



NILS-CHRISTIAN KEMPKE¹, THORSTEN KOCH²

**Fix-and-Propagate Heuristics Using
Low-Precision First-Order LP Solutions
for Large-Scale Mixed-Integer Linear
Optimization**

¹  0000-0003-4492-9818
²  0000-0002-1967-0077

Zuse Institute Berlin
Takustr. 7
14195 Berlin
Germany

Telephone: +49 30 84185-0
Telefax: +49 30 84185-125

E-mail: bibliothek@zib.de
URL: <http://www.zib.de>

ZIB-Report (Print) ISSN 1438-0064
ZIB-Report (Internet) ISSN 2192-7782

Fix-and-Propagate Heuristics Using Low-Precision First-Order LP Solutions for Large-Scale Mixed-Integer Linear Optimization

Nils–Christian Kempke*, Thorsten Koch†

March 3, 2026

Abstract

We investigate the use of low-precision first-order methods (FOMs) within a fix-and-propagate (FP) framework for solving mixed-integer programming problems (MIPs). We employ GPU-accelerated PDLP, a variant of the Primal-Dual Hybrid Gradient (PDHG) method specialized to LP problems, to solve the LP-relaxation of our MIPs to low accuracy. This solution is used to motivate fixings within our FP framework. We evaluate the performance of our heuristic on MIPLIB 2017, demonstrating that low-accuracy LP solutions do not lead to a loss in the quality of the FP heuristic solutions. Further, we use our FP framework to produce high-accuracy solutions for large-scale (up to 243 million nonzeros and 8 million decision variables) unit commitment-based dispatch and expansion planning problems created with the modeling framework REMix. For the largest problems, we can generate solutions with a primal-dual gap of under 2% in less than 4 hours, whereas state-of-the-art commercial solvers cannot produce feasible solutions within 2 days of runtime.


1 Introduction


Linear programming problems (LPs) optimize a linear objective subject to linear constraints:

$$\min_{x \in \mathbb{R}^n} \{c^T x : Ax \leq b, l \leq x \leq u\}, \quad (1)$$

where x are decision variables, $c \in \mathbb{R}^n$ objective, $A \in \mathbb{R}^{m \times n}$ the constraint matrix, $b \in \mathbb{R}^m$ the right-hand sides, and $l, u \in \bar{\mathbb{R}} := \mathbb{R} \cup \{\pm\infty\}$ the variable bounds. LPs are typically solved with Simplex [6] or Interior-Point Methods (IPMs) [19]. Recently, First-Order Methods (FOMs), using gradient information and matrix-vector products, gained attention for large-scale LPs due to their GPU suitability [3, 9, 12].

A mixed-integer program (MIP), extends eq. (1) with integrality constraints on a subset of variables, the *integer variables* or short *integers*: $x_i \in \mathbb{Z} \forall i \in \mathcal{I} \subset \{1, \dots, n\}$. MIPs are NP-hard and the most established solution methods are branch-and-bound approaches relying on IPMs and Simplex methods. Additionally, general purpose MIP

*  0000-0003-4492-9818

†  0000-0002-1967-0077

solvers heavily rely on primal heuristics [1] generating feasible solutions quickly. Fix-and-propagate (FP) heuristics [2] iteratively fix integers and apply domain propagation to reduce the domain of other variables. If the FP heuristic finds a feasible integer assignment, all integers are fixed and the resulting LP is solved generating a MIP feasible solution or aborting the heuristic if the LP is infeasible.

Among FOMs, PDLP has recently been integrated into the Cardinal Optimizer (COPT), Fico Xpress, NVIDIA cuOpt, and Gurobi, enabling GPU accelerated LP solves and providing PDLP as an alternative method for the root LP relaxation of MIPs. Apart from this, little GPU integration has been done into commercial/academic software and this raises the question: can FOMs accelerate heuristics within MIP frameworks while preserving solution quality? Prior work includes Scylla [14], embedding CPU-based PDLP in context of the feasibility pump [7] where already in [7], the possibility of increasing the speed of the heuristic via low-quality LP solves is entertained. During the revision process, cuOpt added a GPU-based feasibility pump using PDLP in combination with propagation and local search [20], as the first commercial standalone GPU-accelerated primal heuristic. Additionally, COPT and cuOpt have both added a GPU accelerated IPM.

Contribution We make three contributions:

- A FP framework exploiting LP solutions of varying accuracy to generate high-quality MIP solutions.
- Performance evaluation on MIPLIB 2017 [8], showing that LP solution accuracy barely affects performance while allowing for speed-ups on large models.
- Application to large-scale energy system optimization models (ESOMs), closing gaps $< 2\%$ within 4 hours, showing speedups up to a factor of 3 over the IPM.

We show with our MIPLIB experiments, that FOMs can readily be embedded into fix-and-propagate type heuristics without sacrificing solution quality. With our ESOM experiments we highlight a class of models where this embedding can be beneficial.

Outline Section 2, details the heuristic framework LP integration strategies. Section 3 introduces our ESOMs. Section 4 presents our MIPLIB and ESOM results.

2 Fix-and-Propagate Using Low-Precision LP Solutions

Our implementation builds on the depth-first-search (*dfs* and *dfsrep*) mode of the fix-propagate-repair framework from [16] where the main FP scheme is described in “Fig. 1 Fix-propagate-repair scheme”. Before running, the framework uses a *variable strategy* sorting the integers and binaries. It then starts the fix-propagate-repair process using a *value strategy* to determine fixing values, fixing variables in the order computed before. We used all propagators from the original code: clique, implication, and linear constraint propagation. We extended the original code in three ways:

- We added LP-based variable strategies (extending Table 1 in [16]): *Frac*, *Duals*, *RedCost*, and *Type* (and several hybrid strategies).
- We added a variable branching strategy better suited for integers, replacing the call in line 14 of Fig. 1 in [16].

