CHAP: A Hybrid GPU-CPU Heuristic for MIP

Gioni Mexi

gionimexi.com

joint work with Gennesaret Kharistio Tjusila, Alexander Hoen, Nils-Christian Kempke, Timo Berthold, Ambros Gleixner, Thorsten Koch, Sebastian Pokutta

MIP Workshop · University of Connecticut

May 19, 2026

Motivation

We consider mixed-integer linear programs

$$\min_{l \le x \le u} \{c^T x \ : \ Ax \le b, \ x_i \in \mathbb{Z} \ \forall i \in \mathcal{I}\}$$

- *Good solutions quickly are needed* by practitioners, and for performant MIP solvers.
- Compute hardware is changing — GPUs have become more capable, and recent first-order LP solvers Applegate et al.2021); Lu et al.2025) make GPU-friendly LP a reality.
- NVIDIA's cuOpt pushes this further: GPU acceleration is being *unraveled* for MIP beyond LP: bound propagation, probing, rapid rounding heuristics, local search, and an extended feasibility pump Çördük et al.2025).

Motivation

We consider mixed-integer linear programs

$$\min_{l \le x \le u} \{c^T x \ : \ Ax \le b, \ x_i \in \mathbb{Z} \ \forall i \in \mathcal{I}\}$$

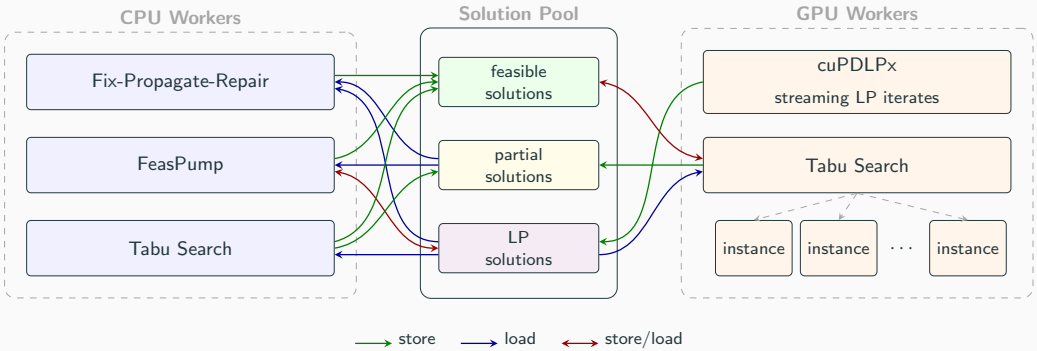
- *Good solutions quickly are needed* by practitioners, and for performant MIP solvers.
- Compute hardware is changing — GPUs have become more capable, and recent first-order LP solvers Applegate et al.2021); Lu et al.2025) make GPU-friendly LP a reality.
- NVIDIA's cuOpt pushes this further: GPU acceleration is being *unraveled* for MIP beyond LP: bound propagation, probing, rapid rounding heuristics, local search, and an extended feasibility pump Çördük et al.2025).

Goal

A portfolio MIP heuristic that leverages **both** *CPU and GPU, exchanging information through a shared solution pool.*

The Architecture of CHAP

The Architecture of CHAP



- CPU workers (left) and GPU workers (right) communicate through a **shared solution pool** (center)
- Pool stores: feasible solutions, partial (infeasible) solutions, LP solutions
- Each component reads what it needs (LP iterate, incumbent, partial sol.) and writes back what it produces

Streaming LP iterates: cuPDLPx

cuPDLPx Lu et al.2025): GPU-based first-order LP solver based on PDHG.

- Mostly matrix-vector ops $\Rightarrow$ fits GPU well, scales to very large LPs
- Convergence to high accuracy can be slow
- Mexi et al.2023); Kempke and Koch2025): **low-precision LP solutions are enough to guide primal heuristics**

Streaming approach. Do not wait for convergence. Snapshot intermediate iterates at 10^2, 10^3, 10^4, 10^5 PDHG iterations and push them into the pool.

Each snapshot contains primal, dual, and reduced costs $\Rightarrow$ CPU heuristics (FPR, FeasPump) start with **near-zero latency**.

Fix-and-Propagate

Iteratively construct a feasible MIP solution Salvagnin et al.2024):

1. select an integer variable
2. fix it to a value
3. propagate constraints (tighten domains)
4. repeat

Once all integer variables are fixed, solve a final LP for the continuous variables.

