The Pindakaas library

Encoding pseudo-Boolean and linear constraints to SAT

Hendrik Henk Bierlee (henk.bierlee@kuleuven.be), post-doc @ KU Leuven

Prof. Tias Guns, ERC grant CHAT-Opt (101002802)

Collaborator: Dr. Jip Dekker @ Monash University, Australia

KU LEUVEN

The unreasonable effectiveness of SAT solver reasoning

SAT solvers can solve complex and hard SAT problems

- MiniZinc Challenge'24
  - Aircraft Disassembly Scheduling
  - Community Detection
- XCSP3 competition'24
- Applications: see SAT handbook [1]

Rank	Solver	Score
#1	cp-sat	**1,557.75**
#2	picat-sat	**1,479.17**
#3	chuffed	**1,478.11**
#4	gurobi	**1,286.07**
#5	cplex	**1,258.59**
#6	izplus	**1,159.11**
#7	cp-optimizer	**1,137.28**
#8	choco-solver_cp	**1,128.72**

Tab. 1: MiniZinc Challenge 2024

[1] A. Biere, M. J. H. Heule, H. van Maaren, and T. Walsh, Eds. [1]

SAT solvers are also versatile

- Incremental solving
- Verifiable solving via proof logging
- Other variants e.g. optimization (via MaxSAT)
- Model counting
- User propagators

Highlight: automated theorem proving

- E.g. of the chromatic packing problem [2]
- Combines many of these techniques to produce a 122TB proof closing the problem

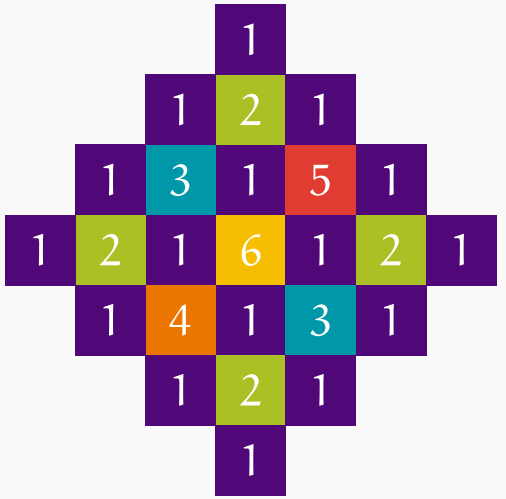
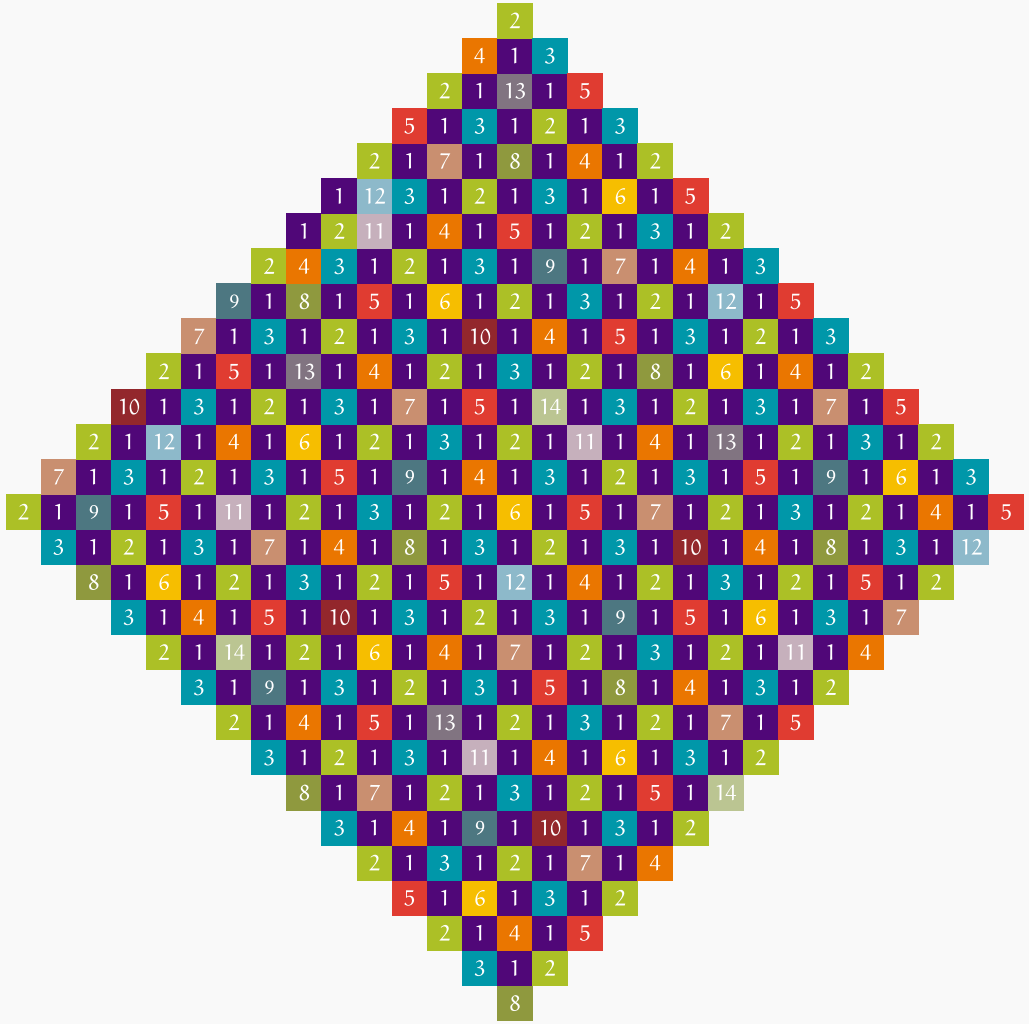


