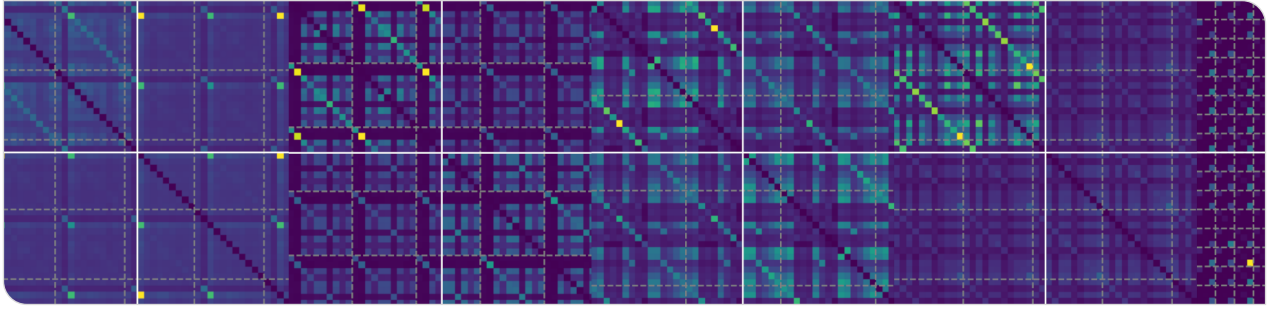
An Empirical Study on Learned Clause Overlaps In Distributed SAT Solving

Pragmatics of SAT 2024, Pune, India

Jannick Borowitz, Dominik Schreiber, Peter Sanders | August 20, 2024

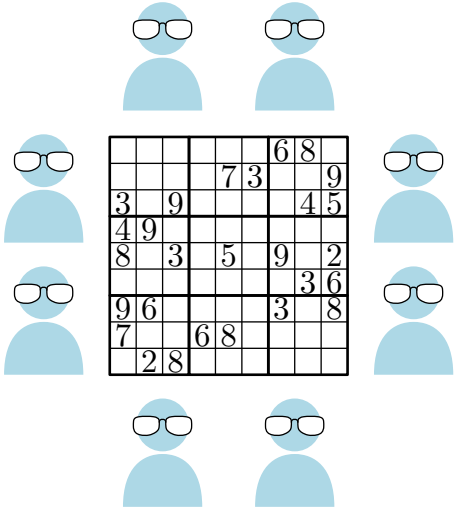


Parallel Logical Reasoning

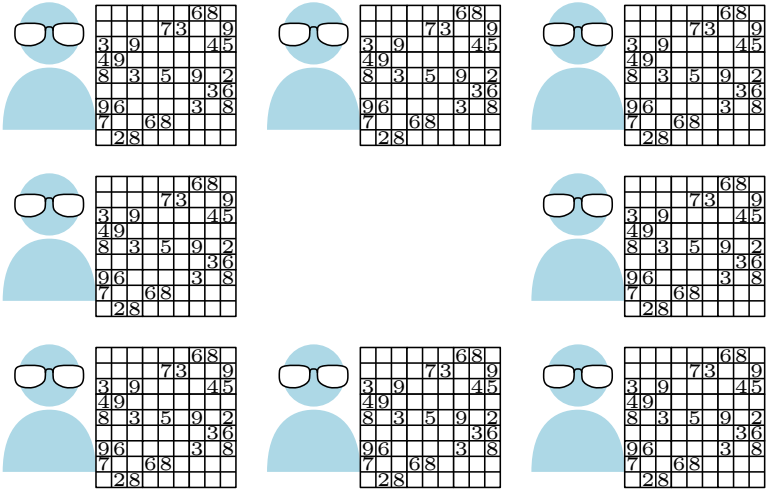
The assembly of logicians

- Complex logic puzzle
- *n* logic experts want to solve the puzzle
- Experts tend to work the best undisturbed

How to coordinate our experts?

