SAT Encodings: From Art to Science

Thesis Proposal

Bernardo Subercaseaux
bersub@cmu.edu

To be presented April 8, 2026

Thesis Committee

Marijn Heule	(*Chair*) Carnegie Mellon University
Jeremy Avigad	Carnegie Mellon University
Ruben Martins	Carnegie Mellon University
Stefan Szeider	TU Wien
Ryan Williams	MIT

Summary

Automated reasoning engines, and SAT solvers in particular, have become a powerful tool for tackling hard combinatorial problems. While SAT solvers have improved dramatically, their effectiveness still depends critically on how problems are encoded into conjunctive normal form (CNF). Encoding choices routinely account for runtime differences of several orders of magnitude, yet their design remains more art than science—guided by intuition and hard-won experience rather than a principled theory. This thesis aims to shed light on the design of effective SAT encodings, from both practical and theoretical perspectives.

The first part is dedicated to the *art* of encodings. We show how clever problem-specific encodings, together with other automated reasoning techniques, have allowed us to solve a variety of open problems in discrete mathematics ranging from graph coloring to discrete geometry. These encodings benefit from mathematical insights incorporated via additional constraints or auxiliary variables that represent semantically important aspects of the problem. Notably, some of these insights are themselves inspired by computational experiments, establishing a symbiotic relationship between automated reasoning and discrete mathematics. We also apply this art to computational problems in explainable AI—uncovering patterns in decision trees, nearest-neighbor classifiers, and other symbolic models—demonstrating the transferability of our techniques beyond mathematics.

The second part is dedicated to the foundations of a *science* of encodings. We present a landscape of theoretical results regarding the number of clauses and auxiliary variables required to encode several building blocks of propositional encodings, from cardinality constraints to arbitrary k-CNF functions. We posit that encoding boolean functions into CNF with as few clauses as possible offers a fascinating yet largely unexplored territory for circuit complexity. By leveraging auxiliary variables and wide clauses, this clause-minimization model permits rich combinatorial structures that surpass known lower bounds for circuits. Furthermore, while clause-minimization does not always correlate with solver performance, theoretical developments in this model have led us to novel encodings that run faster on practical problems. More broadly, this thesis aims to establish that the clause-minimization model is not only an elegant theory, but also a realistic path towards empirical speedups.

Completed Work.

Using problem-specific SAT encodings, as well as symmetry-breaking techniques, and the *Cube And Conquer* paradigm for parallel SAT-solving [HKB18, HKM16], we have made several contributions to problems in discrete mathematics:

- We determined the *packing chromatic number* of the infinite square grid to be 15, settling a 20-year-old problem [SH23b, SH22] *(SAT 2022, TACAS 2023 best paper award nominee).*
- We improved the best bounds for radio-colorings of hypercubes of order 7 and 8, making progress towards a question of Knuth [SH23a] *(LPAR 2023 best paper award).*
- We improved several finite bounds for the maximum number of points in general position without a convex pentagon [SMHM24] *(CICM 2024 best paper award runner-up).*
- We refuted a conjecture on the minimum area of a polyomino that can be folded into three non-isomorphic boxes [QWSH25] *(CADE 2025).*
- We showed how SAT solving could be used to obtain symmetric solutions to problems in discrete geometry, and determined the minimum odd number of points required for a 2-everywhere-unbalanced pointset [SMQH26] *(CICM 2025 best paper award).*
- We verified Norin's antipodal conjecture for hypercubes of order 8, and improved the asymptotic bounds for the general case by a SAT-guided probabilistic argument [KPSS25] *(Under submission).*

Following my master's thesis work on the computational complexity of explainable AI—a problem of increasing societal importance—I have leveraged the knowledge of SAT-solving learned from my advisor Marijn Heule to empower practical methods for automatically explaining the decisions of symbolic models. Namely:

- We solved a conjecture on the difficulty of *probabilistic explanations* for decision trees, a seemingly interpretable class of models, for which explanations turn out to be NP-hard. I designed a practical encoding to compute such explanations [ABOS22] *(NeurIPS 2022).*
- We designed a high-level language for querying the decisions of decision trees, and showed using finite model theory that queries could be evaluated in P^{NP}. We then designed an engine compiling such queries into SAT instances [ABB+24] *(KR 2024).*
- We studied the complexity of explanations for k-nearest-neighbors classifiers, showing several hardness results, and that an elegant SAT encoding could compute explanations in practice [BKR+25] *(PODS 2025 distinguished paper award).*

Motivated by my contribution to applied work using MaxSAT to study metrics of string compressibility [BGI+26] *(Transactions on Algorithms 2026)*, I initiated a line of research studying how the presence of *bicliques* in constraint graphs can be exploited to design more compact encodings [Sub25] (*ModRef 2025*). More generally, I started studying the minimum number of clauses required to encode 2-CNF formulas, as well as cardinality constraints. Since then, I have been joined by the extremely talented young students Andrew Krapivin and Benjamin Przybocki, and together we have greatly expanded this line of work. Concretely,

