Recent Computational Progress on Linear Programming Solvers

LA/OPT SEMINAR

FEBRUARY 14, 2024

Yinyu Ye
Stanford University

Linear Programming and LP Giants

$$\text{max or } min \quad \sum c_j x_j$$

$$\text{s.t.} \quad \sum_j \boldsymbol{a}_j x_j \leq \boldsymbol{b}\,,$$

$$0 \leq x_j \leq 1 \quad \forall\, j = 1, \ldots, n$$

Today’s Talk

- LP Warm-Start: Online Helps Offline
- Smart Crossover: From an Interior Point to a Corner Points
- ABIP: Interior Point Method Meets ADMM
- cuPDLP-C: How GPU Accelerates Solving LP
- Summary

Linear Programming as Combinatorial Classification

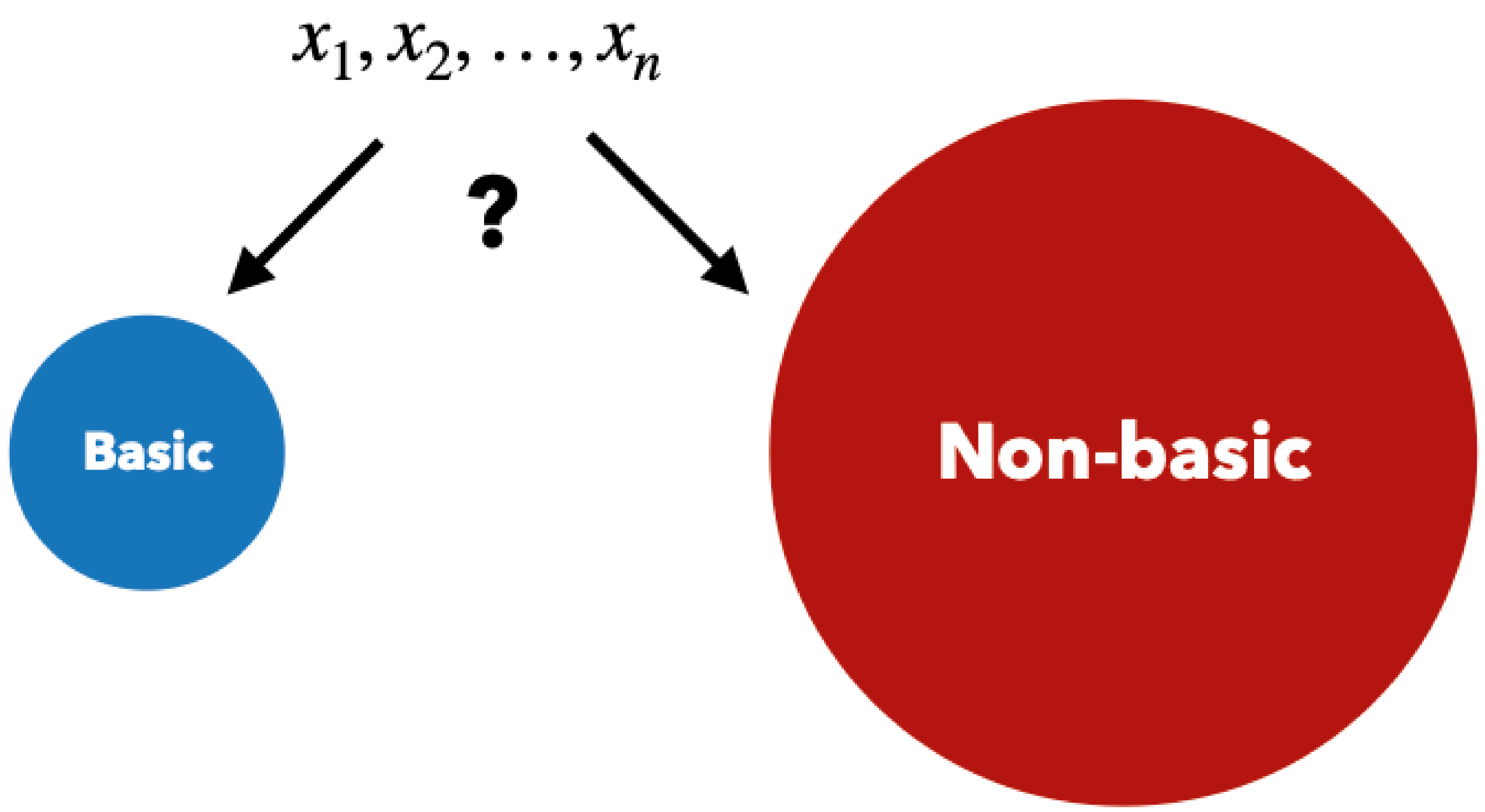
- Basic solution is one of the most important concept in LP

- LP algorithms work towards identifying the optimal basis

  Knowledge of B reduces linear programming to a linear system

$$\begin{aligned} \min_{x} \quad & c^\top x \\ \text{subject to} \quad & Ax = b \\ & x \geq 0 \end{aligned}$$

- LP can be viewed a *classification* task



Can we predict the basis?

Yes! Use the Dual

Classification using Duality

LP duality provides the most powerful classifier for LP

$$\begin{array}{llc} \max_{x} & c^\top x & \text{Dual} \\ \text{subject to} & Ax \le b & y \\ & 0 \le x \le u & s \end{array}$$

Duality ⇒

$$\begin{array}{ll} \min_{y,s} & b^\top y + u^\top s \\ \text{subject to} & s \ge c - A^\top y \\ & (y, s) \ge 0 \end{array}$$

If we get optimal y^*, then optimality condition tells us

$$x_j^* \in \begin{cases} \{0\}, & c_j - a_j^\top y^* < 0 \\ [0,1] & c_j - a_j^\top y^* = 0 \\ \{1\} & c_j - a_j^\top y^* > 0 \end{cases}$$

Dual solution tells us almost all about primal

Fast Training the Classifier *y**

- But solving dual problem is no easier than the primal
- Is there a "*cheap*" way to estimate $\hat{y} \approx y^*$?

$$x_j^* \in \begin{cases} \{0\}, & c_j - a_j^\top y^* < 0 \\ [0,1] & c_j - a_j^\top y^* = 0 \\ \{1\} & c_j - a_j^\top y^* > 0 \end{cases}$$

Dual solution tells us almost all about primal

- No matrix factorization
- No explicit matrix multiplication
- $O(\mathrm{nnz}(\boldsymbol{A}))$ flops
- Reasonable accuracy

The overall budget is only several MatVec

How can we fulfill the goals simultaneously?

Ans: Estimate *on the fly* by Online Linear Programming (OLP)

[Gao et al. ICML, 2023]

What is Online Linear Programming

- Decision maker needs to decide x_t: how much resources are allocated/sold to each customer

- **Online setting**:
  - Customers arrive sequentially and the decision needs to be made instantly upon the customer arrival: Sell or No-sell?

$$\max \sum_{t=1}^{T} r_t x_t$$

$$\text{s.t.} \sum_{t=1}^{T} a_{it} x_t \leq b_i, \quad i = 1, \ldots, m$$

$$0 \leq x_t \leq 1 \ \text{ or } \ x_t \in \{0, 1\}, \quad t = 1, \ldots, T$$

[Agrawal et al. 2010, 2014], [Kesselheim et al., 2014]
[Li/Y, 2019], [Li et al., 2020],

Online Learning of y^*

Re-write the dual as

$$\begin{aligned} \min_{y,s} \quad & b^\top y + u^\top s \\ \text{subject to} \quad & s \geq c - A^\top y \\ & (y, s) \geq 0 \end{aligned}$$

u = e

$$\min_{y \geq 0} \; b^\top y + \sum_{j=1}^{n} [c_j - a_j^\top y]_+$$

- The dual objective is a finite-sum problem with minimal constraints
- When n is large, dual objective is the sample approximation of a stochastic program
- What's the most efficient way for finite-sum problem?

