

Smart Cubing for Graph Search

A Comparative Study

Markus Kirchweger Tomáš Peitl Stefan Szeider Hai Xia

Algorithms and Complexity Group, TU Wien

CP 2026 · Lisbon



Outline

- 1 Cube-and-Conquer
- 2 Symmetry and Propagators
- 3 The Prerun Pipeline
- 4 Benchmarks and Tuning
- 5 Results
- 6 Conclusion

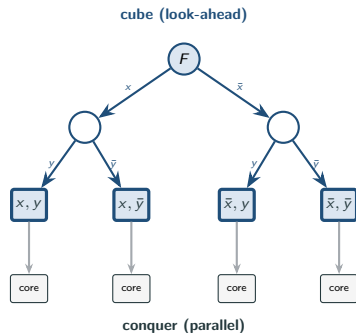
Cube-and-conquer: split, then conquer in parallel

The idea

Split a hard instance into thousands of independent **cubes** (partial assignments); solve them **in parallel** across a cluster.

Look-ahead chooses the split

- ▶ probe each candidate variable both ways;
- ▶ count the propagations a, b in the two branches;
- ▶ branch on the highest score $\sigma(a, b) = a + b + ab$.



four cubes cover every assignment of $\{x, y\}$:
 $\text{mod}(F) = \bigcup_i \text{mod}(F[C_i])$

[first settled the Boolean Pythagorean triples problem, Heule et al. 2016]

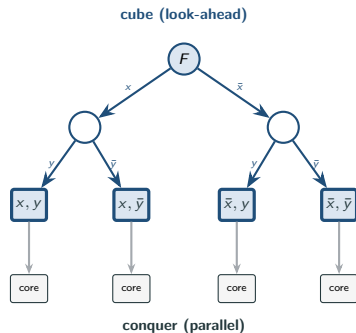
Cube-and-conquer: split, then conquer in parallel

The idea

Split a hard instance into thousands of independent **cubes** (partial assignments); solve them **in parallel** across a cluster.

Look-ahead chooses the split

- ▶ probe each candidate variable both ways;
- ▶ count the propagations a, b in the two branches;
- ▶ branch on the highest score $\sigma(a, b) = a + b + ab$.



four cubes cover every assignment of $\{x, y\}$:
 $\text{mod}(F) = \bigcup_i \text{mod}(F[C_i])$

[first settled the Boolean Pythagorean triples problem, Heule et al. 2016]

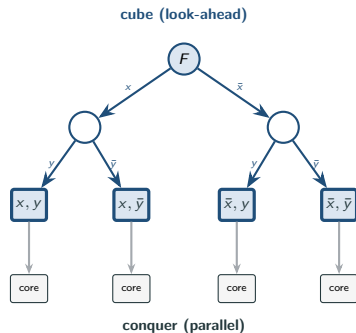
Cube-and-conquer: split, then conquer in parallel

The idea

Split a hard instance into thousands of independent **cubes** (partial assignments); solve them **in parallel** across a cluster.

Look-ahead chooses the split

- ▶ probe each candidate variable both ways;
- ▶ count the propagations a, b in the two branches;
- ▶ branch on the highest score $\sigma(a, b) = a + b + ab$.



four cubes cover every assignment of $\{x, y\}$:
 $\text{mod}(F) = \bigcup_i \text{mod}(F[C_i])$

[first settled the Boolean Pythagorean triples problem, Heule et al. 2016]

Cube-and-conquer: split, then conquer in parallel

The idea

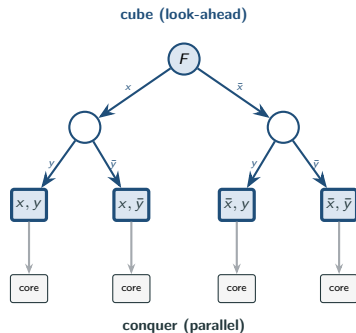
Split a hard instance into thousands of independent **cubes** (partial assignments); solve them **in parallel** across a cluster.

Look-ahead chooses the split

- ▶ probe each candidate variable both ways;
- ▶ count the propagations a, b in the two branches;
- ▶ branch on the highest score $\sigma(a, b) = a + b + ab$.

$$\text{e.g. } \sigma(4, 4) = \mathbf{24} > \sigma(2, 6) = \mathbf{20}$$

[first settled the Boolean Pythagorean triples problem, Heule et al. 2016]



four cubes cover every assignment of $\{x, y\}$:
 $\text{mod}(F) = \bigcup_i \text{mod}(F[C_i])$

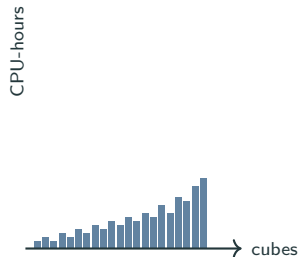
A cube-and-conquer failure

Enumerating extremal diameter-2-critical graphs, $n = 17$

Standard cube-and-conquer produced **3 861 cubes**.

But the work was wildly uneven

- ▶ most cubes finished quickly;
- ▶ **one cube ran 643 CPU-hours** ≈ 27 days;
- ▶ that single cube was $\approx 11\%$ of the entire bill.



3 861 cubes, sorted by cost

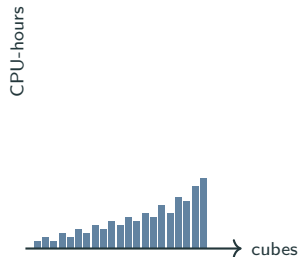
A cube-and-conquer failure

Enumerating extremal diameter-2-critical graphs, $n = 17$

Standard cube-and-conquer produced **3 861 cubes**.