- We solved several open problems regarding biclique partitions on graphs and their natural generalization to k-uniform hypergraphs, as combinatorial groundwork for the clause-minimization model [KPSMS25] *(STOC 2026).*

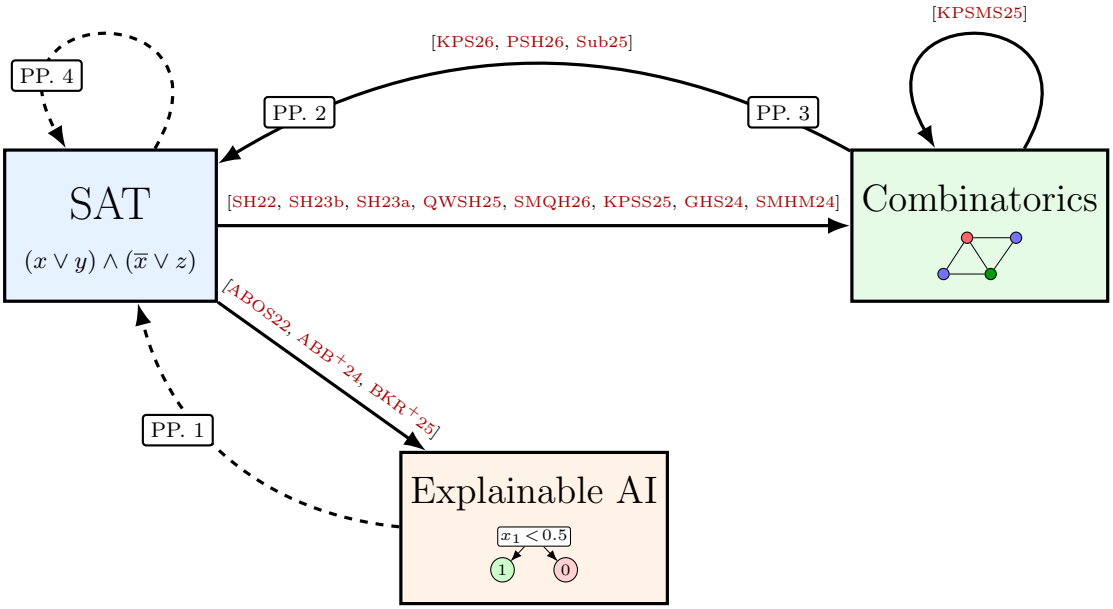


Figure 1: Summary of interactions between fields involved in this thesis proposal. Dashed arrows indicate new directions, whereas solid arrows indicate more established directions. The acronym 'PP' stands for "Proposed Project".

- We improved the best known encodings for cardinality constraints in several regimes, including a more compact encoding for the *at-most-one* constraint which refuted a conjecture of Chen, and led us not only to the first improvement on circuits for the complement *threshold-2* function in 50 years, but also to answer other long-standing questions in circuit complexity. We presented radically new encodings for the *at-most-k* constraint on n variables, which together with our new lower bounds, settled the minimum number of clauses to be $2n + \widetilde{\Theta}(\sqrt{n})$ for any $k \in \log^{O(1)}(n)$. Moreover, we showed that these encodings can sometimes be beneficial in practice, challenging the assumption of propagation completeness being crucial for performance [KPS26] *(submitted to SAT 2026)*.
- We leveraged our work on biclique partitions to better understand and even characterize the *Bounded Variable Addition* (BVA) preprocessing technique, which is now a key ingredient of state-of-the-art solvers. Using an efficient algorithm from our STOC paper in combination with existing implementations of BVA, we improve runtimes on solving the independent set problem for random graphs [PSH26] *(submitted to SAT 2026)*.

I view the first itemized list as *SAT for Combinatorics*,[1] the second itemized list as *SAT for Explainable AI*, and the third itemized list as *Combinatorics for SAT*. I have included in Figure 1 an illustration of the interactions between these fields.

Proposed Work

As suggested by the dashed arrows in Figure 1, I will propose directions of future work that aim for new connections between the aforementioned fields. I will also propose directions that lie neatly within the already explored solid arrows; despite their solid representation, I see them all as explorations in their infancy and ripe with opportunities. My main proposed projects (PPs) are:

[1]The website `sat4math.com`, which I created and maintain, compiles over 100 papers on the topic.

- (**PP 1.** *Explainable AI for SAT*) Why do SAT solvers work so well on structured instances? This question, also known as "the unreasonable effectiveness of SAT solvers", has puzzled the community for decades, and I believe ideas from explainable AI can be leveraged to make progress on it. The remarkable success of Large Language Models (LLMs) has motivated a great deal of techniques in explainable AI, and specifically, in *mechanistic interpretability* [SW24]. While LLMs and SAT solvers are objects of a very different nature, their explanation problems share some similarities (e.g., they both have an autoregressive dynamic, in which earlier decisions may "chaotically" influence later ones), so I propose to initiate a line of work borrowing ideas from the abundant work on mechanistic interpretability for LLMs, such as *causal interventions* [GIZ+25] to help us understand modern solvers.

