The SAT Museum.
Armin Biere and Mathias Fleury and Nils Froleyks and Marijn J.H. Heule.
In Proceedings 14th International Workshop on Pragmatics of SAT (POS’23),
[paper - bibtex - data - zenodo - ceur - workshop - proceedings]
Legacy Solvers on SAT Competition 2023 Benchmarks

- 285 sbva–cadical−2023
- 256 cadical−2019
- 232 kissat−mab−hywalk−2022
- 223 kissat−mab−2021
- 220 kissat−2020
- 203 maple–lcm−disc−cb−dl−v3−2019
- 193 maple−lcm−dist−2017
- 189 maple−lcm−dist−cb−2018
- 181 maple−comsps−drup−2016
- 176 lingeling−2014
- 174 abcdsat−2015
- 169 glucose−2016
- 165 glucose−2011
- 161 lingeling−2010
- 160 cryptominisat−2010
- 158 lingeling−2013
- 156 glucose−2012
- 151 lingeling−2011
- 141 minisat−2008
- 139 precosat−2009
- 125 minisat−2006
- 122 rsat−2007
- 110 satelite−gti−2005
- 110 minisat−2005
- 87 chaff−2001
- 81 siege−2003
- 79 limmat−2002
- 67 picosat−2007
- 62 berkmin−2003
- 58 zchaff−2007
- 57 zchaff−2004
- 55 march−2011
- 51 boehm1−1992
- 30 posit−1995
- 15 grasp−1997
## Legacy Solvers on SAT Competition 2022 Benchmarks

<table>
<thead>
<tr>
<th>Solver</th>
<th>Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>kissat−mab−2021</td>
<td>2021</td>
</tr>
<tr>
<td>kissat−mab−hywalk−2022</td>
<td>2022</td>
</tr>
<tr>
<td>kissat−2020</td>
<td>2020</td>
</tr>
<tr>
<td>sbva−cadical−2023</td>
<td>2023</td>
</tr>
<tr>
<td>cadical−2019</td>
<td>2019</td>
</tr>
<tr>
<td>maple−lcm−disc−cb−dl−v3−2019</td>
<td>2019</td>
</tr>
<tr>
<td>maple−lcm−dist−cb−2018</td>
<td>2018</td>
</tr>
<tr>
<td>maple−lcm−dist−2017</td>
<td>2017</td>
</tr>
<tr>
<td>maple−comsps−drup−2016</td>
<td>2016</td>
</tr>
<tr>
<td>abcdsat−2015</td>
<td>2015</td>
</tr>
<tr>
<td>glucose−2016</td>
<td>2016</td>
</tr>
<tr>
<td>lingeling−2014</td>
<td>2014</td>
</tr>
<tr>
<td>lingeling−2013</td>
<td>2013</td>
</tr>
<tr>
<td>glucose−2012</td>
<td>2012</td>
</tr>
<tr>
<td>lingeling−2011</td>
<td>2011</td>
</tr>
<tr>
<td>glucose−2011</td>
<td>2011</td>
</tr>
<tr>
<td>cryptominisat−2010</td>
<td>2010</td>
</tr>
<tr>
<td>lingeling−2010</td>
<td>2010</td>
</tr>
<tr>
<td>precosat−2009</td>
<td>2009</td>
</tr>
<tr>
<td>satelite−gti−2005</td>
<td>2005</td>
</tr>
<tr>
<td>minisat−2006</td>
<td>2006</td>
</tr>
<tr>
<td>rsat−2007</td>
<td>2007</td>
</tr>
<tr>
<td>minisat−2005</td>
<td>2005</td>
</tr>
<tr>
<td>picosat−2007</td>
<td>2007</td>
</tr>
<tr>
<td>berkmin−2003</td>
<td>2003</td>
</tr>
<tr>
<td>siege−2003</td>
<td>2003</td>
</tr>
<tr>
<td>limmat−2002</td>
<td>2002</td>
</tr>
<tr>
<td>march−2011</td>
<td>2011</td>
</tr>
<tr>
<td>zchaff−2007</td>
<td>2007</td>
</tr>
<tr>
<td>zchaff−2004</td>
<td>2004</td>
</tr>
<tr>
<td>posit−1995</td>
<td>1995</td>
</tr>
<tr>
<td>boehm1−1992</td>
<td>1992</td>
</tr>
<tr>
<td>grasp−1997</td>
<td>1997</td>
</tr>
</tbody>
</table>

