

On Inferring Cumulative Constraints

Konstantin Sidorov

Delft University of Technology

CP 2026

One resource at a time

A typical scheduling CP model bundles m resources into m separate **Cumulative** constraints over shared start-time variables:

$$\text{Cumulative}(s, a_i, d, b_i) \quad (i = 1, \dots, m)$$

One resource at a time

A typical scheduling CP model bundles m resources into m separate **Cumulative** constraints over shared start-time variables:

$$\text{Cumulative}(s, a_i, d, b_i) \quad (i = 1, \dots, m)$$

- Each constraint gets its **own propagator**

One resource at a time

A typical scheduling CP model bundles m resources into m separate **Cumulative** constraints over shared start-time variables:

$$\text{Cumulative}(s, a_i, d, b_i) \quad (i = 1, \dots, m)$$

- Each constraint gets its **own propagator**
- Decades of work made *single*-resource propagation excellent — overload checks, energetic reasoning, explanations

One resource at a time

A typical scheduling CP model bundles m resources into m separate **Cumulative** constraints over shared start-time variables:

$$\text{Cumulative}(s, a_i, d, b_i) \quad (i = 1, \dots, m)$$

- Each constraint gets its **own propagator**
- Decades of work made *single*-resource propagation excellent — overload checks, energetic reasoning, explanations
- Almost none of that reasons about **two resources jointly**

One resource at a time

A typical scheduling CP model bundles m resources into m separate **Cumulative** constraints over shared start-time variables:

$$\text{Cumulative}(s, a_i, d, b_i) \quad (i = 1, \dots, m)$$

- Each constraint gets its **own propagator**
- Decades of work made *single*-resource propagation excellent — overload checks, energetic reasoning, explanations
- Almost none of that reasons about **two resources jointly**
- Interaction between resources happens only **indirectly**, via domain narrowing of s

The $3n$ problem

The $3n$ problem

$3n$ problem

is a family of instances built from $3n$ tasks and three cumulative resources

The $3n$ problem

$3n$ problem

is a family of instances built from $3n$ tasks and three cumulative resources



Equivalent to a **trivially infeasible** single-machine problem — but **intractable** for a conventional lazy-clause-generation solver

The $3n$ problem

$3n$ problem

is a family of instances built from $3n$ tasks and three cumulative resources



Equivalent to a **trivially infeasible** single-machine problem — but **intractable** for a conventional lazy-clause-generation solver



Because no single resource's propagator ever sees the *combined* structure

Prior work: Unite and Lead (UnL)

Prior work: Unite and Lead (UnL)

- Discovers **disjunctive cliques** — sets of tasks that provably never overlap — by probing during search

Prior work: Unite and Lead (UnL)

- Discovers **disjunctive cliques** — sets of tasks that provably never overlap — by probing during search
- Handles the pathological structure well when it's there

Prior work: Unite and Lead (UnL)

- Discovers **disjunctive cliques** — sets of tasks that provably never overlap — by probing during search
- Handles the pathological structure well when it's there
- Limitation 1: **only disjunctive reasoning** — no path to general multi-resource interactions

Prior work: Unite and Lead (UnL)

- Discovers **disjunctive cliques** — sets of tasks that provably never overlap — by probing during search
- Handles the pathological structure well when it's there
- Limitation 1: **only disjunctive reasoning** — no path to general multi-resource interactions
- Limitation 2: probing happens **throughout search**, whether or not any structure exists

Prior work: Unite and Lead (UnL)

- Discovers **disjunctive cliques** — sets of tasks that provably never overlap — by probing during search
- Handles the pathological structure well when it's there
- Limitation 1: **only disjunctive reasoning** — no path to general multi-resource interactions
- Limitation 2: probing happens **throughout search**, whether or not any structure exists



When UnL doesn't help, it often slows the solver down by at least **an order of magnitude**

Occupancy-vector view

Task j occupies the resource on $[s_j, s_j + d_j)$:



Occupancy-vector view

Task j occupies the resource on $[s_j, s_j + d_j)$:

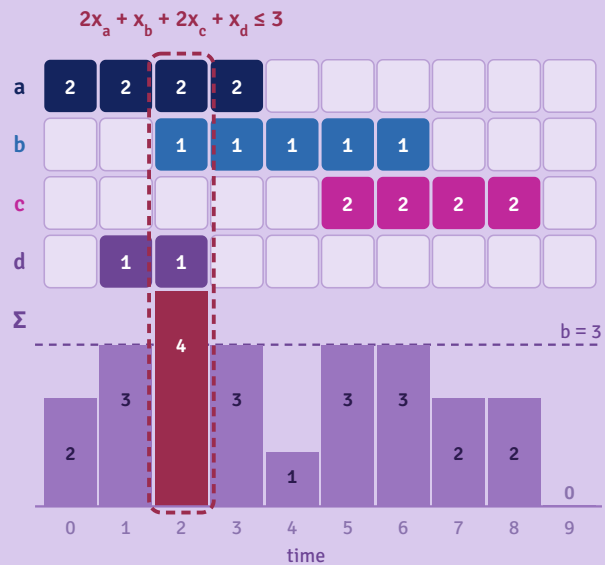


We write this as a binary vector, one entry per time point:

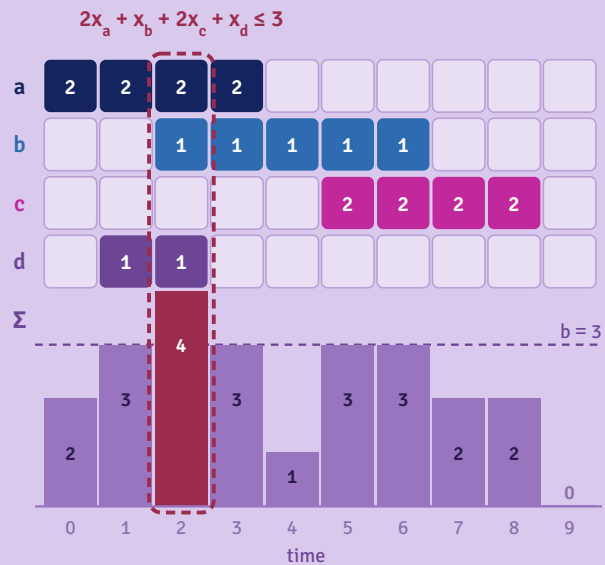
$$\langle x \mid d \rangle_t := \llbracket x \leq t < x + d \rrbracket$$

No different from any other 0/1 vector.

Cumulative is linear on occupancy vectors



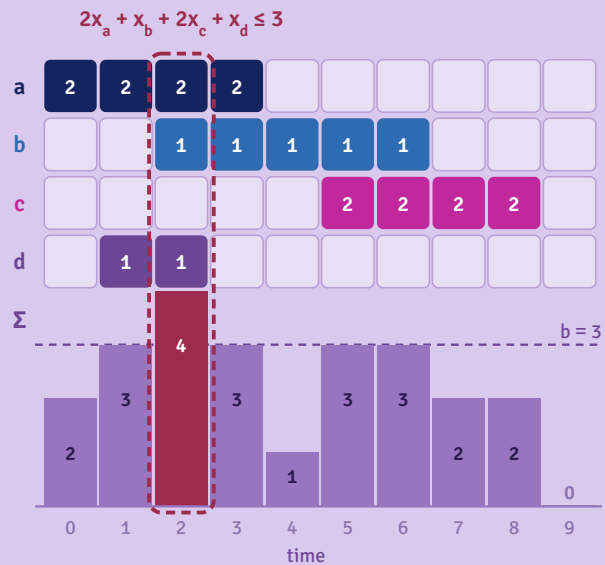
Cumulative is linear on occupancy vectors



Stack occupancy vectors into a matrix, and a system of Cumulative s becomes **one matrix inequality**:

$$A \begin{pmatrix} \langle s_1 | d_1 \rangle \\ \vdots \\ \langle s_n | d_n \rangle \end{pmatrix} \leq (b \quad \dots \quad b)$$

Cumulative is linear on occupancy vectors



Stack occupancy vectors into a matrix, and a system of Cumulative s becomes **one matrix inequality**:

$$A \begin{pmatrix} \langle s_1 | d_1 \rangle \\ \vdots \\ \langle s_n | d_n \rangle \end{pmatrix} \leq (b \quad \dots \quad b)$$

This is just a system of 0/1 linear inequalities — the entire toolbox of integer programming now applies

What *other* inequalities hold there?