- (**PP 2.** *Expanding the theory and practice of preprocessing*) Another path towards making progress in understanding the success of modern solvers is to understand the power and limitations of their multiple different components (e.g., preprocessing, heuristics, inprocessing) individually. As mentioned in the completed work section, we have already submitted a paper on using graph-theoretic techniques to understand *Bounded Variable Addition* (BVA). There, we developed a prototype tool BiVA that uses a biclique-partition algorithm from our STOC paper to perform BVA-preprocessing faster. I propose to develop this prototype further into a more general preprocessing tool that improves upon state-of-the-art preprocessing. On the more theoretical side, I aim to study two theoretical questions regarding BVA for which I have partial progress: (1) Can we prove upper or lower bounds on its reencoding potential over random monotone 2-CNF formulas? (2) Can we prove upper or lower bounds on its proof-complexity impact when restricted to the 2-CNF fragment of formulas?

- (**PP 3.** *Encoding size and Propagation Completeness*) Propagation completeness is a desirable property of CNF encodings, which in very rough terms says that if some of the variables have been assigned and this is enough to semantically imply the value of another variable, then this implication is derivable by unit propagation. Our recent work in cardinality constraints has shown significantly smaller encodings for boolean functions, at the cost of losing propagation completeness. Similarly, the best propagation complete encodings we know for arbitrary 2-CNF functions have size $\Theta(n^2/\lg n)$, whereas using Szemerédi's regularity lemma among other ideas we can get $o(n^2/\lg n)$ clauses but losing propagation completeness. I plan to further investigate this trade-off, and also better understand the practical impact of propagation completeness in modern solvers.

- (**PP 4.** *SAT for SAT*) A central question in my current work is, given a boolean function $f\colon \{0,1\}^n \to \{0,1\}$, what is the minimum number of clauses to encode it? Given a concrete representation of f such as its truth table, this question itself is a computational problem I plan to explore with SAT. Given f's truth table and integer k, the problem of deciding whether f can be encoded in k clauses is in $\Sigma_2\mathsf{P}$; while *a priori* out of the SAT domain, there are several ways of using SAT technology to attack problems in this class. A concrete goal would be to have a program that, given any of the 2^{2^8} possible functions on 8 bits, can respond with an encoding minimizing the number of clauses, and a checkable proof of its correctness and minimality. As detailed in page 20, I have already experienced how such finite search, for $n = 6$, was able to teach me a new encoding for a function I cared about, that I then generalized to arbitrary n. I expect that the higher the n we can reach, the more interesting and complex patterns will emerge. This project could lead to a concrete "taxonomy" of small boolean functions, which could also provide data for how standard properties such as sensitivity or algebraic degree correlate with encoding size.

SAT Encodings: From Art to Science

The classic aphorism "it's not *what* you say, it's *how* you say it" describes quite aptly the problem of SAT encodings: the particular way in which we express a given constraint in CNF can make a significant difference in the ability of SAT solvers to deal with it efficiently.

I will illustrate this point with a seemingly simple example: *Sudokus*—it will turn out that this example, so frequently used to explain SAT to beginners, already hides many interesting tricks that I have not seen published anywhere.

Consider an $n \times n$ Sudoku puzzle, with n a perfect square, and recall that we must enforce that each row, column, and $\sqrt{n} \times \sqrt{n}$ "box"[2] contains a permutation of $[n] := \{1, \dots, n\}$. The most natural choice of variables for the problem is to have $x_{i,j,t}$ represent that number t will go in position (i, j) of the grid. Then, the most direct formulation is to enforce:

1. *Each cell must contain some number:* we have a clause $\bigvee_{t=1}^{n} x_{i,j,t}$ for every $(i, j) \in [n]^2$.
2. *No duplicates:* for each row i, and number $t \in [n]$ we enforce an "*at-most-one* constraint" $\sum_{j=1}^{n} x_{i,j,t} \leq 1$, and proceed analogously for the columns and the boxes.
3. *Respect the clues:* for each given clue saying that cell (i, j) contains the number t, we add the unit clause $(x_{i,j,t})$.

While the first and third constraints are already in clausal form, the second one offers significant freedom for *encodings*. The abstract problem at hand is how to encode into CNF the boolean function $\mathsf{AtMostOne}(x_1, \dots, x_n)$ which is true if and only if at most one of the x_i variables is true. To be more formal, we can define encodings as follows.

Definition 1. Given a boolean function $f\colon \{\bot, \top\}^n \to \{\bot, \top\}$, we say that a triple (φ, X, Y) is an *encoding* for f if: (i) X is a set of n propositional variables, and Y is a set of variables disjoint from X; (ii) φ is a CNF formula over variables $X \sqcup Y$; and (iii) for every assignment $\tau\colon X \to \{\bot, \top\}$,

$$f(\tau(x_1), \dots, \tau(x_n)) = \top \iff \exists \theta\colon Y \to \{\bot, \top\},\ (\tau \cup \theta) \models \varphi.$$

The variables in Y are called *auxiliary* variables, whereas those in X are called *base* variables.