But the work was wildly uneven

- ▶ most cubes finished quickly;
- ▶ one cube ran **643 CPU-hours** $\approx$ 27 days;
- ▶ that single cube was $\approx$ 11% of the entire bill.



3 861 cubes, sorted by cost

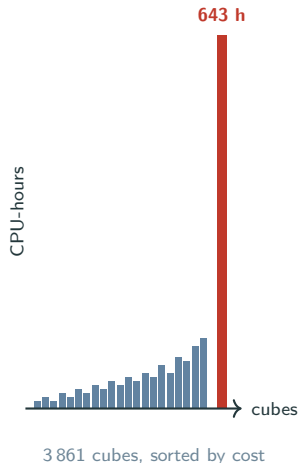
A cube-and-conquer failure

Enumerating extremal diameter-2-critical graphs, $n = 17$

Standard cube-and-conquer produced **3 861 cubes**.

But the work was wildly uneven

- ▶ most cubes finished quickly;
- ▶ **one cube ran 643 CPU-hours** ≈ 27 days;
- ▶ that single cube was $\approx 11\%$ of the entire bill.



A cube-and-conquer failure

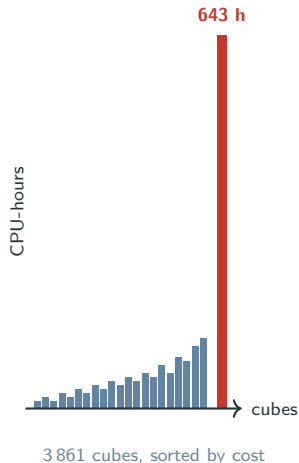
Enumerating extremal diameter-2-critical graphs, $n = 17$

Standard cube-and-conquer produced **3 861 cubes**.

But the work was wildly uneven

- ▶ most cubes finished quickly;
- ▶ **one cube ran 643 CPU-hours** ≈ 27 days;
- ▶ that single cube was $\approx 11\%$ of the entire bill.

Cubes run in parallel, so this one cube **lengthened the whole process significantly**.



Outline

- 1 Cube-and-Conquer
- 2 Symmetry and Propagators**
- 3 The Prerun Pipeline
- 4 Benchmarks and Tuning
- 5 Results
- 6 Conclusion

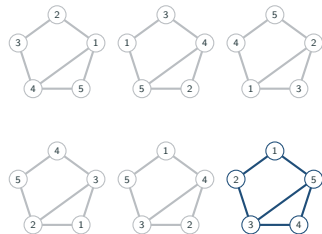
Symmetry drowns the search space

Isomorphic copies

A graph on $n = 22$ vertices has up to $22! \approx 10^{21}$ relabelled twins, and all satisfy *exactly* the same constraints.

Break the symmetry

- ▶ search only the **canonical** representative per class;
- ▶ no compact static encoding is known to remove every copy;
- ▶ testing lex-minimality is coNP-complete. [Crawford et al. 1996]



... same graph, relabelled ... **canonical**

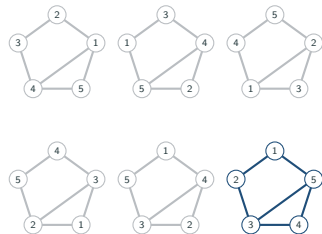
Symmetry drowns the search space

Isomorphic copies

A graph on $n = 22$ vertices has up to $22! \approx 10^{21}$ relabelled twins, and all satisfy *exactly* the same constraints.

Break the symmetry

- ▶ search only the **canonical** representative per class;
- ▶ no compact static encoding is known to remove every copy;
- ▶ testing lex-minimality is coNP-complete. [Crawford et al. 1996]



... same graph, relabelled ... **canonical**

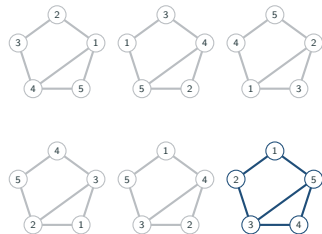
Symmetry drowns the search space

Isomorphic copies

A graph on $n = 22$ vertices has up to $22! \approx 10^{21}$ relabelled twins, and all satisfy *exactly* the same constraints.

Break the symmetry

- ▶ search only the **canonical** representative per class;
- ▶ no compact static encoding is known to remove every copy;
- ▶ testing lex-minimality is coNP-complete. [Crawford et al. 1996]



... same graph, relabelled ... **canonical**

SAT Modulo Symmetries: a solver that learns

SMS = CDCL + propagators

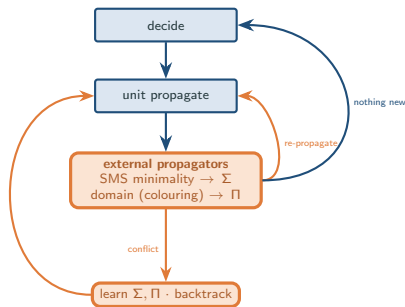
After unit propagation, an **external propagator** is consulted.

It does one of three things

- ▶ forces a **conflict**; or
- ▶ triggers more propagation; or
- ▶ passes to the next decision.

Its clauses are *not* in F

- ▶ Σ — symmetry (minimality check);
- ▶ Π — domain, e.g. colourability.



SAT Modulo Symmetries: a solver that learns

SMS = CDCL + propagators

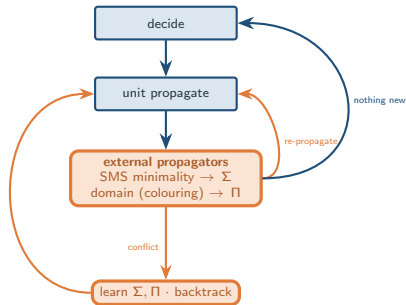
After unit propagation, an **external propagator** is consulted.