Ans: Online Sub-Gradient

Online Sub-Gradient Method

Solve finite-sum problem by OSG?

$$\min_{y \geq 0} \; b^\top y + \sum_{j=1}^{n} [c_j - a_j^\top y]_+$$

How to estimate $\{x_j\}$?

$$x_j^* \in \begin{cases} \{0\}, & c_j - a_j^\top y^* < 0 \\ [0,1] & c_j - a_j^\top y^* = 0 \\ \{1\} & c_j - a_j^\top y^* > 0 \end{cases}$$

On the dual side

- When read in a column (c_j, a_j) data

Compute subgradient $g_j = \frac{b}{n} - a_j I\{c_j > a_j^\top y^j\}$

- Update y^j using (projected) subgradient

On the primal side

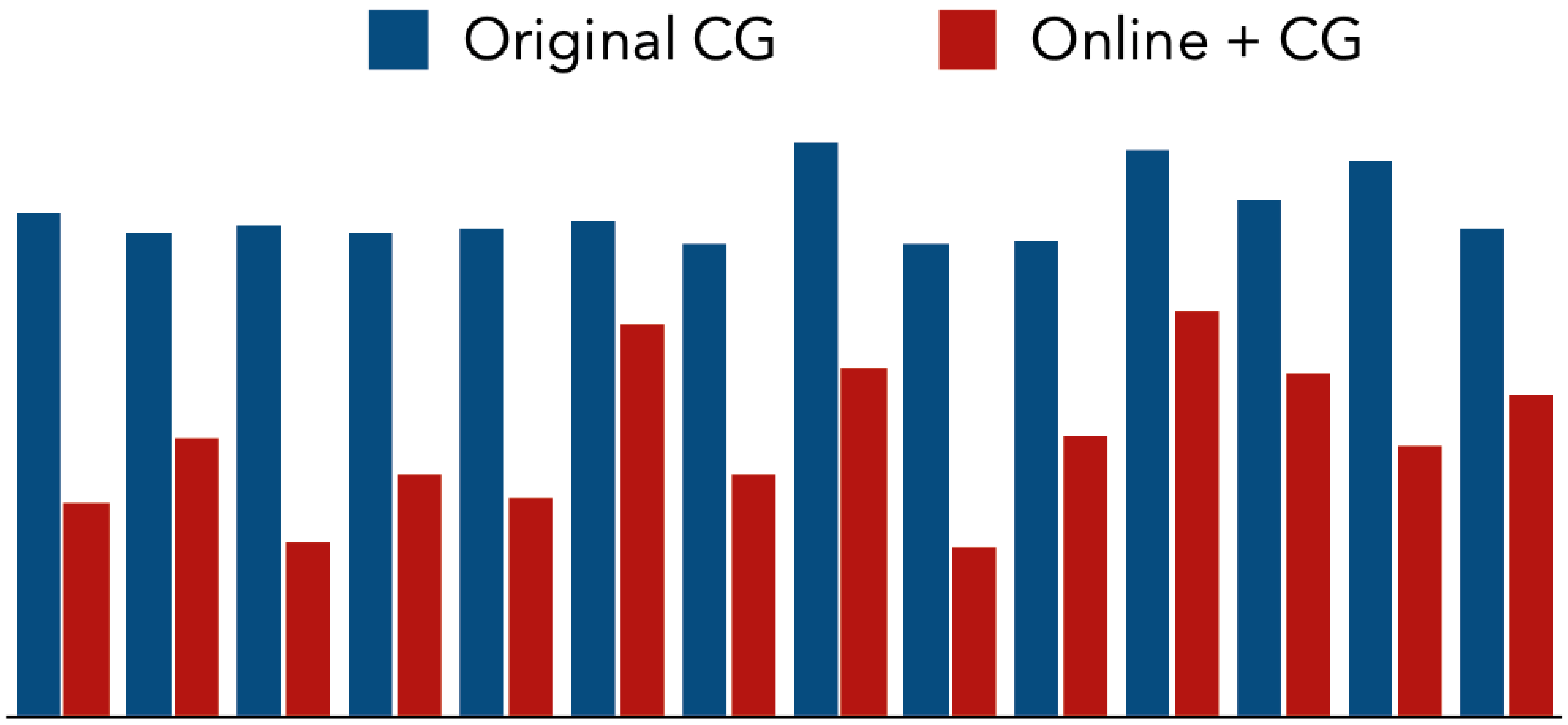
- Apply optimality condition on the fly

$$x_j = I\{c_j > a_j^\top y^j\}$$

- May randomly sample columns multiple times and take average

Computational Results

Experiments on MIPLIB 2017 and MKP instances using Column-Generation



Data	Acc	Data	Acc
scpm1	100%	rail507	90%
scpn2	100%	rail516	88%
scpl4	100%	rail2586	94%
scpk4	100%	rail4284	96%

Accuracy of classification

- 2x speedup on instances with many variables
- Simple, efficiently and almost no-cost
- Online LP helps pre-solving offline LP for Warm Start

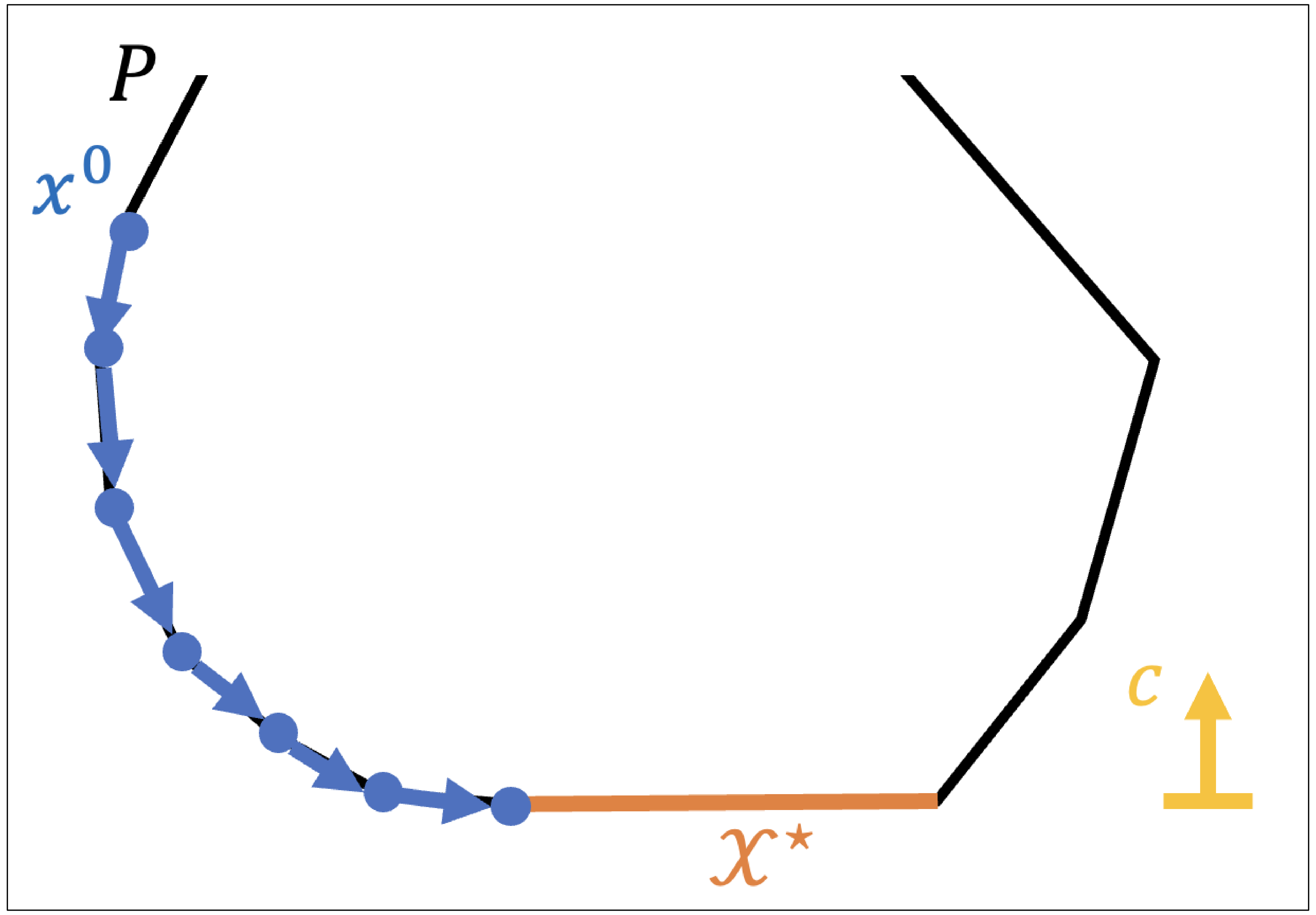
Today's Talk

- LP Warm-Start: Online Helps Offline
- Smart Crossover: From an Interior Point to a Corner Point
- ABIP: Interior Point Method Meets ADMM
- cuPDLP-C: How GPU Accelerates Solving LP
- Summary