If we restrict ourselves to encodings of $\mathsf{AtMostOne}(x_1, \dots, x_n)$ without auxiliary variables (i.e., $Y = \varnothing$), the only reasonable encoding is

$$\bigwedge_{1 \leq i < j \leq n} (\overline{x_i} \vee \overline{x_j}). \tag{1}$$

Since this encoding uses $\Omega(n^2)$ clauses, it is highly undesirable for large values of n. Fortunately, we can do much better by using auxiliary variables. The first encodings of linear size for $\mathsf{AtMostOne}$ were proposed independently by several authors [War98, Sin05, AM05, GN04] and relied on introducing auxiliary variables y_i for $1 \leq i \leq n-1$ that intuitively mean "some variable in $\{x_1, \dots, x_i\}$ is true". The encoding then consists of the following $3n - 4$ clauses:

$$(\overline{x_i} \vee y_i), \text{ for } 1 \leq i \leq n-1 \tag{2}$$
$$(\overline{y_i} \vee y_{i+1}), \text{ for } 1 \leq i \leq n-2 \tag{3}$$
$$(\overline{y_i} \vee \overline{x_{i+1}}), \text{ for } 1 \leq i \leq n-1. \tag{4}$$

[2] An unambiguous definition is to partition the $n \times n$ grid into n isomorphic subgrids, and call each of them a *box*; there is only one such partition.

To verify that (2)–(4) are indeed an encoding for AtMostOne, we must check both that setting $x_i = \top$ and $x_j = \top$ for $i < j$ leads to a contradiction (i.e., no assignment to the y variables can satisfy all clauses), and conversely, that if at most one x variable is true then there is indeed an assignment for the y variables that satisfies the clauses. For this encoding, both obligations are simple: if $x_i = \top$ and $x_j = \top$ for $i < j$, then by (2) we derive $y_i = \top$, and by successively considering (3) we have the chain of implications $y_i \to y_{i+1} \to \cdots \to y_{j-1}$, which leads to $y_{j-1} = \top$, and together with $x_j = \top$ contradicts (4). Conversely, if at most one x variable is true, namely x_i, then it suffices to assign $y_j = \top$ for $j \geq i$ and check that all clauses are satisfied.

This encoding, called the *sequential encoding*, not only uses fewer clauses than (1), called the *pairwise* encoding, but it also preserves a nice "propagation" property: upon setting any x_i variable to true we deduce that all other x_j variables must be false by an efficient process called *unit propagation*. To define this process formally we will need some notation. A *partial assignment* τ for φ is a partial function from the variables of φ to $\{\bot, \top\}$. We then define $\varphi_{\lfloor\tau}$ as the formula obtained by eliminating from φ all clauses already satisfied by τ, and from the remaining clauses removing all literals that are falsified by τ.

Example 1. Let $\varphi := (x_1 \vee \overline{x_2}) \wedge (x_1 \vee x_2 \vee x_3)$, and $\tau := \{x_1 \mapsto \bot, x_2 \mapsto \bot\}$. Then $\varphi_{\lfloor\tau} = (x_3)$.

A clause containing a single literal, as in the end of the previous example, is called a *unit clause*. Naturally, when encountering a unit clause (ℓ), the only option towards satisfaction is to extend the partial assignment one currently has by assigning the variable of literal ℓ to its prescribed value. In Example 1, this amounts to assigning $x_3 \mapsto \top$. With this notation, unit propagation is the following procedure:

Algorithm 1 Unit propagation

```
1: while φ|τ contains a unit clause (ℓ) do
2:     extend τ by setting the variable of ℓ so that ℓ becomes true
```

Whenever line 2 is executed, we say that the literal ℓ is derived by unit propagation from φ, and denote this by $\varphi \vdash_1 \ell$. Similarly, if φ ends up with an empty clause as a result of this process, and is thus unsatisfiable, we write $\varphi \vdash_1 \bot$. Unit propagation is one of the key subroutines of modern SAT solvers and thus highly optimized for efficiency (more on this later!). As a consequence, if a partial assignment τ semantically entails a literal ℓ in a given formula φ, it is desirable that $\varphi_{\lfloor\tau} \vdash_1 \ell$, since then solvers will efficiently derive ℓ. This desirable property motivates the definition of *propagation completeness*, introduced formally by Bordeaux and Marques-Silva [BMS12]:

Definition 2 (Propagation Completeness [KSV19])**.** Let $f\colon \{\bot, \top\}^n \to \{\bot, \top\}$ be a boolean function and (φ, X, Y) be a CNF encoding for f. We say that φ is *propagation complete* if for every partial assignment τ of X, and literal ℓ (over variables in X) unassigned by τ, we have

$$\varphi_{\lfloor\tau} \models \ell \implies (\varphi_{\lfloor\tau} \vdash_1 \ell \text{ or } \varphi_{\lfloor\tau} \vdash_1 \bot)\,.$$

We thus say, for instance, that both the pairwise encoding and the sequential encoding are propagation-complete encodings for AtMostOne.