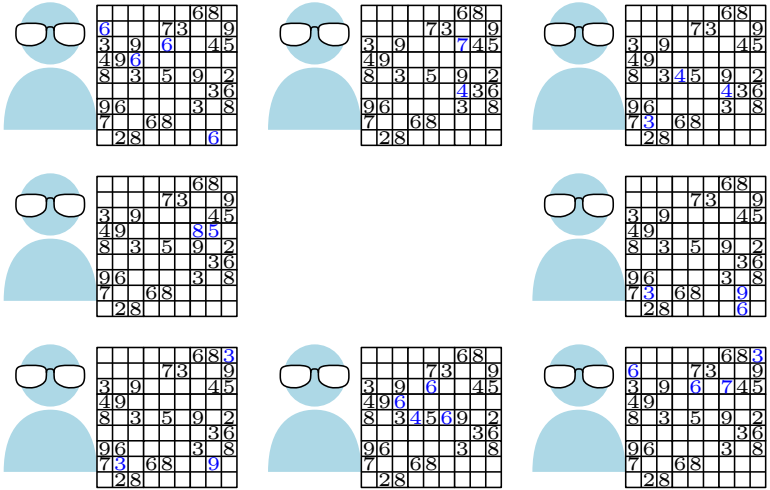


Cooperative Portfolio



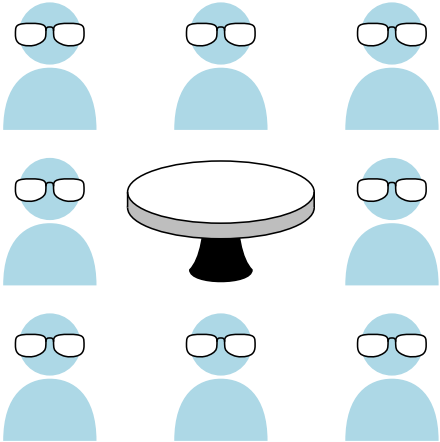
- All experts work on original problem independently

Cooperative Portfolio



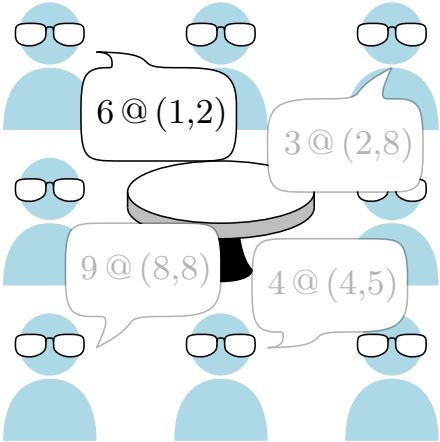
- All experts work on original problem independently

Cooperative Portfolio



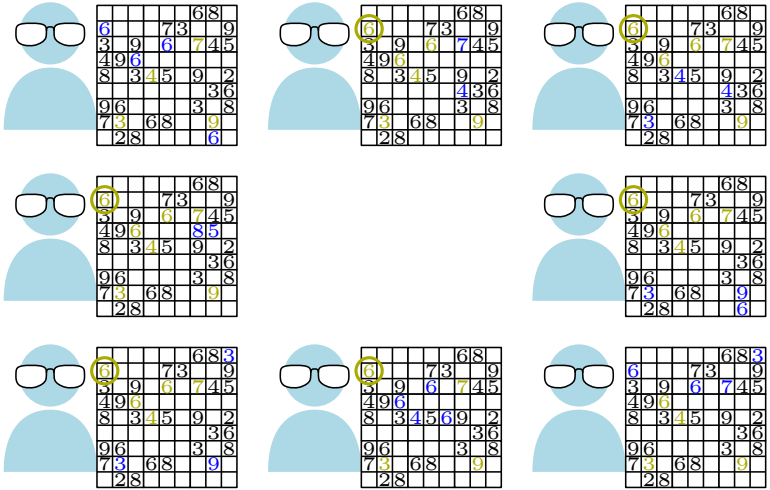
- All experts work on original problem independently
- Brief meetings to exchange crucial insights