Fig. 1: Colouring a grid so that colour ***i*** is at least ***i*** steps away from the closest ***i***.[3]

[2] B. Subercaseaux and M. J. H. Heule [2]

[3] https://bsubercaseaux.github.io/blog/2023/packingchromatic/

Should we checkerboard the grid?



None of these problems are SAT problems

Pindakaas: solve high-level problems with low-level solvers

- SAT solvers are efficient solvers of the low-level SAT problem
- But they can solve high-level problems only once translated (i.e. encoded)
- The quality of the encoding is crucial for solver performance

Pindakaas is a library which provides effective encoding methods and convenient solver access

Pindakaas: features and integrations

- Features
  - The encoding of high-level constraints to SAT:
    - Propositional
    - Pseudo-Boolean
    - Integer linear
  - Efficient and open-source Rust implementation + Python interface
  - Three SOTA, incremental back-end SAT solvers
- Integrations

  CPMpy A Constraint Programming and Modeling library @ KU Leuven

  Huub A Lazy Clause Generation solver developed @ Monash University

The SAT problem in Conjunctive Normal Form (CNF)

- A conjunction (i.e. AND, $\wedge$)
  - of disjunctions (i.e., ORs, clauses, $\vee$)
    - of literals:
      - variable, x
      - negated variable, $\neg x$

The SAT problem in Conjunctive Normal Form (CNF)

- A conjunction (i.e. AND, $\wedge$)
  - of disjunctions (i.e., ORs, clauses, $\vee$)
    - of literals:
      - variable, x
      - negated variable, $\neg x$

Encoding and solving the XOR-problem for Boolean variables x, y:

$$x \oplus y$$

$$(x \vee y) \wedge (\neg x \vee \neg y) \quad \textit{dec.}\ x$$

$$\neg y \quad \Longrightarrow \quad \neg y$$

$$\top \quad \textit{fixp.}$$

$$\{(x \mapsto \text{true}), (y \mapsto \text{false})\}$$

Usage from CPMpy

```python
import cpmpy as cp
x = cp.boolvar()
y = cp.boolvar()
model = cp.Model(cp.Xor([x, y]))
assert model.solve(solver="pindakaas") is True
assert x.value() != y.value()
```

