.12	7	2	625513308.12	7	2	625513308.12	7	2
lotsize	N/A	4422	1800	3631054.0	617	221	3631054.0	617	223
mad	2.47	2	0	2.47	2	0	2.47	2	0
map10	0.0	2	10	0.0	2	10	0.0	2	10
map16715-04	0.0	2	11	0.0	2	9	0.0	2	8
markshare2	693.0	4	0	693.0	4	0	693.0	4	0
markshare_4_0	91.0	2	0	91.0	2	0	91.0	2	0
mas74	19142.39	2	0	19142.39	2	0	19142.39	2	0
mas76	48566.14	2	0	48566.14	2	0	48566.14	2	0
mc11	37721.0	191	14	37721.0	191	14	37721.0	191	13
milo-v12-6-r2-40-1	480761.1	70	12	480761.1	70	11	480761.1	70	13
momentum1	N/A	253	1800	N/A	324	1800	N/A	254	1800
n2seq36q	69800.0	13	6	69800.0	13	6	69800.0	14	7
n3div36	344200.0	6	1	344200.0	6	1	344200.0	6	2
neos-1122047	162.0	3	12	162.0	3	10	162.0	3	11
neos-1171448	0.0	2	0	0.0	2	0	0.0	2	0

neos-1171737	0.0	2	0	0.0	2	0	0.0	2	0
neos-1445765	N/A	1777	1800	N/A	1785	1800	N/A	1750	1800
neos-2075418-temuka	N/A	2	1800	N/A	2	1800	N/A	2	1800
neos-2978193-inde	44.53	2	0	44.53	2	0	44.53	2	0
neos-2987310-joes	321815756.4	11	11	321815756.4	11	11	-483321246.81	7	13
neos-3216931-puriri	N/A	1688	1800	N/A	1355	1800	N/A	1357	1800
neos-3402294-bobin	N/A	213	1800	N/A	212	1800	N/A	210	1800
neos-3402454-bohle	N/A	54	1800	N/A	53	1801	N/A	56	1801
neos-3555904-turama	N/A	231	1800	N/A	230	1801	N/A	192	1800
neos-3627168-kasai	1006772.44	18	0	1006772.44	18	0	1006772.44	18	0
neos-3754480-nidda	29227.37	2	0	29227.37	2	0	29227.37	2	0
neos-3988577-wolgan	N/A	280	1800	N/A	374	1800	N/A	230	1800
neos-4300652-rahue	N/A	200	1800	N/A	185	1800	N/A	136	1800
neos-4387871-tavua	N/A	1975	1800	N/A	1927	1800	N/A	1914	1800
neos-4413714-turia	N/A	162	1801	N/A	94	1800	N/A	93	1802
neos-4532248-waihi	N/A	250	1800	N/A	249	1801	N/A	239	1801
neos-4647030-tutaki	39830.21	21	89	39830.21	21	88	39830.21	21	88
neos-4763324-toguru	5626.97	2	11	5626.97	2	11	5626.97	2	12
neos-4954672-berkel	11182960.0	2	0	11182960.0	2	0	11182960.0	2	0
neos-5049753-cuanza	N/A	35	1800	N/A	37	1800	N/A	42	1801
neos-5052403-cygnet	198.0	4	388	198.0	4	400	198.0	4	393
neos-5093327-huahum	9608.0	38	70	9852.0	57	107	9852.0	57	109
neos-5104907-jarama	N/A	8	1800	N/A	8	1800	N/A	7	1800
neos-5107597-kakapo	N/A	916	1801	N/A	934	1800	N/A	935	1800
neos-5114902-kasavu	N/A	21	1800	N/A	22	1801	N/A	22	1801
neos-5188808-nattai	5.19	13	9	5.19	13	9	5.49	12	9
neos-5195221-niemur	N/A	1011	1800	0.14	106	334	0.17	45	164
neos-631710	266.0	157	1475	266.0	157	1440	266.0	85	556
neos-787933	316.0	216	1371	316.0	216	1233	316.0	216	1372
neos-827175	121.0	12	7	121.0	12	6	121.0	12	7
neos-848589	29648194.7	2	5	29648194.7	2	5	29648194.7	2	5
neos-860300	6069.0	6	1	6069.0	6	1	6069.0	6	1
neos-873061	239.2	89	1010	239.2	89	831	239.2	89	730
neos-911970	207.8	2	0	207.8	2	0	207.8	2	0
neos-933966	77297.0	2	0	77297.0	2	0	77297.0	2	0