- We enabled the use of PDLP (or any other LP solver) for solving LP relaxations within the framework with different tolerances.

For any LP-based variable or value strategy, the framework first solves an *initial LP* using Simplex, IPM, or PDLP. The relative tolerance of the initial solve for PDLP and IPM can be adjusted, for Simplex we use solver’s default tolerances. The LP solution guides variable selection and/or fixing throughout the FP branching procedure. Periodic re-solving to improve the LP solution information is possible in principle but not used here, as it incurs substantial computational cost. After finding a partial assignment of integers, the *final LP* is solved with an IPM to produce a feasible solution.

LP-based variable strategies *Frac* sorts variables by fractionality, the distance of the LP value to the nearest integer: $\text{frac}_i = \min(x_i - \lfloor x_i \rfloor, \lceil x_i \rceil - x_i) < 1$. This strategy is commonly used in diving or FP heuristics [4].

RedCost uses a variable’s reduced costs to determine its ordering. The reduced cost indicates how much the objective would change if the variable moved from its current value. Variables with high absolute reduced cost are less likely to deviate in an optimal solution (assuming the optimum is close to the LP solution value). Greedily fixing these variables first aims to stay close to the objective value.

Dual combines dual constraint values with reduced costs. Intuitively, constraints with large absolute duals often have a strong influence on the objective and tend to be tight in optimal solutions. We sort constraints by absolute dual value, iterate through them, and extract integer variables not yet considered, ordering these by reduced cost. This heuristic prioritizes variables likely to impact the objective associated with tight constraints and follows an intuition rather than a formally proven property.

Type uses the pre-existing variable strategy *type* sorting variables stably, putting binaries first and integers last.

LP-based fixing In our experiments we pair *Frac*, *Redcosts*, *Dual*, and *Type* with an LP-based fixing strategy. Given an integer variable x_i with LP value $x_{LP,i}$, we compute its fractional distance to the lower integer: $d_i = x_{LP,i} - \lfloor x_{LP,i} \rfloor$. We then sample $d \sim U(0,1)$ and fix x_i to its lower bound if $d > d_i$, otherwise to its upper bound. This choice of fixing is meant to stay close to the LP solution while also allowing for some diversification of the search space and avoiding deterministic patterns when running the heuristic more than once or with similar LP solutions. Fractionality is treated as a “confidence” of the variable being a particular integer or not and for values close to 0.5 the strategy becomes random. Strictly rounding to the closest integer usually leads to infeasibility and, known as *simple rounding*, is also discussed in [4].

Integer-aware branching strategy Our branching strategy, shown in algorithm 1, generates two or three *branching bound changes* depending on the domain of the integer variable x_i and its fixing value $a_i \in \mathbb{Z}$. Each branching bound change consists of the new lower and upper bound that should be enforced on a variable and the variable’s index. If the fixing value matches a bound (line 6), we create two branching bound changes (line 8 or 10), fixing the variable to either bound, preferring the one equal to the fixing value. Otherwise, three nodes are generated: one fixing the variable to the suggested value, and two restricting its domain above and below it. When generating three nodes (line 3 or 5) we prefer fixing to the branching value, and then the restriction in objective direction. These restrictions do not fix the variable but ensure it is soon

branched on again. LP-based fixing often suggests values inside the domain, helping explore regions around LP solutions. This change was especially important for our ESOMs as shown in detail in appendix B.

Algorithm 1 Branching strategy

Input: Integer variable x_i with bounds $l_i \leq x_i \leq u_i$, branching value $a_i \in \mathbb{Z}$ with $l_i \leq$

$$a_i \leq u_i$$

Output: $(l^1, u^1, i), \dots, (l^k, u^k, i)$ k branching bound changes in **increasing** priority

```

1: if  $l_i < a_i < u_i$  then
2:   if  $c_i > 0$  then
3:     return  $[(a_j + 1, u_j, i), (l_i, a_i - 1, i), (a_j, a_j, i)]$ 
4:   else
5:     return  $[(l_i, a_i - 1, i), (a_j + 1, u_j, i), (a_j, a_j, i)]$ 
6: else
7:   if  $a_i = l_i$  then
8:     return  $[(u_j, u_j), (l_j, l_j)]$ 
9:   else
10:    return  $[(l_j, l_j), (u_j, u_j)]$ 

```

Tiebreaking and hybrid strategies When two or more variables are ranked equally by a given variable strategy, a tiebreaking strategy can chose the potentially “better” variable for fixing next. In section 4.1 we considered *Frac* with ties broken by *Dual*, *Frac* with ties broken by *RedCost*, and vice versa. These hybrid approaches do not fundamentally change the underlying heuristic, but provide additional flexibility in variable selection and help avoid arbitrary decisions when multiple variables have similar LP characteristics. We note that the original code did not feature any tiebreaking and we run all strategies in [16] Table 3 without additional tiebreaking.

3 Energy system optimization models

The ESOMs used in the experiments in section 4 were developed in the UNSEEN project and are publicly available [17]; all sets are used except *miso_244k*. Based on the REMIX modeling framework [18], the instances represent the German power system and neighboring countries and optimize the power sector for 2030.

Existing conventional capacities follow the current fleet, while a CO₂ price incentivizes renewable expansion. Renewable generation (wind and solar), lithium-ion batteries, and pumped-storage hydropower are modeled with continuous variables; discrete technologies and transmission expansions use integer variables. Natural-gas CCGT is the only expandable conventional technology; lignite, coal, and OCGT capacities are fixed. Unit commitment includes minimum up-/down-times and partial-load constraints for CCGT, coal, and lignite. The transmission network comprises HVAC and HVDC corridors. AC lines use a DC load-flow ($B-\theta$) formulation, and HVDC links are modeled as capacity-constrained transport arcs. Grid topology is fixed and expandable only in capacity. Investments are static, chosen once for the modeled year; commissioning times and staged investments are not considered.