Inference transfer: any valid inequality $\pi^\top x \leq \pi_0$ of this polyhedron translates back into a *new valid* `Cumulative`(s, π, d, π_0) constraint

What *other* inequalities hold there?

Inference transfer: any valid inequality $\pi^\top x \leq \pi_0$ of this polyhedron translates back into a *new valid* `Cumulative`(s, π, d, π_0) constraint

It doesn't matter *how* we found (π, π_0) : valid for the occupancy polyhedron $\Rightarrow$ valid for the scheduling problem.

Multi-resource Cumulative inference
is *cutting-plane generation*

Discovering better constraints by lifting

	a	b	c	d	e	Capacity	Polyhedral form
Duration	2	1	1	1	1		
Resource 1 usage	4	4	4	2	2	10	$4x_a + 4x_b + 4x_c + 2x_d + 2x_e \leq 10$
Resource 2 usage	3	3	3	4	2	9	$3x_a + 3x_b + 3x_c + 4x_d + 2x_e \leq 9$

Discovering better constraints by lifting

	a	b	c	d	e	Capacity	Polyhedral form
Duration	2	1	1	1	1		
Resource 1 usage	4	4	4	2	2	10	$4x_a + 4x_b + 4x_c + 2x_d + 2x_e \leq 10$
Resource 2 usage	3	3	3	4	2	9	$3x_a + 3x_b + 3x_c + 4x_d + 2x_e \leq 9$

- **Seed cover:** $\{a, b, c\}$ – on R1, $4 + 4 + 4 = 12 > 10$, so not all three can run together

Discovering better constraints by lifting

	a	b	c	d	e	Capacity	Polyhedral form
Duration	2	1	1	1	1		
Resource 1 usage	4	4	4	2	2	10	$4x_a + 4x_b + 4x_c + 2x_d + 2x_e \leq 10$
Resource 2 usage	3	3	3	4	2	9	$3x_a + 3x_b + 3x_c + 4x_d + 2x_e \leq 9$

- **Seed cover:** $\{a, b, c\}$ – on R1, $4 + 4 + 4 = 12 > 10$, so not all three can run together
- **Cover inequality:** $x_a + x_b + x_c \leq 2$

Discovering better constraints by lifting

	a	b	c	d	e	Capacity	Polyhedral form
Duration	2	1	1	1	1		
Resource 1 usage	4	4	4	2	2	10	$4x_a + 4x_b + 4x_c + 2x_d + 2x_e \leq 10$
Resource 2 usage	3	3	3	4	2	9	$3x_a + 3x_b + 3x_c + 4x_d + 2x_e \leq 9$

- **Seed cover:** $\{a, b, c\}$ – on R1, $4 + 4 + 4 = 12 > 10$, so not all three can run together
- **Cover inequality:** $x_a + x_b + x_c \leq 2$
- **Assume that d is running:** if $x_d = 1$, then by R2 we have $3(x_a + x_b + x_c) \leq 5 \implies x_a + x_b + x_c \leq 1$

Discovering better constraints by lifting

	a	b	c	d	e	Capacity	Polyhedral form
Duration	2	1	1	1	1		
Resource 1 usage	4	4	4	2	2	10	$4x_a + 4x_b + 4x_c + 2x_d + 2x_e \leq 10$
Resource 2 usage	3	3	3	4	2	9	$3x_a + 3x_b + 3x_c + 4x_d + 2x_e \leq 9$

- **Seed cover:** $\{a, b, c\}$ – on R1, $4 + 4 + 4 = 12 > 10$, so not all three can run together
- **Cover inequality:** $x_a + x_b + x_c \leq 2$
- **Assume that d is running:** if $x_d = 1$, then by R2 we have $3(x_a + x_b + x_c) \leq 5 \implies x_a + x_b + x_c \leq 1$
- **Lift d into the constraint:** $(x_a + x_b + x_c) + x_d \leq 2$ is also valid

Discovering better constraints by lifting

	a	b	c	d	e	Capacity	Polyhedral form
Duration	2	1	1	1	1		
Resource 1 usage	4	4	4	2	2	10	$4x_a + 4x_b + 4x_c + 2x_d + 2x_e \leq 10$
Resource 2 usage	3	3	3	4	2	9	$3x_a + 3x_b + 3x_c + 4x_d + 2x_e \leq 9$