Cooperative Portfolio



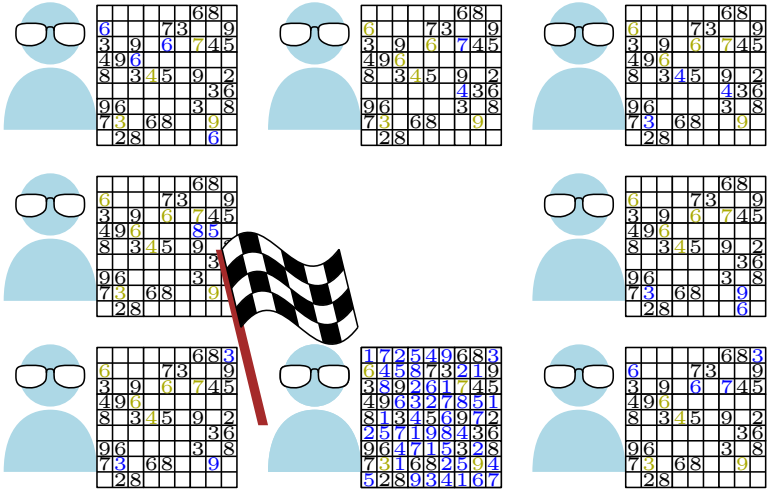
- All experts work on original problem independently
- Brief meetings to exchange crucial insights

Cooperative Portfolio



- All experts work on original problem independently
- Brief meetings to exchange crucial insights
- Insights accelerate solving

Cooperative Portfolio



- All experts work on original problem independently
- Brief meetings to exchange crucial insights
- Insights accelerate solving
- Only one expert needs to find a solution!

Motivation

Parallel & Distributed SAT solving

- Experts $\equiv$ diversified sequential solver threads
- Shared information $\equiv$ learned conflict clauses

Motivation

Parallel & Distributed SAT solving

- Experts ≡ diversified sequential solver threads
- Shared information ≡ learned conflict clauses

State of the art: MALLOBSAT (two talks later!)

- Periodic all-to-all clause sharing with duplicate detection and filtering of repeated clauses
- Strongly sublinear scaling in most cases
- How diverse, how redundant is the work performed by our solver threads?

Motivation

Parallel & Distributed SAT solving

- Experts ≡ diversified sequential solver threads
- Shared information ≡ learned conflict clauses

State of the art: MALLOBSAT (two talks later!)

- Periodic all-to-all clause sharing with duplicate detection and filtering of repeated clauses
- Strongly sublinear scaling in most cases
- How diverse, how redundant is the work performed by our solver threads?

Research questions

- How big is the overlap in learned clause sets across solver threads?
- Can these overlaps serve as a proxy for redundant work performed?
- Can we gain insights on suboptimal solver design based on learned clause overlaps?

Clause Logging Scheme

Goal: Log all clauses produced by all solver threads together with some meta data.

Problem: Many millions of clauses per second

Clause Logging Scheme

Goal: Log all clauses produced by all solver threads together with some meta data.

Problem: Many millions of clauses per second

Solution: Apply 64-bit high-quality hash function h to c, report clause **only if** $h(c) \bmod 16 = 0$.
$\Rightarrow$ Sampling factor 1:16 (-4 bits)

Clause Logging Scheme

Goal: Log all clauses produced by all solver threads together with some meta data.

Problem: Many millions of clauses per second

Solution: Apply 64-bit high-quality hash function h to c, report clause **only if** $h(c) \bmod 16 = 0$.
$\Rightarrow$ Sampling factor 1:16 (-4 bits)

Logging scheme: Report $r = (\frac{h(c)}{16}, t_c, p_c, s_c, d_c, g_c)$

- t_c: timestamp of logging
- p_c: index of producing process
- s_c: producing solver's local solver ID within the process
- d_c: clause length
- g_c: clause LBD score

Clause Logging Scheme

Goal: Log all clauses produced by all solver threads together with some meta data.

Problem: Many millions of clauses per second

Solution: Apply 64-bit high-quality hash function h to c, report clause **only if** $h(c) \bmod 16 = 0$.
$\Rightarrow$ Sampling factor 1:16 (-4 bits)

Logging scheme: Report $r = (\frac{h(c)}{16}, t_c, p_c, s_c, d_c, g_c)$

- t_c: timestamp of logging
- p_c: index of producing process
- s_c: producing solver's local solver ID within the process
- d_c: clause length
- g_c: clause LBD score

Accuracy: ≤ 1 expected hash collision at one billion hashed objects

Metrics

Assume we have a set of reports $\mathcal{R}$ whose set of unique hashes is $\mathcal{H}(\mathcal{R})$.

Metrics

Assume we have a set of reports $\mathcal{R}$ whose set of unique hashes is $\mathcal{H}(\mathcal{R})$.