The resulting large-scale MIPs span 8,760 hourly time steps. Model sizes and instance counts are reported in table 1. These undecomposed formulations capture strong

Table 1: Sizes and number of UNSEEN ESOMs.

	Variables	Integers	Constraints	Non-zeros	#instances
miso_2M	880,606	35,052	928,790	2,829,975	100
miso_4M	1,103,903	157,692	1,279,124	4,590,593	96
miso_47M	13,597,290	972,477	16,313,022	47,310,667	100
miso_82M	24,356,961	867,465	24,698,935	82,773,747	20
remix_105M	28,455,474	3,364,138	33,194,746	105,960,838	100
remix_243M	66,714,360	7,726,975	77,498,047	243,495,063	100

seasonal effects in renewable-dominated systems (see [5] for comparisons with rolling-horizon and temporal-zooming approaches).

4 Numerical results

Hardware and software The MIPLIB experiments in section 4.1 ran on four identical machines, each with four NVIDIA A100 GPUs (80 GB), 128 GB RAM, and two AMD EPYC 7513 CPUs (32 cores each). One CPU socket was allocated per run, with up to two concurrent runs per machine, all runs being allocated 32 threads. The ESOM experiments in sections 4.2 and 4.3 used a single NVIDIA GH200 Grace-Hopper system with a 72-core ARM Neoverse-V2 CPU, 480 GB CPU memory, and one NVIDIA H200 GPU (96 GB). All runs used 32 threads (affecting only the IPM); two parallel runs were allowed, but never simultaneously using the GPU.

Instances were presolved with Gurobi 11, which is essential for tractability; pre-solve time is included in total runtimes. Only presolved solutions are reported, as postsolve is not exported by Gurobi. Although the framework also supports CPLEX for pre- and postsolve, but when starting this paper CPLEX offered only limited support on Linux ARM. Open-source presolvers such as PaPILO were considered but performed poorly on our ESOMs. We use COPT for PDLP, IPM, and Simplex solves using COPT 7.2.0 on x86 and 7.2.3 on ARM.

FP experiments We conducted three experiments: (i) in section 4.1, each FP strategy, including the presets from Table 3 [16], was run on the MIPLIB 2017 benchmark [8]; after removing seven infeasible instances and applying five random permutations per feasible instance [13], this yielded 1,165 instances; (ii) in section 4.2, we compared IPM and PDLP runtimes on large-scale ESOMs; and (iii) in section 4.3, we evaluated LP-based strategies using PDLP and IPM on the ESOMs.

The IPM is used for the final LP relaxation with primal, dual, and gap shiftetolerances of 10^{-8} . The LP method for the initial relaxation depends on the strategy: Dual Simplex (presets *zerolp* and *lp*) uses COPT defaults of 10^{-6} absolute primal and dual tolerances, while PDLP uses relative tolerances. IPMs often yield absolute primal-dual feasibility even for larger gap tolerances, whereas FOMs provide only relative feasibility. A solution is *feasible* if it satisfies relative primal/dual tolerances of 10^{-6} .

Other Unless stated otherwise, results are reported using the shifted geometric mean with a shift of one second or percent reducing sensitivity to outliers. Gaps and solution times are reported only for instances with feasible solutions; unsolved instances typically have shorter runtimes as the final LP is not executed. Gaps are computed relative

to known optima for MIPLIB and to the LP-relaxation value for ESOMs. Results often aggregate multiple configurations by taking the best outcome per instance, reducing the impact of performance variability [13]. Full tables are provided in the appendix, detailed logs in [10]. Finally, as NVIDIA released cuOpt during revision, limited results on cuOpt for ESOMs and MIPLIB perturbations are reported in appendix C and table 19.

4.1 MIPLIB experiments

The MIPLIB experiments were run with a timelimit of one hour and 20 GB CPU memory. Neither limit was ever hit.

LP-based FP compared with LP-free variants We ran all presets in Table 3 [16] and our additions with and without repair and with and without our newly implemented branching rules. Full results can be found in appendix A (tables 13 and 14). In table 2 we aggregated our LP-based variants and compare them against all presets and only LP-free presets. Our additions produce fewer solutions but also result in over eight

Table 2: MIPLIB results aggregated by LP-based and LP-free FP strategies.

Strategies	# found	gap (%)	time (s)
Presets	897	24.44	2.37
Presets (LP-free)	849	33.90	1.35
LP based (ours)	840	16.33	3.74
Best	922	15.30	3.66

percent better gaps while also taking more time as they always rely on the initial LP relaxation. LP-based methods dominate the aggregated runtime of “Best” and as they produce better solutions.

Table 3: MIPLIB results aggregated by branching strategy.

Methods	old branching			new branching		
	# found	gap (%)	time (s)	# found	gap (%)	time (s)
Presets	895	26.34	2.7	900	24.36	2.41
LP based (ours)	832	18.47	3.65	834	16.81	3.77

Impact of branching strategy To evaluate our branching strategy, we aggregated all results using the old and new branching strategies over the presets and our LP based additions in table 3. The extended branching strategy is able to reduce the gap produced on average by about two percent, solving slightly more instances at the cost of 0.25s higher runtimes for the presets and 0.1s for our additions.

Tiebreaking/hybrid strategies We ran each of our tiebreaking variants using IPM and PDLP (tolerances 10^{-4} each) for the initial LP solve and repair disabled (to decrease random influence) and compare the tiebreaking variants against their LP counterparts. The aggregated results are presented in table 4. Overall, while the tiebreaking

Table 4: MIPLIB results for LP-based and tiebreaking strategies aggregated by LP method (tolerance 10^{-4}).