---

### Diagram

The diagram illustrates the performance of various legacy solvers on SAT Competition 2022 benchmarks. Each solver is represented by a specific marker, with the corresponding year indicated in the legend.
Overview

Part I

- competitions and quantum leaps
- preprocessing: bounded variable elimination / addition
- portfolio / stable-focused-mode
- target phases, rephasing

Part II

- random local search, restarts
- Las Vegas algorithm, probabilistic approximate complete (PAC)
- focused random search, random walks
- critical clauses, make-, break-value
- dynamic local search, clause weighting
Bounded Variable Elimination  

\[ (\overline{x} \lor a)_1 \quad (x \lor \overline{a} \lor \overline{b})_4 \quad (a \lor \overline{a} \lor \overline{b})_{14} \quad (a \lor d)_{15} \]

\[ (\overline{x} \lor b)_2 \quad (x \lor d)_5 \quad (b \lor \overline{a} \lor \overline{b})_{24} \quad (b \lor d)_{25} \]

\[ (\overline{x} \lor c)_3 \quad (c \lor \overline{a} \lor \overline{b})_{34} \quad (c \lor d)_{45} \]

- number of clauses *not increasing*
- strengthen and remove subsumed clauses too
- most important and most effective preprocessing we have

Bounded Variable Addition  

\[ (a \lor d) \quad (a \lor e) \quad \overline{x} \lor a \quad \overline{x} \lor b \quad \overline{x} \lor c \quad (x \lor d) \quad (x \lor e) \]

- number of clauses has to *decrease strictly*
- reencodes for instance naive at-most-one constraint encodings
- actually an interesting form of extended resolution
Chanseok Oh conjectured and proved empirically:
- solving *satisfiable* instances is different from
- solving *unsatisfiable* instances
- if we do not know whether the formula is satisfiable or not
  run an interleaved portfolio of two different solver configurations

**SAT** phase tries to solve satisfiable instances faster:
- benefits from few restarts (surprisingly) and
- from slower score decay (interesting variables slowly emerge)
- all-in-all forms more *stable* search behavior

**UNSAT** phase tries to solve unsatisfiable instances faster:
- benefits from frequent aggressive restarts (also surprisingly) and
- from fast score decay (chase variables involved in recent conflicts)
- all-in-all forms more *focused* search behavior

introduced in COMSPS solver, became effective in MapleCOMSPS
Target Phases

- idea: maximize “consistent” trail (motivated by “blocking restarts” on progress)
- save maximum consistent trail as \textit{phase} (value) assigned to a variable and after picking a decision variable assign it to this \textit{target phase}
- mostly useful in “stable mode” focusing on satisfiable instances
- earlier work (use \textit{saved phase}) used the last assigned value

Rephasing

- complements the \textit{intensification} technique of target phases with the (obvious) \textit{diversification} technique of \textit{resetting phases}:
  - set to constant \textit{original} or \textit{inverted original} phase, or
  - the \textit{cached best} (save best target phases on rephasing), or
  - phases are minimized through \textit{local-search}
Random Local Search Algorithm

flip-variable \( (\text{variable assignment } \alpha, \text{ variable } x) \quad // \alpha \text{ is call-by-reference} \\
1 \quad \alpha \leftarrow \alpha' \text{ with } \alpha'(y) = \begin{cases} 
-\alpha(x) & \text{if } x = y \\
\alpha(x) & \text{otherwise } (x \neq y)
\end{cases}

random-local-search (CNF \( F \))
2 \quad \alpha \leftarrow \text{random total assignment of variables in } F \\
3 \quad \textbf{while } \alpha(F) \neq 1 \\
4 \quad \text{pick variable } x \text{ in } F \text{ randomly} \\
5 \quad \text{flip-variable } (\alpha, x) \\
6 \quad \textbf{return } \alpha
Completeness

- considered local search algorithms are “incomplete” in two ways:
  - either might not terminate or return “unknown”
  - only can return satisfying assignments
- fall in the class of *Las Vegas* randomized algorithms
  - Monte Carlo: might return incorrect result with some (low) probability
  - Las Vegas: result guaranteed correct but with varying runtime
- applications
  - practical instances: generate test cases, hit coverage holes, …
  - excellent for randomly generated instances or hard combinatorial problems
  - “Local Search for SMT” [Fröhlich…-AAAI’15] [Niemetz…-FMCAD’15]
  - phase assignment for complete algorithms (CaDiCaL, Kissat, …)
    [CaiZhangFleuryBiere-JAIR’22] plus target phases
- probabilistic approximate complete (PAC) — [Hoos99]
  - algorithm does not get trapped and can escape local minima
  - always has a non-zero probability to find satisfying assignment
random-local-search-with-restarts (CNF $F$)
1 $\alpha \leftarrow$ random total assignment of variables in $F$
2 while $\alpha(F) \neq 1$
3 if it is time to restart then
4 $\alpha \leftarrow$ random total assignment of variables in $F$
5 else
6 pick variable $x$ in $F$ randomly
7 flip-variable ($\alpha$, $x$)
8 return $\alpha$
When it is time to restart?