Measuring the overall overlap of produced clauses:
Duplicate Clause Production Ratio (DCPR)

$$DCPR(\mathcal{R}) = \begin{cases} \frac{|\mathcal{R}| - |\mathcal{H}(\mathcal{R})|}{|\mathcal{R}|} & \text{if } |\mathcal{R}| \geq 1 \\ 0 & \text{otherwise} \end{cases}$$

Metrics

Assume we have a set of reports $\mathcal{R}$ whose set of unique hashes is $\mathcal{H}(\mathcal{R})$.

Measuring the overall overlap of produced clauses:
Duplicate Clause Production Ratio (DCPR)

$$DCPR(\mathcal{R}) = \begin{cases} \frac{|\mathcal{R}| - |\mathcal{H}(\mathcal{R})|}{|\mathcal{R}|} & \text{if } |\mathcal{R}| \geq 1 \\ 0 & \text{otherwise} \end{cases}$$

Measuring the overlap between two particular solvers:
Pairwise Produced Clause Overlap (PPCO)

$$PPCO(\mathcal{R}_x, \mathcal{R}_y) = \begin{cases} \frac{|\mathcal{H}(\mathcal{R}_x) \cap \mathcal{H}(\mathcal{R}_y)|}{|\mathcal{H}(\mathcal{R}_x) \cup \mathcal{H}(\mathcal{R}_y)|} & \text{if } |\mathcal{H}(\mathcal{R}_x)| + |\mathcal{H}(\mathcal{R}_y)| \geq 1 \\ 0 & \text{otherwise} \end{cases}$$

(a.k.a. Jaccard index for measuring similarity between two sets)

Experimental Setup

Solver: MALLOBSAT

- Cycling through KISSAT, CADICAL, and LINGELING
- Allow sharing, logging of clauses up to length 60 (and LBD 60)
- Full diversification: Seeds, sparse random variable phases, configuration options

Experimental Setup

Solver: MALLOBSAT

- Cycling through KISSAT, CADICAL, and LINGELING
- Allow sharing, logging of clauses up to length 60 (and LBD 60)
- Full diversification: Seeds, sparse random variable phases, configuration options

Hardware: HPC clusters SuperMUC-NG and HoreKa

- SuperMUC-NG (LRZ Munich): per node 2×24 cores, 96 GB RAM
- HoreKa (SCC Karlsruhe): per node 2×38 cores, 256 GB RAM

Experimental Setup

Solver: MALLOBSAT

- Cycling through KISSAT, CADICAL, and LINGELING
- Allow sharing, logging of clauses up to length 60 (and LBD 60)
- Full diversification: Seeds, sparse random variable phases, configuration options

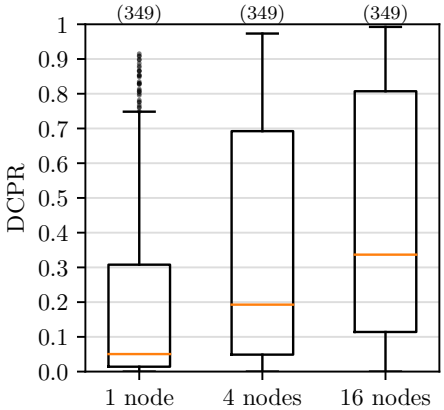
Hardware: HPC clusters SuperMUC-NG and HoreKa

- SuperMUC-NG (LRZ Munich): per node 2 $\times$ 24 cores, 96 GB RAM
- HoreKa (SCC Karlsruhe): per node 2 $\times$ 38 cores, 256 GB RAM

Benchmarks: 400 instances of SAT Competition 2022

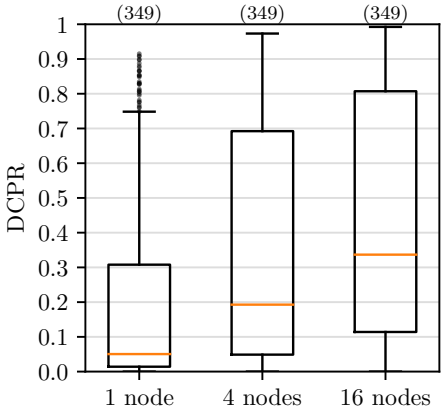
- Some experiments: only 349 instances which some solver @ SAT comp '22 solved
- 300 s wallclock time per instance

