A Parameter Studies

We now report two additional parameter studies. They were done with the setting denoted “Ours”, with one difference: original LBDs were still used instead of the Deactivated-LBDs setting. Each run consisted of 396 instances with 300 s timeout on 8 nodes (384 cores).

The first study explores clause database reduction. Kissat offers parameters `reduce-low` (default 500) and `reduce-high` (default 900), which control the aggressiveness of database reductions. The default parameters correspond to reductions by 50% early in the run and by up to 90% later in the run. We tested four parameter settings, described briefly.

Reduce setting	#	+	−	PAR2
Default	323	153	170	137.9
Point	323	154	169	139.2
Gaussian	321	151	170	140.8
Range	318	149	169	144.2

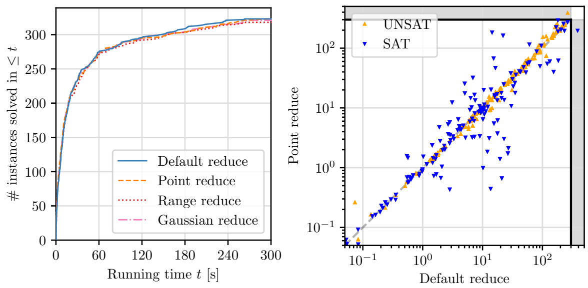


Figure 12 Effects of diversifying Kissat’s `reduce-low` and `reduce-high` parameters.

Default reduce: Default reduce parameters. **Point reduce**: Value $r \in [0, ..., 1000]$ is uniformly sampled per solver and both `reduce-low` and `reduce-high` are set to it. This creates some extreme solvers that keep all clauses forever ($r = 0$) and others that delete every clause almost immediately ($r = 1000$). **Range reduce**: A value $r \in [-200, 1200]$ is uniformly sampled per thread; then we set `reduce-low=max(0,r-200)` and `reduce-high=min(1000,r+200)`. Intuitively this slides the default interval [500,900] randomly to higher or lower values and leaves per solver the flexiblity of shifting from low to high reductions. **Gaussian reduce**: A value r is sampled from a Gaussian distribution with mean 700 and standard deviation 150, then r is clipped to be within [300,980] and both `reduce-low` and `reduce-high` are set to it. This sampling specifically avoids the extremes from the other two settings.

The results of the four reduce settings are shown in Fig. 12. Default reduce performs overall best, whereas Point reduce performs strongly on some SAT instances, which might be due to some aggressive solvers being allowed to eliminate almost all learned clauses.

The second study focuses on the decay of (E)VSIDS scores controlled by the `decay` parameter. Its default value is 50 (per mille), corresponding to an update of variables activity scores to 95% of their former value. Higher `decay` results in more aggressive updates.

Decay setting	#	+	−	PAR2
Uniform [1,50]	324	154	170	135.5
Default [50]	323	153	170	137.9
Uniform [1,200]	322	152	170	140.1
Uniform [50,200]	321	151	170	141.1

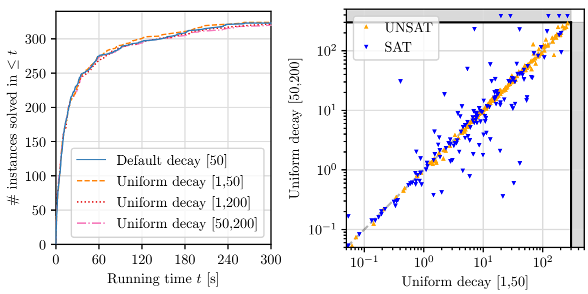


Figure 13 Effects of diversifying Kissat’s `decay` parameter.

Kissat accepts values in the range of [1,...,200]. We explore this full range by testing three settings: **Uniform[1,50]**, **Uniform[1,200]** and **Uniform[50,200]**. In each setting every solver thread samples its `decay` value uniformly from the given interval. The third setting is thus much more eager than the default, while the first is more conservative.

The results of the different decay settings are shown in Fig. 13. Regarding PAR2 scores, the conservative updating with `decay` at or below 50 performs better than the more aggressive choices. However, similar to the database reductions, the more eager approaches perform better on some SAT instances, observable in the direct comparison plot.