neos-957323	-228.77	2	1	-228.77	2	1	-228.77	2	1	
neos-960392	N/A	528	1800	-204.0	129	226	-204.0	125	210	
neos17	0.85	21	0	0.85	21	0	0.85	21	0	
neos5	26.0	2	0	26.0	2	0	26.0	2	0	
net12	337.0	56	10	337.0	56	9	316.0	81	15	
netdiversion	491.0	18	183	491.0	18	186	491.0	18	183	
nexp-150-20-8-5	576.0	64	51	576.0	64	53	576.0	64	53	
ns1116954	N/A	940	1800	N/A	512	1800	N/A	512	1800	
ns1208400	N/A	4427	1800	N/A	1500	1800	N/A	1398	1800	
ns1644855	-500.0	3	224	-500.0	3	230	-500.0	3	224	
ns1760995	-150.95	88	1368	-168.11	48	1043	-168.11	48	1011	
ns1830653	N/A	31241	1800	N/A	2721	1800	N/A	1834	1800	
nw04	19792.0	2	2	19792.0	2	2	19792.0	2	2	
opm2-z10-s4	-1874.0	2	36	-1874.0	2	38	-1874.0	2	39	
p200x1188c	17028.0	6	0	17028.0	6	0	17028.0	6	0	
peg-solitaire-a3	N/A	3711	1800	N/A	1680	1800	N/A	748	1800	
pg	-5310.96	4	0	-5310.96	4	0	-5310.96	4	0	
pg5_34	-10840.58	2	0	-10840.58	2	0	-10840.58	2	0	
physicianssched3-3	N/A	15	1800	N/A	14	1800	N/A	14	1800	
physicianssched6-2	N/A	200	1800	N/A	209	1800	N/A	199	1800	
pk1	109.0	2	0	109.0	2	0	109.0	2	0	
qap10	550.0	3	13	N/A	3	15	N/A	3	14	
rail01	N/A	394	1800	N/A	240	1800	-14.46	55	1090	
rail02	-139.58	2	881	-139.58	2	866	-139.58	2	834	
rail507	179.0	3	3	179.0	3	3	179.0	3	3	
ran14x18-disj-8	4747.0	10	0	4747.0	10	0	4747.0	10	0	
rd-rplusc-21	N/A	1742	1800	N/A	1616	1800	N/A	1694	1800	
reblock115	-17475307.28	56	3	-17854773.9	27	1	-17754872.8	77	5	
rmatr100-p10	755.0	2	0	755.0	2	0	755.0	2	0	
rmatr200-p5	6425.0	2	14	6425.0	2	13	6425.0	2	11	
rocII-5-11	N/A	620	1800	N/A	500	1800	N/A	452	1800	
roi2alpha3n4	0.0	57	15	0.0	57	16	0.0	21	6	
roi5alpha10n8	N/A	260	1800	N/A	261	1800	N/A	243	1801	
s100	N/A	67	1800	N/A	53	1800	N/A	53	1800	
s250r10	2.83	5	169	2.83	5	163	2.83	5	170	