Overview

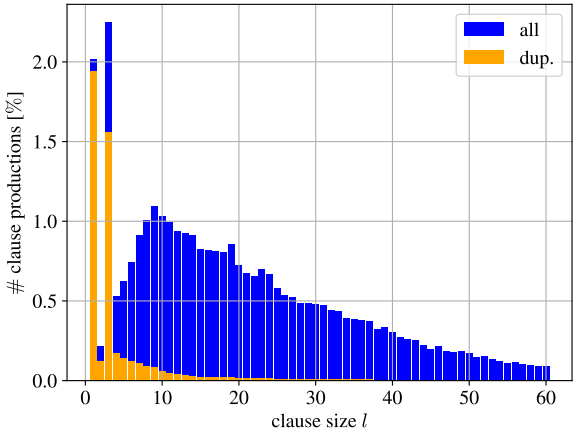


At 16 nodes (768 cores), two thirds of produced clauses are still unique.

Overview



At 16 nodes (768 cores), two thirds of produced clauses are still unique.



Clauses of length < 10 have highest overlaps.

By Benchmark Family

Family, result	#	*DCPR* min	median	mean ▲	max
`pigeon-hole-unsat`	2	0.004	0.082	0.019	0.082
`minimum-disagreement-parity-unsat`	3	0.018	0.046	0.034	0.047
`grid-coloring-sat`	10	0.007	0.043	0.042	0.311
`algorithm-equivalence-checking-unsat`	13	0.018	0.049	0.050	0.161
`graph-isomorphism-unsat`	8	0.024	0.063	0.052	0.091
...					
`planning-unsat`	2	0.817	0.992	0.900	0.992
`planning-sat`	3	0.943	0.966	0.964	0.982
`graceful-production-sat`	13	0.969	0.981	0.979	0.984
`sat-x-sat`	2	0.974	0.986	0.980	0.986
`software-verification-unsat`	14	0.971	0.980	0.981	0.990

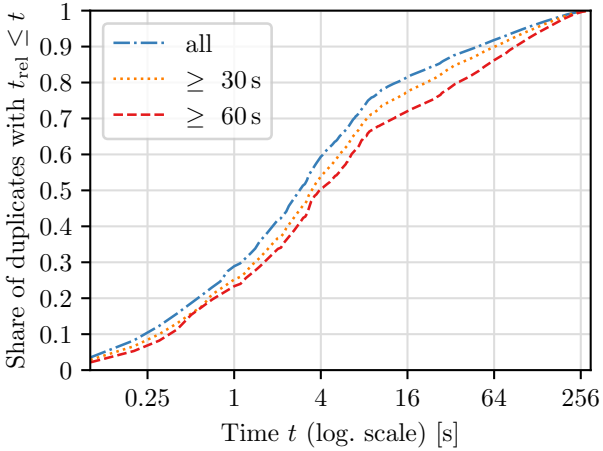
Temporal distribution

- Vast majority of duplicates are produced in the first few seconds of solving
- E.g., median DCPR of 0.28 after 3 s – $\approx 80\%$ of full run's duplicates!

Temporal distribution

- Vast majority of duplicates are produced in the first few seconds of solving
- E.g., median DCPR of 0.28 after 3 s – $\approx$ 80% of full run's duplicates!

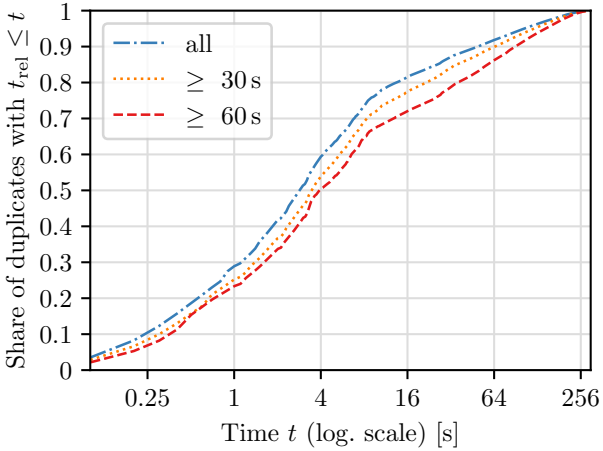
- 60% of duplicates are produced within 4 s after the clause's 1st production
- Similar results for "long runs" (e.g., $\geq$ 60 s)