B Supplementary Data

Procedure	Kissat	768×d	768×s-o
backbone	0.12	0.18	—
congruence	0.46	1.85	—
eliminate	1.01	1.33	—
extend	0.00	0.00	—
factor	0.55	—	—
fastel	0.30	2.96	—
focused	43.39	44.65	49.97
lucky	0.08	1.34	1.59
parse	0.15	—	—
preprocess	0.83	4.77	—
probe	11.15	13.52	—
reduce	1.37	2.68	3.24
search	86.32	79.15	95.41
simplify	12.61	13.86	1.77
stable	42.93	34.50	45.44
substitute	0.66	1.39	—
subsume	0.39	0.36	—
sweep	2.18	1.54	—
transitive	0.17	0.20	—
vivify	7.03	8.35	—
walking	0.96	0.65	1.77

Table 2 Percentages of total ("CPU") time spent in different procedures of Kissat's SAT solving, for sequential Kissat, 768-core default setup, and 768-core search-only setup. Note that some categories subsume each another (e.g., focused and stable are disjoint sub-categories of search).

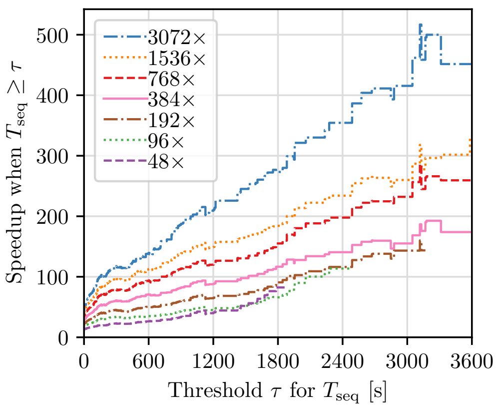


Figure 14 Weak Scaling of KCL configuration (as in Fig. 11).

Family	#	Avg. time	Speedup
miter-unsat	14	34.86	0.30
profitable-robust-production-sat	5	7.19	0.35
heule-nol-sat	7	18.50	0.43
social-golfer-sat	6	8.23	0.52
grs-fp-comm-unsat	8	75.60	0.62
scheduling-unsat	9	26.66	0.80
software-verification-unsat	10	56.29	0.97
cryptography-simon-sat	8	0.13	0.98
random-circuits-sat	5	24.89	1.01
hamiltonian-unsat	11	7.11	1.05
cryptography-ascon-unsat	6	8.58	1.05
argumentation-unsat	18	7.45	1.06
satcoin-unsat	5	16.36	1.28
brent-equations-sat	7	0.96	1.29
hamiltonian-sat	12	2.14	1.35
maxsat-optimum-sat	5	5.78	1.36
scheduling-sat	9	40.00	1.59
heule-folkman-sat	5	81.82	1.82
school-timetabling-sat	8	21.01	2.00
cryptography-sat	6	12.77	2.05
set-covering-sat	14	11.07	2.09
mutilated-chessboard-unsat	6	31.48	2.27

Table 3 Geom. mean speedup of "search-only" configuration over default Kissat-only configuration, at 768 cores, for each GBD benchmark family and result (SAT/UNSAT) group with data on ≥ 5 instances ("#"). "Avg. time" denotes the default configuration's according average running time.

Family	#	Avg. time	Speedup
set-covering-sat	6	0.21	0.04
heule-folkman-sat	6	5.82	0.08
register-allocation-unsat	11	0.12	0.10
random-circuits-sat	10	23.27	0.60
rbsat-sat	5	13.93	0.68
maxsat-optimum-unsat	5	19.11	0.70
hamiltonian-unsat	9	5.19	0.78
brent-equations-sat	9	0.63	0.81
argumentation-unsat	16	12.00	0.84
profitable-robust-production-sat	8	70.91	0.84
cryptography-ascon-unsat	10	8.12	0.88
quantum-kochen-specker-unsat	6	9.92	0.90
hamiltonian-sat	8	1.46	1.03
scheduling-unsat	6	28.64	1.04
scheduling-sat	17	24.12	1.08
satcoin-unsat	10	13.73	1.16
cryptography-sat	8	24.14	1.18
school-timetabling-sat	9	11.98	1.25
miter-sat	9	87.61	1.25
miter-unsat	23	29.96	1.26
coloring-unsat	5	24.48	1.26
software-verification-unsat	5	34.61	1.38
hashtable-safety-unsat	7	100.41	1.48
cryptography-ascon-sat	8	3.00	1.57
grs-fp-comm-unsat	6	65.93	1.63