satellites2-40	N/A	523	1801	N/A	534	1800	N/A	528	1801	
satellites2-60-fs	N/A	555	1801	N/A	574	1800	N/A	548	1800	
savsched1	16815.4	19	100	16815.4	19	97	16815.4	19	97	
seymour	476.0	6	0	476.0	6	0	474.0	6	0	
seymour1	1163.0	2	0	1163.0	2	0	1163.0	2	0	
sing326	79143558.84	2	6	79143558.84	2	6	79143558.84	2	6	
sing44	89002621.68	2	5	89002621.68	2	4	89002621.68	2	5	
sorrell3	-1.0	11	8	-1.0	11	7	-1.0	11	8	
sp150x300d	70.0	62	0	70.0	62	0	70.0	62	0	
sp97ar	1087742657.6	10	4	1087742657.6	10	4	1153986523.52	11	5	
sp98ar	794324109.92	9	5	794324109.92	9	4	794324109.92	8	4	
supportcase10	N/A	20	1800	N/A	20	1800	N/A	18	1800	
supportcase18	N/A	1113	1801	71.0	376	400	67.0	462	618	
supportcase22	N/A	102	1800	N/A	105	1800	N/A	107	1800	
supportcase26	2201.88	2	0	2201.88	2	0	2201.88	2	0	
supportcase40	82002.6	2	0	82002.6	2	0	82002.6	2	0	
swath1	1214.88	6	0	2260.4	7	0	2260.4	7	0	
swath3	2399.18	11	0	2399.18	11	0	3292.45	14	1	
tbfp-network	103.29	7	39	103.29	7	39	103.29	7	38	
tr12-30	290602.0	48	1	290602.0	48	1	290602.0	48	1	
trento1	10000000000.0	2	5	10000000000.0	2	5	10000000000.0	2	5	
uccase12	1007203.91	7	10	1007203.91	7	8	1007203.91	7	8	
uct-subprob	2245.0	2	0	2245.0	2	0	2245.0	2	0	
unitcal_7	N/A	731	1800	N/A	663	1800	N/A	662	1801	
var-smallemery-m6j6	-97.44	2	2	-97.44	2	2	-97.44	2	2	

7 Conclusion and Final Remarks

Algorithm 2 or its modified versions did not out-perform the original feasibility pump from [11]. The advantage they give in terms of cutting out parts of the feasible region for the continuous relaxation of a given 0-1 MILP seems to be outweighed by the performance cost of the extra constraints introduced throughout the execution of the algorithms, which slow down the optimizer used to solve linear programs and lead to significantly fewer iterations within the time limit.

With respect to the distance function 1, neither the rounding step or the projection step is guaranteed to have unique solutions. Without this guarantee, *Lemma 3* of [12] fails, and algorithm 2 is susceptible to cycles of length 2 or more, for which it has no procedure in place. Config5 and config6 (see Table 2) address this issue, but it could also be resolved by the use of a different function to represent distances between points that guarantees unique solutions for the projection and/or rounding steps. However, such a function may not be linear and it may result in slower iterations.

Another observed issue that led to cycling involved the constraint 2 not being able to cut out certain points from the feasibility region of a continuous relaxation when $\lceil \beta \rceil - \beta$ is too small. This was also addressed by config5 and config6, but the corresponding algorithms may cut out some feasible points from the program.

8 References

[1] Miplib 2017 - the mixed integer programming library. https://miplib.zib.de/download.html.

[2] Tobias Achterberg and Timo Berthold. Improving the feasibility pump. *Discrete Optimization*, 4(1):77–86, 2007.

[3] Hadi Charkhgard Aritra Pal. A feasibility pump and local search based heuristic for bi-objective pure integer linear programming. *INFORMS Journal on Computing*, 31(1):115–133, 2019.

[4] Hadi Charkhgard Aritra Pal. FPBH: A feasibility pump based heuristic for multi-objective mixed integer linear programming. *Computers & Operations Research*, 112, 2019.

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

[6] Timo Berthold, Andrea Lodi, and Domenico Salvagnin. Ten years of feasibility pump, and counting. *EURO Journal on Computational Optimization*, 7(1), 2018.

[7] Pierre Bonami, Gérard Cornuéjols, Andrea Lodi, and François Margot. A feasibility pump for mixed integer nonlinear programs. *Mathematical Programming*, 119(2):331–352, 2009.

[8] Pierre Bonami and João Gonçalves. Heuristics for convex mixed integer nonlinear programs. *Computational Optimization and Applications*, 51:729–747, 2010.

[9] Claudia D'Ambrosio, Frangioni Antonio, Leo Liberti, and Andrea Lodi. A storm of feasibility pumps for nonconvex minlp. *Mathematical Programming*, 136, 2010.

[10] Matteo Fischetti. *Introduction to Mathematical Optimization*. 2019.

[11] Matteo Fischetti, Fred Glover, and Andrea Lodi. The feasibility pump. *Mathematical Programming*, 104(1):91–104, 2005.

[12] Björn Geißler, Antonio Morsi, Lars Schewe, and Martin Schmidt. Penalty alternating direction methods for mixed-integer optimization: A new view on feasibility pumps. *SIAM Journal on Optimization*, 27(3):1611–1636, 2017.