Temporal distribution

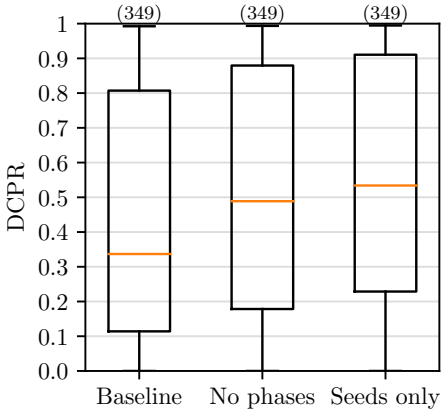
- Vast majority of duplicates are produced in the first few seconds of solving
- E.g., median DCPR of 0.28 after 3 s – $\approx$ 80% of full run's duplicates!

- 60% of duplicates are produced within 4 s after the clause's 1st production
- Similar results for "long runs" (e.g., $\geq$ 60 s)
- Explains why filtering recently shared clauses works well for short horizons



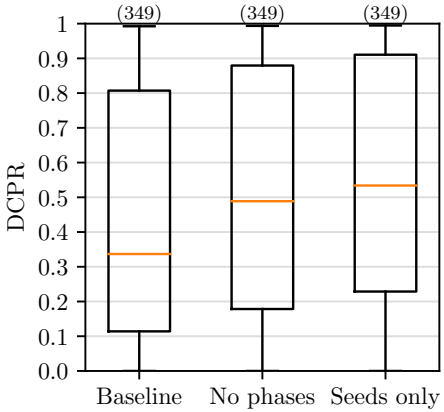
Diversification and Sharing

Impact of diversification



Diversification and Sharing

Impact of diversification

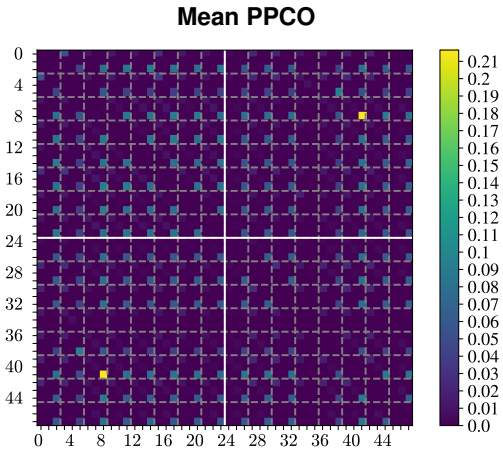


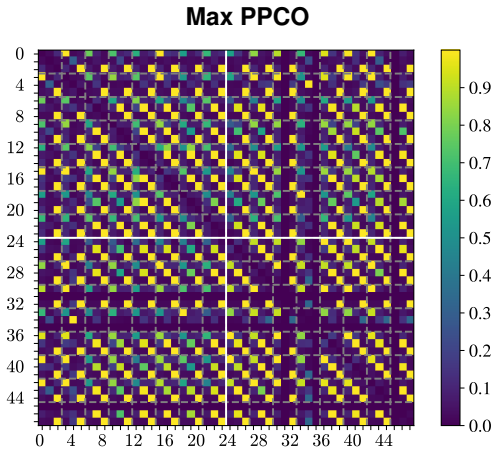
Impact of disabling clause sharing

- DCPR: 0.34 → 0.41

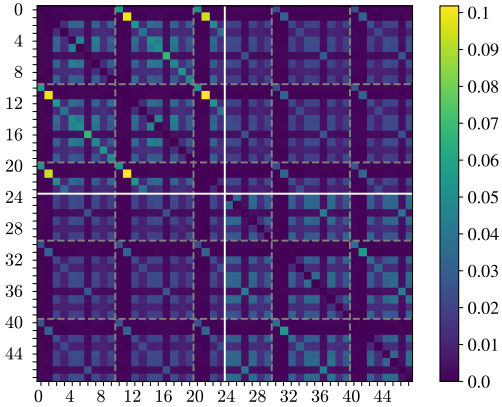
  ⇒ Clause sharing diversifies produced clauses
- 87 fewer instances solved

Pairwise Overlaps: Baseline (KISSAT-CADICAL-LINGELING)