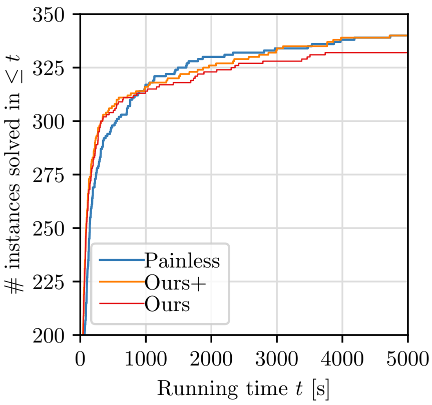
Table 4 Geometric mean speedup of our new setup over KCL, at 3072 cores, split as in Table 3. "Avg. time" denotes the average running time of KCL on the respective instances.

cores	K #	+	−	PAR	S_g	S_t	KCL #	+	−	PAR	S_g	S_t	Ours #	+	−	PAR	S_g	S_t
48	122	57	65	429.9	7.6	20.0	134	55	79	411.5	12.8	16.7	142	61	81	400.1	8.7	25.0
96	128	62	66	419.7	10.9	40.9	136	55	81	406.6	16.7	23.0	146	62	84	394.0	12.3	39.1
192	129	62	67	418.0	13.1	43.5	141	57	84	398.8	20.4	39.6	148	62	86	388.7	15.5	56.9
384	132	62	70	412.3	16.5	72.5	147	58	89	391.6	23.5	60.2	151	63	88	384.0	16.8	91.3
768	137	64	73	406.5	17.0	70.5	150	61	89	386.5	28.0	74.5	159	67	92	373.2	21.5	116.9
1536	142	66	76	398.8	18.1	80.5	151	60	91	384.4	30.8	81.5	157	65	92	374.5	23.4	120.6
3072	146	67	79	393.8	18.9	83.5	160	68	92	372.2	33.2	97.8	159	65	94	371.3	25.2	158.9

Table 5 Performance as in Tab. 1, limited to the 209 randomly chosen instances from SAT Competition 2023.

cores	K #	+	−	PAR	S_g	S_t	KCL #	+	−	PAR	S_g	S_t	Ours #	+	−	PAR	S_g	S_t
48	148	75	73	396.5	8.1	16.6	148	74	74	397.1	9.3	19.1	152	74	78	391.1	8.2	16.8
96	158	78	80	382.4	11.3	22.6	157	77	80	382.1	12.5	25.1	156	75	81	383.0	10.9	25.0
192	163	81	82	373.3	13.7	32.6	168	83	85	366.5	15.6	33.7	163	80	83	372.4	14.1	31.3
384	165	82	83	368.4	17.6	39.7	168	82	86	363.5	20.1	42.5	167	82	85	363.6	17.6	45.3
768	168	83	85	363.0	20.8	46.9	169	82	87	360.4	24.4	52.1	168	82	86	360.4	21.9	53.7
1536	172	85	87	357.0	24.1	55.8	172	84	88	354.6	29.6	64.4	172	84	88	352.9	25.8	76.8
3072	172	85	87	354.7	28.0	67.6	175	86	89	350.5	35.6	81.0	175	86	89	348.1	28.1	76.4

Table 6 Performance as in Tab. 1, limited to the 191 randomly chosen instances from SAT Competition 2024.



	#	+	−	PAR2
Ours	332	148	184	1857.0
Ours+	**340**	154	**186**	**1705.3**
Painless	**340**	**159**	181	1721.4

Figure 15 Performance of our approach from the paper ("Ours"), an enhanced version ("Ours+") with added Lingeling-based preprocessing and each 20th thread running YalSAT, and state-of-the-art shared-memory solver PL-PRS-BVA-KISSAT, at a single node (48 cores) and for up to 5000 s of running time, as in the SAT Competition Parallel tracks. Note the y axis offset.