Advantage: many more supported constraint types and other solvers

Native usage through Python

```
import pindakaas as pdk
cadical = pdk.CaDiCaL()
x, y = cadical.new_vars(2)
cadical += x ^ y
with cadical.solve() as result:
    assert result.status == pdk.solver.Status.SATISFIED
    assert result.value(x) != result.value(y)
```

Advantage: wider scope, more features (e.g. more control over encoding)

Encoding a n region, k colour map-colouring problem

Fig. 3: Map colouring where $n = 12, k = 3$

Let integer variable x denote the colour assigned to North-Holland, y the colour to South-Holland, z to Utrecht, ...

$$x \in [1..k] \wedge y \in [1..k] \wedge z \in [1..k] \wedge \ldots$$
$$\wedge\, x \neq y \wedge x \neq z \wedge \ldots$$

A partial assignment:

$$\{(x \mapsto 1), (y \mapsto 2), (z \mapsto 3)\}$$

The direct encoding

- Encode variable x using k Boolean variables $[\![x = 1]\!], \ldots, [\![x = k]\!]$:

	$[\![x = 1]\!]$	$[\![x = 2]\!]$	$[\![x = 3]\!]$
$(x \mapsto 1)$	true	false	false
$(x \mapsto 2)$	false	true	false
$(x \mapsto 3)$	false	false	true

- Encode domain constraint $x \in [1..k]$
  - To prevent e.g. $\{([\![x = 1]\!] \mapsto \text{true}), ([\![x = 2]\!] \mapsto \text{true})\}$

$$[\![x = 1]\!] + \cdots + [\![x = k]\!] = 1$$

- Encode disequality constraint on adjacent nodes, e.g. $x \neq y$:

$$[\![x = 1]\!] \oplus [\![y = 1]\!] \wedge \cdots \wedge [\![x = k]\!] \oplus [\![y = k]\!]$$

Map colouring via CPMpy-pindakaas

```
import cpmpy as cp
# Create 12 integer variables with domains `[1..k]`
xs = cp.intvar(1, 3, shape=12)
# Create model with adjacency constraints
model = cp.Model(
  [
    xs[0] != xs[1],
    xs[0] != xs[2],
    # [..]
  ]
)
model.solve(solver="pindakaas")
```

Encoding with partial assignment $\{(x \mapsto 1), (y \mapsto 2), (z \mapsto 3)\}$