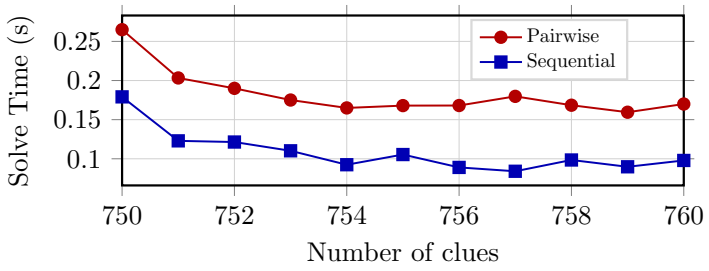
Okay, but can we do better?

Do better at what, exactly? Well, ultimately we want to solve Sudoku puzzles faster, but this doesn't tell us much about what encoding to choose until we have decent proxy metrics to judge how different encodings will perform. Thus far, three natural metrics arise: the number of clauses, the

number of auxiliary variables, and whether our encodings are propagation complete or not. Let us first focus on the number of clauses, assuming for now the intuitive idea that fewer clauses can lead us to faster solving—we will see that while not always the case, this remains a decent rule of thumb. With the above formulation for encoding Sudokus, the number of clauses is given by the sum of:

1. n^2, the number of clauses to enforce that each cell contains some number,
2. $3n^2 \cdot \mathsf{AtMostOne}(n)$, where $\mathsf{AtMostOne}(n)$ denotes the number of clauses we use to encode for each $\mathsf{AtMostOne}$ constraint over n variables,
3. T, the number of clues in the puzzle.

We can immediately choose to ignore the T term since it counts unit clauses that will be immediately eliminated by unit propagation—in fact, increasing T will lead to smaller encodings, not larger, since more clues serve to eliminate other clauses by unit propagation. With the pairwise encoding for $\mathsf{AtMostOne}$, the total number of clauses is $\sim \frac{3}{2}n^4$, whereas with the sequential encoding we get $\sim 9n^3$ clauses. In practice, using the sequential encoding is a clear improvement for a wide range of parameters. For instance, for $n = 36$, and $T \in \{750, \ldots, 760\}$ uniformly random clues (i.e., $\approx 57\%$ of the cells are given as clues), we get the following data:



Encoding	Clauses	Variables
Pairwise	2.4 M	46 K
Sequential	406 K	182 K

While *"let's reduce the number of clauses even further!"* is a tempting takeaway from the previous data, we will shortly see that things are more complicated. Nonetheless, it is worth following that path anyways. It turns out that encodings for $\mathsf{AtMostOne}$ can get much smaller than the sequential encoding. Before our work, the smallest known encoding was Chen's product encoding [Che10], which uses $2n + 4\sqrt{n} + O(\sqrt[4]{n})$ clauses and $2\sqrt{n} + O(\sqrt[4]{n})$ auxiliary variables while also being propagation complete.

Chen's product encoding for $\mathsf{AtMostOne}(x_1, \ldots, x_n)$ is recursive, and usually presented in terms of a grid, as we show next. The base case of the recursion is when $n \leq 4$ (or some other constant), where the pairwise encoding is used. For $n > 4$, place the n variables $x_1, \ldots, x_n$ into a $p \times q$ grid with $p \cdot q \geq n$, as illustrated in Figure 2. We can assume without affecting the asymptotic analysis that $p = q = \lceil \sqrt{n} \rceil$. For ease of notation, rename the variables so that $x_{i,j}$ is the variable placed on the intersection of row i and column j. We denote by E the set of indices (i, j) such that variable $x_{i,j}$ exists.

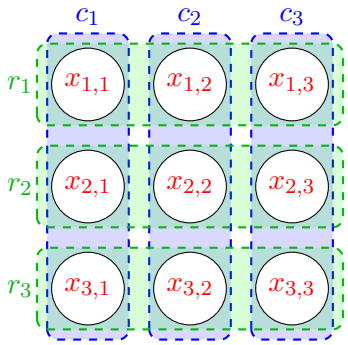


Figure 2: Grid interpretation.