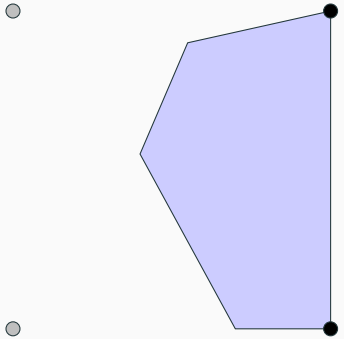
Propagation example. For a row $a^\top x + y \leq b$,

$$y \leq b - a^\top x \leq b - \min_{x \in [l,u]} a^\top x.$$

Portfolio in CHAP. Strategies = (*variable ranker*, *value chooser*, *search setting*); we run a diverse mix in parallel: *LP-guided* (LP fractionality / duals / reduced costs), *tabu-guided* (consume tabu iterates, prioritize infeasible rows), and an *LP-free fallback*.

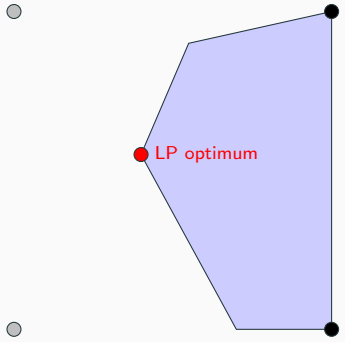
Feasibility Pump

Iteratively project onto integers and onto the feasible region in alternation Fischetti et al.2005).



Feasibility Pump

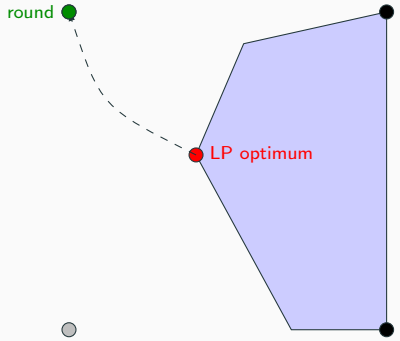
Iteratively project onto integers and onto the feasible region in alternation Fischetti et al.2005).



1. Solve LP relaxation.

Feasibility Pump

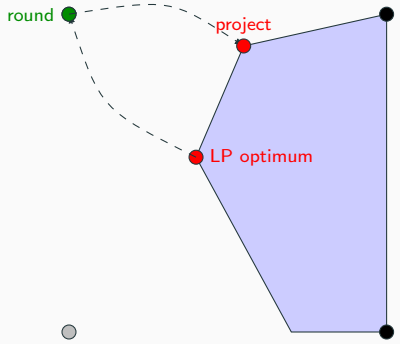
Iteratively project onto integers and onto the feasible region in alternation Fischetti et al.2005).



2. Round to the nearest integer point.

Feasibility Pump

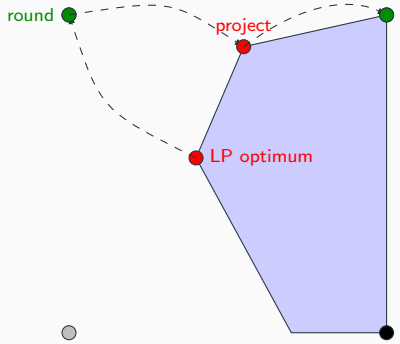
Iteratively project onto integers and onto the feasible region in alternation Fischetti et al.2005).



3. Infeasible: project back into the polytope (ℓ^1 minimization).

Feasibility Pump

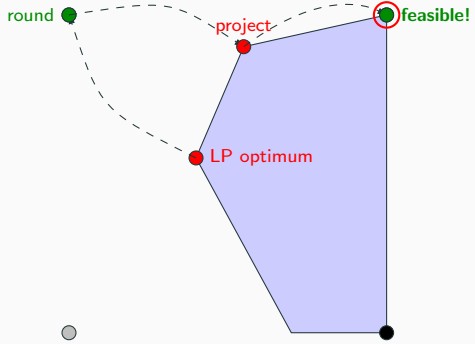
Iteratively project onto integers and onto the feasible region in alternation Fischetti et al.2005).



4. Round again.

Feasibility Pump

Iteratively project onto integers and onto the feasible region in alternation Fischetti et al.2005).