- **Seed cover:** $\{a, b, c\}$ – on R1, $4 + 4 + 4 = 12 > 10$, so not all three can run together
- **Cover inequality:** $x_a + x_b + x_c \leq 2$
- **Assume that d is running:** if $x_d = 1$, then by R2 we have $3(x_a + x_b + x_c) \leq 5 \implies x_a + x_b + x_c \leq 1$
- **Lift d into the constraint:** $(x_a + x_b + x_c) + x_d \leq 2$ is also valid



Model constraints alone: **LB = 2**. Lifted constraint: **LB = 2½**.

The algorithm, zoomed out

The algorithm, zoomed out

- 1 **Enumerate covers** — short covers (pairs/triples) and long covers, scored by capacity bound L

The algorithm, zoomed out

- 1 **Enumerate covers** — short covers (pairs/triples) and long covers, scored by capacity bound L
- 2 **Lift** each cover inequality against the occupancy polyhedron, one variable at a time

The algorithm, zoomed out

- 1 **Enumerate covers** — short covers (pairs/triples) and long covers, scored by capacity bound L
- 2 **Lift** each cover inequality against the occupancy polyhedron, one variable at a time
- 3 **Select** the N_{out} most restrictive lifted constraints (highest L), inject them back as new

Cumulative s

SOTA bounds

Preprocessing 736 RCPSP + 349 RCPSP/max benchmark instances:

SOTA bounds

Preprocessing 736 RCPSP + 349 RCPSP/max benchmark instances:

5

new best-known solutions

SOTA bounds

Preprocessing 736 RCPSP + 349 RCPSP/max benchmark instances:

5

new best-known solutions

25

new best-known lower bounds

SOTA bounds

Preprocessing 736 RCPSP + 349 RCPSP/max benchmark instances:

5

new best-known solutions

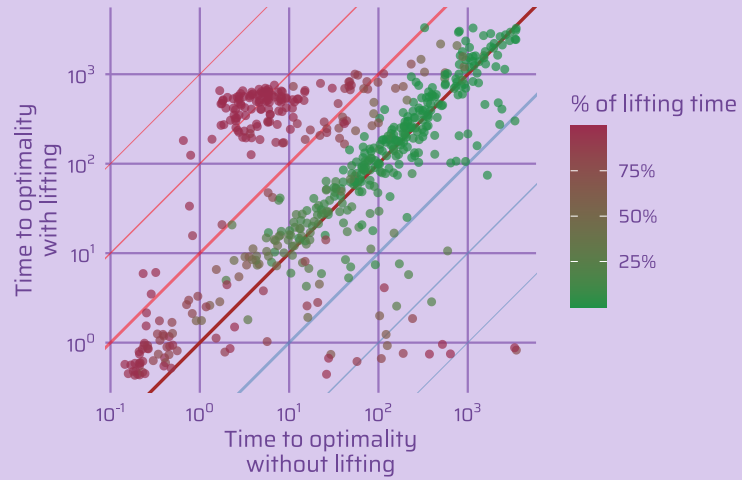
25

new best-known lower bounds

7

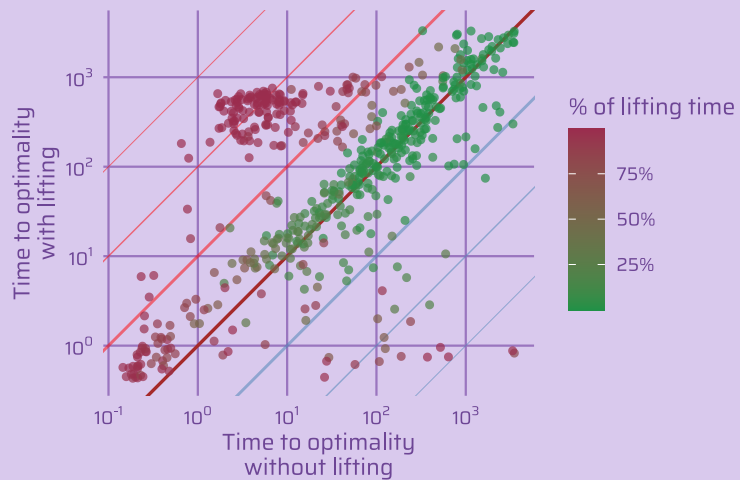
of those provable without any
search

Lifting vs. no-op



Diagonals: $y = x$, then 10× steps either way

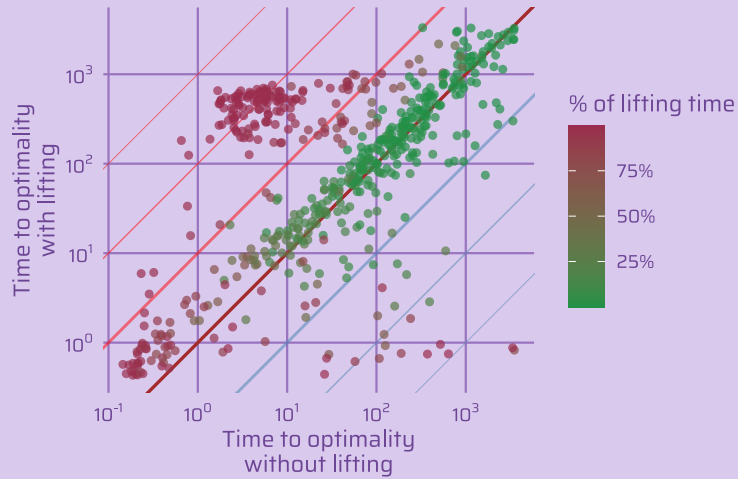
Lifting vs. no-op



Diagonals: $y = x$, then 10× steps either way

- Helps a lot when it helps — sometimes minutes down to sub-second

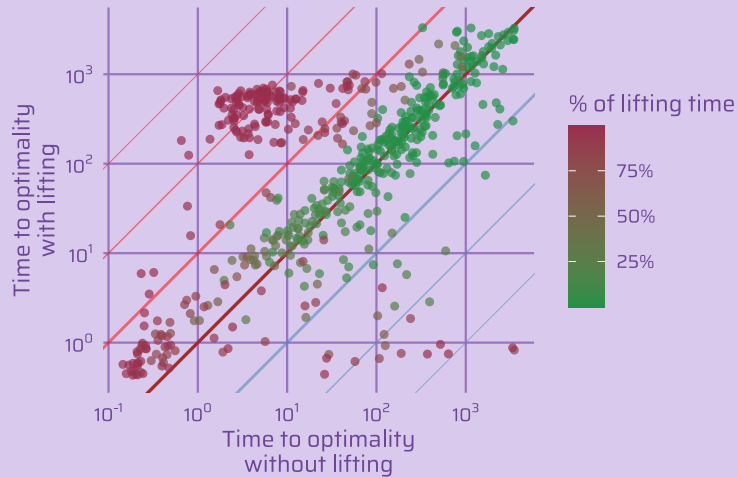
Lifting vs. no-op



Diagonals: $y = x$, then $10\times$ steps either way

- Helps a lot when it helps — sometimes minutes down to sub-second
- Rarely backfires badly; when it does, lifting time itself dominates (large instances)

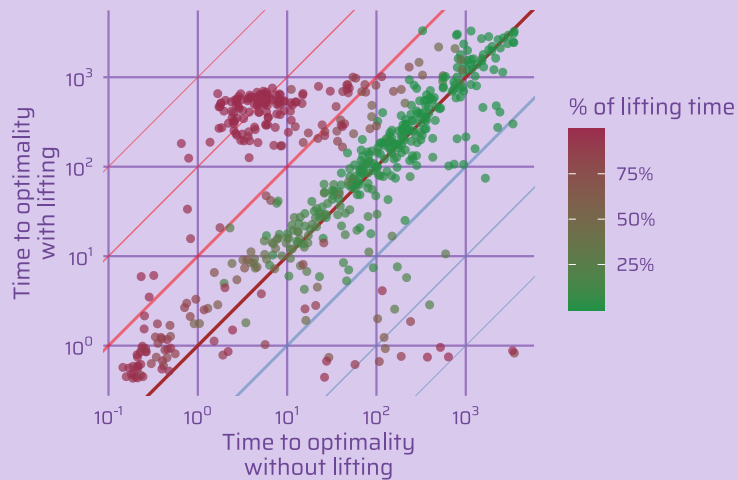
Lifting vs. no-op



Diagonals: $y = x$, then $10\times$ steps either way

- Helps a lot when it helps — sometimes minutes down to sub-second
- Rarely backfires badly; when it does, lifting time itself dominates (large instances)
- As its predecessor, **only shows clear edge in UNSAT $\rightarrow$ SAT direction**