It does one of three things

- ▶ forces a **conflict**; or
- ▶ triggers more propagation; or
- ▶ passes to the next decision.

Its clauses are *not* in F

- ▶ Σ — symmetry (minimality check);
- ▶ Π — domain, e.g. colourability.



SAT Modulo Symmetries: a solver that learns

SMS = CDCL + propagators

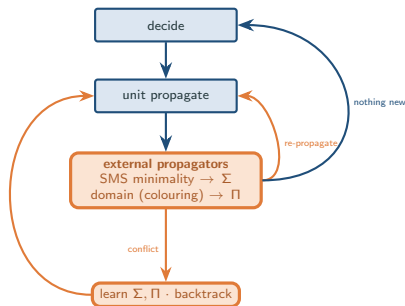
After unit propagation, an **external propagator** is consulted.

It does one of three things

- ▶ forces a **conflict**; or
- ▶ triggers more propagation; or
- ▶ passes to the next decision.

Its clauses are *not* in F

- ▶ Σ — symmetry (minimality check);
- ▶ Π — domain, e.g. colourability.



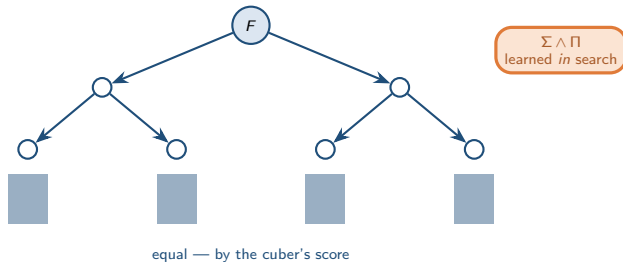
Why standard cubing breaks

The cuber is blind

- ▶ Its look-ahead runs on F alone.
- ▶ It is blind to the clauses Σ, Π that the propagators add *during* search.

Consequence

- ▶ the split looks balanced to the cuber.
- ▶ the *true* costs come out uneven.
- ▶ one cube becomes the **monster**.



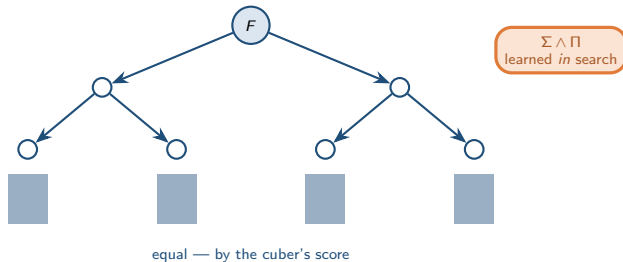
Why standard cubing breaks

The cuber is blind

- ▶ Its look-ahead runs on F alone.
- ▶ It is blind to the clauses Σ, Π that the propagators add *during* search.

Consequence

- ▶ the split looks balanced to the cuber.
- ▶ the *true* costs come out uneven.
- ▶ one cube becomes the **monster**.



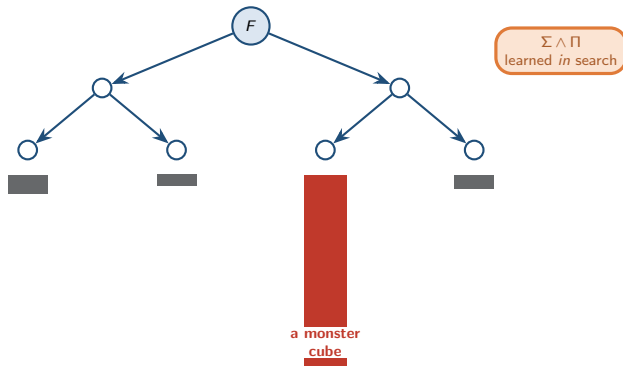
Why standard cubing breaks

The cuber is blind

- ▶ Its look-ahead runs on F alone.
- ▶ It is blind to the clauses Σ, Π that the propagators add *during* search.

Consequence

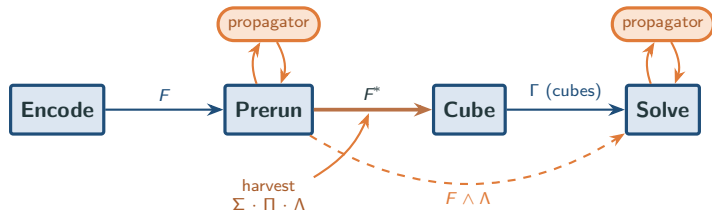
- ▶ the split looks balanced to the cuber.
- ▶ the *true* costs come out uneven.
- ▶ one cube becomes the **monster**.



Outline

- 1 Cube-and-Conquer
- 2 Symmetry and Propagators
- 3 The Prerun Pipeline**
- 4 Benchmarks and Tuning
- 5 Results
- 6 Conclusion

Prerun: cube the formula your solver *discovered*



Make the hidden clauses visible

Run the propagator-augmented solver briefly and harvest

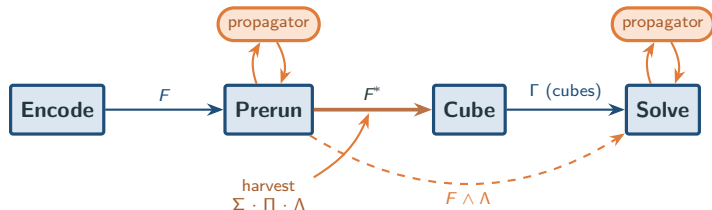
$$F^* = F \wedge \Sigma \wedge \Pi \wedge \Lambda.$$

- ▶ Σ symmetry, Π domain, Λ learned clauses;
- ▶ then cube F^* instead of F .