$$
\begin{aligned}
&(\llbracket x_1 = 1 \rrbracket \vee \llbracket x_1 = 2 \rrbracket \vee \llbracket x_1 = 3 \rrbracket) \wedge (\llbracket x_2 = 1 \rrbracket \vee \llbracket x_2 = 2 \rrbracket \vee \llbracket x_2 = 3 \rrbracket) \wedge (\llbracket x_3 = 1 \rrbracket \vee \llbracket x_3 = 2 \rrbracket \vee \llbracket x_3 = 3 \rrbracket) \wedge (\llbracket x_4 = 1 \rrbracket \vee \llbracket x_4 = 2 \rrbracket \vee \llbracket x_4 = 3 \rrbracket) \\
&\wedge (\llbracket x_5 = 1 \rrbracket \vee \llbracket x_5 = 2 \rrbracket \vee \llbracket x_5 = 3 \rrbracket) \wedge (\llbracket x_6 = 1 \rrbracket \vee \llbracket x_6 = 2 \rrbracket \vee \llbracket x_6 = 3 \rrbracket) \wedge (\llbracket x_7 = 1 \rrbracket \vee \llbracket x_7 = 2 \rrbracket \vee \llbracket x_7 = 3 \rrbracket) \wedge (\llbracket x_8 = 1 \rrbracket \vee \llbracket x_8 = 2 \rrbracket \vee \llbracket x_8 = 3 \rrbracket) \\
&\wedge (\llbracket x_9 = 1 \rrbracket \vee \llbracket x_9 = 2 \rrbracket \vee \llbracket x_9 = 3 \rrbracket) \wedge (\llbracket x_{10} = 1 \rrbracket \vee \llbracket x_{10} = 2 \rrbracket \vee \llbracket x_{10} = 3 \rrbracket) \wedge (\llbracket x_{11} = 1 \rrbracket \vee \llbracket x_{11} = 2 \rrbracket \vee \llbracket x_{11} = 3 \rrbracket) \wedge (\llbracket x_{12} = 1 \rrbracket \vee \llbracket x_{12} = 2 \rrbracket \vee \llbracket x_{12} = 3 \rrbracket) \\
&\wedge (\neg\llbracket x_4 = 1 \rrbracket \vee \neg\llbracket x_6 = 1 \rrbracket) \wedge (\neg\llbracket x_4 = 2 \rrbracket \vee \neg\llbracket x_6 = 2 \rrbracket) \wedge (\neg\llbracket x_4 = 3 \rrbracket \vee \neg\llbracket x_6 = 3 \rrbracket) \wedge (\neg\llbracket x_4 = 1 \rrbracket \vee \neg\llbracket x_8 = 1 \rrbracket) \\
&\wedge (\neg\llbracket x_4 = 2 \rrbracket \vee \neg\llbracket x_8 = 2 \rrbracket) \wedge (\neg\llbracket x_4 = 3 \rrbracket \vee \neg\llbracket x_8 = 3 \rrbracket) \wedge (\neg\llbracket x_4 = 1 \rrbracket \vee \neg\llbracket x_{11} = 1 \rrbracket) \wedge (\neg\llbracket x_4 = 2 \rrbracket \vee \neg\llbracket x_{11} = 2 \rrbracket) \\
&\wedge (\neg\llbracket x_4 = 3 \rrbracket \vee \neg\llbracket x_{11} = 3 \rrbracket) \wedge (\neg\llbracket x_5 = 1 \rrbracket \vee \neg\llbracket x_6 = 1 \rrbracket) \wedge (\neg\llbracket x_5 = 2 \rrbracket \vee \neg\llbracket x_6 = 2 \rrbracket) \wedge (\neg\llbracket x_5 = 3 \rrbracket \vee \neg\llbracket x_6 = 3 \rrbracket) \\