5. Feasible MIP solution — and all this works also with low-accuracy **cuPDLPx** iterates.

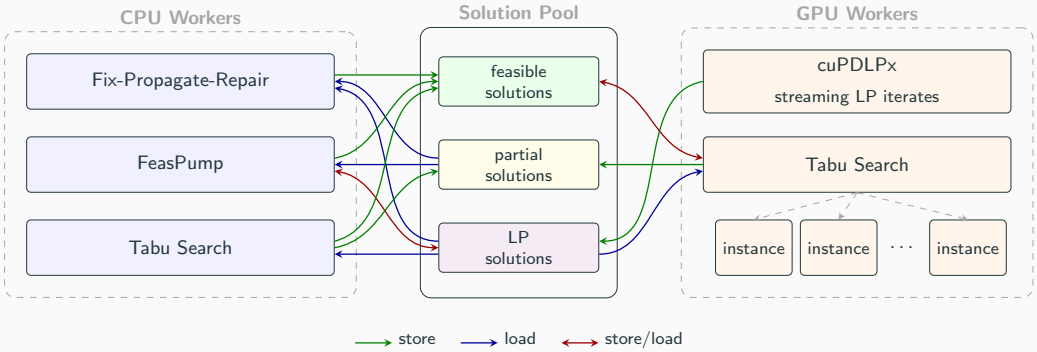
Tabu search: the basic loop

Iterative primal heuristic Glover1989, 1990); recent example: *LocalMIP* Lin et al.2025).

Given an initial iterate $\bar{x}$, repeat until termination:

- **generate** candidate moves on $\bar{x}$, dropping moves on *tabu* variables
- **score** each move and pick $m^\star \leftarrow \arg\max_{m \in \mathcal{M}} \mathrm{score}(m, \bar{x})$
- **apply** $m^\star$ to update $\bar{x}$, and mark the touched variables as tabu
- if $\bar{x}$ is feasible and improves the incumbent, **store** it

Tabu-driven repair loop



Repair:

- **FP polishing** — ≤ 5 FP iterations per tabu iterate.
- **Guided FPR** — tabu iterate as reference, prioritize infeasible rows.
- **Objective cutoff** $c^\top x \leq c^\top x_{\text{inc}}$ shared by all workers.

GPU Tabu Search

Scoring function and best shift moves

Scoring. Constraints carry weights $w_1, \ldots, w_m$. For a move of x_j from $\bar{x}_j$ to $\hat{x}_j$, with $\bar{y}_i$ the current activity and y_{ij} the activity after the move, each constraint i contributes

$$p_{ij} = \begin{cases} -w_i & \bar{y}_i \le b_i,\ y_{ij} > b_i \\ w_i & \bar{y}_i > b_i,\ y_{ij} \le b_i \\ \frac{1}{2} w_i & \bar{y}_i > b_i,\ y_{ij} > b_i,\ y_{ij} < \bar{y}_i \\ -\frac{1}{2} w_i & \bar{y}_i > b_i,\ y_{ij} > b_i,\ y_{ij} > \bar{y}_i \\ 0 & \text{otherwise.} \end{cases}$$

We define the total score of the move as

$$s_j(\hat{x}_j, \bar{x}) := \sum_{i=1}^{m} p_{ij}.$$

Best shift moves

Over $D_j := \{x_j : l_j \leq x_j \leq u_j,\ x_j \in \mathbb{Z} \text{ if } j \in \mathcal{I}\}$, a best shift move assigns

$$\hat{x}_j \in \arg\max_{x_j \in D_j} s_j(x_j, \bar{x}).$$

Observation. As a function of $\hat{x}_j$, the score $s_j(\cdot, \bar{x})$ is a **step function** — D_j reduces to a finite candidate set Luteberget and Sartor2023).

Evaluating binary flip candidates

For each binary variable j, we want $s_j(1 - \bar{x}_j, \bar{x})$. Iterate over the binary nonzeros of A and *scatter* contributions to per-variable scores.