Correctness

- ▶ verify cubes against the *original* F ($F \wedge \neg \Gamma$ UNSAT);
- ▶ cheap, and recovers any lost model.

Prerun: cube the formula your solver *discovered*



Make the hidden clauses visible

Run the propagator-augmented solver briefly and harvest

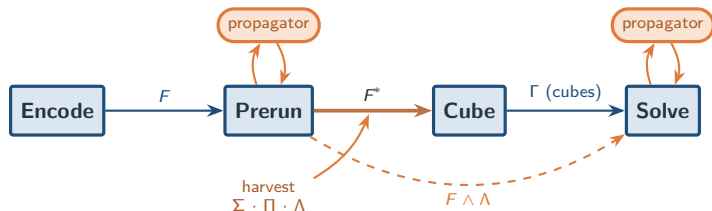
$$F^* = F \wedge \Sigma \wedge \Pi \wedge \Lambda.$$

- ▶ Σ symmetry, Π domain, Λ learned clauses;
- ▶ then cube F^* instead of F .

Correctness

- ▶ verify cubes against the *original* F ($F \wedge \neg \Gamma$ UNSAT);
- ▶ cheap, and recovers any lost model.

Prerun: cube the formula your solver *discovered*



Make the hidden clauses visible

Run the propagator-augmented solver briefly and harvest

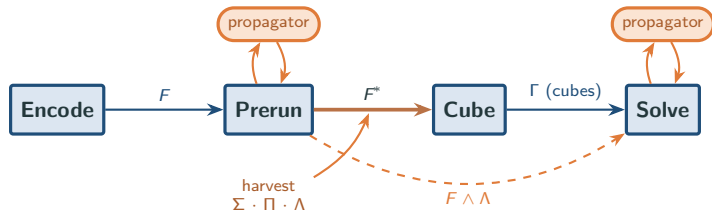
$$F^* = F \wedge \Sigma \wedge \Pi \wedge \Lambda.$$

- ▶ Σ symmetry, Π domain, Λ learned clauses;
- ▶ then cube F^* instead of F .

Correctness

- ▶ verify cubes against the *original* F ($F \wedge \neg \Gamma$ UNSAT);
- ▶ cheap, and recovers any lost model.

Prerun: cube the formula your solver *discovered*



Make the hidden clauses visible

Run the propagator-augmented solver briefly and harvest

$$F^* = F \wedge \Sigma \wedge \Pi \wedge \Lambda.$$

- ▶ Σ symmetry, Π domain, Λ learned clauses;
- ▶ then cube F^* instead of F .

Correctness

- ▶ verify cubes against the *original* F ($F \wedge \neg \Gamma$ UNSAT);
- ▶ cheap, and recovers any lost model.

Four ways to cube F^*

CDCL

prior SMS practice

cutoff on assigned edges; *blindly follows* the CDCL path.

Four ways to cube F^*

CDCL

prior SMS practice

cutoff on assigned edges; *blindly follows* the CDCL path.

CDCL-LA- $\forall$

in-solver look-ahead, *all* variables.

$$\sigma = \min(a, b) + \varepsilon(a+b)$$

Four ways to cube F^*

CDCL

prior SMS practice

cutoff on assigned edges; *blindly follows* the CDCL path.

CDCL-LA- $\forall$

in-solver look-ahead, *all* variables.

$$\sigma = \min(a, b) + \varepsilon(a+b)$$

CDCL-LA-E

look-ahead, *edge* variables only, cheaper.

$$\sigma = \min(a, b) + \varepsilon(a+b)$$

Four ways to cube F^*

CDCL

prior SMS practice

cutoff on assigned edges; *blindly follows* the CDCL path.

CDCL-LA- $\forall$

in-solver look-ahead, *all* variables.

$$\sigma = \min(a, b) + \varepsilon(a+b)$$

CDCL-LA-E

look-ahead, *edge* variables only, cheaper.

$$\sigma = \min(a, b) + \varepsilon(a+b)$$

march_cu

selected

dedicated look-ahead solver; richer scoring.

$$\sigma = a + b + ab$$

All cubers calibrated to ≈ 5000 cubes: march_cu wins overall and is selected.

Outline

- 1 Cube-and-Conquer
- 2 Symmetry and Propagators
- 3 The Prerun Pipeline
- 4 Benchmarks and Tuning**
- 5 Results
- 6 Conclusion

Three benchmark problems

Kochen–Specker



quantum contextuality;

lower bound **24**

Three benchmark problems

Kochen–Specker



quantum contextuality;
lower bound **24**

Murty–Simon



diameter-2-critical;
 $\leq \lfloor n^2/4 \rfloor$ edges

Three benchmark problems

Kochen–Specker



quantum contextuality;
lower bound **24**

Murty–Simon



diameter-2-critical;
 $\leq \lfloor n^2/4 \rfloor$ edges

Triangle-free



high chromatic number;
 $k \geq 6$ **open**

Three benchmark problems

Kochen–Specker



quantum contextuality;
lower bound **24**

Murty–Simon



diameter-2-critical;
 $\leq \lfloor n^2/4 \rfloor$ edges

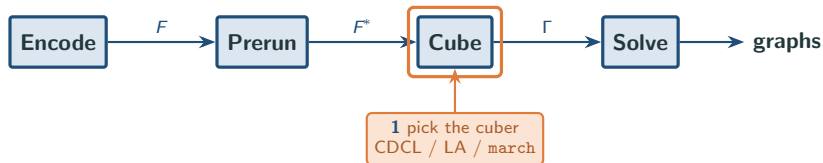
Triangle-free



high chromatic number;
 $k \geq 6$ **open**

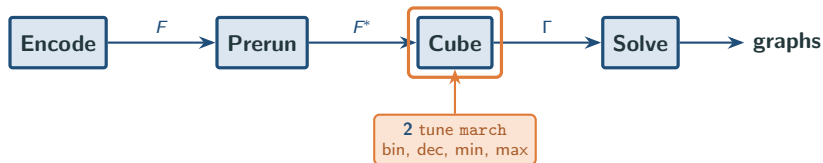
Train on n , test on $n+1$: the space of graphs on n vertices is $2^{\binom{n}{2}}$ (one more vertex multiplies it by 2^n).