- literature on local search has run-time parameter *max-flipped*
- often number of restarts is limited by *max-tries* too (i.e., outer loop)
- restart scheduling methods:
  - **never** or **always** restart
  - **constant** number of flips (i.e., *max-flipped*)
  - **arbitrarily** increase restart interval (increment \( i > 0 \))
    - \[ \text{max-flipped} \leftarrow \text{max-flipped} + i \]
  - **geometrically** increase restart interval (times factor \( f > 1 \))
    - \[ \text{max-flipped} \leftarrow f \cdot \text{max-flipped} \]
  - **inner-outer** geometric increase (as in PicoSAT)
    - use *inner* for *max-flipped* initialized with *initial*
      - \[ \text{inner} \leftarrow f \cdot \text{inner} \] after every restart and
      - if \( \text{inner} > \text{outer} \) then reset \( \text{inner} \leftarrow \text{initial} \) and \( \text{outer} \leftarrow f \cdot \text{outer} \)
  - **Luby** scheme (also called **reluctant doubling** by Knuth)
divide-distribute-fixed-weights (CNF $F$)

1. initialize clause weights $\omega : F \rightarrow \mathbb{Q}$ with $\omega(C) = \omega_0$

2. for $i = 1$ to max-tries

3. $\alpha \leftarrow$ random total assignment of variables in $F$

4. for $j = 1$ to max-flipped

5. if $\alpha(F) = 1$ then return “SATISFIABLE”

6. if exists weight-reducing-variable

7. flip-variable $(\alpha, x)$ with $x$ most reducing one and continue

8. if exists sideways-variable and with probability $p$

9. flip-variable $(\alpha, x)$ with $x$ is sideways and continue

10. for all falsified clauses $C$

11. $D \leftarrow$ maximum-weighted-satisfied clause neighboring $C$

12. if $\omega(D) < \omega_0$ or with probability $q$

13. $D \leftarrow$ random satisfied clause with $\omega(C) \leq \omega_0$

14. if $\omega(D) > \omega_0$ then transfer $\omega_>$ from $D$ to $C$

15. else transfer $\omega_=$ from $D$ to $C$

16. return “UNKNOWN”
Inner-Outer Restart Intervals

378 restarts in 104408 conflicts
int inner = 100, outer = 100;
int restarts = 0, flipped = 0;

for (;;) {
    ... // run SAT core loop for 'inner' flips

    restarts++;
    flipped += inner;

    if (inner >= outer) {
        outer *= 1.1;
        inner = 100;
    } else {
        inner *= 1.1;
    }
}
Luby’s Restart Intervals

70 restarts in 104448 conflicts
Luby Restart Scheduling

```c
unsigned
luby (unsigned i)
{
    unsigned k;

    for (k = 1; k < 32; k++)
        if (i == (1 << k) - 1)
            return 1 << (k - 1);

    for (k = 1;; k++)
        if ((1 << (k - 1)) <= i && i < (1 << k) - 1)
            return luby (i - (1 << (k-1)) + 1);
}

limit = 512 * luby (++restarts);
...  // run SAT core loop for 'limit' flips
```
Reluctant Doubling Sequence

[Knuth’12]

\[(u_1, v_1) := (1, 1)\]

\[(u_{n+1}, v_{n+1}) := (u_n \& -u_n = v_n ? (u_n + 1, 1) : (u_n, 2v_n))\]

\[(1, 1), (2, 1), (2, 2), (3, 1), (4, 1), (4, 2), (4, 4), (5, 1), \ldots\]
focused-random-walk-with-restarts (CNF $F$)