Input: binary part of A in COO form $\{(i, j, a_{ij})\}$ sorted by row, incumbent $\bar{x}$, activities $\bar{y}$, weights w
Output: s_j for every binary j

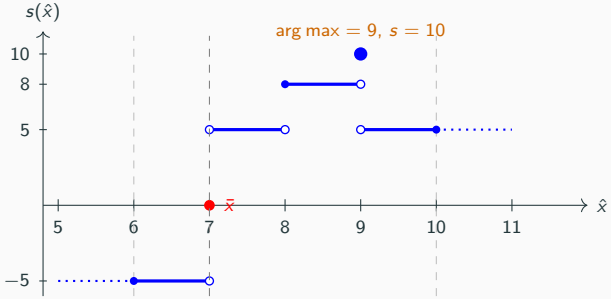
1: $s_j \leftarrow 0$ for all binary j
2: **for** each nonzero (i, j, a_{ij}) **in parallel do**
3: $\quad \delta \leftarrow (1 - 2\bar{x}_j) \cdot a_{ij}$ $\quad \{\text{flip changes } x_j \text{ by } \pm 1\}$
4: $\quad y_{ij} \leftarrow \bar{y}_i + \delta$ $\quad \{\text{activity after flip}\}$
5: $\quad$ compute p_{ij} from $(\bar{y}_i, y_{ij}, b_i, w_i)$
6: $\quad s_j \mathrel{+}= p_{ij}$ $\quad \{\texttt{atomicAdd}\}$
7: **end for**
8: **return** s

Optimizations (see Tjusila et al.2026)): row-sorted COO $\Rightarrow$ $a_{ij}, \bar{y}_i, w_i$ **coalesced**; $\bar{x}$ as a **bitset**, L2-resident; **residuals** $r_i := \bar{y}_i - b_i$ halve memory loads; $\leq$-normalized rows $\Rightarrow$ **one shared formula**.

Best shift for continuous variables: a worked example

Variable x with incumbent $\bar{x} = 7$, bounds $[l, u] = [6, 10]$. Five constraints, each shown *after substituting the other variables at their incumbent values*:

Constraint	w	At $\bar{x} = 7$
$x \geq 4$	2	satisfied
$x \geq 8$	6	violated
$x \geq 9$	4	violated
$x \leq 9$	5	satisfied
$x \leq 11$	3	satisfied



Best shift via sort-scan-argmax

The score $s(\hat{x})$ moves only at **breakpoints**. Each constraint emits 1–2 deltas; sweep $\hat{x}$ from $-\infty$ to $+\infty$ and accumulate. Worked example, $\bar{x} = 7$, bounds $[6, 10]$:

position	4	8	9	9+	11	11+
emitted by	C_1	C_2	C_3, C_4	C_4	C_5	C_5
Δs	$+2$	$+3$	$+2$	-5	0	-3
prefix	$+2$	$+5$	$+7$	$+2$	$+2$	-1
$\beta+$ prefix	-5	-2	0	-5	-5	-8
$+\alpha$ if pos $> \bar{x}$	no	yes	yes	yes	yes	yes
σ	-5	$+8$	$+10$	$+5$	$+5$	$+2$

Algorithm (one variable, in parallel across all variables):

1. **Emit breakpoints.** For each constraint with $a_{ij} \neq 0$: push 1–2 entries into B at $t_{ij} = (b_i - \sum_{k \neq j} a_{ik}\bar{x}_k)/a_{ij}$; accumulate β (score at $\hat{x} \to -\infty$) and α ($\bar{x}$-crossing offset).
2. **Sort** B lexicographically. **Inclusive scan** the deltas $\Rightarrow$ prefix P.
3. **Argmax** of $\sigma_i = \beta + P_i + \alpha \cdot \mathbf{1}[B_1^i > \bar{x}_j]$ over positions in $[l_j, u_j]$.

Sort, scan, argmax are well-known GPU primitives, but we hand-implement them; see paper for details.

Computational results

Computational results

Setup. MIPcc26 instances (50 presolved), 5-minute time limit.
Hardware: Intel Xeon 8481C + NVIDIA H100. Gap% and Primal Integral (PI) w.r.t. best solution[1].

Configuration	Found	Wins	Gap%	PI
Pure CPU	47	3	19.57	72.59
CPU + GPU LP (cuPDLPx)	46	13	12.17	51.01
CPU + GPU LP + GPU tabu	**47**	**22**	**8.84**	**41.84**

Solver	Found	Wins	Only
CHAP	**47**	11	1
Gurobi (5 min)	44	25	1
cuOpt (heuristics only)	43	6	0

[1] best solution by Gurobi/Xpress in 8 h.

Takeaways

- **Both sides matter.** CPU and GPU primal-heuristic workers each pull their weight; CHAP composes them through a shared solution pool.
- **The CPU side is already very strong.** On its own it competes with commercial MIP solvers on MIPcc26.
- **The GPU side is a real lever.** Streaming cuPDLPx LP iterates and GPU tabu search with *globally optimal* best-shift moves cut Gap% and primal integral substantially on top of the CPU portfolio Tjusila et al.2026).