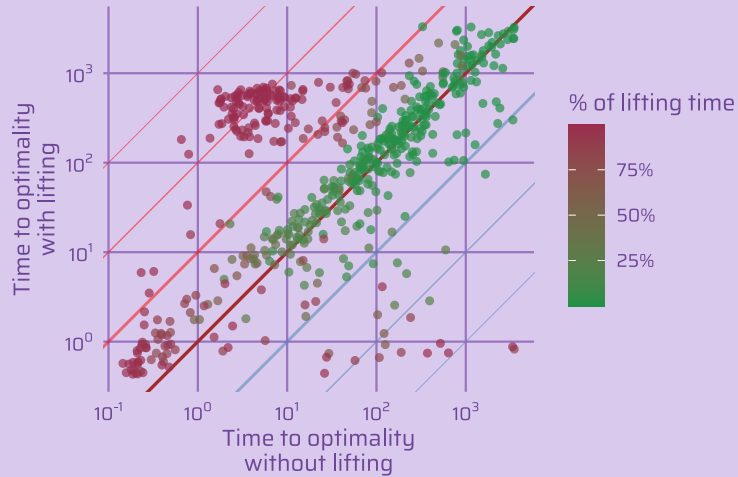
Lifting vs. no-op



Diagonals: $y = x$, then $10\times$ steps either way

- Helps a lot when it helps — sometimes minutes down to sub-second
- Rarely backfires badly; when it does, lifting time itself dominates (large instances)
- As its predecessor, **only shows clear edge in UNSAT $\rightarrow$ SAT direction**
- Restricting to `Disjunctive` -only covers often **leaves gains on the table**

Lifting vs. no-op



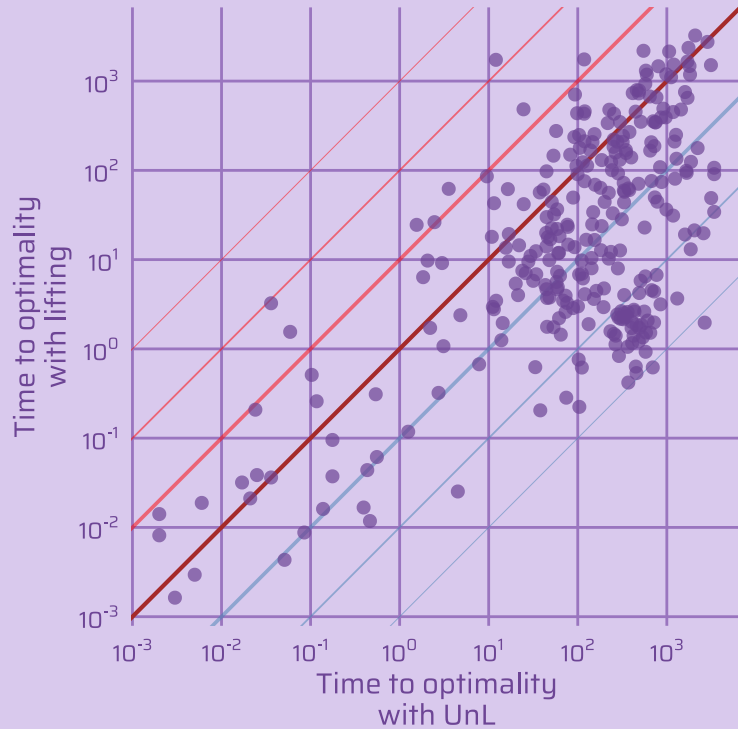
Diagonals: $y = x$, then $10 \times$ steps either way

- Helps a lot when it helps — sometimes minutes down to sub-second
- Rarely backfires badly; when it does, lifting time itself dominates (large instances)
- As its predecessor, **only shows clear edge in UNSAT $\rightarrow$ SAT direction**
- Restricting to **Disjunctive** -only covers often **leaves gains on the table**