1. $\alpha \leftarrow$ random total assignment of variables in $F$
2. while $\alpha(F) \neq 1$
3. pick unsatisfied clause $C \in F$ randomly // with $\alpha(C) = 0$
4. pick literal $\ell \in C$ randomly
5. let $x = |\ell|$
6. flip-variable ($\alpha$, $x$) // flip variable in falsified clause
7. return $\alpha$
focused-random-walk-with-restarts (CNF $F$)
1 $\alpha \leftarrow$ random total assignment of variables in $F$
2 while $\alpha(F) \neq 1$
3 \hspace{1em} if it is time to restart then
4 \hspace{2em} $\alpha \leftarrow$ random total assignment of variables in $F$
5 \hspace{1em} else
6 \hspace{2em} pick unsatisfied clause $C \in F$ randomly \hspace{1em} // with $\alpha(C) = 0$
7 \hspace{2em} pick literal $\ell \in C$ randomly
8 \hspace{2em} let $x = |\ell|$\hspace{1em}
9 \hspace{2em} flip-variable ($\alpha$, $x$) \hspace{1em} // flip variable in falsified clause
10 return $\alpha$
**Make-Value, Critical Clauses, Break-Value**

*make-value* (CNF $F$, variable $x$, assignment $\alpha$)

1. `return $|\{ C \in F \mid \ell \in C, |\ell| = x, \alpha(C) = 0 \}|$` // number of falsified clauses with $x$

*is-critical-clause* (clause $C$, assignment $\alpha$)

2. `return $|\{ \ell \in C \mid \alpha(\ell) = 1 \}| = 1$` // exactly one literal of $C$ set to true

*break-value* (CNF $F$, variable $x$, assignment $\alpha$)

3. `return $|\{ C \in F \mid \ell \in C, |\ell| = x, \alpha(\ell) = 1, \text{is-critical-clause}(C, \alpha) \}|$` // #critical

*make-break-value* (CNF $F$, variable $x$, assignment $\alpha$)

4. `return make-value($F$, $x$, $\alpha$) − break-value($F$, $x$, $\alpha$)` // made minus broken
Example: Make-Value, Critical Clauses, Break-Value

consider flipping \( x \) with \( \alpha(x) = 1 \) in the following CNF

\[
C_1 \left( \neg x \lor x_1 \lor \neg x_2 \right) \quad \text{unsatisfied (falsified)}
\]
\[
C_2 \left( x \lor x_1 \lor x_3 \right) \quad \text{critical (single satisfied by } x)\]
\[
C_3 \left( x \lor \neg x_2 \lor \neg x_3 \right) \quad \text{double satisfied}
\]
\[
C_3 \left( x \lor x_2 \lor \neg x_3 \right) \quad \text{triple satisfied}
\]

and after flipping \( x \), so setting \( \alpha(x) = 0 \), we have

\[
C_1 \left( \neg x \lor x_1 \lor \neg x_2 \right) \quad \text{made (and critical)}
\]
\[
C_2 \left( x \lor x_1 \lor x_3 \right) \quad \text{broken}
\]
\[
C_3 \left( x \lor \neg x_2 \lor \neg x_3 \right) \quad \text{critical}
\]
\[
C_3 \left( x \lor x_2 \lor \neg x_3 \right) \quad \text{double satisfied}
\]

thus both make-value and break-value are 1
greedy-local-search-with-restarts (CNF $F$)
1    $\alpha \leftarrow$ random total assignment of variables in $F$
2    while $\alpha(F) \neq 1$
3        if it is time to restart then
4            $\alpha \leftarrow$ random total assignment of variables in $F$
5        else
6            $X \leftarrow \arg\max_x \text{make-break-value}(F, x, \alpha)$
7                pick $x$ randomly from $X$
8            flip-variable ($\alpha$, $x$)
9            return $\alpha$
greedy-local-search-with-random-walk (CNF $F$)
1 $\alpha \leftarrow$ random total assignment of variables in $F$
2 while $\alpha(F) \neq 1$
3     if it is time to restart then
4         $\alpha \leftarrow$ random total assignment of variables in $F$
5     else
6         with probability $p$ // random walk with probability $p$
7             pick random falsified clause $C$ // $\alpha(C) = 0$
8         select random literal $\ell \in C$
9         set variable $x \leftarrow |\ell|$
10        otherwise // greedy step with probability $1 - p$
11             $X \leftarrow \text{arg max} \ make-break-value (F, x, \alpha)$
12                 $x$ variable in $F$
13             pick $x$ randomly from $X$
14        flip-variable $(\alpha, x)$
15 return $\alpha$
walksat-local-search (CNF $F$)