Our tuning roadmap: four phases on one pipeline



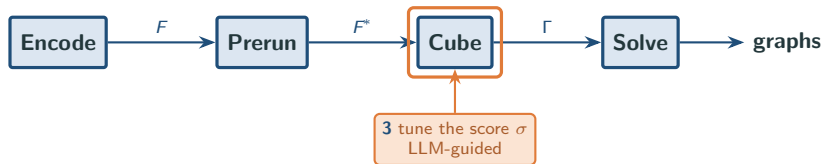
Phase 1: compare four cubers, keep the best (march).

Our tuning roadmap: four phases on one pipeline



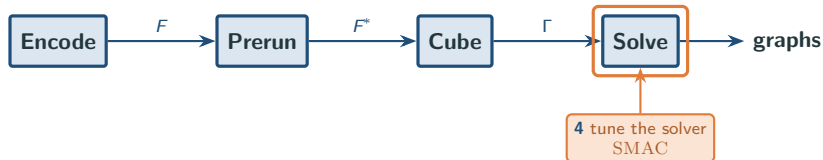
Phase 2: grid-search march's four parameters. modest gain

Our tuning roadmap: four phases on one pipeline



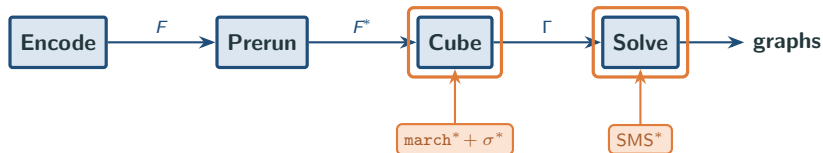
Phase 3: an LLM proposes scoring functions σ . modest gain

Our tuning roadmap: four phases on one pipeline



Phase 4: configure the conquer solver with SMAC.

Our tuning roadmap: four phases on one pipeline



The tuned pipeline — everything tuned *only* on training (n), evaluated *once* on test ($n+1$).

Outline

- 1 Cube-and-Conquer
- 2 Symmetry and Propagators
- 3 The Prerun Pipeline
- 4 Benchmarks and Tuning
- 5 Results**
- 6 Conclusion

Does the prerun help? With vs. without

Total time and hardest cube (CPU-hours), default solver

cuber	prerun	KS		MS		TF	
		sum	max	sum	max	sum	max
CDCL	no	42.8	5.9	124.6	3.4	42.5	2.1
CDCL	10 min	15.0	0.2	213.8	2.6	45.6	0.5
march	no	29.0	0.7	107.5	4.6	30.3	1.0
march	10 min	15.4	0.0	33.9	0.7	31.6	0.2

sum = total solving time max = hardest cube

Does the prerun help? With vs. without

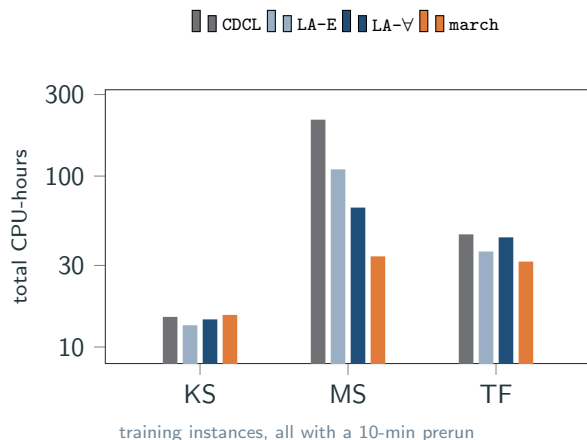
Total time and hardest cube (CPU-hours), default solver

cuber	prerun	KS		MS		TF	
		sum	max	sum	max	sum	max
CDCL	no	42.8	5.9	124.6	3.4	42.5	2.1
CDCL	10 min	15.0	0.2	213.8	2.6	45.6	0.5
march	no	29.0	0.7	107.5	4.6	30.3	1.0
march	10 min	15.4	0.0	33.9	0.7	31.6	0.2

sum = total solving time max = hardest cube

Prerun helps *look-ahead* cubing most: march's total on MS 107.5 → 33.9 h, and its **hardest cube** drops on all three (MS 4.6 → 0.7 h).

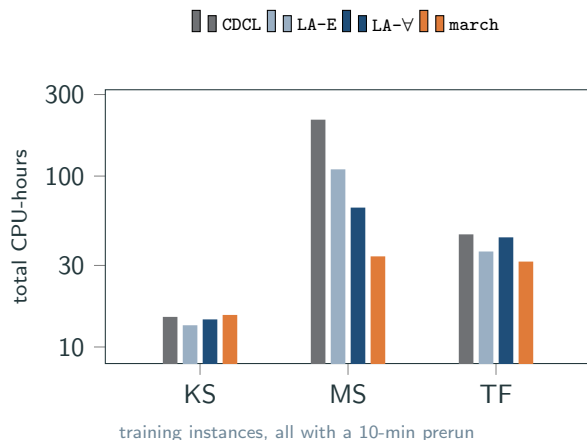
Comparing the four cubers



Verdict

- ▶ **march** best on **MS** and **TF**, competitive on **KS**;
- ▶ look-ahead turns F^* into better-balanced cubes;
- ▶ CDCL's gains are inconsistent (MS: 214 vs 34 h).