&\wedge (\neg\llbracket x_5 = 1 \rrbracket \vee \neg\llbracket x_7 = 1 \rrbracket) \wedge (\neg\llbracket x_5 = 2 \rrbracket \vee \neg\llbracket x_7 = 2 \rrbracket) \wedge (\neg\llbracket x_5 = 3 \rrbracket \vee \neg\llbracket x_7 = 3 \rrbracket) \wedge (\neg\llbracket x_5 = 1 \rrbracket \vee \neg\llbracket x_1 = 1 \rrbracket) \\
&\wedge (\neg\llbracket x_5 = 2 \rrbracket \vee \neg\llbracket x_1 = 2 \rrbracket) \wedge (\neg\llbracket x_5 = 3 \rrbracket \vee \neg\llbracket x_1 = 3 \rrbracket) \wedge (\neg\llbracket x_5 = 1 \rrbracket \vee \neg\llbracket x_{11} = 1 \rrbracket) \wedge (\neg\llbracket x_5 = 2 \rrbracket \vee \neg\llbracket x_{11} = 2 \rrbracket) \\
&\wedge (\neg\llbracket x_5 = 3 \rrbracket \vee \neg\llbracket x_{11} = 3 \rrbracket) \wedge (\neg\llbracket x_5 = 1 \rrbracket \vee \neg\llbracket x_3 = 1 \rrbracket) \wedge (\neg\llbracket x_5 = 2 \rrbracket \vee \neg\llbracket x_3 = 2 \rrbracket) \wedge (\neg\llbracket x_5 = 3 \rrbracket \vee \neg\llbracket x_3 = 3 \rrbracket) \\
&\wedge (\neg\llbracket x_6 = 1 \rrbracket \vee \neg\llbracket x_8 = 1 \rrbracket) \wedge (\neg\llbracket x_6 = 2 \rrbracket \vee \neg\llbracket x_8 = 2 \rrbracket) \wedge (\neg\llbracket x_6 = 3 \rrbracket \vee \neg\llbracket x_8 = 3 \rrbracket) \wedge (\neg\llbracket x_6 = 1 \rrbracket \vee \neg\llbracket x_1 = 1 \rrbracket) \\
&\wedge (\neg\llbracket x_6 = 2 \rrbracket \vee \neg\llbracket x_1 = 2 \rrbracket) \wedge (\neg\llbracket x_6 = 3 \rrbracket \vee \neg\llbracket x_1 = 3 \rrbracket) \wedge (\neg\llbracket x_6 = 1 \rrbracket \vee \neg\llbracket x_{11} = 1 \rrbracket) \wedge (\neg\llbracket x_6 = 2 \rrbracket \vee \neg\llbracket x_{11} = 2 \rrbracket) \\
&\wedge (\neg\llbracket x_6 = 3 \rrbracket \vee \neg\llbracket x_{11} = 3 \rrbracket) \wedge (\neg\llbracket x_7 = 1 \rrbracket \vee \neg\llbracket x_9 = 1 \rrbracket) \wedge (\neg\llbracket x_7 = 2 \rrbracket \vee \neg\llbracket x_9 = 2 \rrbracket) \wedge (\neg\llbracket x_7 = 3 \rrbracket \vee \neg\llbracket x_9 = 3 \rrbracket) \\
&\wedge (\neg\llbracket x_7 = 1 \rrbracket \vee \neg\llbracket x_{10} = 1 \rrbracket) \wedge (\neg\llbracket x_7 = 2 \rrbracket \vee \neg\llbracket x_{10} = 2 \rrbracket) \wedge (\neg\llbracket x_7 = 3 \rrbracket \vee \neg\llbracket x_{10} = 3 \rrbracket) \wedge (\neg\llbracket x_7 = 1 \rrbracket \vee \neg\llbracket x_{11} = 1 \rrbracket) \\