[13] Min Li and Qian Liu. Inexact feasibility pump for mixed integer nonlinear programming. *Information Processing Letters*, 118:110–116, 2017.

[14] Shaurya Sharma, Brage Knudsen, and Bjarne Grimstad. Towards an objective feasibility pump for convex MINLPs. *Computational Optimization and Applications*, 63:737–753, 2016.

A Appendix

```
import gurobipy as gp
from gurobipy import GRB

from enum import Enum

from random import randint, uniform
from time import time

import argparse

import json
import os


class Mode(Enum):
    RANDOM = 1      # Original FP cycle handling (configs 1-3)
    SMART = 2       # configs (4-6)

class FixedLenQueue:
    """Data structure to handle fixed length queues. Useful for keeping track of
    previous objective function coefficients"""
    def __init__(self, queueLen : int):
        # main storage of the items
        self.queue = [None] * queueLen
        # points to the item at the head of the queue (the earliest added item)
        self.queueHeadIndex = 0

        self.queueLen = queueLen

    def get(self, virtualIndex : int):
        """Get the item at the given virtual index. Virtual index 0 is the
        earliest added item of the currently stored items"""
        realIndex = (virtualIndex + self.queueHeadIndex) % self.queueLen
        return self.queue[realIndex]

    def push(self, item):
        """Remove the earliest added item, replace it with this item and
        increment queueHeadIndex"""
        self.queue[self.queueHeadIndex] = item
        self.queueHeadIndex += 1
        self.queueHeadIndex %= self.queueLen


class FP:
```

```
def __init__(self, model : gp.Model, mode : Mode, timeLimit : int,
             verbose : bool, randomFlipParam : int,
             maxCycleLen : int, magic : bool, flip_smart : bool,
             incr_k : bool, force1 : bool):

    self.model = model

    self.relaxed = model.relax()

    integral = []
    for i, var in enumerate(model.getVars()):
        if var.VType == GRB.BINARY or var.VType == GRB.INTEGER:
            integral.append(i)

    self.integralVarIndices = set(integral)

    self.intFeasTol = model.params.IntFeasTol
    self.vars = self.relaxed.getVars()

    # keep track of the most recent objective function coefficients
    self.ObjCoeffs = None

    # Data structure to keep track of the last maxCycleLen objective
    # function coefficients to detect cycles of length <= maxCycle len
    self.maxCycleLen = maxCycleLen
    self.prevObjCoeffsQueue = FixedLenQueue(maxCycleLen)

    self.verbose = verbose

    # cycle handling mode
    self.mode = mode

    self.timeLimit = timeLimit  # in seconds

    # parameter to help determine the number of variables to
    # flip in the random mode
    self.randomFlipParam = randomFlipParam

    self.magic = magic # whether or not to add the "magic" constraint

    # random flipping to escape cycles in smart mode
    self.flip_smart = flip_smart

    # increment the lower bound when obj val is too close to
    # ceiling in smart mode 1-cycle handling
    self.incr_k = incr_k

    # Force 1-cycle when ObjVal doesn't drop from rounding
    self.force1 = force1
```

```
def checkIntFeas(self) -> bool:
    """Check if all integral variables in the original model have integer
    values (upto the tolerance parameter)"""

    for i, var in enumerate(self.vars):
        if i not in self.integralVarIndices:
            continue
        rounding = int(var.X + 0.5)
        if abs(var.X - rounding) > self.intFeasTol:
            return False

    return True

def generateNewObjective(self) -> tuple[float, float]:
    """Generate the coefficents of a linear expression representing the l1
    distance from the coordinate-wise rounding of the vector
    considering only integral variables of the original model.
    It is assumed that the model being considered only consists of binary
    and continuous variables. Return L1 and L-inf distance from rounding"""

    self.ObjCoeffs = []
    l1 = 0
    linf = -1
    for i, var in enumerate(self.vars):
        if i not in self.integralVarIndices:
            self.ObjCoeffs.append(0)
            continue
        rounding = int(var.X + 0.5)
        diff = abs(var.X - rounding)
        l1 += diff
        linf = max(linf, diff)
        self.ObjCoeffs.append(1 - 2 * rounding)
    if self.verbose:
        print(f"Rounding with l1 = {l1}, Linf = {linf}")

    return l1, linf

def flipkMostFractional(self, k : int) -> None:
    """flip the coefficients of the k most fractional ones from the
    integral variables of the original model"""

    sortedIndices = sorted(list(self.integralVarIndices),
                           key = lambda i : abs(self.vars[i].X - 0.5))

    for j in range(min(k, len(sortedIndices))):
        i = sortedIndices[j]
        self.ObjCoeffs[i] *= -1          # flip coefficient
```

```
        if self.verbose:
            print(f"Flipped {k} variables")


    def updateObjective(self) -> None:
        """update the objective function of the relaxed model based on the
        current coefficents and constant"""

        obj = gp.LinExpr(self.ObjCoeffs, self.vars)
        obj.addConstant(self.ObjCoeffs.count(-1))

        self.relaxed.setObjective(obj, GRB.MINIMIZE)
        self.relaxed.update()


    def cyclesLen1(self) -> int:
        """Detect and handle cycles of length 1. Return the number of
        variables to flip"""
        if self.prevObjCoeffsQueue.get(-1) == self.ObjCoeffs:
                # Handle it based on the mode selected
                if self.mode == Mode.RANDOM:
                    k = randint(self.randomFlipParam // 2,
                                (self.randomFlipParam // 2) * 3)

                elif self.mode == Mode.SMART:
                    # ceiling of objective value.
                    k = int(self.relaxed.ObjVal + 1)

                    # Handle case where additional constraint does nothing
                    if self.incr_k and \
                        abs(k - self.relaxed.ObjVal) < \
                            self.relaxed.params.FeasibilityTol:
                        if self.verbose:
                            print(f"Objective value of \
                                {self.relaxed.ObjVal} too close to ceiling")
                        k += 1

                return k
        return 0

    def randomizedFlipAll(self) -> None:
        """Make random decisions on whether or not to flip each coefficient"""
        flipped = 0
        for j, var in enumerate(self.vars):
            if j not in self.integralVarIndices:
                continue
            pj = uniform(-0.3, 0.7)
            rounding = int(var.X + 0.5)
            if abs(rounding - var.X) + max(pj, 0) > 0.5:
```

```
                self.ObjCoeffs[j] *= -1
                flipped += 1

        if self.verbose:
            print(f"Flipped {flipped} variables")

    def cyclesLenGt1(self) -> int:
        """Detect cycles of 2 <= length <= maxCycleLen. Return length of
        cycle (0 if no cycle)"""

        for i in range(self.maxCycleLen - 2, -1, -1):
            if self.ObjCoeffs == self.prevObjCoeffsQueue.get(i):
                return self.maxCycleLen - i # length of the cycle detected
        return 0



    def run(self) -> dict[str]:
        """run the iterations of FP with the specified cycle handling mode.
        The model given is assumed to be a binary program."""

        startTime = time()
        elapsed = 0
        # keep original objective as secondary objective with 0 weight to
        # query its value when desired
        self.relaxed.setObjectiveN(self.relaxed.getObjective(), 1, 0, 0.0)

        self.relaxed.params.OutputFlag = 0
        self.relaxed.params.Threads = 1

        iterCount = 0

        summary = dict()

        summary["sense"] = self.model.ModelSense
        summary["name"] = self.model.ModelName
        summary["nvars"] = len(self.vars)
        summary["nconstraints"] = len(self.relaxed.getConstrs())
        summary["nIntegral"] = len(self.integralVarIndices)

        summary["mode"] = self.mode.name
        summary["cycles"] = []
        summary["LPsolTimes"] = []
        summary["ObjVals"] = []
        summary["AdditionalConstrs"] = []

        if self.verbose:
            print(f"running FP in {self.mode.name} mode on a model with \
                {len(self.vars)} variables and {summary['nconstraints']} \
```

```
                    constraints. Time Limit {self.timeLimit}")

        lastC11Iter = -1

        while elapsed < self.timeLimit:
            iterCount += 1
            # main loop for FP iterations

            # in seconds
            self.relaxed.params.TimeLimit = self.timeLimit - elapsed

            self.relaxed.update()

            t = time()
            # Step 1 solve the LP relaxation
            self.relaxed.optimize()
            dt = time() - t
            summary["LPsolTimes"].append(dt)
            if self.verbose:
                print(f"Solved LP in {dt} seconds")

            if self.relaxed.Status == GRB.TIME_LIMIT:
                if self.verbose:
                    print(f"Failed to find feasible solution in {iterCount} \
                    iterations and {int(time() - startTime)} seconds")
                break # Time Limit Exceeded

            if self.relaxed.Status == GRB.INFEASIBLE:

                if self.verbose:
                    print(f"infeasible relaxation at iteration {iterCount}")

                summary["infeasible"] = iterCount
                break

            if self.magic and iterCount == 1:
                self.relaxed.addConstr(self.relaxed.getObjective() <= \
                    self.relaxed.ObjVal + (50.5 * abs(self.relaxed.ObjVal)))

            if self.verbose:
                print(f"iteration {iterCount} : Objective function value : \
                    {self.relaxed.ObjVal}")

            summary["ObjVals"].append(self.relaxed.ObjVal)
            # Step 2 check integer feasibility and return if feasible

            if self.checkIntFeas():
                self.relaxed.params.ObjNumber = 1
                if self.verbose:
```

```
                print(f"Found feasible solution with objective value \
                        {self.relaxed.ObjNVal} in {iterCount} iterations \
                            and {int(time() - startTime)} seconds")

            summary["feasSolObjVal"] = self.relaxed.ObjNVal
            summary["foundFeas"] = True
            summary["iterationsCount"] = iterCount
            summary["elapsedTime"] = int(time() - startTime)
            return summary