CHAP paper · arxiv.org/abs/2605.05086

What's next?

How far can we push a parallel SCIP on CPU?

What's next?

How far can we push a parallel SCIP on CPU?

And can a GPU push it even further?

ReXi: Race · eXchange · improve

A separate, ongoing project — not part of CHAP.

ReXi is a parallel CPU solver for mixed-integer programming, based on **SCIP**, focused on quickly finding strong primal solutions.

Joint work with Daniel Rehfeldt.

- Includes customized and optimized versions of **SCIP** and **SoPlex**.
- All SCIP/SoPlex improvements will be released

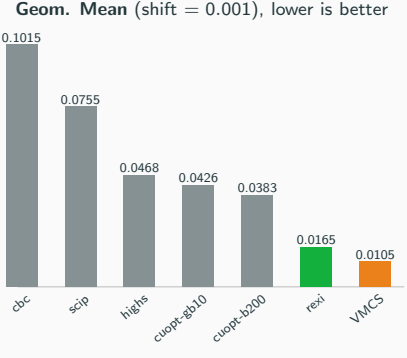
On Mittelmann's `mipfeas`

`plato.asu.edu/ftp/mipfeas.html`:

- closing the gap to VMCS

ReXi Opt

`rexi-opt.org`



Thank you!

mexi@zib.de

References i

Applegate et al. Practical Large-Scale Linear Programming using Primal-Dual Hybrid Gradient. In *Advances in Neural Information Processing Systems*, volume 34, pages 20243–20257. Curran Associates, Inc., 2021.

A. Çördük, P. Sielski, A. Boucher, and K. Aatish. Gpu-accelerated primal heuristics for mixed integer programming, 2025. URL `https://arxiv.org/abs/2510.20499`.

M. Fischetti, F. Glover, and A. Lodi. The feasibility pump. *Mathematical Programming*, 104(1): 91–104, 3 2005. ISSN 1436-4646. doi: 10.1007/s10107-004-0570-3.

F. Glover. Tabu search—part i. *ORSA Journal on Computing*, 1(3):190–206, Aug. 1989. ISSN 2326-3245. doi: 10.1287/ijoc.1.3.190. URL `http://dx.doi.org/10.1287/ijoc.1.3.190`.

F. Glover. Tabu search—part ii. *ORSA Journal on Computing*, 2(1):4–32, Feb. 1990. ISSN 2326-3245. doi: 10.1287/ijoc.2.1.4.

References ii

N.-C. Kempke and T. Koch. Low-precision first-order method-based fix-and-propagate heuristics for large-scale mixed-integer linear optimization, 2025. URL `https://arxiv.org/abs/2503.10344`.

P. Lin, S. Cai, M. Zou, and J. Lin. Local-mip: Efficient local search for mixed integer programming. *Artificial Intelligence*, 348:104405, 2025. ISSN 0004-3702. doi: 10.1016/j.artint.2025.104405.

H. Lu, Z. Peng, and J. Yang. cupdlpx: A further enhanced gpu-based first-order solver for linear programming, 2025. URL `https://arxiv.org/abs/2507.14051`.

B. Luteberget and G. Sartor. Feasibility jump: an lp-free lagrangian mip heuristic. *Mathematical Programming Computation*, 15(2):365–388, Mar. 2023. ISSN 1867-2957. doi: 10.1007/s12532-023-00234-8.

G. Mexi, M. Besançon, S. Bolusani, A. Chmiela, A. Hoen, and A. Gleixner. Scylla: A matrix-free fix-propagate-and-project heuristic for mixed-integer optimization. In *International Conference on Operations Research*, pages 65–72. Springer, 2023. doi: 10.1007/978-3-031-58405-3_9.

References iii

D. Salvagnin, R. Roberti, and M. Fischetti. A fix-propagate-repair heuristic for mixed integer programming. *Mathematical Programming Computation*, Oct. 2024. ISSN 1867-2957. doi: 10.1007/s12532-024-00269-5.

G. K. Tjusila, A. Hoen, N.-C. Kempke, G. Mexi, T. Berthold, A. Gleixner, T. Koch, and S. Pokutta. Chap: A hybrid gpu-cpu heuristic for mip, 2026. URL `https://arxiv.org/abs/2605.05086`.