&\wedge (\neg\llbracket x_7 = 2 \rrbracket \vee \neg\llbracket x_{11} = 2 \rrbracket) \wedge (\neg\llbracket x_7 = 3 \rrbracket \vee \neg\llbracket x_{11} = 3 \rrbracket) \wedge (\neg\llbracket x_7 = 1 \rrbracket \vee \neg\llbracket x_2 = 1 \rrbracket) \wedge (\neg\llbracket x_7 = 2 \rrbracket \vee \neg\llbracket x_2 = 2 \rrbracket) \\
&\wedge (\neg\llbracket x_7 = 3 \rrbracket \vee \neg\llbracket x_2 = 3 \rrbracket) \wedge (\neg\llbracket x_7 = 1 \rrbracket \vee \neg\llbracket x_3 = 1 \rrbracket) \wedge (\neg\llbracket x_7 = 2 \rrbracket \vee \neg\llbracket x_3 = 2 \rrbracket) \wedge (\neg\llbracket x_7 = 3 \rrbracket \vee \neg\llbracket x_3 = 3 \rrbracket) \\
&\wedge (\neg\llbracket x_9 = 1 \rrbracket \vee \neg\llbracket x_{10} = 1 \rrbracket) \wedge (\neg\llbracket x_9 = 2 \rrbracket \vee \neg\llbracket x_{10} = 2 \rrbracket) \wedge (\neg\llbracket x_9 = 3 \rrbracket \vee \neg\llbracket x_{10} = 3 \rrbracket) \wedge (\neg\llbracket x_{10} = 1 \rrbracket \vee \neg\llbracket x_2 = 1 \rrbracket) \\
&\wedge (\neg\llbracket x_{10} = 2 \rrbracket \vee \neg\llbracket x_2 = 2 \rrbracket) \wedge (\neg\llbracket x_{10} = 3 \rrbracket \vee \neg\llbracket x_2 = 3 \rrbracket) \wedge (\neg\llbracket x_{10} = 1 \rrbracket \vee \neg\llbracket x_{12} = 1 \rrbracket) \wedge (\neg\llbracket x_{10} = 2 \rrbracket \vee \neg\llbracket x_{12} = 2 \rrbracket) \\
&\wedge (\neg\llbracket x_{10} = 3 \rrbracket \vee \neg\llbracket x_{12} = 3 \rrbracket) \wedge (\neg\llbracket x_1 = 1 \rrbracket \vee \neg\llbracket x_2 = 1 \rrbracket) \wedge (\neg\llbracket x_1 = 2 \rrbracket \vee \neg\llbracket x_2 = 2 \rrbracket) \wedge (\neg\llbracket x_1 = 3 \rrbracket \vee \neg\llbracket x_2 = 3 \rrbracket) \\
&\wedge (\neg\llbracket x_1 = 1 \rrbracket \vee \neg\llbracket x_3 = 1 \rrbracket) \wedge (\neg\llbracket x_1 = 2 \rrbracket \vee \neg\llbracket x_3 = 2 \rrbracket) \wedge (\neg\llbracket x_1 = 3 \rrbracket \vee \neg\llbracket x_3 = 3 \rrbracket) \wedge (\neg\llbracket x_2 = 1 \rrbracket \vee \neg\llbracket x_3 = 1 \rrbracket) \\
&\wedge (\neg\llbracket x_2 = 2 \rrbracket \vee \neg\llbracket x_3 = 2 \rrbracket) \wedge (\neg\llbracket x_2 = 3 \rrbracket \vee \neg\llbracket x_3 = 3 \rrbracket) \wedge (\neg\llbracket x_2 = 1 \rrbracket \vee \neg\llbracket x_{12} = 1 \rrbracket) \wedge (\neg\llbracket x_2 = 2 \rrbracket \vee \neg\llbracket x_{12} = 2 \rrbracket) \\
&\wedge (\neg\llbracket x_2 = 3 \rrbracket \vee \neg\llbracket x_{12} = 3 \rrbracket)
\end{aligned}
$$