Linear Programming: the Need of Basic Feasible Solutions

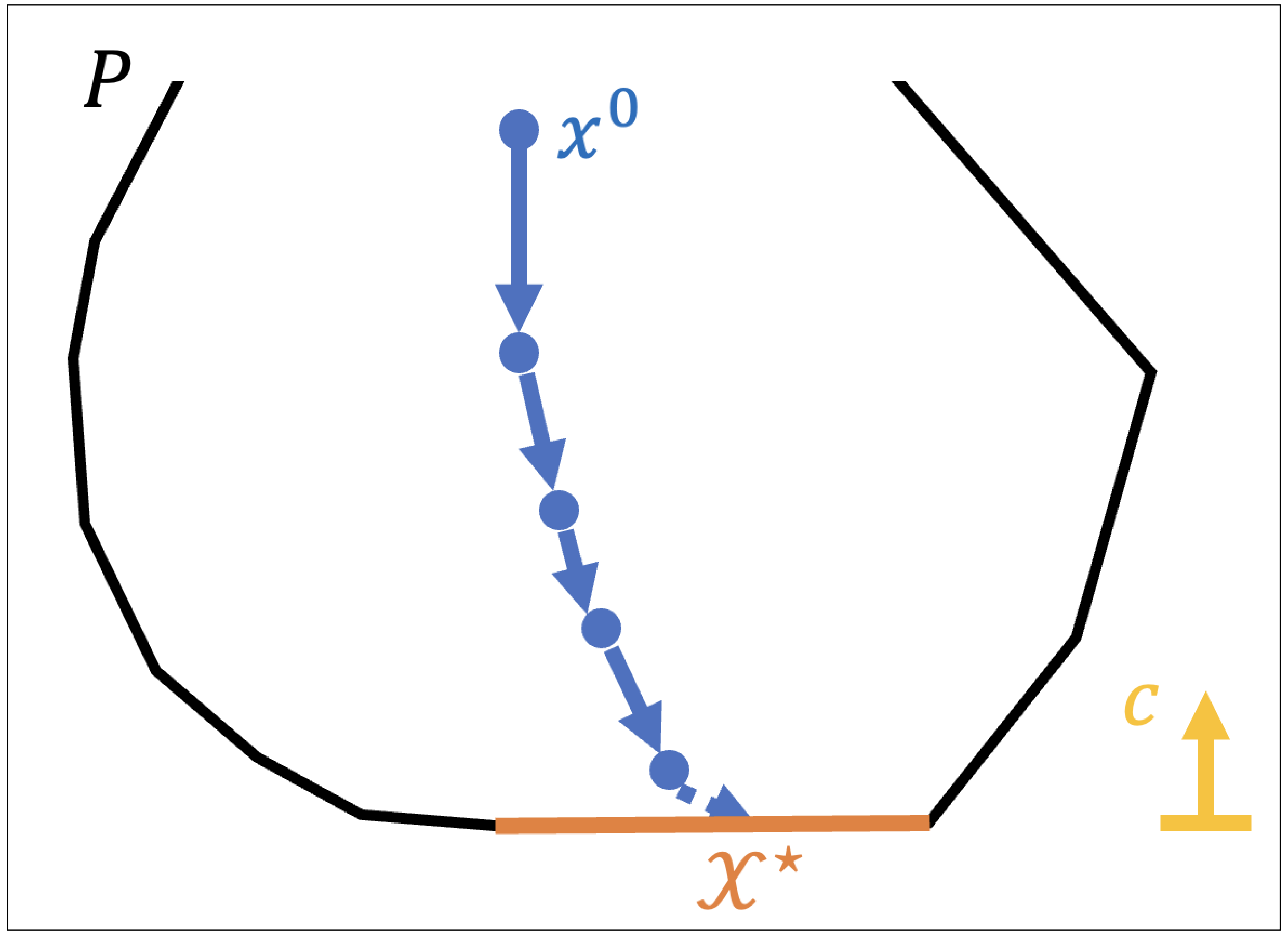
$$x^\star = \operatorname{argmin}_{x \in P} c^\top x$$

Simplex Method



Move along vertices

Interior Point Method



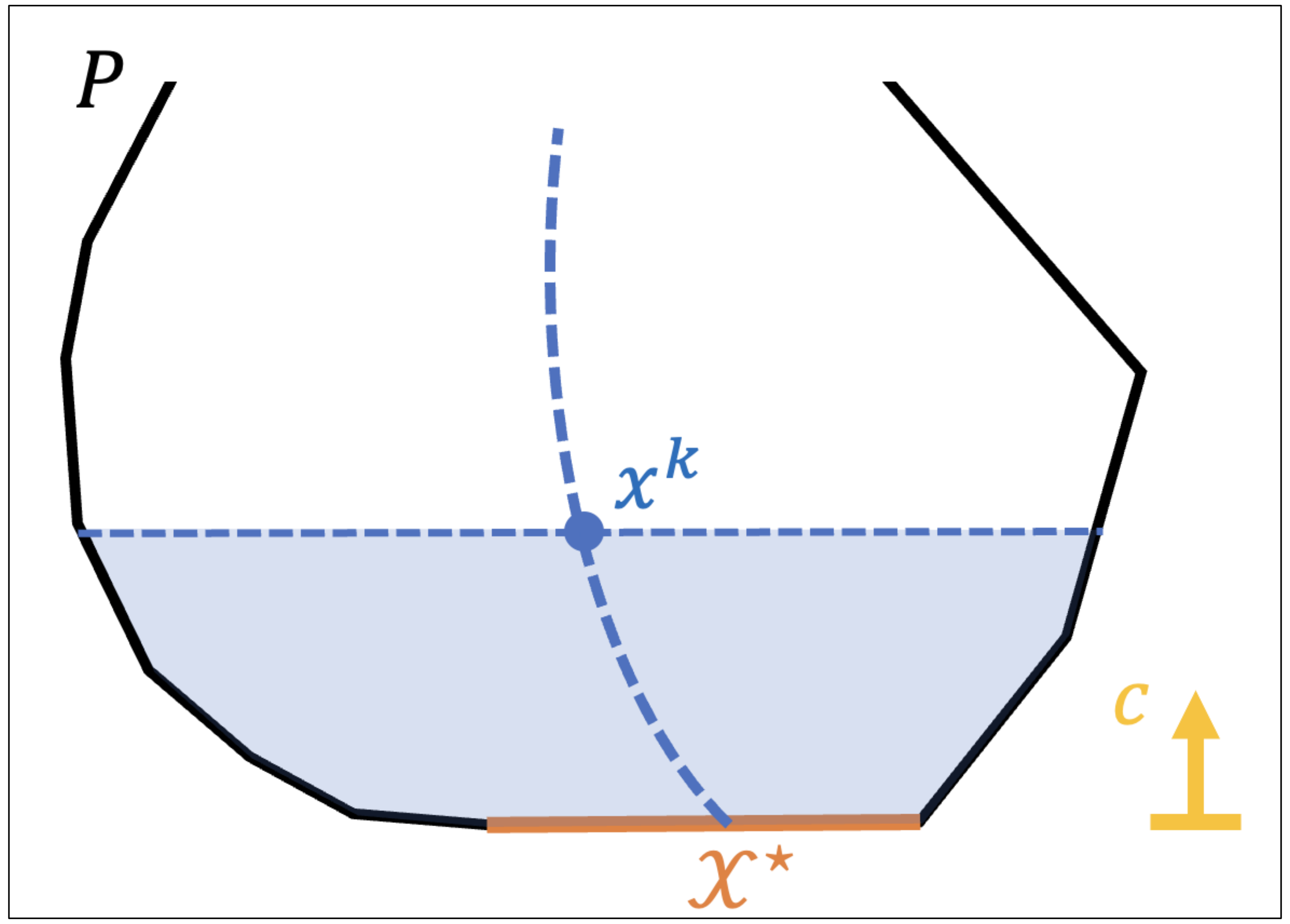
Move in the interior

Crossover is the procedure from an interior-point solution to a BFS [Andersen/Y, 1996]

From an Interior Point to a Corner Point [Ge et al. 2021]

$x^\star = \mathrm{argmin}_{x \in P} c^\top x$

IPM Stops at x^k



Goal: Find a BFS that is in the sublevel set (enough for regular tolerance)

$$P \cap \{x: c^\top x \le c^\top x^k\}$$

Our approach: Solve a randomly-perturbed-objective problem

$$\hat{x} = \mathrm{argmin}_{x \in P} (c + \Delta c)^\top x$$

- If Δc is too <u>tiny</u>, identifying the BFS $\hat{x}$ is still hard
- If Δc is too <u>large</u>, $\hat{x}$ is no longer in the sublevel set

- <u>We need theoretical guarantees to keep a balance on the size of Δc!</u>

How Large Can the Perturbation be?