Then, create auxiliary variables $r_1, \ldots, r_p$ and $c_1, \ldots, c_q$, where intuitively r_i will represent that some variable in row i is true, and c_j that some variable in column j is true. We enforce this by adding, for each variable $x_{i,j}$, clauses $(\overline{x_{i,j}} \vee r_i)$ and $(\overline{x_{i,j}} \vee c_j)$, resulting in a total of

$2n$ clauses. Finally, we add constraints $\mathsf{AtMostOne}(r_1, \ldots, r_p)$ and $\mathsf{AtMostOne}(c_1, \ldots, c_q)$, encoded recursively with this same encoding. The resulting formula is then of the form:

$$\mathsf{PE}(\{x_{i,j} \mid (i,j) \in E\}) \equiv \left(\bigwedge_{(i,j)\in E} (\overline{x_{i,j}} \vee r_i) \wedge (\overline{x_{i,j}} \vee c_j) \right) \wedge \mathsf{PE}(r_1, \ldots, r_p) \wedge \mathsf{PE}(c_1, \ldots, c_q). \quad (5)$$

Intuitively, this encoding enforces that at most one cell of a grid is *active* by enforcing that at most one row is *active* and at most one column is *active*. More formally, correctness can be argued by contradiction: if we suppose a satisfying assignment with variables x_{i_1,j_1} and x_{i_2,j_2} assigned to $\top$ (true), then every variable in $S := \{r_{i_1}, r_{i_2}, c_{j_1}, c_{j_2}\}$ must be assigned to $\top$ as well, and since $(i_1, j_1) \neq (i_2, j_2)$, we have $|S| \geq 3$, from where the pigeonhole principle implies there must be either 2 row variables or 2 column variables assigned to $\top$, contradicting the encoding.

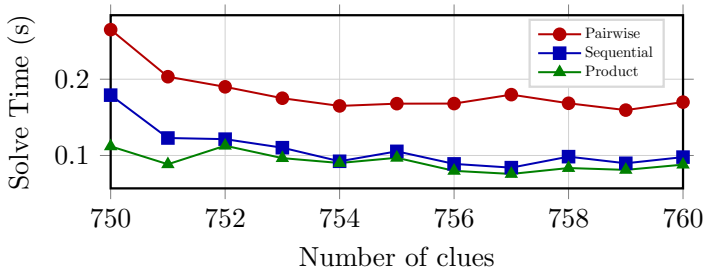
Now, to analyze its size, let $C_{\mathsf{PE}}(n)$ be the number of clauses used by the product encoding for $\mathsf{AtMostOne}(x_1, \ldots, x_n)$, and $A_{\mathsf{PE}}(n)$ the number of auxiliary variables used. Then, from Equation (5), we have:

$$C_{\mathsf{PE}}(n) = \begin{cases} \binom{n}{2} & \text{if } n \leq 4 \\ 2n + 2C_{\mathsf{PE}}(\lceil \sqrt{n} \rceil) & \text{otherwise} \end{cases} \quad \text{and} \quad A_{\mathsf{PE}}(n) = \begin{cases} 0 & \text{if } n \leq 4 \\ 2\lceil \sqrt{n} \rceil + 2A_{\mathsf{PE}}(\lceil \sqrt{n} \rceil) & \text{otherwise.} \end{cases}$$

A simple inductive argument then yields the following.

Proposition 3 ([Che10]). *$C_{PE}(n) = 2n + 4\sqrt{n} + o(\sqrt{n})$ and $A_{PE}(n) = 2\sqrt{n} + o(\sqrt{n})$.*

So, do we get even faster running times using this encoding? Sadly, only by a small margin:



Encoding	Clauses	Variables
Pairwise	2.4 M	46 k
Sequential	406 k	182 k
Product	391 k	132 k

The Art of Encodings. To improve more substantially, we will need the *art*: the application of problem-specific ideas that are not yet captured by a principled framework. In this case, we can use a variation of Chen's product encoding tailored specifically to the geometry of Sudoku's constraints. The main idea is that, when we use Chen's product encoding on a $\sqrt{n} \times \sqrt{n}$ box, our r_i and c_j auxiliary variables can be repurposed to encode the row and column constraints intersecting said box. More concretely, for each number $t \in [n]$, we introduce auxiliary variables $h_{r,b,t}$ for $1 \leq r \leq n$ and $1 \leq b \leq \sqrt{n}$, which will represent that t appears in row r between columns $(b-1)\sqrt{n}+1$ and $b\sqrt{n}$; that is, in the intersection between row r and the b-th box intersecting that row. Symmetrically, we introduce auxiliary variables $v_{b,c,t}$ for $1 \leq b \leq \sqrt{n}$ and $1 \leq c \leq n$, which will represent that t appears in column c between rows $(b-1)\sqrt{n}+1$ and $b\sqrt{n}$. Figure 3 illustrates these auxiliary variables. Now, in order to turn this into an actual encoding which we shall call the *geometric* encoding, we use the following constraints:

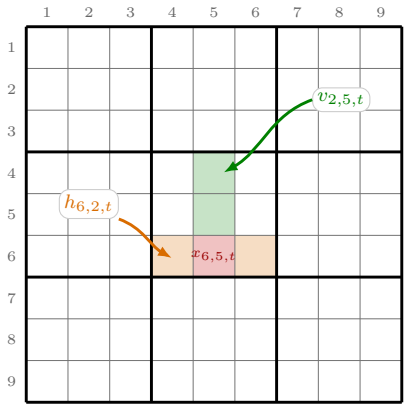


Figure 3: Geometric encoding.