What is a good encoding?

- This encoding is large but propagates efficiently
  - The solver infers this partial assignment will not lead to a solution without search decisions
- Other encodings are small yet propagate poorly
  - Trade-off between encoding size and propagation strength
- These and other complex choices arise during encoding

Fig. 4: Solution for $k = 4$

Single-constant multiplication

Encoding a integer linear constraint for integer variables, x_i:

$$\sum c_i x_i \leq c$$

- We can encode x_i with a binary representation
- But then we need to multiply by a constant: $c_i x_i = y_i$
- Fortunately, well-studied in the hardware circuit design community, so:

[CPAIOR'24] [4] Do the benefits carry over from hardware to software?

[4] H. Bierlee, J. J. Dekker, V. Lagoon, P. J. Stuckey, and G. Tack [3]

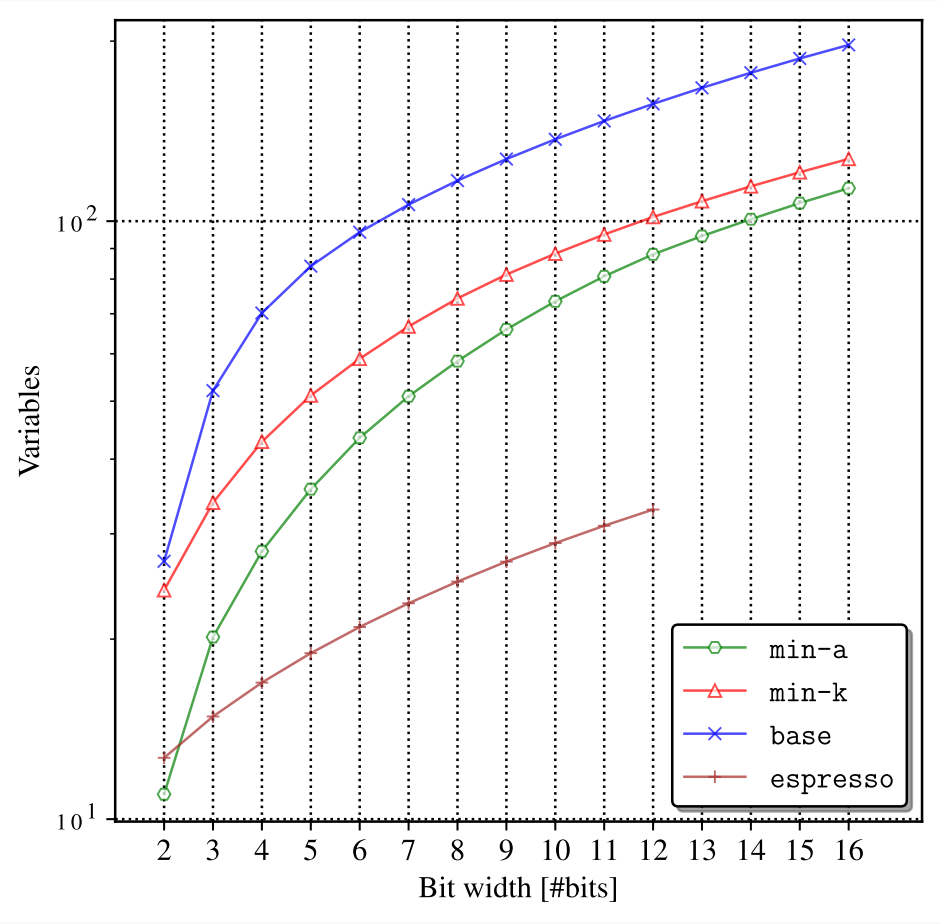
Finding optimal SCM encodings

- We decompose the multiplication into additions, subtractions, and multiplications by 2^s (i.e. left-shift by $s \geq 0$)
  - E.g. for $5x$, we encode $x + 4x = 5x$
  - E.g. for $11x$, we encode $x + 8x = 9x \wedge 9x + 2x = 11x$
- Why? Because additions and subtractions are cheap(er) and shifts are free
  - But minimizing number of additions/subtractions is NP-hard [5]
  - We cannot compute the optimal decomposition at encode time

Instead, pre-compute $c \cdot x$ for $1 \leq c \leq 2047$

[5] P. Cappello and K. Steiglitz [4]

Static results (i.e. encoding sizes)