Theorem:

Let x^k **be any central-path solution of** $\min_x c^\top x \ \text{s.t.} Ax = b, x \geq 0$**. Then for any** Δc **such that**

$$\|X_k \Delta c\|_2 \leq \frac{\|X_k(I - A^\top (A X_k^2 A^\top)^{-1} A X_k) c\|_2}{4\text{n} + 2},$$

let $\hat{x}$ **be the optimal solution of the perturbed problem, and then**

$$c^\top \hat{x} \leq c^\top x^k.$$

Insight:

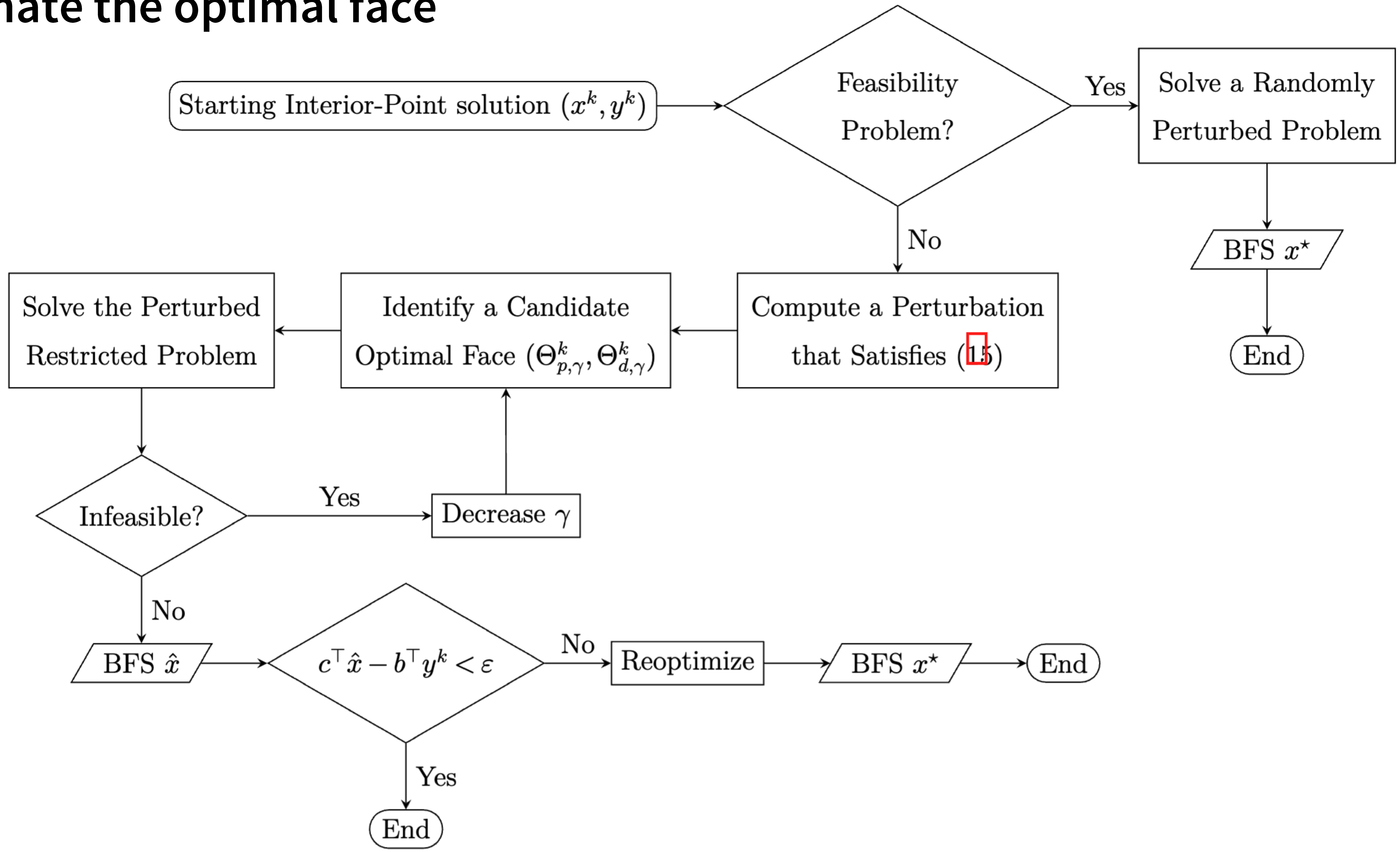
We can generate the random perturbation Δc **within this range but as large as possible.**

Flowchart of the Perturbation Crossover Method

Other heuristics:

1. Identify the feasibility problems.

2. Estimate the optimal face



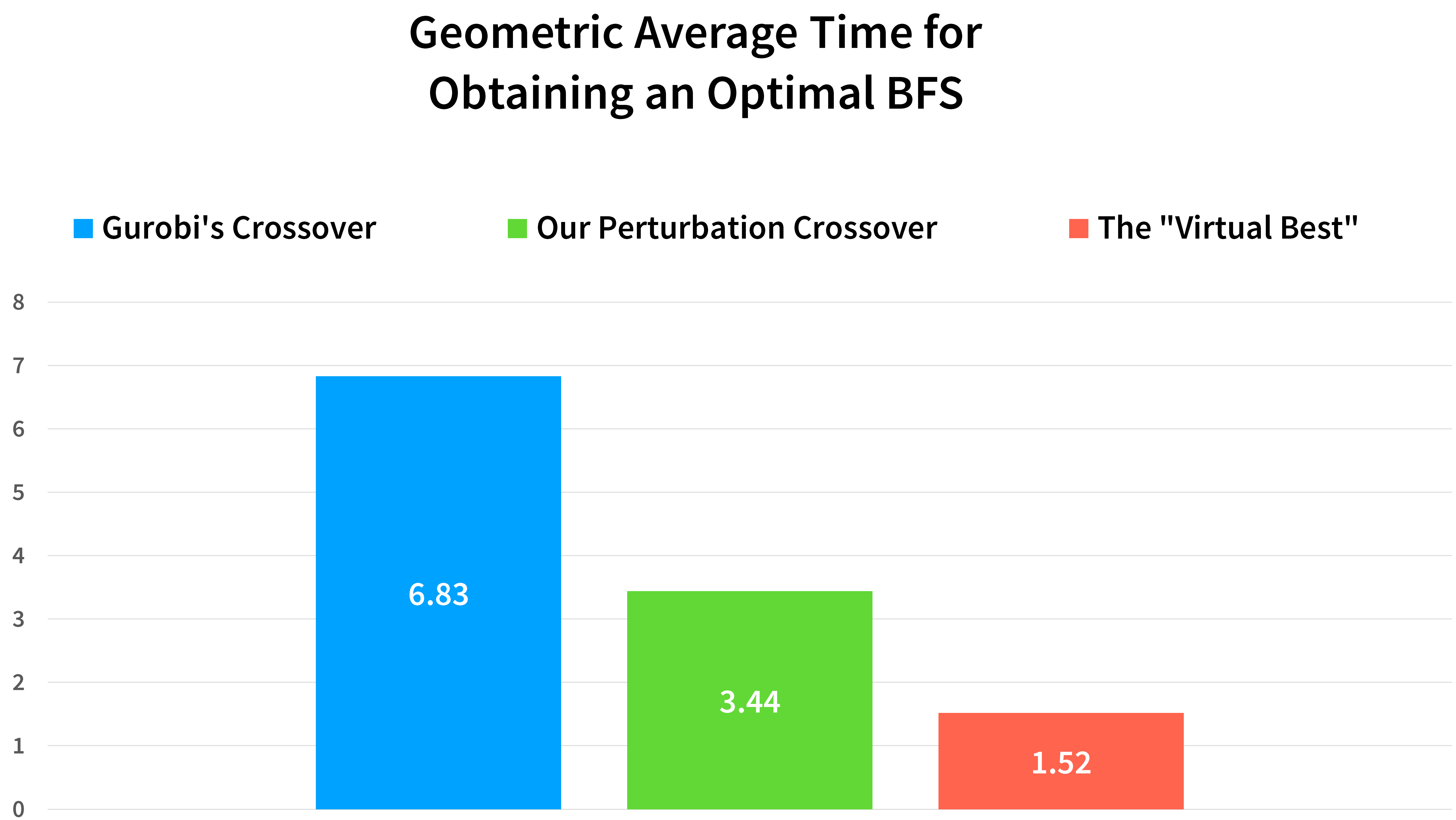
Computational Results on some LP relaxations in MIPLIB

LP relaxation of some max cut packing problems:

Problem	Dimension of optimal face	Gurobi Barrier Method (seconds)	Gurobi Crossover (seconds)	Perturbation Crossover (seconds)
graph20-20-1rand	2035	0.01	0.05	0.04
graph20-80-1rand	15912	0.05	2.42	1.11
graph40-20-1rand	20773	0.09	15.82	8.33
graph40-40-1rand	101700	0.41	323.41	50.79
graph40-80-1rand	282112	1.4	>10000	872.07

Our crossover is much faster especially when the dimension of the optimal face is large.