Variable strategies	PDLP			IPM		
	# found	gap (%)	time (s)	# found	gap (%)	time (s)
LP-based tiebreaking	649	20.18	3.63	641	20.15	2.16
LP-based	671	22.86	3.57	660	21.92	2.00
Best	693	19.65	3.67	699	19.11	2.19

variants produce about 2% smaller gaps, they also found solutions for about 20 instances less. More notably, aggregating the tiebreaking and standard variants together, we see that they seem to complement each other, producing smaller gaps and more solutions. Primal heuristics are strongly susceptible to variance. By adding more variants, this randomness can be exploited to generate a more complete probing of the overall solution space which we believe is the primary benefit of the tiebreaking variants.

Impact of LP algorithm and precision Table 5 shows results for our LP-based variants using IPM and PDLP as LP solver and precisions 10^{-4} and 10^{-6} . Overall, solution quality is largely insensitive to LP algorithm and precision for the MIPLIB instances varying by only 0.2% over all aggregations. As IPMs converge quickly and often close the final gap in few iterations for small instances (such as in MIPLIB) reducing precision only yields minimal time savings. PDLP on the other hand exhibits slower final convergence but faster initial progress; reducing precision yields roughly a 1.5x speed-up. On MIPLIB, PDLP is still slower than IPM in absolute terms, a result in line with

Table 5: MIPLIB results for LP-based FP variants aggregated by LP method and tolerance.

LP Method	10^{-4}			10^{-6}		
	# found	gap (%)	time (s)	# found	gap (%)	time (s)
best PDLP	801	19.39	3.70	791	19.23	5.96
best IPM	794	19.25	2.19	799	19.49	2.26

PDLP literature as most instances in MIPLIB are small and thus not well suited for FOMs to outperform IPMs. In table 6 we count the percentage of times the IPM or PDLP variants were faster at finding a solution for all experiments from table 13. Additionally, we supply the average number of nonzeros of these instances. PDLP is more

Table 6: Percentage of total runs where LP method found a solution quicker vs average instance nonzeros.

LP Method	% faster	avg. nonzeros
PDLP	18.2	790 462.6
IPM	81.8	456 624.3

likely to outperform on larger or denser models with more non-zeros, a fact that will be reinforced in sections 4.2 and 4.3.

Timing analysis Lastly, we present a breakdown of the FP algorithm by time spent in its major components: presolve, the initial LP solve (if any), the fix-and-propagate scheme (possibly including repairs), and the final LP solve. Table 7 shows the average time spent per FP component for instances where FP was able to find a solution. Again we compare all LP-free presets from [16] and our additions, split by LP method. For

Table 7: Average percentage breakdown of FP time by components.

Methods	reading	presolve	fix-and-propagate	initial LP	final LP
Presets (LP-free)	11.97	75.82	5.44	0.0	6.77
LP-based with PDLP	1.30	9.12	1.05	87.64	0.90
LP-based with IPM	5.86	43.63	4.25	41.71	4.55

the MIPLIB instances, presolve time is relatively significant, especially on the LP-free methods taking up to 75% of the total runtime. For PDLP-based variants the initial LP time dominates taking 88% on average, the IPM-based methods spend 42% in the initial LP. The fix-and-propagate loop itself is relatively cheap for all variants taking at most 5.5% of the total runtime. For MIPLIB, solving the final LP is relatively inexpensive taking at most 7% of total runtime for all aggregations. We provide a similar breakdown in section 4.3 for the sets miso_2M and miso_4M.

4.2 PDLP vs. IPM performance on large ESOMs

We compare PDLP and IPM on the LP relaxations of the ESOMs, crossover disabled. Table 8 shows solution times for PDLP and IPM across instance sets and tolerances and we supply the CPU and GPU memory used as reported by slurm and “nvidia-smi”.

Table 8: Time to solution (in seconds) and CPU/GPU used memory (in GB) for COPT 7.1 PDLP and IPM with varying tolerances on ESOMs.

Test Set	PDLP		MEM		IPM		MEM	
	10^{-4}	10^{-6}	CPU	GPU	10^{-4}	10^{-6}	CPU	GPU
miso_2M	13.08	22.21	0.003	1.0	29.62	31.91	1.53	0.0
miso_4M	9.18	26.15	0.003	1.1	72.34	86.74	2.4	0.0
miso_47M	104.19	166.34	9.6	3.4	1 002.05	1 216.62	18.9	0.0
miso_82M	394.54	1 669.78	17.2	6.4	4 354.58	6 893.95	38.3	0.0
remix_105M	147.82	552.33	21.5	6.8	10 067.46	12 893.38	113.1	0.0
remix_243M	436.92	2 131.95	49.4	13.4	TIMEOUT	TIMEOUT	611.5	0.0

All runs used 32 threads, tolerances of 10^{-4} and 10^{-6} , and a time limit of 43,200s (12 h). As instance size increases, PDLP becomes increasingly competitive. For the largest models remix_243M, the IPM failed to converge within the time limit. PDLP benefits more from relaxed accuracy: for large instances, reducing the tolerance to 10^{-4} yields speedups of up to a factor of 5, while the IPM achieves only a factor of 1–2. PDLP only stores the system matrix and its required memory scales roughly linearly with matrix size, for our instances GPU memory stays below 50 GB. IPMs require storing the matrix factor(s) and memory requirements scale closer to quadratically with matrix size reaching more than 600 GB for our largest models.

The observed advantage of PDLP stems from both the algorithm and GPU acceleration. Both, PDLP and IPM are memory bound algorithms. As PDLP relies only on vector and sparse matrix-vector operations this effect is even stronger than for the IPM, where more computations are done relative to memory loaded. Thus, PDLP benefits strongly from the high-bandwidth memory of GPUs. GPU implementations of IPMs quickly exceed the available memory and are more complex to implement efficiently on GPU. They still benefit (but less) from the increased memory bandwidth. Consequently, the relative performance of PDLP and IPM depends on available hardware: while PDLP excels on modern high-bandwidth GPUs, IPMs may outperform it on systems with weaker or no GPU support.