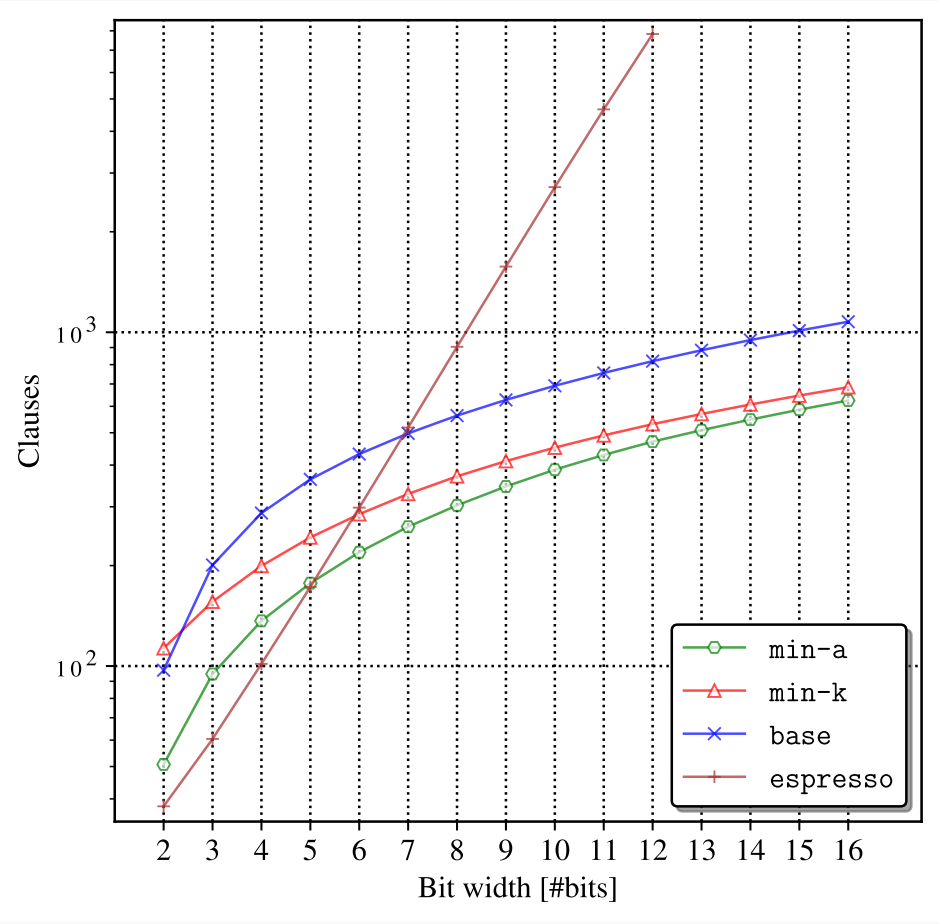


Fig. 5: Avg. #variables/#clauses for $c \cdot x, 1 \leq c \cdot \leq 2047, 2 \leq \text{bits}(x) \leq 16$

Challenges and future research

Challenges

- Ongoing challenge to release advanced features
  - Learning to cut features rather than stabilize them (now)
- Stiff competition from `RustSAT` (but competition is good)
  - Supporting a wider range of traditional encodings and features
  - But we focus more on higher-level constraints

Future research

- Proof-logging to certify the encoding is correct
- Inclusion in next version of CPMpy to run on wider sets of benchmarks
- Open to discuss new features to support collaborations!

Thank you for listening

Questions?

https://github.com/pindakaashq/pindakaas

References

[1] A. Biere, M. J. H. Heule, H. van Maaren, and T. Walsh, Eds., *Handbook of Satisfiability*, 2nd ed. IOS Press, 2021.

[2] B. Subercaseaux and M. J. H. Heule, "The Packing Chromatic Number of the Infinite Square Grid Is 15," in *Tools and Algorithms for the Construction and Analysis of Systems - 29th International Conference, TACAS 2023, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Paris, France, April 22-27, 2023, Proceedings, Part I*, S. Sankaranarayanan and N. Sharygina, Eds., in Lecture Notes in Computer Science, vol. 13993. Springer, 2023, pp. 389–406. doi: 10.1007/978-3-031-30823-9_20.

[3] H. Bierlee, J. J. Dekker, V. Lagoon, P. J. Stuckey, and G. Tack, "Single Constant Multiplication for SAT," in *Integration of Constraint Programming, Artificial Intelligence, and Operations Research - 21st International Conference, CPAIOR 2024, Uppsala, Sweden, May 28-31, 2024, Proceedings, Part I*, B. Dilkina, Ed., in Lecture Notes in Computer Science, vol. 14742. Springer, 2024, pp. 84–98. doi: 10.1007/978-3-031-60597-0_6.

[4] P. Cappello and K. Steiglitz, "Some Complexity Issues in Digital Signal Processing," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 32, no. 5, pp. 1037–1041, 1984.