More Experiments on the LP Benchmark Problems (LPopt)



Some hard LP instances for crossover in Gurobi:

- datt256

  415.94 -> 18.19
- s82

  881.19 -> 0.53
- set_cover_model

  281.14 -> 1.28
- …

"Optimal": the regular relative objective gap < 1e-8

Today’s Talk

- **LP Warm-Start: Online Helps Offline**
- **Smart Crossover: from an Interior Point to a Corner Point**
- **ABIP: Interior Point Method Meets ADMM**
- **cuPDLP-C: How GPU Accelerates Solving LP**
- **Summary**

ABIP [Lin et al., 2021]

- An ADMM based interior point method solver for LP problems
- The primal-dual pair of LP:

$$(P)\quad \begin{array}{ll} \min & \mathbf{c}^\top \mathbf{x} \\ \text{s.t.} & A\mathbf{x} = \mathbf{b} \\ & \mathbf{x} \geq 0 \end{array} \qquad (D)\quad \begin{array}{ll} \max & \mathbf{b}^\top \mathbf{y} \\ \text{s.t.} & A^\top \mathbf{y} + \mathbf{s} = \mathbf{c} \\ & \mathbf{s} \geq 0 \end{array}$$

- For IPM, initial feasible interior solutions are hard to find
- So we consider homogeneous and self-dual (HSD) LP here!

$$\begin{array}{ll} \min & \beta(n+1)\theta + \mathbf{1}(\mathbf{r} = 0) + \mathbf{1}(\xi = -n-1) \\ \text{s.t.} & Q\mathbf{u} = \mathbf{v}, \\ & \mathbf{y} \text{ free}, \mathbf{x} \geq 0, \tau \geq 0, \theta \text{ free}, \mathbf{s} \geq 0, \kappa \geq 0 \end{array}$$

where

$$Q = \begin{bmatrix} 0 & A & -\mathbf{b} & \bar{\mathbf{b}} \\ -A^\top & 0 & \mathbf{c} & -\bar{\mathbf{c}} \\ \mathbf{b}^\top & -\mathbf{c}^\top & 0 & \bar{\mathbf{z}} \\ -\bar{\mathbf{b}}^\top & \bar{\mathbf{c}}^\top & -\bar{\mathbf{z}} & 0 \end{bmatrix}, \quad \mathbf{u} = \begin{bmatrix} \mathbf{y} \\ \mathbf{x} \\ \tau \\ \theta \end{bmatrix}, \quad \mathbf{v} = \begin{bmatrix} \mathbf{r} \\ \mathbf{s} \\ \kappa \\ \xi \end{bmatrix}, \quad \bar{\mathbf{b}} = \mathbf{b} - A\mathbf{e}, \quad \bar{\mathbf{c}} = \mathbf{c} - \mathbf{e}, \quad \bar{\mathbf{z}} = \mathbf{c}^\top \mathbf{e} + 1$$

ABIP – Subproblem

- Add log-barrier penalty for HSD LP and solve

$$\begin{aligned} \min \quad & B(\mathbf{u}, \mathbf{v}, \mu) \\ \text{s.t.} \quad & Q\mathbf{u} = \mathbf{v} \end{aligned}$$

- Traditional IPM applies Newton's method to solve the subproblem, which can be too expensive when problem is large!

- Apply ADMM (with splitting) to solve the kth subproblem inexactly

$$\begin{aligned} \min \quad & \mathbf{1}(Q\tilde{\mathbf{u}} = \tilde{\mathbf{v}}) + B\big(\mathbf{u}, \mathbf{v}, \mu^k\big) \\ \text{s.t.} \quad & (\tilde{\mathbf{u}}, \tilde{\mathbf{v}}) = (\mathbf{u}, \mathbf{v}) \end{aligned}$$

where the augmented Lagrangian function

$$\mathcal{L}_\beta\big(\tilde{\mathbf{u}}, \tilde{\mathbf{v}}, \mathbf{u}, \mathbf{v}, \mu^k, \mathbf{p}, \mathbf{q}\big) := \mathbf{1}(Q\tilde{\mathbf{u}} = \tilde{\mathbf{v}}) + B\big(\mathbf{u}, \mathbf{v}, \mu^k\big) - \langle \beta(\mathbf{p}, \mathbf{q}), (\tilde{\mathbf{u}}, \tilde{\mathbf{v}}) - (\mathbf{u}, \mathbf{v}) \rangle + \frac{\beta}{2} \|(\tilde{\mathbf{u}}, \tilde{\mathbf{v}}) - (\mathbf{u}, \mathbf{v})\|^2$$

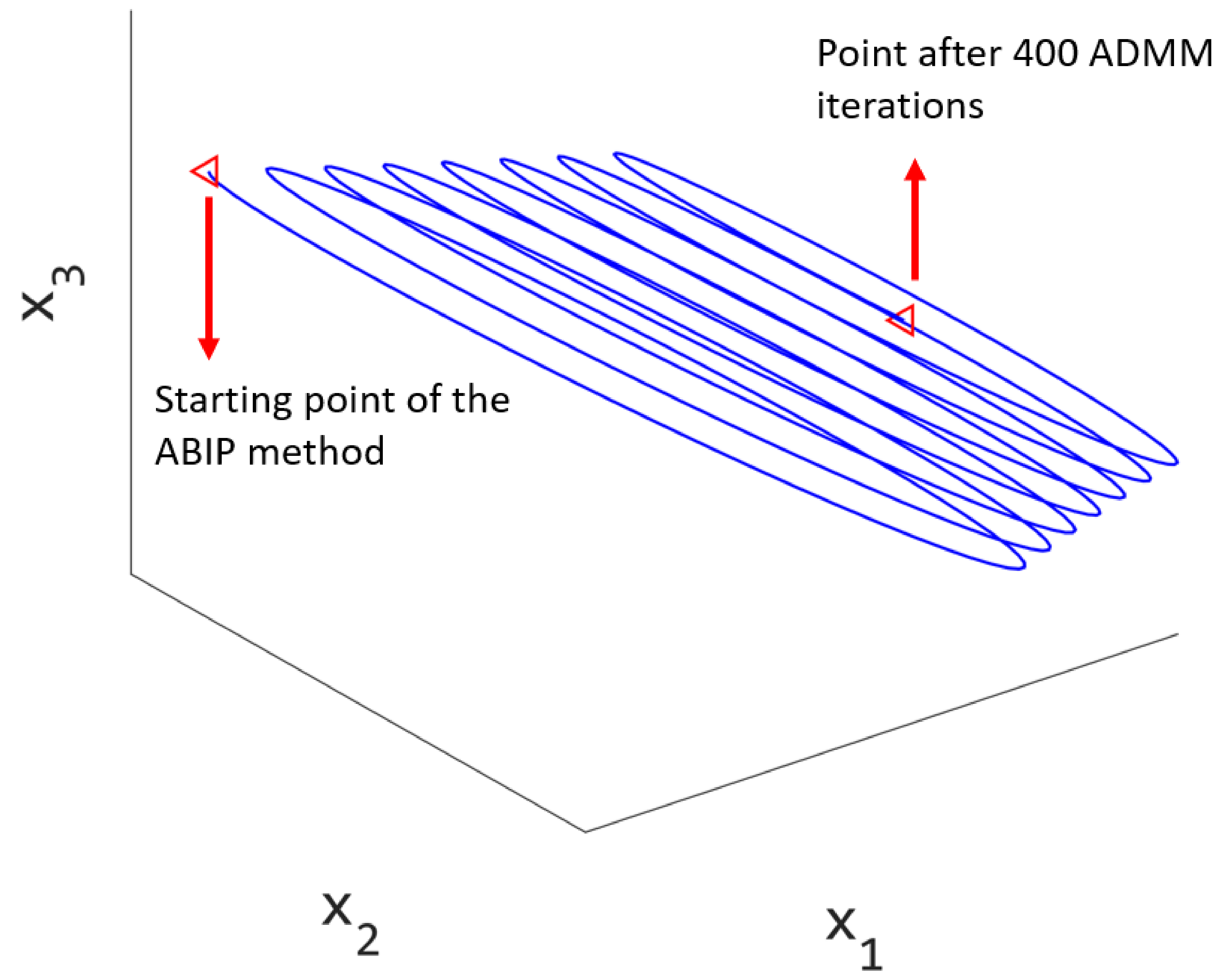
ABIP+ – Enhancements [Deng et al., 2022]

Motivation	Enhancement
ADMM	Rescaling
	Restart
	Half-update
IPM	Adaptive barrier parameter
Practice	Inner loop convergence check
	Strategy integration
Extension	Quadratic conic programming