1. Each cell (i, j) must contain some number: $\bigvee_{t=1}^{n} x_{i,j,t}$.

2. Each cell implies its corresponding *segment* variables: $(\overline{x_{r,c,t}} \vee h_{r,\lceil c/\sqrt{n}\rceil,t}) \wedge (\overline{x_{r,c,t}} \vee v_{\lceil r/\sqrt{n}\rceil,c,t})$ for all $1 \leq r, c \leq n$ and $1 \leq t \leq n$.

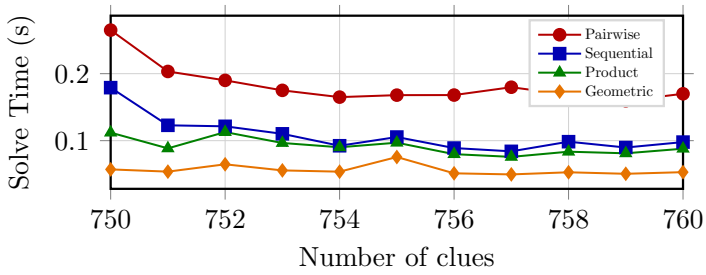
3. For each row, column, and box, at most one segment can take number t (for every $t \in [n]$):

$$\bigwedge_{r=1}^{n} \mathsf{AtMostOne}(\{h_{r,b,t} \mid 1 \leq b \leq \sqrt{n}\}) \wedge \bigwedge_{c=1}^{n} \mathsf{AtMostOne}(\{v_{b,c,t} \mid 1 \leq b \leq \sqrt{n}\})$$
$$\wedge \bigwedge_{b=1}^{\sqrt{n}} \bigwedge_{s=1}^{\sqrt{n}} \mathsf{AtMostOne}(\{h_{r,b,t} \mid (s-1)\sqrt{n}+1 \leq r \leq s\sqrt{n}\})$$
$$\wedge \bigwedge_{b=1}^{\sqrt{n}} \bigwedge_{s=1}^{\sqrt{n}} \mathsf{AtMostOne}(\{v_{b,c,t} \mid (s-1)\sqrt{n}+1 \leq c \leq s\sqrt{n}\}).$$

Adding these together, the number of clauses used by this encoding is:

$$n^2 + 2n^3 + 2n^2\mathsf{AtMostOne}(\sqrt{n}) + 2n^2\mathsf{AtMostOne}(\sqrt{n}) \sim 2n^3.$$

We now observe the following data:



Encoding	Clauses	Variables
Pairwise	2.4 M	46 k
Sequential	406 k	182 k
Product	391 k	132 k
Geometric	168 k	88 k

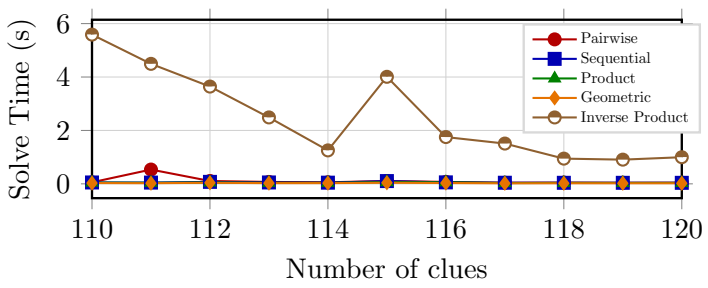
The danger of metrics. It turns out that a different formulation of the Sudoku problem leads to a more compact encoding, but has terrible performance in practice. The alternative formulation is simply:

1. Each cell contains at most one number: $\bigwedge_{i,j} \mathsf{AtMostOne}(x_{i,j,1}, \ldots, x_{i,j,n})$.

2. Each row, column, and box contains each number at least once:

$$\bigwedge_{r=1}^{n}\bigwedge_{t=1}^{n}\left(\bigvee_{c=1}^{n} x_{r,c,t}\right) \wedge \bigwedge_{c=1}^{n}\bigwedge_{t=1}^{n}\left(\bigvee_{r=1}^{n} x_{r,c,t}\right) \wedge \bigwedge_{b=1}^{\sqrt{n}}\bigwedge_{s=1}^{\sqrt{n}}\bigwedge_{t=1}^{n}\left(\bigvee_{r=(b-1)\sqrt{n}+1}^{b\sqrt{n}}\bigvee_{c=(s-1)\sqrt{n}+1}^{s\sqrt{n}} x_{r,c,t}\right).$$

This encoding, which we shall call the *inverse* encoding, also has $\sim 2n^3$ clauses, but with a smaller second order term if we use the product encoding for $\mathsf{AtMostOne}$. More importantly, the inverse encoding is *so* bad in practice that we cannot even benchmark it on the previous family of instances without spending days! Here is the data for the, much smaller, $n = 16$:



Encoding	Clauses	Variables
Pairwise	92 k	4 k
Sequential	34 k	15 k
Product	34 k	10 k
Geometric	16 k	9 k
Inverse	12 k	6 k

Why is the inverse encoding so bad? I invite the reader to take a few moments to ponder this question.