4.3 Large-scale ESOM experiments

We next evaluated our FP heuristic on the large-scale ESOMs using different LP methods and precisions. We ran all presets and our additions on the `miso_2M` and `miso_4M` instances (see tables 15 to 18). In table 9 we provide a similar time breakdown as in table 7 for the sets `miso_2M` and `miso_4M`. The time spend in the initial and final LP

Table 9: Average percentage breakdown of FP time by components.

Set	Methods	reading	presolve	fix-and-propagate	initial LP	final LP
miso_2M	Presets (LP-free)	18.76	45.78	0.13	0.00	35.32
	LP-based with PDLP	5.48	13.12	0.00	70.36	11.04
	LP-based with IPM	5.88	15.59	0.20	66.42	11.91
miso_4M	Presets (LP-free)	9.24	22.06	0.34	0.00	68.36
	LP-based with PDLP	3.88	9.23	0.00	54.42	32.47
	LP-based with IPM	1.41	3.52	0.32	83.03	11.72

rises for growing instance size taking 67% and 70% for PDLP- and IPM-based variants respectively on `miso_2M`. For `miso_4M` the IPM based methods even spend 84% of the time in the initial LP while for PDLP this fraction decreases to 54% as the final IPM LP solve becomes more expensive relatively. Presolve and reading time become less significant for LP-based variants and the fix-and-propagate logic itself continues to play no role in the overall runtime.

Table 10: ESOM results aggregated by LP-based and LP-free FP strategies.

Strategies	miso_2M			miso_4M		
	# found	gap (%)	time (s)	# found	gap (%)	time (s)
Presets	100	0.05	13.6	96	0.02	163.15
Presets (LP-free)	100	2.91	4.30	96	27.36	12.41
LP based (ours)	100	0.04	11.20	96	0.02	41.58
Best	100	0.04	11.12	96	0.01	77.57

Table 10 shows LP-based FP variants were significantly more successful (while more expensive) on the ESOMs. Our additions, as they make use of the fast and low precision PDLP solutions were also significantly faster than the presets. We note that the presets performances is mainly driven by the *core* strategy, similar to our *type*, using

IPM with standard accuracy. We decided for the most successful (compare tables 16 and 18) in terms of gap and instances solved strategy, *type*, combined with the repair mechanism from [16] and our new branching strategy. While *type* was not the most successful strategy on MIPLIB, for our specific instance class it is, displaying the strong variability of primal heuristics performance for diverse sets of MIPs. Table 11 shows the performance metrics of *type* for all model sets, LP methods, and precisions with a timelimit of 12 hours.

Table 11: Performance of type FP heuristic on large-scale ESOMs for IPM and PDLP.

Method	Testset	1e-4			1e-6			MEM (GB)	
		# found	gap (%)	time (s)	# found	gap (%)	time (s)	CPU	GPU
IPM	miso_2M	100	0.05	12.58	99	0.05	14.10	0.003	0.0
	miso_4M	96	0.01	73.82	96	0.01	86.54	3.0	0.0
	miso_47M	100	0.07	1 391.72	100	0.11	1 584.05	22.1	0.0
	miso_82M	15	0.78	7 707.47	16	0.65	9 647.04	47.3	0.0
	remix_105M	100	1.64	10 590.73	95	1.52	13 252.69	58.22	0.0
PDLP	miso_2M	99	0.05	6.98	100	0.05	16.17	0.003	0.6
	miso_4M	95	0.04	18.07	96	0.04	34.79	0.003	0.9
	miso_47M	100	0.07	486.95	100	0.07	558.78	20.1	3.1
	miso_82M	18	0.77	3 461.62	15	0.65	5 281.46	42.5	6.1
	remix_105M	96	1.74	3 322.69	96	1.55	5 095.18	45.6	6.5
	remix_243M	87	1.57	12 587.51	82	1.48	26 334.98	105.8	13.1

Most gaps remain below 1% for the smaller ESOMs, and below 2% for two largest sets. Runtime is dominated by the final barrier solve; the initial LP solve with PDLP is comparatively cheaper. As we solve the final LP using an IPM, memory requirements for PDLP based variants also grow, though slightly less than the IPM counterpart as here the larger initial LP is also solved by an IPM. Memory requirements are smaller than in table 8, as we additionally apply MIP presolve implemented by Gurobi. For individual instance sets, either variant can slightly outperform the other, a result which we mostly attribute to performance variability within the method. E.g., on remix_105M the IPM variant produces solutions better by 0.03% gap and finds four more solutions overall, whereas on miso_82M, the PDLP variant found more solutions with 0.01% smaller gap. Overall, the heuristic is effective at exploiting low-quality LP solutions, without sacrificing solution quality but within less time.

Table 12: Performance of Gurobi (1% target gap) on ESOMs.

Instance set	# found	gap (%)	time (s)
miso_2M	100	0.40	130.81
miso_4M	96	0.05	881.53
miso_47M	89	0.06	25 283.24
miso_82M	0	-	-
remix_105M	0	-	-
remix_243M	0	-	-

In table 12 we compare our results to Gurobi (timelimit two days). Our PDLP-based

FP heuristic can generate solutions for most instances, often in significantly less time. Gurobi, taking a different, global solution approach is slowed down by the initial LP solve and crossover, failing for the larger sets.