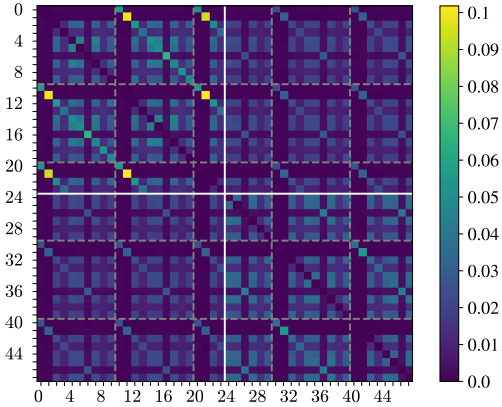


Pairwise Overlaps: CaDiCaL only



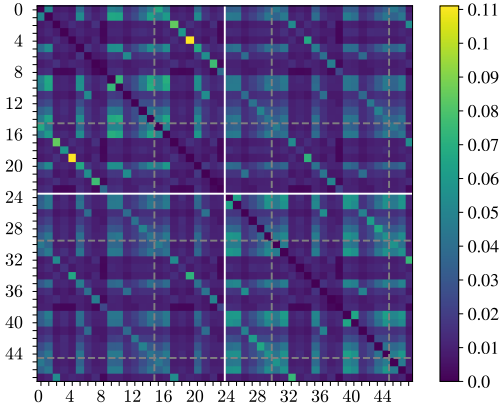
- Diagonals: Pairs of the same configuration have larger overlaps
- Smaller overlaps for modified restart intervals (C1, C6), flipped default phase (C0)

Pairwise Overlaps: CaDiCaL only



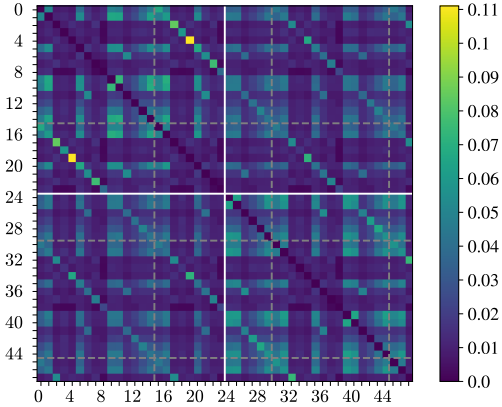
- Diagonals: Pairs of the same configuration have larger overlaps
- Smaller overlaps for modified restart intervals (C1, C6), flipped default phase (C0)
- Higher overlaps for pairs of the same process
  - 1st process always "lives the longest"
    $\rightarrow$ possible bias
  - Tree structure of sharing
    $\rightarrow$ clauses arrive at different points in time

Pairwise Overlaps: KISSAT'20 only



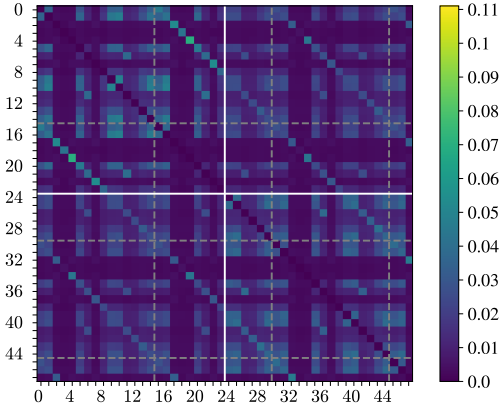
- Higher overlaps than for CADICAL!
  - Possible explanation for relatively poor performance of KISSAT vs. CADICAL in MALLOBSAT
- Low overlaps for altered restart intervals and disabling simplification techniques

Pairwise Overlaps: KISSAT'20 only



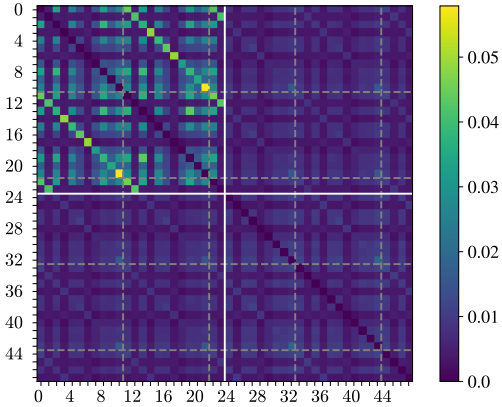
- Higher overlaps than for CADICAL!
  - Possible explanation for relatively poor performance of KISSAT vs. CADICAL in MALLOBSAT
- Low overlaps for altered restart intervals and disabling simplification techniques
- Together with excessive ternary clauses: primary suspect hyper-ternary resolution (HTR)