Comparing the four cubers



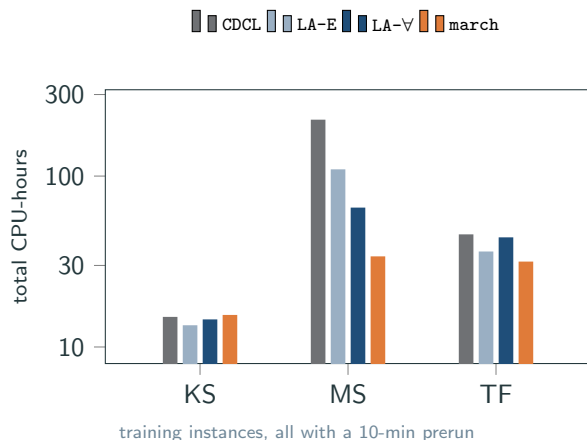
Verdict

- ▶ **march** best on **MS** and **TF**, competitive on **KS**;
- ▶ look-ahead turns F^* into better-balanced cubes;
- ▶ CDCL's gains are inconsistent (MS: 214 vs 34 h).

Hardest cube (CPU-hours)

cuber	KS	MS	TF
CDCL	0.2	2.6	0.5
LA-E	0.0	0.9	0.2
LA-∀	0.0	1.1	0.3
march	0.0	0.7	0.2

Comparing the four cubers



Verdict

- ▶ **march** best on **MS** and **TF**, competitive on **KS**;
- ▶ look-ahead turns F^* into better-balanced cubes;
- ▶ CDCL's gains are inconsistent (MS: 214 vs 34 h).

Hardest cube (CPU-hours)

cuber	KS	MS	TF
CDCL	0.2	2.6	0.5
LA-E	0.0	0.9	0.2
LA- $\forall$	0.0	1.1	0.3
march	0.0	0.7	0.2

Tuning march's parameters

Grid search over $3^4 = 81$ configurations

four parameters: bin, dec, min, max.

Tuning march's parameters

Grid search over $3^4 = 81$ configurations

four parameters: bin, dec, min, max.

Total solving time (CPU-hours), training instances

config	KS	MS	TF
default	15.4	35.2	29.5
best	13.5	34.3	27.2

Tuning march's parameters

Grid search over $3^4 = 81$ configurations

four parameters: bin, dec, min, max.

Total solving time (CPU-hours), training instances

config	KS	MS	TF
default	15.4	35.2	29.5
best	13.5	34.3	27.2

A **modest** gain; the best configuration is problem-dependent.

Tuning the scoring function σ

LLM-guided search

An LLM proposes scoring functions $\sigma(a, b)$; we evaluate them and feed the timings back.

Tuning the scoring function σ

LLM-guided search

An LLM proposes scoring functions $\sigma(a, b)$; we evaluate them and feed the timings back.

Best σ and speedup over the default

problem	best $\sigma(a, b)$	speedup
KS	$\min(a, b) + \varepsilon(a+b)$	3.0%
MS	$a + b + ab + 0.5 a - b $	2.9%
TF	$a + b + ab + 0.5 a - b $	7.9%

Tuning the scoring function σ

LLM-guided search

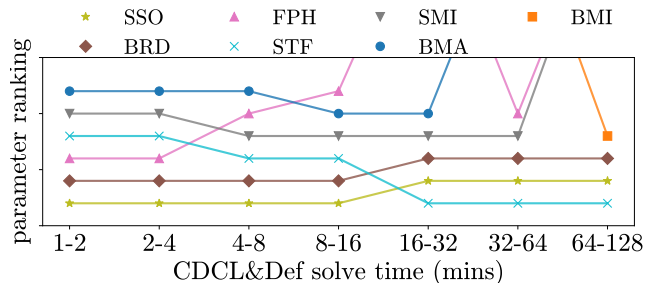
An LLM proposes scoring functions $\sigma(a, b)$; we evaluate them and feed the timings back.

Best σ and speedup over the default

problem	best $\sigma(a, b)$	speedup
KS	$\min(a, b) + \varepsilon(a+b)$	3.0%
MS	$a + b + ab + 0.5 a - b $	2.9%
TF	$a + b + ab + 0.5 a - b $	7.9%

Again **modest**; on KS it rediscovered a classic look-ahead score.

Tuning the conquer solver with SMAC

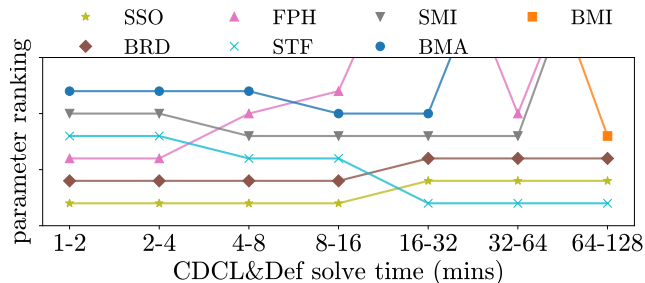


parameter ranking on **KS cubes**, grouped by default solving time

Sensitivity analysis → SMAC

- ▶ toggle each parameter to an extreme;
- ▶ keep the **12** most impactful (10 CaDiCaL + 2 SMS);
- ▶ tune per problem ⇒ **SMS***.