5 Conclusion

Embedding FOMs via PDLP into the FP heuristic enables systematic exploitation of low-quality LP solutions without compromising solution quality. On MIPLIB 2017 we show that the low quality solutions produced by FOMs marginally affect solution quality and on the large-scale ESOMs, this approach achieves gaps below 2%, often below 1%, and can solve instances where traditional IPM-based heuristics or commercial solvers are impractical. For future research we believe exploration of more hybrid LP-based variants to be promising for FP itself. The exploitation of PDLP’s cycling convergence behavior to generate multiple LP reference solutions starting FP searches as done in [15] for crossover seems promising. Using PDLP for the final LP solve might further speedup the heuristic but solutions would be less accurate.

Acknowledgements We thank Domenico Salvagnin for making his FP repair code [16] available.

Funding The described research activities are funded by the Federal Ministry for Economic Affairs and Energy within the project UNSEEN (ID: 03EI1004D). Part of the work for this article has been conducted in the research Campus MODAL funded by the German Federal Ministry of Education and Research (BMBF) (fund numbers 05M14ZAM, 05M20ZBM).

Data availability The ESOMs in this study are available at [17, 11], logs and evaluation scripts at [10]. The source code is not publicly available at this time, but compiled binaries can be obtained upon request from kempke@zib.de.

Conflict of Interest All authors declare that they have no conflicts of interest.

References

- [1] Achterberg, T.: Constraint integer programming. Ph.D. thesis, Technical University Berlin (2007). DOI 10.14279/depositonce-1634
- [2] Achterberg, T., Wunderling, R.: Mixed Integer Programming: Analyzing 12 Years of Progress, pp. 449–481. Springer, Berlin, Heidelberg (2013). DOI 10.1007/978-3-642-38189-8_18
- [3] Applegate et al.: Practical Large-Scale Linear Programming using Primal-Dual Hybrid Gradient. In: Advances in Neural Information Processing Systems, vol. 34, pp. 20243–20257. Curran Associates, Inc. (2021)
- [4] Berthold, T.: Primal Heuristics for Mixed Integer Programs. Ph.D. thesis, Technical University Berlin (2006)
- [5] Cao et al.: Classification and Evaluation of Concepts for Improving the Performance of Applied Energy System Optimization Models. *Energies* **12**(24) (2019). DOI 10.3390/en12244656

- [6] Dantzig, G.B., Orden, A., Wolfe, P.: The generalized simplex method for minimizing a linear form under linear inequality restraints. *Pacific Journal of Mathematics* **5**(2), 183 – 195 (1955)
- [7] Fischetti, M., Glover, F., Lodi, A.: The feasibility pump. *Mathematical Programming* **104**(1), 91–104 (2005). DOI 10.1007/s10107-004-0570-3
- [8] Gleixner et al: MIPLIB 2017: Data-Driven Compilation of the 6th Mixed-Integer Programming Library. *Mathematical Programming Computation* (2021). DOI 10.1007/s12532-020-00194-3
- [9] Han et al.: A Low-Rank ADMM Splitting Approach for Semidefinite Programming (2024). DOI 10.48550/arXiv.2403.09133
- [10] Kempke, N.C., Koch, T.: Experimental Logs and Results for “Fix-and- Propagate Heuristics Using Low-Precision First- Order LP Solutions for Large-Scale Mixed-Integer Linear Optimization” (2025). DOI 10.5281/zenodo.17831842
- [11] Kempke, N.C., Koch, T.: Mixed-Integer Energy System Optimization Models for Germany: Seven Spatial Aggregations with Monte Carlo–Sampled UC, Dispatch, and Expansion Scenarios - Supplement: Gurobi Presolved Models (2026). DOI 10.5281/zenodo.18777743
- [12] Lin et al.: An ADMM-Based Interior-Point Method for Large-Scale Linear Programming (2020). DOI 10.48550/arXiv.1805.12344
- [13] Lodi, A., Tramontani, A.: Performance Variability in Mixed-Integer Programming, p. 1–12. *INFORMS* (2013). DOI 10.1287/educ.2013.0112
- [14] Mexi et al.: Scylla: a matrix-free fix-propagate-and-project heuristic for mixed-integer optimization. In: *Operations Research Proceedings* (2023)
- [15] Rothberg, E.: Concurrent Crossover for PDHG (2025). DOI 10.48550/arXiv.2510.24429
- [16] Salvagnin, D., Roberti, R., Fischetti, M.: A fix-propagate-repair heuristic for mixed integer programming. *Mathematical Programming Computation* (2024). DOI 10.1007/s12532-024-00269-5
- [17] Sasanpour, S., Breuer, T.: Mixed-Integer Energy System Optimization Models for Germany: Seven Spatial Aggregations with Monte Carlo–Sampled UC, Dispatch, and Expansion Scenarios (2025). DOI 10.5281/zenodo.17702892
- [18] Wetzel et al.: REMix: A GAMS-based framework for optimizing energy system models. *Journal of Open Source Software* **9**(99), 6330 (2024). DOI 10.21105/joss.06330
- [19] Wright, S.J.: Primal-dual interior-point methods. *SIAM* (1997). DOI 10.1137/1.9781611971453
- [20] Çördük et al.: GPU-Accelerated Primal Heuristics for Mixed Integer Programming (2025). DOI 10.48550/ARXIV.2510.20499

A MIPLIB2017 results

Table 14: Geo. mean of gap (shift 1%) and time (shift 1s) for LP-based FP variants on MIPLIB. Gaps and time are reported for instances where a feasible solution could be found.