Pairwise Overlaps: KISSAT'20 only



- Higher overlaps than for CADICAL!
  - Possible explanation for relatively poor performance of KISSAT vs. CADICAL in MALLOBSAT
- Low overlaps for altered restart intervals and disabling simplification techniques
- Together with excessive ternary clauses: primary suspect hyper-ternary resolution (HTR)
- Disabling HTR: DCPR 0.47 $\rightarrow$ 0.33, +3 solved

Pairwise Overlaps: KISSAT'23 only



- Very low overlaps!
- "Missing" several inprocessing techniques like hyper-ternary resolution
- 322 solved (KISSAT'20: 315 solved)

Mitigations and Improvements

Can our findings translate to solver improvements?

- Turn off KISSAT's HTR (combat excessive ternary duplicates)
- Let the i-th LINGELING only export units c with $h_c \bmod n \in \{i, i+1\}$ (combat excessive unit duplicates)
- Delay i-th solver thread's first successful clause import by $i \bmod 11$ import queries

Mitigations and Improvements

Can our findings translate to solver improvements?

- Turn off KISSAT's HTR (combat excessive ternary duplicates)
- Let the i-th LINGELING only export units c with $h_c \bmod n \in \{i, i+1\}$ (combat excessive unit duplicates)
- Delay i-th solver thread's first successful clause import by $i \bmod 11$ import queries

Results on validation set (SAT comp '23)

- Drastic reduction of DCPR: $0.37 \rightarrow 0.22$
- Mild performance improvements: geom. mean speedup 2.7%, 58% of instances solved faster

Mitigations and Improvements

Can our findings translate to solver improvements?

- Turn off KISSAT's HTR (combat excessive ternary duplicates)
- Let the i-th LINGELING only export units c with $h_c \bmod n \in \{i, i+1\}$ (combat excessive unit duplicates)
- Delay i-th solver thread's first successful clause import by $i \bmod 11$ import queries

Results on validation set (SAT comp '23)

- Drastic reduction of DCPR: $0.37 \rightarrow 0.22$
- Mild performance improvements: geom. mean speedup 2.7%, 58% of instances solved faster

Why are clause overlaps not an accurate measure for redundant work?

- Some tasks in a solver are not reflected by an exported clause
- Some bursts of exported clauses (e.g., from some inprocessing) are very inexpensive (per clause)

Conclusion

Central findings

- Produced clauses in distributed solving are less redundant than one might expect
- Duplicates often tied to in–/preprocessing, often co-occur in close succession
- Diversification: Variable phases, restart intervals have large impact on produced clauses
- Found explanations for some earlier observations (KISSAT performance, short-horizon clause filtering)

Conclusion

Central findings

- Produced clauses in distributed solving are less redundant than one might expect
- Duplicates often tied to in–/preprocessing, often co-occur in close succession
- Diversification: Variable phases, restart intervals have large impact on produced clauses
- Found explanations for some earlier observations (KISSAT performance, short-horizon clause filtering)

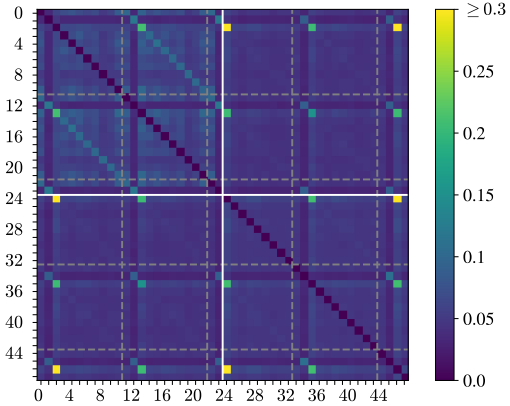
Limitations

- Only considered syntactical equality of clauses
- Black-box approach with unmodified solver backends vs. deeper look into provenance of individual clauses
- Only MALLOBSAT – what about other systems?

Try our analysis tool on your solver!

`github.com/jabo17/clause-lab`

Pairwise Overlaps: LINGELING only



- Smooth structure
- Configuration C1 disables simplification techniques → Smaller overlaps!