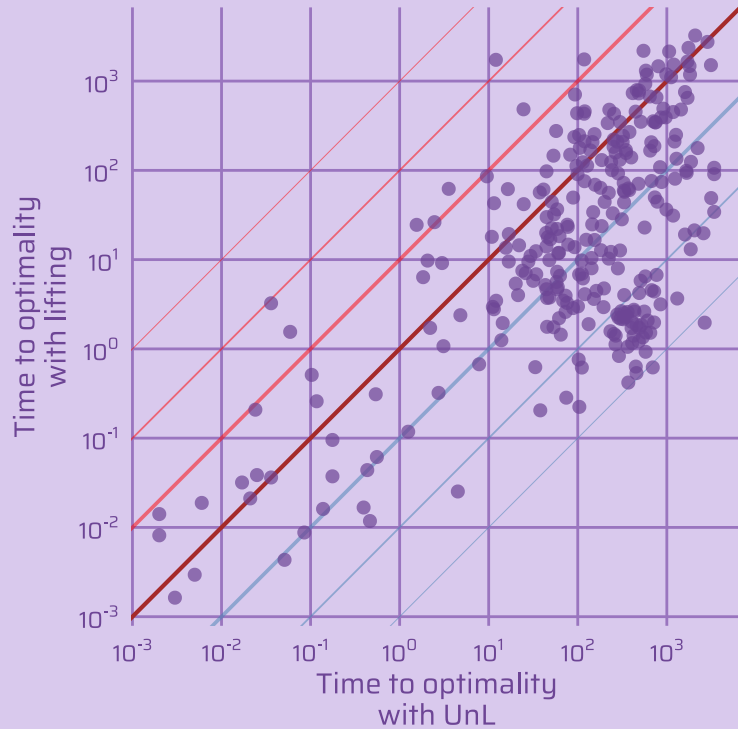
The SOTA bound on **UBO1000 #6** comes from a genuinely non-unit inferred constraint of capacity **3**