Strategy	method	Tol.	no repair + old branching			repair + old branching			no repair + new branching			repair + new branching		
			# found	gap (%)	time (s)	# found	gap (%)	time (s)	# found	gap (%)	time (s)	# found	gap (%)	time (s)
duals	PDLP	10^{-4}	516	41.33	3.34	658	41.07	3.34	504	35.34	3.28	659	34.74	3.27
		10^{-6}	517	41.18	5.95	652	39.56	5.29	505	33.95	5.86	654	32.82	5.22
	IPM	10^{-4}	525	39.56	1.79	658	40.23	1.95	504	33.46	1.79	649	33.98	1.86
		10^{-6}	528	39.82	1.81	668	39.48	1.97	508	33.89	1.76	655	32.86	1.93
redcost	PDLP	10^{-4}	459	34.33	3.12	679	34.85	3.22	441	27.66	3.040	682	30.66	3.05
		10^{-6}	450	32.79	5.53	666	37.87	5.21	442	27.32	5.45	663	31.59	5.20
	IPM	10^{-4}	473	33.52	1.69	681	37.20	2.04	458	29.74	1.63	679	32.31	2.04
		10^{-6}	460	29.67	1.77	679	33.30	2.03	455	26.60	1.66	679	30.52	1.99
frac	PDLP	10^{-4}	465	27.33	2.93	698	33.36	3.19	457	23.03	2.87	691	28.71	3.10
		10^{-6}	458	27.97	5.53	676	33.10	5.56	452	23.71	5.55	675	28.96	5.30
	IPM	10^{-4}	457	24.56	1.77	684	32.31	1.95	450	20.96	1.74	696	27.46	1.86
		10^{-6}	443	25.86	1.82	690	32.03	1.92	436	21.60	1.68	694	27.00	1.89
type	PDLP	10^{-4}	566	46.60	3.64	695	41.06	3.39	562	41.60	3.58	694	35.87	3.34
		10^{-6}	555	46.34	5.89	691	41.88	5.36	551	41.12	5.92	689	36.28	5.34
	IPM	10^{-4}	549	45.99;	1.91	690	40.00	1.97	546	41.12	1.97	688	34.91	1.84
		10^{-6}	550	46.02	1.95	691	40.02	1.99	547	41.15	1.95	689	35.10	1.90

B ESOM results

Table 15: Geo. mean of gap (shift 1%) and time (shift 1s) for FP variants as reported in [16] on miso_2M. Gaps and time are reported for instances where a feasible solution could be found.

Preset	no repair + old branching			repair + old branching			no repair + new branching			repair + new branching		
	# found	gap (%)	time (s)	# found	gap (%)	time (s)	# found	gap (%)	time (s)	# found	gap (%)	time (s)
random	100	7.34	4.34	100	7.34	4.35	100	5.57	4.44	100	5.65	4.48
random2	100	6.75	4.54	100	6.75	4.52	0	-	-	0	-	-
badobj	100	2.97	4.29	100	2.97	4.30	100	2.97	4.31	100	2.97	4.31
badobjcl	100	2.97	4.98	100	2.97	4.87	100	2.97	4.86	100	2.97	4.91
goodobj	100	6.57	4.34	100	6.57	4.36	100	6.57	4.33	100	6.57	4.34
goodobjcl	100	6.57	5.03	100	6.57	5.04	100	6.57	4.88	100	6.57	4.95
locks	100	2.97	4.32	100	2.97	4.30	100	2.97	4.29	100	2.97	4.30
locks2	100	2.97	4.31	100	2.97	4.30	100	2.97	4.31	100	2.97	4.31
cliques	100	2.79	14.03	100	2.79	14.13	100	2.79	14.18	100	2.79	14.06
cliques2	100	2.79	14.08	100	2.79	14.05	100	2.79	14.01	100	2.79	14.12
zerocore	100	98.81	5.95	100	98.81	5.98	100	98.81	5.98	100	98.81	5.98
zerolp	100	98.82	4.05	100	98.82	4.05	100	98.82	4.05	100	98.82	4.11
core	100	0.26	13.70	100	0.26	13.78	100	0.05	13.59	99	0.05	13.55
lp	99	0.25	26.17	99	0.25	26.08	100	0.05	25.95	100	0.05	25.92

Table 16: Geo. mean of gap (shift 1%) and time (shift 1s) for LP-based FP variants on miso_2M. Gaps and time are reported for instances where a feasible solution could be found.

Strategy	method	no repair + old branching			repair + old branching			no repair + new branching			repair + new branching			
		Tol.	# found	gap (%)	time (s)	# found	gap (%)	time (s)	# found	gap (%)	time (s)	# found	gap (%)	time (s)
duals	PDLP	10^{-4}	100	0.31	7.23	100	0.31	7.23	100	0.13	7.03	100	0.13	7.05
		10^{-6}	100	0.31	16.51	100	0.31	16.37	100	0.13	16.06	100	0.13	16.13
	IPM	10^{-4}	100	0.33	12.77	100	0.33	12.81	100	0.13	12.61	100	0.13	12.60
		10^{-6}	100	0.32	14.22	100	0.32	14.29	100	0.13	14.13	100	0.13	14.11
redcost	PDLP	10^{-4}	100	0.52	7.22	100	0.52	7.25	65	0.44	7.12	94	0.46	7.06
		10^{-6}	100	0.44	16.38	100	0.44	16.35	71	0.28	16.88	94	0.29	16.10
	IPM	10^{-4}	100	0.29	12.78	100	0.29	12.71	98	0.05	12.59	100	0.09	12.61
		10^{-6}	100	0.27	14.10	100	0.27	14.34	94	0.05	13.83	99	0.08	13.72
frac	PDLP	10^{-4}	100	0.28	7.22	100	0.28	7.24	65	0.04	6.98	94	0.11	7.05
		10^{-6}	99	0.29	16.38	99	0.29	16.49	61	0.04	17.81	96	0.11	16.14
	IPM	10^{-4}	100	0.29	12.82	100	0.29	12.89	61	0.05	12.21	89	0.12	12.43
		10^{-6}	100	0.27	14.31	100	0.27	14.37	75	0.04	13.94	96	0.09	14.21
type	PDLP	10^{-4}	100	0.26	7.18	100	0.26	7.19	100	0.05	6.88	99	0.05	6.98
		10^{-6}	100	0.26	16.36	100	0.26	16.44	99	0.05	16.03	100	0.05	16.17
	IPM	10^{-4}	100	0.27	12.75	100	0.27	12.73	100	0.05	12.61	100	0.05	12.58
		10^{-6}	100	0.26	14.22	100	0.26	13.83	100	0.05	14.08	99	0.05	14.10

Table 17: Geo. mean of gap (shift 1%) and time (shift 1s) for FP variants as reported in [16] on miso_4M. Gaps and time are reported for instances where a feasible solution could be found.