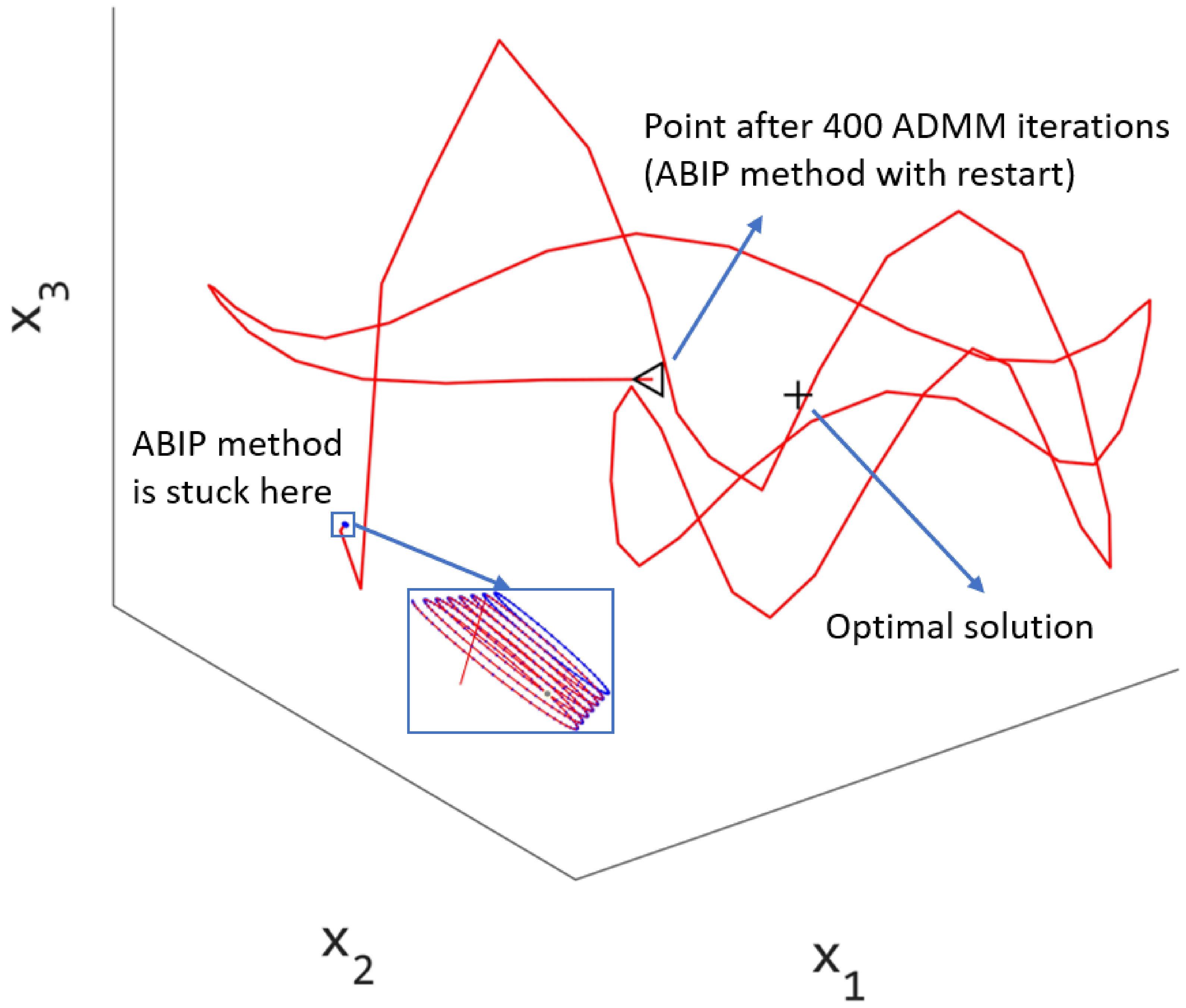
Various enhancements significantly improve ABIP!

ABIP+ – Restart

- Idea: Let the uniform average of the past few points be the new starting point
- ABIP (or first-order method in general) tends to induce a spiral trajectory
- After restart, ABIP moves more aggressively and converges faster (reduce almost 70% ADMM iterations) !



Before restart



After restart

Instance SC50B (only plot the first two dimension)

Computational Results on Netlib

- **Selected 105 Netlib instances**
- $\boldsymbol{\epsilon = 10^{-6}, 10^{6}}$ **max ADMM iterations**

Method	# Solved	# IPM	# ADMM	Avg.Time (s)
ABIP	65	74	265418	87.07
+ restart	68	74	88257	23.63
+ rescale	84	72	77925	20.44
+ hybrid μ (=ABIP+)	**86**	**22**	**73738**	**14.97**

- **Hybrid $\boldsymbol{\mu}$: If $\boldsymbol{\mu > \epsilon}$ use the aggressive strategy, otherwise use the LOQO strategy**
- **ABIP+ decreases both # IPM iterations and # ADMM iterations significantly**

Computational Results on PageRank Problems

- 117 instances, generated from sparse matrix datasets: DIMACS10, Gleich, Newman and SNAP, where Second order methods in commercial solver fail in most of these instances.

- $\epsilon = 10^{-4}$, 5000 max ADMM iterations.

Method	# Solved	SGM
PDLP(Julia)	**117**	**1**
ABIP+	114	1.28

- In staircase matrix case (# nodes = # edges), ABIP+ is significantly faster than PDLP!

# nodes	PDLP (Julia)	ABIP+
10^4	8.60	**0.93**
10^5	135.67	**10.36**
10^6	2248.40	**60.32**

[PDLP, Applegate et al., 2021, 2023]

Today’s Talk

- **Online Warm-Start: Online Helps Offline**
- **Smart Crossover: From an Interior Point to a Corner Point**
- **ABIP: Interior Point Method meets ADMM**
- **cuPDLP-C: How GPU Accelerates Solving LP**
- **Summary**

Drawbacks for the simplex method and IPMs

Factorization is memory demanding

- A sparse matrix may induce dense decomposition
- Factorization is difficult for huge-size problems (>10^9 variables)

Difficult for GPU and parallelization

- Factorization is not as efficient on GPU
- Operations like pivoting are hard to parallelize
- CPU and GPU communication problems.

Recent progresses

- Parallelizing first-order methods for Linear programming on GPU
- Utilizing matrix-vector products on GPU
- Julia prototype: cuPDLP.jl (Lu/Yang, 2023)
- C implementation and solver enhancements: cuPDLP-C (Lu et al., 2024)

Primal-Dual Hybrid Gradient for Linear Programming

- **cuPDLP uses the saddle-point formulation of LP**

$$\begin{aligned} \min_{x \in \mathbb{R}^n} \quad & c^\top x \\ \text{s.t.} \quad & Gx \geq h \\ & Ax = b \\ & l \leq x \leq u \ , \end{aligned}$$

$$\min_{x \in X} \max_{y \in Y} \ L(x, y) := c^\top x - y^\top K x + q^\top y \ ,$$

$$\begin{cases} x^{t+1} \leftarrow \text{proj}_X(x^t - \tau(c - K^\top y^t)) \\ y^{t+1} \leftarrow \text{proj}_Y(y^t + \sigma(q - K(2x^{t+1} - x^t))) \end{cases} ,$$

An Iteration of PDHG [Esser at al. 2010]:

- **Computing $Kx, K^T y$ by sparse matrix-vector product (spmv)**
- **Choosing step sizes: τ, σ**
- **PDLP Adaptive line-search: Applegate et al. (2021,2023), Lu/Yang (2023)**
- **All operations can be done on GPU!**

Selected MIPLIB Instances

Instances	Variables	Constraints	Non-zeros
Packing Cuts in Undirected Graphs.			
graph20-80-1rand	16263	55107	191997
graph40-20-1rand	31243	99067	345557
graph40-40-1rand	102600	360900	1260900
graph40-80-1rand	283648	1050112	3671552
Open Pit Mining over a cube considering multiple time periods and two knapsack constraints per period.			
rmine11	12292	97389	241240
rmine13	23980	197155	485784
rmine15	42438	358395	879732
rmine21	162547	1441651	3514884
rmine25	326599	2953849	7182744
Unit Commitment problems (electricity production planning problems)			
uccase7	33020	47132	335644
uccase8	37413	53709	214625
uccase9	33242	49565	332316
uccase10	110818	196498	787045
uccase12	62529	121161	419447

Computational Results on Selected MIPLIB instances

Instances	cuPDLP.jl V100	cuPDLP.jl H100	cuPDLP-C H100	Gurobi Barrier	COPT Barrier 1th, 16G	COPT Barrier 12 th, 128G
graph20-80-1rand	1.16	0.86	0.13	0.21	0.04	0.04
graph40-20-1rand	1.16	0.87	0.15	0.36	0.06	0.06
graph40-40-1rand	1.19	0.84	0.30	1.62	0.12	0.14
graph40-80-1rand	1.73	1.02	0.88	5.72	0.43	0.44
rmine11	42.81	32.80	16.70	9.79	5.06	2.26
rmine13	28.35	56.62	12.09	38.31	15.23	4.20
rmine15	35.14	32.02	22.40	149.59	68.90	13.55
rmine21	441.16	830.18	148.49	2674.46	1361.07	207.33
rmine25	1411.57	409.39	246.33	> 3600.00	> 3600.00	1839.05
uccase7	62.26	82.04	38.34	3.98	2.57	1.66
uccase8	14.57	14.92	7.04	2.62	1.86	1.18
uccase9	66.49	58.31	13.40	4.46	3.09	2.04
uccase10	65.49	99.36	20.76	2.68	1.22	0.90
uccase12	45.53	37.41	20.22	1.53	0.59	0.62

- **GPU solver is less influenced by problem sizes**

Strengthening with other LP Techniques

Dataset	Optimizer	Presolver	Tol.	SGM10	Solved
MIPLIB (383)	COPT	-	10^{-8}	3.11	383
	cuPDLP-C	COPT	10^{-4}	5.43	379
			10^{-8}	18.53	369
		HiGHS	10^{-4}	6.12	373
			10^{-8}	20.08	365
		CLP	10^{-4}	7.95	372
			10^{-8}	21.89	362
		No Presolve	10^{-4}	10.28	370
			10^{-8}	27.15	359
	cuPDLP.jl	No Presolve	10^{-4}	17.49	370
			10^{-8}	35.69	355
Mittelmann (49)	COPT	-	10^{-8}	13.81	48
	cuPDLP-C	COPT	10^{-4}	25.29	46
			10^{-8}	110.22	41
		HiGHS	10^{-4}	31.84	46
			10^{-8}	128.39	41
		CLP	10^{-4}	33.97	45
			10^{-8}	125.95	38
		No Presolve	10^{-4}	57.54	43
			10^{-8}	172.98	39