        # Step 3 generate objective based on rounding. Force 1-cycle
        # when the option is active and ObjVal doesn't drop
        l1, _ = self.generateNewObjective()

        if iterCount > 1 and self.mode == Mode.SMART and self.force1 \
            and abs(l1 - self.relaxed.ObjVal) < \
                self.relaxed.params.FeasibilityTol:
            self.ObjCoeffs = self.prevObjCoeffsQueue.get(-1)

        # Step 4 check for cycles of length 1 and handle them
        cl1 = self.cyclesLen1()
        if cl1:
            summary["cycles"].append((1, iterCount))
            if self.verbose:
                print(f"Detected cycle of length 1 at iteration {iterCount}")

            if self.mode == Mode.SMART:
                # add constraint
                self.relaxed.addConstr(self.relaxed.getObjective() >= cl1).\
                Lazy = 0
                summary["AdditionalConstrs"].append(iterCount)
                if self.verbose:
                    print(f"Additional constraint added at iteration \
                            {iterCount}")


            # choose new integer point and corresponding objective
            self.flipkMostFractional(cl1)
            lastCl1Iter = iterCount

        # Step 5 check for cycles of length >= 2 and handle them
        clgt1 = self.cyclesLenGt1()
        if clgt1:
            summary["cycles"].append((clgt1, iterCount))
            if self.verbose:
                print(f"Detected cycle of length {clgt1} at iteration \
                        {iterCount}")

            if self.mode == Mode.RANDOM:
```

```
                    self.randomizedFlipAll()

                if self.mode == Mode.SMART and self.flip_smart and \
                    iterCount - clgt1 > lastCl1Iter:
                    self.randomizedFlipAll()

            # Step 6 update the objective funtion based on the current
            # coefficients
            self.updateObjective()

            self.prevObjCoeffsQueue.push(self.ObjCoeffs)

            elapsed = int(time() - startTime)

        summary["foundFeas"] = False
        summary["iterationsCount"] = iterCount
        summary["elapsedTime"] = int(time() - startTime)
        return summary     # No feasible solution found



parser = argparse.ArgumentParser(description="Program to run the original \
    or a slightly modified verion of Feasibility Pump")
parser.add_argument("filename", help="file containing an encoding of \
    a model. e.g. an mps file")
parser.add_argument("-m", "--mode", choices=['random', 'smart'],
                    default='random', help="cycle handling scheme to use")
parser.add_argument("-t", "--time_limit", type=int,
                    help="time limit in seconds", default=1800)
parser.add_argument("-v", "--verbose", help="print information \
    about objective function values, cycle detection etc. at \
        each iteration", action="store_true")
parser.add_argument("-g", "--magic", help="add the magic constraint",
                    action="store_true")
parser.add_argument("-r", "--random_flip_param",
                    help="parameter to determine number of \
                        variables to flip when a cycle is found \
                            in random mode", type=int, default=20)
parser.add_argument("-c", "--max_cycle_len",
                    help="maximum length of cycle to detect",
                    type=int, default=3)
parser.add_argument("-s", "--summary_folder",
                    help="where to store json file generated",
                    default="summaries")
parser.add_argument("-f", "--flip_smart",
                    help="flip some vars when a cycle is \
                        encountered in smart mode", action="store_true")
parser.add_argument("-i", "--incr_k",
                    help="handle contraints that do nothing \
```

```
                        due to obj vals very close to their ceiling \
                        in smart mode", action="store_true")
parser.add_argument("-o", "--force1", help="force 1-cycle when ObjVal \
    doesn't drop in smart mode", action="store_true")
args = parser.parse_args()

filename : str = args.filename
time_limit : int = args.time_limit
verbose : bool = args.verbose
magic : bool = args.magic
max_cycle_len : int = args.max_cycle_len
random_flip_param : int = args.random_flip_param
summary_folder_path : str = args.summary_folder
flip_smart : bool = args.flip_smart
incr_k : bool = args.incr_k
force1 : bool = args.force1

if args.mode == "random":
    mode = Mode.RANDOM
elif args.mode == "smart":
    mode = Mode.SMART

model = gp.read(filename)

fp = FP(model, mode, time_limit, verbose, random_flip_param, max_cycle_len,
        magic, flip_smart, incr_k, force1)

summary = fp.run()

if not os.path.exists(summary_folder_path):
    os.makedirs(summary_folder_path)
jsonFileName = filename.split('/')[-1].split('.')[0]
jsonFileName = f"{summary_folder_path}/{jsonFileName}_{args.mode}_\
    {time_limit}_{'magic' if magic else 'no_magic'}_{max_cycle_len}_\
        {random_flip_param}_{'incrk' if incr_k else 'no_incrk'}_\
            {'flip_smart' if flip_smart else 'no_flip_smart'}_\
                {'force1' if force1 else 'no_force1'}.json"

with open(jsonFileName, 'w') as f:
    json.dump(summary, f)
    print(f"Wrote summary to {jsonFileName}")
```