1. $\alpha \leftarrow$ random total assignment of variables in $F$
2. while $\alpha(F) \neq 1$
   3. if it is time to restart then
      4. $\alpha \leftarrow$ random total assignment of variables in $F$
   5. else
      6. pick random falsified clause $C$ // $\alpha(C) = 0$
      7. select literal $\ell \leftarrow$ walksat-selection-heuristic ($F$, $C$, $\alpha$)
      8. flip-variable ($\alpha$, $|\ell|$)

9. return $\alpha$
WalkSAT Literal Selection Heuristic according to [Hoos-AAAI’99]

walksat-selection-heuristic (CNF $F$, clause $C$, assignment $\alpha$)

1. $L \leftarrow \{\ell \in C \mid \text{break-value}(F, |\ell|, \alpha) = 0\}$
2. If $L \neq \emptyset$
3. Return $\ell \in L$ randomly // greedily select literal with break-value zero
4. Else with probability $p$
5. Return $\ell \in C$ randomly // with probability $p$ select random literal
6. Otherwise
7. $K \leftarrow \arg \min_{\ell \in C} \text{break-value}(F, |\ell|, \alpha)$
8. Return $\ell \in K$ randomly // with probability $1 - p$ minimize break-value
Other WalkSAT Heuristics

- **Tabu Search**
  - maintain a “tabu” list of recently flipped variables
  - those are not allowed to be flipped

- **Novelty [McAllesterSelmanKautz-AAAI’97]**
  - use variable with best *make-break-value*
  - select best variable if not most-recently-flipped (*age*)
  - otherwise with probability $p$ choose second best
  - finally (with probability $1 - p$) still pick best
  - ties broken by *age*

- **Novelty+ [Hoos-AAAI’99]**
  - adds random walk step to Novelty
  - as otherwise Novelty is not PAC

- **AdoptNovelty+ [Hoos-AAAI’02]**
  - self-adapting noise mechanism
**probsat-score** (CNF $F$, literal $\ell$, assignment $\alpha$)

1. $b \leftarrow \text{break-value}(F, |\ell|, \alpha)$
2. **return** $2^{-b}$ // or other monotonically decreasing function $f(b)$ instead of $b$

**probsat-local-search** (CNF $F$)

3. $\alpha \leftarrow \text{random total assignment of variables in } F$
4. **while** $\alpha(F) \neq 1$
5.  pick random falsified clause $C$ // $\alpha(C) = 0$
6.  $S \leftarrow \sum_{\ell \in C} \text{probsat-score}(F, \ell, \alpha)$
7.  sample $\ell \in C$ over probability distribution $\ell \mapsto \frac{\text{probsat-score}(F, \ell, \alpha)}{S}$
8.  $\text{flip-variable}(\alpha, |\ell|)$
9. **return** $\alpha$
Clause Weights, Make-Score, Break-Score

make-score \((\text{CNF } F, \text{ variable } x, \text{ assignment } \alpha, \text{ clause weights } \omega)\)

$$G \leftarrow \{C \in F \mid \ell \in C, |\ell| = x, \alpha(C) = 0\}$$

1. return $$\sum_{C \in G} \omega(C)$$ // sum of weights of falsified clauses with $$x$$

break-score \((\text{CNF } F, \text{ variable } x, \text{ assignment } \alpha, \text{ clause weights } \omega)\)

$$G \leftarrow |\{C \in F \mid \ell \in C, |\ell| = x, \alpha(\ell) = 1, \text{ is-critical-clause } (C, \alpha)\}|$$

3. return $$\sum_{C \in G} \omega(C)$$ // sum of weights clauses where $$x$$ is critical

make-break-score \((\text{CNF } F, \text{ variable } x, \text{ assignment } \alpha, \text{ clause weights } \omega)\)

5. return $$\text{make-score}(F, x, \alpha) - \text{break-score}(F, x, \alpha)$$ // improved score
**dynamic-local-search** (CNF $F$)

1. $\alpha \leftarrow$ random total assignment of variables in $F$
2. initialize clause weights $\omega : F \rightarrow \mathbb{Q}$ with $\omega(C) = 1$
3. **while** $\alpha(F) \neq 1$
4. use *make/break* scores instead of values
5. same principles: “falsified clause”, “random walks”, “restarts”
6. additionally *scale* (increase) weights $\omega$ of unsatisfied clauses
7. optionally *smooth* (move to average) weights $\omega$ of all clauses
8. **return** $\alpha$