- **Julia Prototype: cuPDLP.jl (Lu/Yang, 2023)**
- **C Implementation: cuPDLP-C (Lu et al., 2024)**
- **LP scaling and presolving techniques significantly improve the GPU solver**
- **cuPDLP-C with HiGHS backend are open-sourced at:**

  `github.com/COPT-Public/cuPDLP-C`

Milestones of Solving a Well-Known "Intractable" Instance

In a workshop in January 2008 on the *Perspectives in Interior Point Methods for Solving Linear Programs*, the instance `zib03` with 29,128,799 columns, 19,731,970 rows and 104,422,573 non-zeros was made public. As it turned out, the simplex algorithm was not suitable to solve it and barrier methods needed at least about 256 GB of memory, which was not easily available at that time. The first to solve it was Christian Bliek in April 2009, running CPLEX out-of-core with eight threads and converging in 12,035,375 seconds (139 days) to solve the LP without crossover. Each iteration took 56 hours! Using modern codes on a machine with 2 TB memory and 4 E7-8880v4 CPUs @ 2.20 GHz with a total of 88 cores, this instance can be solved in 59,432 seconds = 16.5 hours with just 10% of the available memory used. This is a speed-up of 200 within 10 years. However, when the instance was introduced in 2008, none of the codes was able to solve it. Therefore there was infinite progress in the first year. Furthermore, 2021 was the first time we were able to compute an optimal *basis* solution.

2008: Instance zib03[1]
29,128,799 variables
19,731,970 constraints
104,422,573 non-zeros
Presolve can't really reduce it

2009: Cplex Barrier (without crossover)
139 days (56 hours/IPM-iteration)

2019: IPM on a more advanced machine
16.5 hours

2023-24: cuPDLP-C (to 1e-6 tolerance)
1.7 hours on NVIDIA A6000
27 minutes on NVIDIA H100!

[1]Koch, Thorsten, et al. "Progress in mathematical programming solvers from 2001 to 2020." EURO Journal on Computational Optimization 10 (2022): 100031.

Today’s Talk

- **LP Warm-Start: Online Helps Offline**
- **Smart Crossover: From an Interior Point to a Corner Point**
- **ABIP: Interior Point Method Meets ADMM**
- **cuPDLP-C: How GPU Accelerates Solving LP**
- **Summary**

Scientific Research Drives (Conic) LP Solver Development

COPT Barrier solver [User guide Ge at al. 2022]

- Added in COPT 1.4, October 2020
- Leading in Barrier Benchmark since June 2021 (COPT 2)
- Continue to lead in new LP benchmarks since October 2022

There are 49 public and 16 undisclosed LP problems in new LP benchmark.

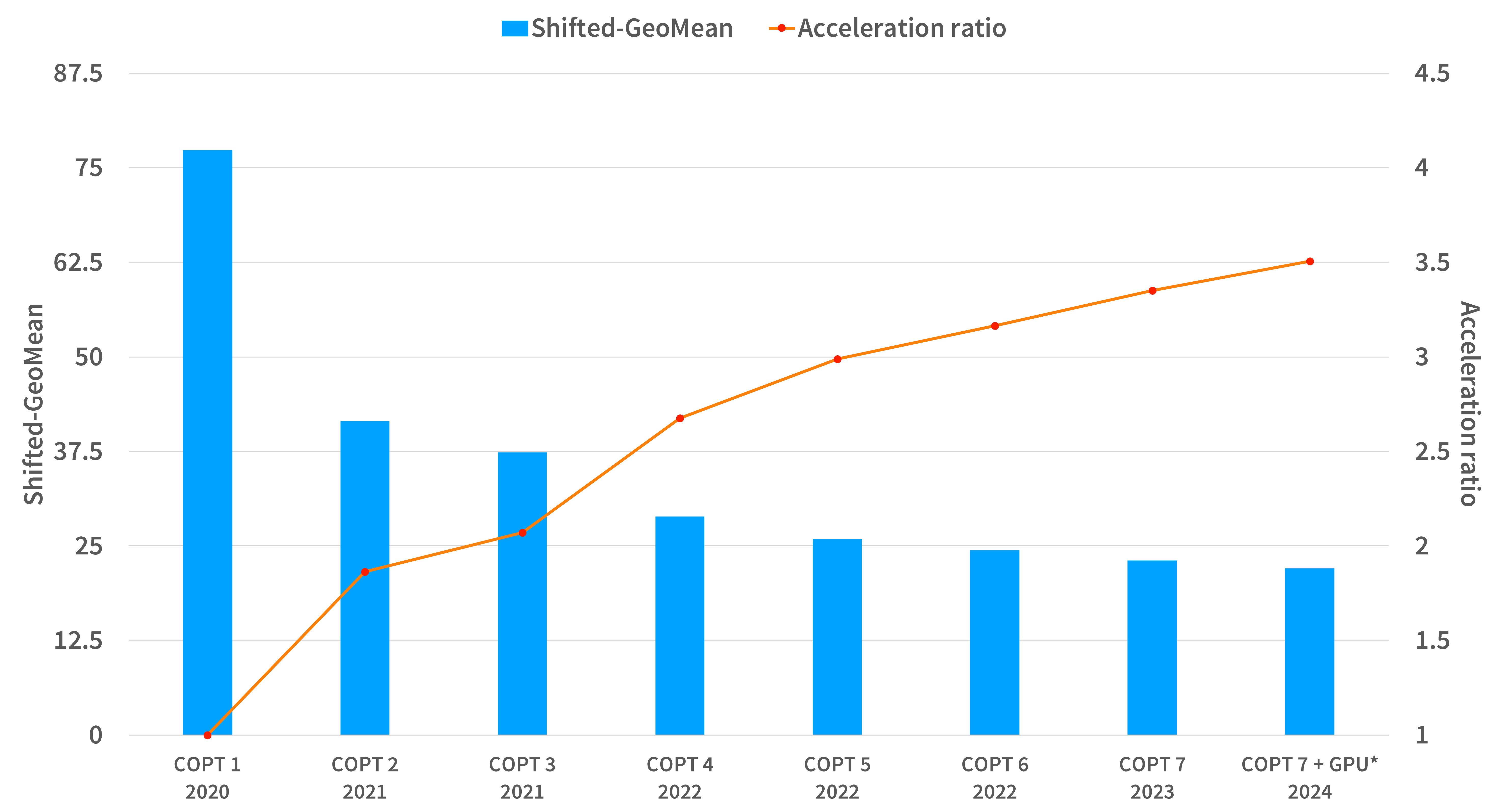
COPT is the only solver that can solve all of them in time.

Barrier is more often the best choice for soling LP.

Key Features

- High performance presolver
- Deterministic Parallel Cholesky
- # threads-independent behaviors
- Parallel crossover
- Smart crossover

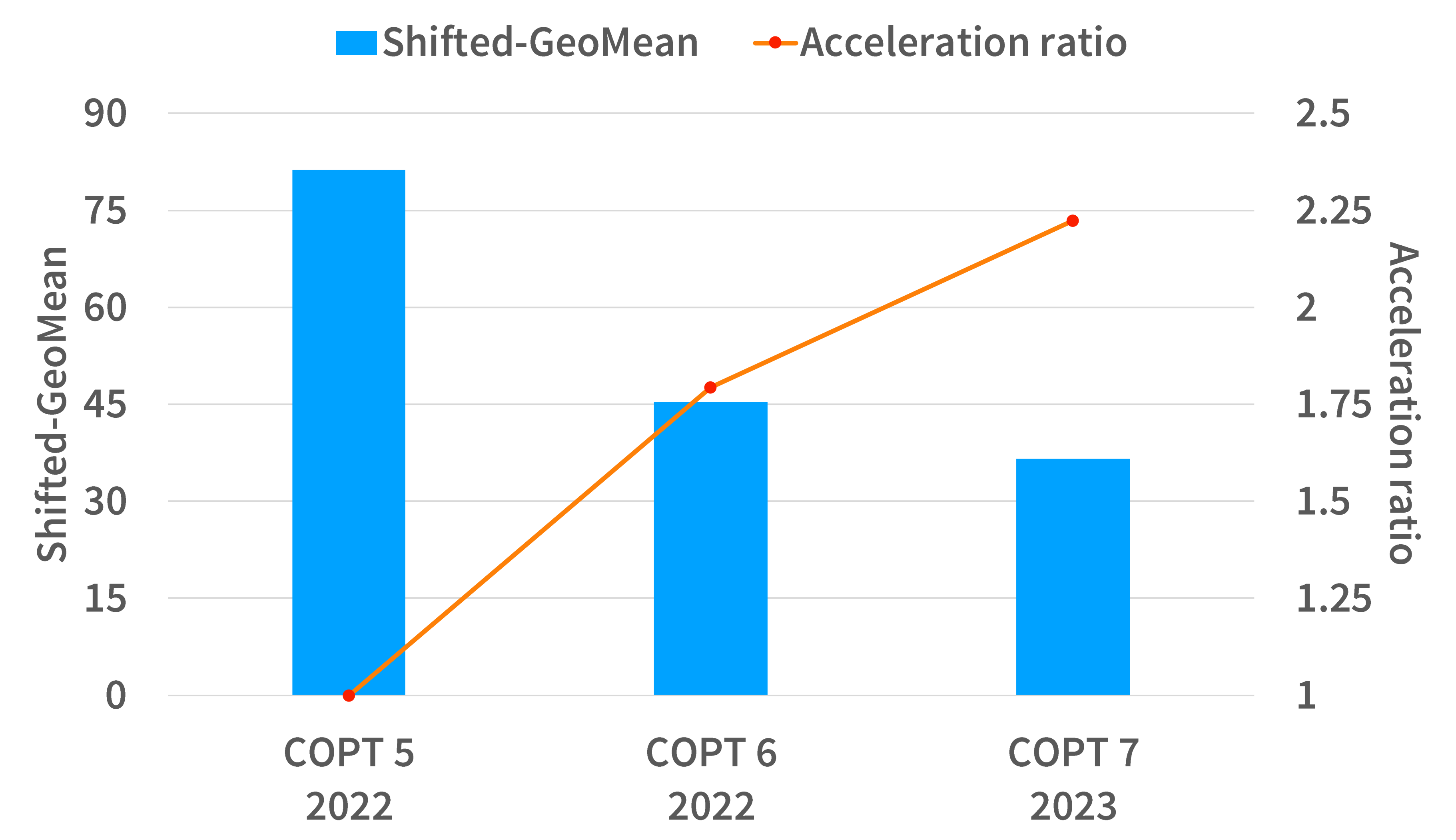
Performance Advances COPT 1 – 7 on Solving LP