Tuning the conquer solver with SMAC

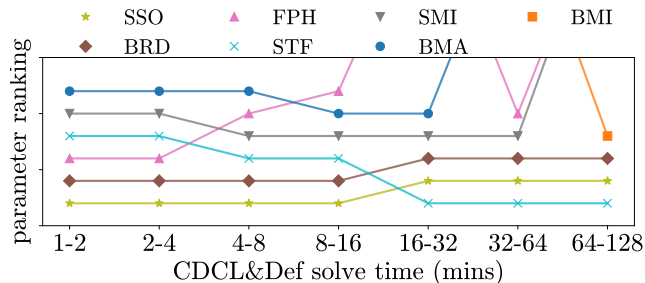


parameter ranking on **KS cubes**, grouped by default solving time

Sensitivity analysis → SMAC

- ▶ toggle each parameter to an extreme;
- ▶ keep the **12** most impactful (10 CaDiCaL + 2 SMS);
- ▶ tune per problem ⇒ **SMS***.

Tuning the conquer solver with SMAC



parameter ranking on **KS cubes**, grouped by default solving time

Sensitivity analysis → SMAC

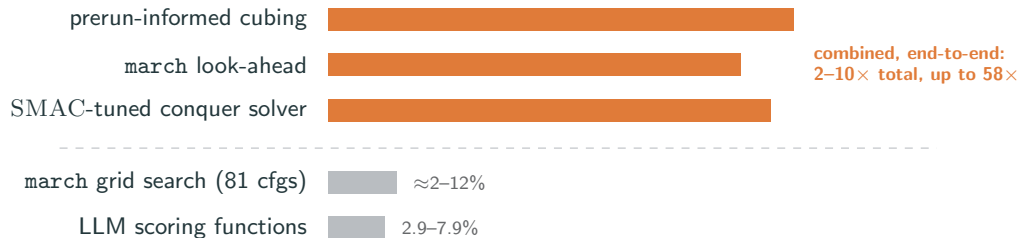
- ▶ toggle each parameter to an extreme;
- ▶ keep the **12** most impactful (10 CaDiCaL + 2 SMS);
- ▶ tune per problem ⇒ **SMS***.

Finding

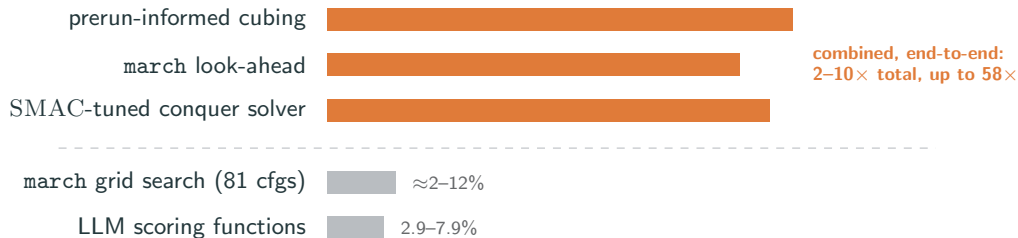
BRD dominates; restart-free “stable” phases help.

SSO stabilizeonly, BRD bumpreasondepth, FPH forcephase, ...

What did help and what did not

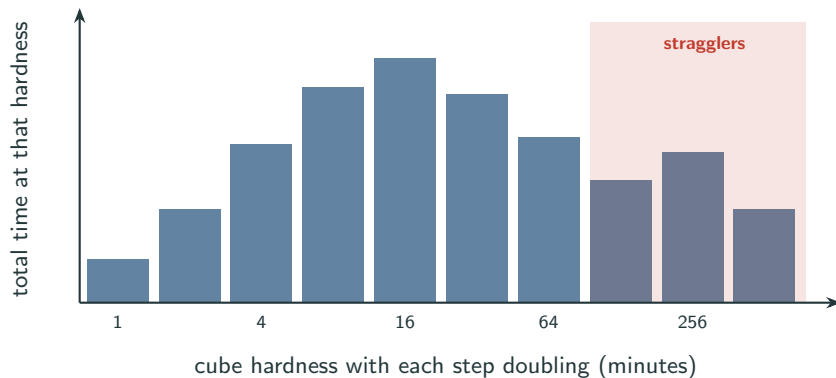


What did help and what did not

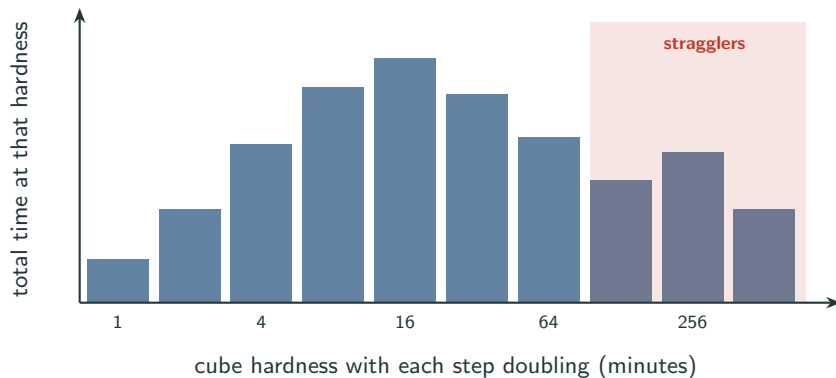


The LLM's best **Kochen–Specker** scoring function was $\min(a, b) + \varepsilon(a+b)$: *exactly* the default of our other cubers. It rediscovered a classic look-ahead score.

