Feature-Driven SAT Instance Generation

Benchmarking Model Counting Solvers Using Horn-Clause Variations

Author: Vuk Jurišić[1]
Responsible Professor: Dr. Anna L.D. Latour[1]
[1]Delft University of Technology

Background

CNF Formula:

$$F(X) := (x \vee y) \wedge (y \vee \neg z)$$

Horn Clause Definition:

A clause with at most one positive literal is called a **Horn Clause**.

$C_1 := (\neg x \vee \neg y)$ *(Horn Clause)*
$C_2 := (x \vee \neg y)$ *(Horn Clause)*
$C_3 := (x \vee y)$ *(Not Horn Clause)*

Truth Table for Model Counting:

x	y	z	Formula
0	0	0	UNSAT
0	0	1	UNSAT
0	1	0	SAT
0	1	1	SAT
1	0	0	SAT
1	0	1	UNSAT
1	1	0	SAT
1	1	1	SAT
Model Count			5

Motivation

- Solver performance depends on input instance characteristics:
  - *"Harder"* problems challenge solvers and can reveal bugs, weaknesses and strenghts.
  - Feedback can be extracted from solving instances that enduce such behaviour.
- We propose generating #SAT instances by varying horn-clauses-fractions feature:
  - Horn clauses have been researched before in both SAT and #SAT, however not in generation.
  - Existing generators do not explore full feature space of horn clauses, covering only 40% of it.

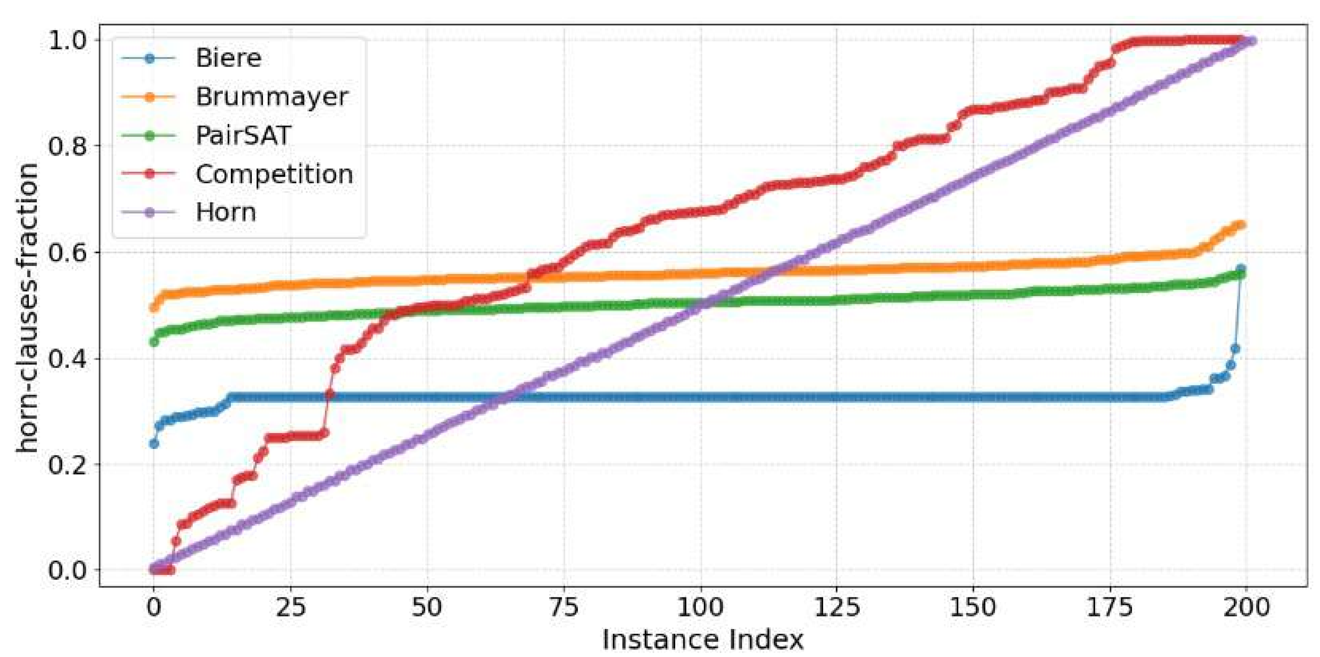

Figure 1. Fraction of Horn clauses in instances produced by existing generators.