Barrier	Time	Improvement	Note
COPT 1	2020.10	Initial barrier LP solver release.	
COPT 2	2021.05	Independently development efficient alternatives for MKL/Pardiso, allows for better parallelization and numerical handling.	Solves set-cover-model 1.95 times faster.
COPT 3	2021.10	Developed and Implemented smart crossover.	Solves datt256 18.8 times faster.
COPT 4	2022.01	Improved parallel crossover implementation.	Solves a2864-99blp 2.02 times faster.
COPT 5	2022.06	Improved barrier ordering.	Solves dlr1 36% faster.
COPT 6	2022.10	Improved LP presolver.	Solves rail02 28% faster.
COPT 7	2023.09	Revised starting point computation.	Solves s82 45% faster.
COPT 7 + GPU*	2024.01	Added PDLP with GPU support.	Solves thk_63 40% faster.

- Tested on 49 public LP benchmark problems from Hans Mittelmann, using time limit 15000.
- The PDLP GPU version also solves to optimal basis, where the crossover is finished on CPU.
- COPT 7 + GPU* = Best of COPT 7 and PDLP with GPU support.
- Hardware: CPU: AMD 5900X (12 Threads) with 128G memory and NVIDIA 4090 with 24G memory.

Performance Advances COPT 5 – 7 on Solving SDP



SDP	Time	Improvement	Note
COPT 5	2022.06	Initial SDP solvers release with all of Primal-Dual, ABIP/ADMM and Dual method.	
COPT 6	2022.10	• Rewrote and improved ABIP/ADMM implementation. • Rewrote and improved Dual method implementation.	• Solves theta12 7.5 times faster. • Solves G55mc 6.85 times faster.
COPT 7	2023.09	Improved Primal-Dual method parallelism for large SDPs with many cones.	Solves Bex2_1_5 93% faster.

- Testing machine AMD 5900X with 128G memory.
- Testing time limit 40000s.
- **COPT 7.0 leads in the Mittelmann SDP benchmark (Feb. 1, 2024).**

```
1 Feb 2024  ==================================================
            Several SDP-codes on sparse and other SDP problems
            ==================================================
                 Hans D. Mittelmann (mittelmann@asu.edu)

Scaled shifted geometric means of runtimes ("1" is fastest solver)
                  1     5.21     3.64     10.5     5.14     28.9     7.86     1.44
---------------------------------------------------------------------------------
count of "a"      6        5        0       17       13        2       11       12
solved of 75     75       70       73       61       69       62       70       75
=================================================================================
problem        COPT     CSDP    MOSEK     SDPA    SDPT3   SeDuMi    HDSDP    MDOPT
=================================================================================
```

COPT Standings

- In 2019, COPT first stood on the solver stage with its high-performance LP simplex solver.
- At present, COPT 7.0 has become one of the fastest solver in the world for various problem types.

Benchmarks for Optimization Software

http://plato.asu.edu/guide.html

by Prof. Hans Mittelmann

16 May 2019

Benchmark of Simplex LP solvers

H. Mittelmann (mittelmann@asu.edu)

Logfiles of these runs at: plato.asu.edu/ftp/lp_logs/

This benchmark was run on a Linux-PC (i7-4790K, 4.0GHz, 32GB).
The MPS-datafiles for all testcases are in one of (see column "s")

miplib.zib.de/ [1]
plato.asu.edu/ftp/lptestset/ [2]
www.netlib.org/lp/data/ [3,7]
www.sztaki.hu/~meszaros/public_ftp/lptestset/
(MISC[4], PROBLEMATIC[5], STOCHLP[6], INFEAS[8])

NOTE: files in [2-8] need to be expanded with emps in same directory!

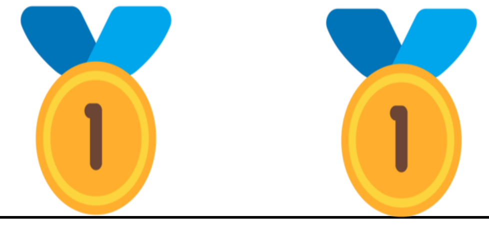
The simplex methods were tested of the codes:

MOSEK-9.0.86	www.mosek.com
CLP-1.17.0	projects.coin-or.org/Clp (with openblas)
Google-GLOP	LP with Glop
SOPLEX-4.0.0	soplex.zib.de/
LP_SOLVE-5.5.2	lpsolve.sourceforge.net/
GLPK-4.64	www.gnu.org/software/glpk/glpk.html
MATLAB-R2018a	mathworks.com (dual-simplex)
SAS-OR-14.3:	SAS
HiGHS-1.0.0:	HiGHS
COPT-1.0.0:	COPT

Unscaled and scaled shifted (by 10 sec) geometric mean of runtimes

40 probs	MSK	CLP	GLOP	SPLX	LPSLV	GLPK	MATL	SAS	HiGHS	COPT
	137	45.4	292	461	5068	1180	298	147	240	34.5
	3.97	1.32	8.45	13.4	147	34.2	8.65	4.26	6.96	1
solved	38	40	40	36	23	31	32	38	37	40
L1_sixm	350	402	f	13342	11965	2536	f	f	3030	39
Linf_520c	f	48	249	t	523	1358	1433	3396	1212	121
buildingen	382	158	267	316	14128	652	309	97	207	27
cont1	208	277	656	7508	398	f	32	449	1185	451
cont11	19268	1070	3025	16851	10537	f	f	1413	2103	2580
cont4	700	216	338	907	503	f	f	289	f	285
dano3mip	10	4	3	14	17455	5	49	13	17	20
dbic1	55	26	17	226	345	137	157	14	448	24
ds-big	156	218	318	t	t	712	338	355	276	160
fome12	54	25	64	78	506	571	38	45	45	29
fome13	139	49	232	233	6498	3574	179	99	111	8
gen4	1	5	11	8	463	25	2	1	3	1
ken-18	4	2	44	65	1215	541	8	11	6	1
l30	6	12	39	35	f	14	5	8	4	5
mod2	16	17	42	82	92	210	26	11	29	8
neos	67	29	105	61	1616	5510	387	319	353	34
neos1	1	4	50	13	11644	13	39	10	9	5
neos2	1	5	163	19	f	15	314	11	32	12
neos3	8	29	404	9881	t	3617	f	552	1390	4
ns1644855	236	20	77	118	t	29	220	86	671	82
ns1687037	449	408	1501	725	t	3247	f	f	1036	2400
ns1688926	t	17	t	104	t	t	f	18	12	17
nug15	9796	13	230	12533	t	398	371	3104	9997	12
nug08-3rd	2311	177	f	1178	f	t	f	3954	149	33

Simplex Benchmark, 2019

Problem Types	Ranking
Linear Programming	
Mixed Integer Linear Programming	
Second-Order Cone Programming	
Convex Quadratic Programming and Convex Quadratically Constrained Programming	
Semi-Definite Programming	
Mixed Integer Second-Order Cone Programming	
Mixed Integer Convex Quadratic Programming	

Optimization Benchmark, Oct. 25, 2023

LP Real-World Applications (from Cardinal Operations)

Education and Academic Research

Energy and Electricity

Industry 4.0

Supply Chain

Aviation

Transportation

Finance

Warehouse and Logistics

Long Live – Linear Programming