How to read one run

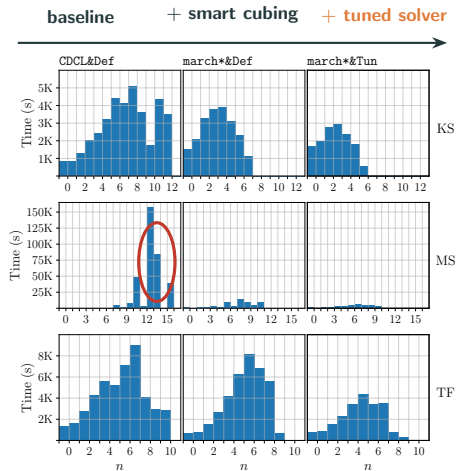


How to read one run



Each bar: total time spent on cubes of that hardness. The **area of all bars** is the whole CPU time.

Main results: three problems, one signature



Held-out test ($n+1$)

► total CPU-time:

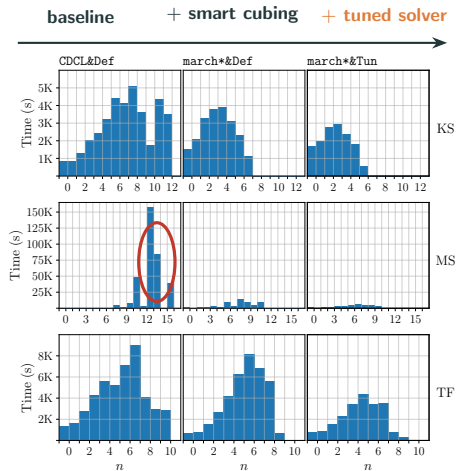
$2.7\times$ / $10.6\times$ / $2.2\times$

► hardest cube:

$58\times$ / $57\times$ / $1.9\times$

KS / MS / TF; area shrinks and moves left across the row.

Main results: three problems, one signature



Held-out test ($n+1$)

- ▶ total CPU-time:
 $2.7\times$ / $10.6\times$ / $2.2\times$
- ▶ hardest cube:
 $58\times$ / $57\times$ / $1.9\times$

KS / MS / TF; area shrinks and moves left across the row.

The straggler, tamed

old

643.6 CPU-hours

≈ 27 days

full pipeline



$\leftarrow$ **11.2 h** $\approx$ overnight

(both bars to scale)

The straggler, tamed

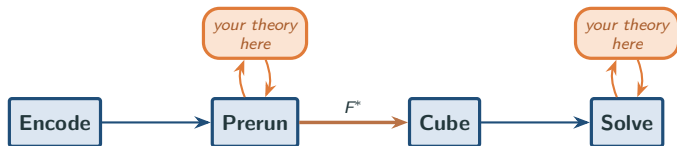


Wall-clock time is bounded below by the hardest cube: the baseline's dominant straggler (643.6 h) shrinks to 11.2 h, a **57×** smaller worst cube.

Outline

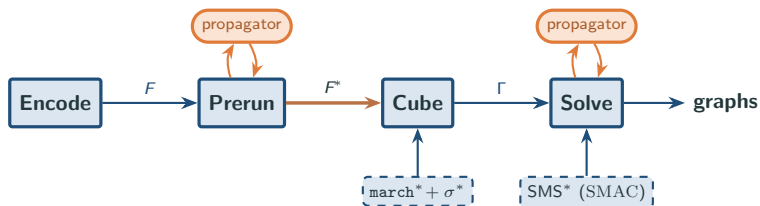
- 1 Cube-and-Conquer
- 2 Symmetry and Propagators
- 3 The Prerun Pipeline
- 4 Benchmarks and Tuning
- 5 Results
- 6 Conclusion

Beyond symmetry: a recipe for any propagator



Nothing here is symmetry-specific: *whenever* an IPASIR-UP propagator hides constraints from the cuber, **prerun** makes them visible. [cf. Wilson et al. 2023, Wu et al. 2025]

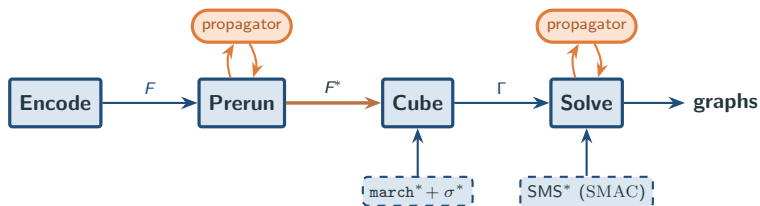
Take-home



Message 1

Cube the formula your solver *discovered*, instead of the one you wrote.

Take-home



Message 1

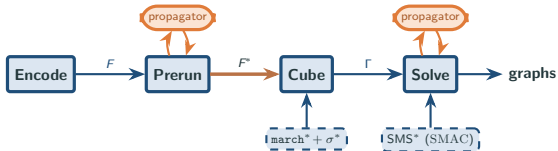
Cube the formula your solver *discovered*, instead of the one you wrote.

Message 2

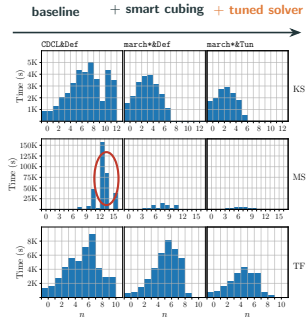
Then *tune* the conquer solver for the cubes it will see.

Thank you!

Questions?



cube the formula the solver *discovered* · tune the conquer solver for its cubes



2-10x total CPU · up to 58x on the hardest cube



code & data — Zenodo

These slides were created with the help of LLMs.