Preset	no repair + old branching		repair + old branching		no repair + new branching		repair + new branching		
	# found	gap (%)	time (s)	# found	gap (%)	time (s)	# found	gap (%)	time (s)
random	96	51.72	9.74	96	51.72	9.81	96	27.71	12.19
random2	96	53.03	9.30	96	52.99	9.53	0	-	-
badobj	96	53.17	11.71	96	53.17	11.75	96	53.17	11.77
badobjcl	96	53.17	12.65	96	53.17	12.75	96	53.17	12.63
goodobj	93	43.82	19.00	93	43.82	19.02	93	44.09	19.17
goodobjcl	93	43.82	20.28	93	43.82	20.37	93	43.82	20.29
locks	96	53.57	11.69	96	53.57	11.72	96	53.57	11.75
locks2	96	53.57	11.71	96	53.57	11.72	96	53.57	11.72
cliques	96	53.13	84.58	96	53.13	84.56	96	53.13	84.85
cliques2	96	53.13	84.11	96	53.13	84.64	96	53.13	84.68
zerocore	96	95.72	26.34	96	95.72	26.46	96	95.31	26.56
zerolp	96	95.68	12.50	96	95.68	12.55	96	96.01	12.44
core	96	51.54	92.46	96	51.54	92.30	95	0.03	87.51
lp	96	43.82	408.43	96	43.82	413.02	96	0.05	408.13

Table 18: Geo. mean of gap (shift 1%) and time (shift 1s) for LP-based FP variants on miso_4M. Gaps and time are reported for instances where a feasible solution could be found.

Strategy	method	Tol.	no repair + old branching			repair + old branching			no repair + new branching			repair + new branching		
			# found	gap (%)	time (s)	# found	gap (%)	time (s)	# found	gap (%)	time (s)	# found	gap (%)	time (s)
duals	PDLP	10^{-4}	96	52.53	23.20	96	52.53	23.22	96	13.45	17.85	23	12.19	16.48
	IPM	10^{-6}	96	52.56	40.30	96	52.56	40.36	96	13.39	34.73	35	12.98	30.45
	PDLP	10^{-4}	96	52.63	80.19	96	52.63	80.20	96	13.11	73.99	44	13.09	75.62
	IPM	10^{-6}	96	52.83	91.78	96	52.83	91.86	96	13.37	86.40	36	11.97	83.71
redcost	PDLP	10^{-4}	96	52.48	23.22	96	52.48	23.24	17	3.58	17.61	31	0.94	19.31
	IPM	10^{-6}	96	51.25	40.73	100	51.25	40.64	26	1.53	33.17	44	0.73	37.68
	PDLP	10^{-4}	96	51.32	81.90	96	51.32	80.69	23	1.58	74.17	62	0.75	74.56
	IPM	10^{-6}	96	51.52	92.25	96	51.52	92.44	21	0.92	99.21	48	0.43	87.45
frac	PDLP	10^{-4}	96	52.45	23.24	96	52.45	23.32	30	12.20	18.48	52	2.91	18.84
	IPM	10^{-6}	96	52.50	40.33	96	52.50	40.40	32	0.04	34.44	61	2.60	35.35
	PDLP	10^{-4}	96	52.59	81.05	96	52.59	81.39	12	10.02	76.40	39	2.63	74.98
	IPM	10^{-6}	96	52.77	92.44	93	52.89	93.15	21	9.48	81.38	49	2.26	85.63
type	PDLP	10^{-4}	96	51.21	22.89	96	51.21	23.03	96	0.22	17.71	96	0.04	18.07
	IPM	10^{-6}	96	51.19	40.15	96	51.19	40.19	96	0.28	34.61	96	0.04	34.79
	PDLP	10^{-4}	87	51.04	78.72	95	51.26	79.80	96	0.07	73.51	96	0.05	73.82
	IPM	10^{-6}	96	51.53	91.74	96	51.53	91.71	96	0.08	86.48	96	0.03	86.54

C cuOpt heuristics-only results

Results of cuOpt in *heuristics-only*¹ mode on MIPLIB and our ESOMs. All instances were presolved using Gurobi 11.0.0 and (presolve times are not included in the cuOpt runtimes) and run on an ARM Neoverse-V2 CPU with 72 cores, 480 GB of CPU memory, and one NVIDIA H200 GPU with 96 GB of device memory. Each run executed exclusively using 32 threads; cuOpt’s heuristics-only run almost purely on GPU. Note that cuOpt is not designed for fast-fail solution computation but rather as a standalone heuristic solver. We enforced tight time-limits on our cuOpt runs to still get some comparability against our primal heuristics.

Table 19: Geo. mean of gap (shift 1%) and time (shift 1s) for cuOpt “heuristics-only” on the miso_2M set with different time limits. Gaps and time are reported for instances where a feasible solution could be found.

set	timelimit	# found	gap (%)
MIPLIB	30s	1021	11.76
	300s	1097	6.25
miso_2M	10s	18	2.85
	30s	100	2.12
miso_4M	30s	71	12.50
	60s	91	5.29

¹<https://docs.nvidia.com/cuopt/user-guide/latest/lp-milp-settings.html#heuristics-only>