Lifting vs. UnL



Diagonals: $y = x$, then 10 $\times$ steps either way

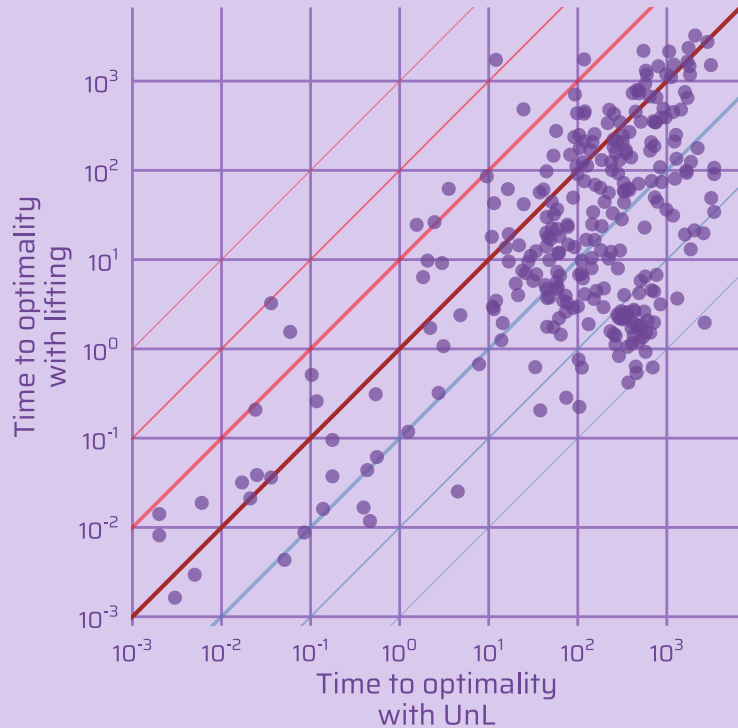
Lifting vs. UnL



Diagonals: $y = x$, then $10\times$ steps either way

- Faster to optimality on the majority of comparable instances

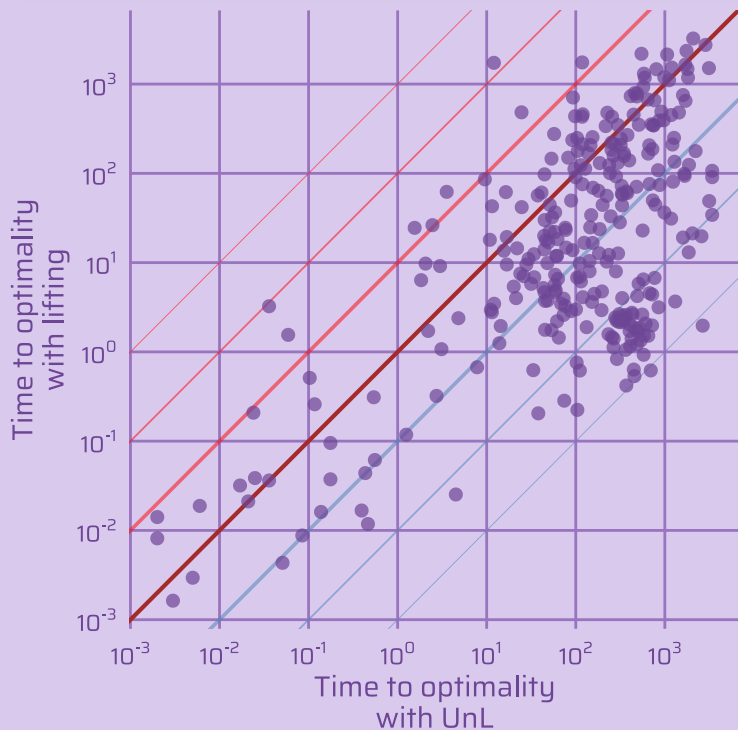
Lifting vs. UnL



Diagonals: $y = x$, then $10\times$ steps either way

- Faster to optimality on the majority of comparable instances
- Far less variance on the downside: **19 pairs of runs** degrade by $2\times$ vs. UnL's own reported order-of-magnitude degradations on many instances

Lifting vs. UnL



Diagonals: $y = x$, then $10\times$ steps either way

- Faster to optimality on the majority of comparable instances
- Far less variance on the downside: **19 pairs of runs** degrade by $2\times$ vs. UnL's own reported order-of-magnitude degradations on many instances
- Doesn't fully subsume UnL — a few instances still favor UnL's during-search discovery

Takeaways

Takeaways

Cumulative constraints can be seen as **linear inequalities over occupancy vectors** and handled with polyhedral methods

Takeaways

Cumulative constraints can be seen as **linear inequalities over occupancy vectors** and handled with polyhedral methods

Lifting operationalizes this insight into a CP-native, root-node inference

Takeaways

Cumulative constraints can be seen as **linear inequalities over occupancy vectors** and handled with polyhedral methods

Lifting operationalizes this insight into a CP-native, root-node inference

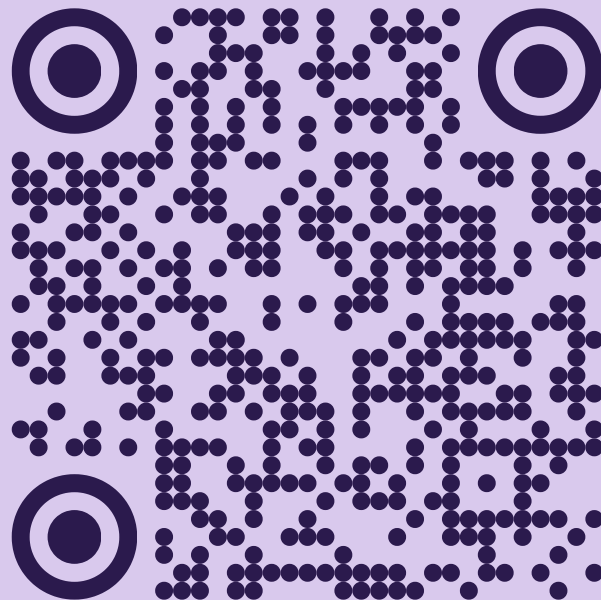
A helpful preprocessing technique with improved robustness over UnL

Takeaways

Cumulative constraints can be seen as **linear inequalities over occupancy vectors** and handled with polyhedral methods

Lifting operationalizes this insight into a CP-native, root-node inference

A helpful preprocessing technique with improved robustness over UnL



[Read the blog post](#)