Research Question

- *How can we design a #SAT instance generator that systematically varies the fraction of Horn clauses while keeping values of other features stable?*
- *How can analysing solver performance on instances produced by our generator reveal solver strengths, weaknesses, and opportunities for improvement?*

Methodology

- Selected 8 additional features [1]:
  - To benchmark horn-clause-fractions independently, instances should be similar in other properties.
- Designed a metric:
  - We used NCV to measure how well feature values vary between their theoretical amplitudes.
- Developed a custom generator:
  - Takes in an instance and outputs N instances with evenly distributed horn clause fractions (0% to 100%).
  - Utilizes concepts of post-processing by flipping literal polarity and solution fitting [2].
- Benchmarked state-of-the-art solvers:
  - Created large instance sets with different clause-to-variable ratios.

$$\text{NCV} = \frac{\sigma}{\mu} \cdot \frac{\text{Observed\_Max} - \text{Observed\_Min}}{\text{Theoretical\_Max} - \text{Theoretical\_Min}}$$

Figure 2. Normalized Coefficient of Variation (NCV) formula.

Solve time = f(Model count)?

Results

Feature Name	NCV Value		
	CNFuzzDD	Competition	Horn
horn-clauses-fraction	0.06118	0.35889	0.57053
BINARY+	0.23235	0.68741	0.00001
VCG-VAR-mean	0.13415	0.22757	0.00001
VCG-CLAUSE-mean	0.13410	0.23705	0.00001
cluster-coeff-mean	0.08751	1.50337	0.01160
vars-clauses-ratio	0.04495	0.50339	0.00064
reducedClauses	0.00729	0.29252	0.00009
reducedVars	0.00677	0.41309	0.00025
TRINARY+	0.06654	0.36689	0.00000

Table 1. NCV values for selected features instances generated with CNFuzzDD generator, Track 1 of 2024 MC Competition and our Horn Generator.

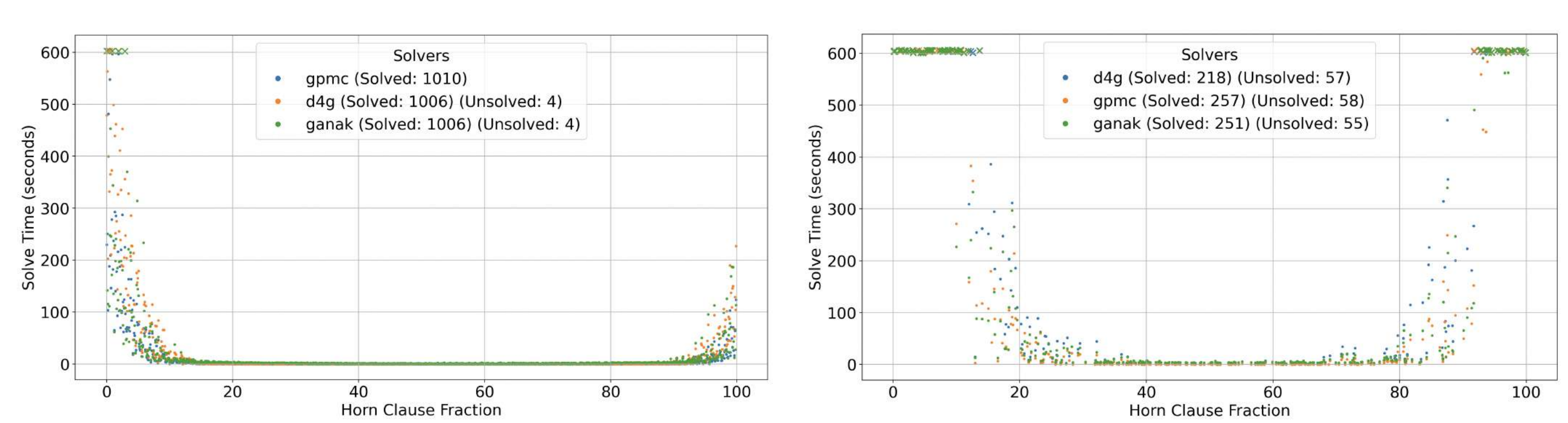

Figure 3. Solver performance on instances with 400 variables and clause counts: 90 and 110.

- Applying the transformation function $\sqrt[4]{\text{model count}}$, we observe a Pearson correlation coefficient of 0.862 and a Spearman rank correlation coefficient of 0.972 between model count and solving time.