- scaling and smoothing global
- slower (but better?) flips per second than ProbSAT
- much more complex algorithmics
- CCASat [CaiSu-AAA1’12] successful over the years
- TaSSAT [ChowdhuryCodelHeule-NFM’23, ChowdhuryCodelHeule-TACAS’24] extension of YaLSAT [Biere’14]
divide-distribute-fixed-weights (CNF $F$)

1. initialize clause weights $\omega : F \rightarrow \mathbb{Q}$ with $\omega(C) = \omega_0$

2. for $i = 1$ to max-tries

3. $\alpha \leftarrow$ random total assignment of variables in $F$

4. for $j = 1$ to max-flipped

5. if $\alpha(F) = 1$ then return “SATISFIABLE”

6. if exists weight-reducing-variable

7. flip-variable $(\alpha, x)$ with $x$ most reducing one and continue

8. if exists sideways-variable and with probability $p$

9. flip-variable $(\alpha, x)$ with $x$ is sideways and continue

10. for all falsified clauses $C$

11. $D \leftarrow$ maximum-weighted-satisfied clause neighboring $C$

12. if $\omega(D) < \omega_0$ or with probability $q$

13. $D \leftarrow$ random satisfied clause with $\omega(C) \leq \omega_0$

14. if $\omega(D) > \omega_0$ then transfer $\omega >$ from $D$ to $C$

15. else transfer $\omega =$ from $D$ to $C$

16. return “UNKNOWN”
TaSSAT: Transfer and Share SAT

Md Solimul Chowdhury(✉), Cayden R. Codel(✉), and Marijn J. H. Heule(✉)

Carnegie Mellon University, Pittsburgh, PA, USA
{mdsolimc,ccodel,mheule}@cs.cmu.edu

Abstract. We present TaSSAT, a powerful local search SAT solver that effectively solves hard combinatorial problems. Its unique approach of transferring clause weights in local minima enhances its efficiency in solving problem instances. Since it is implemented on top of YalSAT, TaSSAT benefits from practical techniques such as restart strategies and thread parallelization. Our implementation includes a parallel version that shares data structures across threads, leading to a significant reduction in memory usage. Our experiments demonstrate that TaSSAT outperforms similar solvers on a vast set of SAT competition benchmarks. Notably, with the parallel configuration of TaSSAT, we improve lower bounds for several van der Waerden numbers.

Keywords: Local Search for SAT · Weight Transfer · Memory Efficiency
Fig. 2: Performance profiles for solver modifications on the anni benchmark set show that TaSSAT significantly outperforms the others. Since all solvers can quickly solve 600 instances, we start the y-axis at 600 to improve readability.
Legacy Solvers on SAT Competition Anniversary SAT Benchmarks

- 1932 kissat−mab−2021
- 1911 kissat−mab−hywalk−2022
- 1867 sbva−cadical−2023
- 1860 kissat−2020
- 1833 cadical−2019
- 1707 maple−lcm−disc−cb−dl−v3−2019
- 1670 maple−lcm−dist−2017
- 1663 maple−lcm−dist−cb−2018
- 1657 maple−comsps−drup−2016
- 1538 lingeling−2014
- 1517 minisat−2008
- 1515 abcdsat−2015
- 1499 lingeling−2013
- 1489 precosat−2009
- 1484 glucose−2012
- 1483 glucose−2011
- 1406 cryptominisat−2010
- 1323 minisat−2006
- 1306 rsat−2007
- 1262 satelite−gti−2005
- 1071 berkmin−2003
- 910 chaff−2001
- 907 zchaff−2004
- 776 limmat−2002
- 488 boehm1−1992
- 482 grasp−1997
- 466 posit−1995
Conclusion

- SAT solvers get faster and faster
  - SAT museum shows steady progress in the last 30 years
- every 3-5 years quantum leaps in performance improvement
  - due to clever new algorithms and search heuristics
  - pushed forward by the yearly SAT competition and
  - by a steady increase of applications
- old and new ideas have to be revisited
  - check-out 2nd edition of the “Handbook of Satisfiability”  
    [BiereHeuleMaarenWalsh’21]
  - intuitions need to be checked empirically
  - we also need to explain the progress better
- local-search reappeared as interesting research topic
Some Solvers on SAT Competition 2022 Benchmarks

- 301 new-kissat
- 276 kissat−2020
- 272 sbva−cadical−2023
- 266 cadical−2019
- 191 glucose−2011
- 147 minisat−2006
- 44 march−2011
- 32 zchaff−2004
- 7 grasp−1997