Table 6: Server Specifications

Architecture	x86_64
CPU op-mode(s)	32-bit, 64-bit
Byte Order	Little Endian
CPU(s)	128
On-line CPU(s) list	0-127
Thread(s) per core	2
Core(s) per socket	32
Socket(s)	2
NUMA node(s)	2
Vendor ID	GenuineIntel
CPU family	6
Model	106
Model name	Intel(R) Xeon(R) Platinum 8362 CPU @ 2.80GHz
Stepping	6
CPU MHz	2800.000
BogoMIPS	5600.00
L1d cache	48K
L1i cache	32K
L2 cache	1280K
L3 cache	49152K
NUMA node0 CPU(s)	0-31,64-95
NUMA node1 CPU(s)	32-63,96-127
Flags	fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm constant_tsc art arch_perfmon pebs bts rep_good nopl xtopology nonstop_tsc cpuid aperfmperf pni pclmulqdq dtes64 monitor ds_cpl smx est tm2 ssse3 sdbg fma cx16 xtpr pdcm pcid dca sse4_1 sse4_2 x2apic movbe popcnt tsc_deadline_timer aes xsave avx f16c rdrand lahf_lm abm 3dnowprefetch cpuid_fault epb cat_l3 invpcid_single ssbd mba ibrs ibpb stibp ibrs_enhanced fsgsbase tsc_adjust bmi1 avx2 smep bmi2 erms invpcid cqm rdt_a avx512f avx512dq rdseed adx smap avx512ifma clflushopt clwb intel_pt avx512cd sha_ni avx512bw avx512vl xsaveopt xsavec xgetbv1 xsaves cqm_llc cqm_occup_llc cqm_mbm_total cqm_mbm_local split_lock_detect wbnoinvd dtherm ida arat pln pts avx512vbmi umip pku ospke avx512_vbmi2 gfni vaes vpclmulqdq avx512_vnni avx512_bitalg tme avx512_vpopcntdq la57 rdpid fsrm md_clear pconfig flush_l1d arch_capabilities