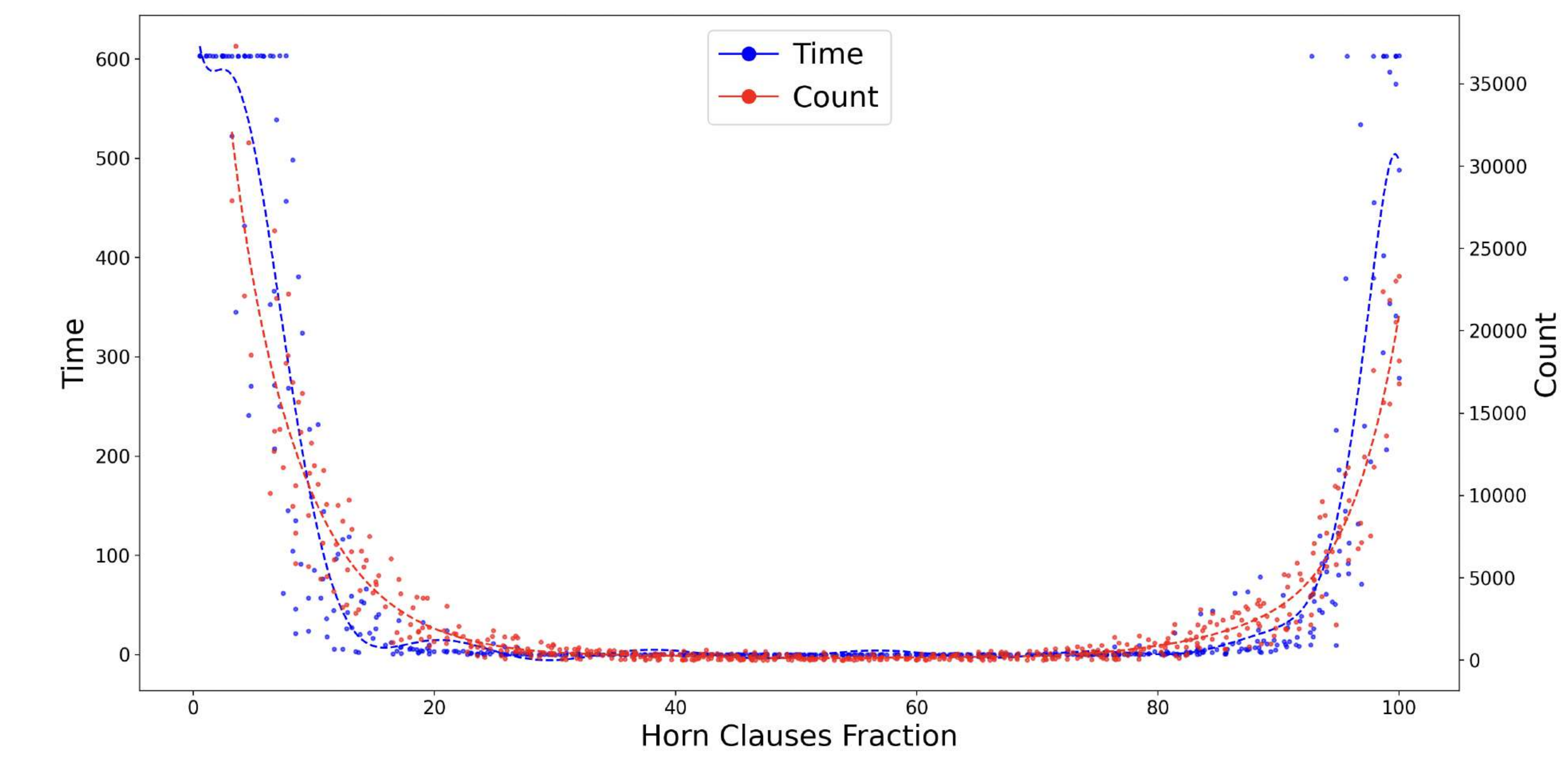

Figure 4. *gpmc* solver runtime and model count for instances with 100 clauses, with transformation function $f(x) = \sqrt[4]{x}$ applied to model count.

Conclusion

- Implemented a new generator exploring the full feature space of Horn-clause fractions.
- Solvers were particularly challenged by instances with extreme Horn-clause fractions.
- Comparison of solvers showed:
  - `ganak` took 4 times more time on instances with standard Horn-clause fractions, comparing to `d4` and `gpmc`.
  - `d4` was slower then other 2 when solving on problems with extreme Horn-clause fractions.
- A strong correlation is suspected between model count and solve time for all solvers.

Future Work

- Improve the generator by performing informed instead of random modifications.
- Conduct experiments with clauses of varying arities for greater versatility.
- Further investigate the relationship between model count and solve time across diverse instance sets.
- Establish connections between solver algorithms and observed performance results.

References

[1] E. Nudelman, K. Leyton-Brown, H. H. Hoos, A. Devkar, and Y. Shoham, "Understanding Random SAT: Beyond the Clauses-to-Variables Ratio," in *Principles and Practice of Constraint Programming – CP 2004*, M. Wallace, Ed. Berlin, Heidelberg: Springer, 2004, pp. 438–452.

[2] G. Escamocher and B. O'Sullivan, "Generation and Prediction of Difficult Model Counting Instances," Dec. 2022, arXiv:2212.02893. [Online]. Available: http://arxiv.org/abs/2212.02893

V.Jurisic@student.tudelft.nl