While it may appear simple at first, I believe this example serves to motivate the desire for rigorous explainability methods (PP. 1). As an experiment, I asked three frontier LLMs (ChatGPT 5.4 Pro, Claude 4.6 Sonnet, Gemini 3.1 Pro) to attempt their best explanation, and while all of them gestured at it being a matter of "propagation", none was able to produce a falsifiable hypothesis, or a clean mathematical model that would predict these results. More generally, I posit that the SAT community continuously faces an epistemic problem that is also shared by the machine learning community: we are prone to make plausible hypotheses based on vague intuitions, but lack tools to rigorously test them. A priori, we can think of SAT solvers as "transparent" (i.e., state-of-the-art and competition solvers are open source), but I claim that until we develop tooling to nicely extract, visualize, and causally intervene on their executions, we will keep operating as if they were black-boxes that we only interface with through high-level intuitions.

Let me give an example from my work on explainable AI regarding how even decision trees, often presented as the gold standard of *interpretable* models, can offer interesting explainability problems. Consider a decision tree for judging loan applications, and an applicant Jane Doe who gets *approved* by the model. As depicted in Figure 4, we can have full information of the decision path taken by the model that led to the approval outcome. However, did Jane Doe get approved *because* of her prior fraud convictions or *despite* her prior fraud convictions? Based on our human intuition we would think of the latter, but actually validating this hypothesis requires an argument based on the rest of the tree. For instance, we can observe that changing that feature in Jane's application still leads to the approval outcome, and moreover that in fact any applicant with: a *good credit score*, *many assets*, and a *clean history of payments* will get approved by the model. As we argue in [ABK+25], this makes the computation of rigorous explanations a challenging problem even for decision trees. On the positive side, however, in that same work we showed how SAT solvers could be used to compute explanations in practice, and more generally, there has been significant work in the SAT community regarding practical proof minimization (see e.g., [WHH14]). I claim that this *"because/despite/irrelevant"* problem arises naturally when understanding the performance of encodings (e.g., does the geometric encoding perform better *because* it has fewer clauses, or is that irrelevant and the speed-up is due to some other metric?), and that minimization techniques can be useful for dealing with it.

Now, going back to the inverse encoding, I will give my best *attempt* of an explanation. In the direct formulation, assigning a cell (i, j) to number t propagates via the AtMostOne constraint (assuming propagation completeness) into discarding t as potential value for all other cells in the row, column, and box of (i, j). On the other hand, in the inverse formulation, the propagation simply discards all other values t' for the same cell (i, j). With the former kind, assigning $x_{i,j,t} \mapsto \top$ directly assigns to false up to $3n - 2\sqrt{n} - 1$ other base variables, whereas the latter form only assigns up to $n - 1$. However, my best guess is that more so than this number being the main factor at play,

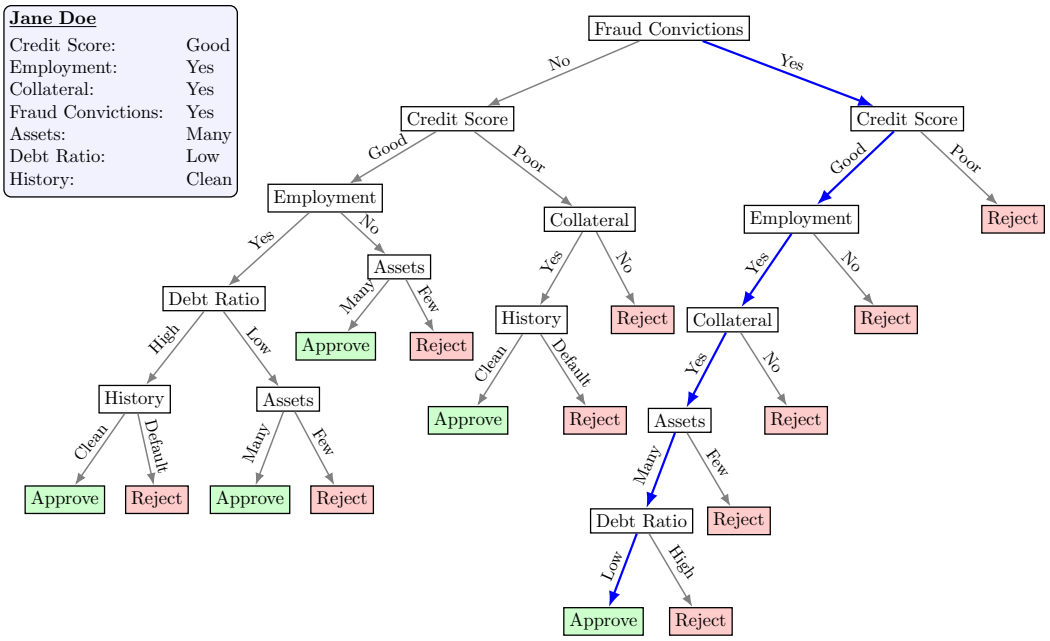


Figure 4: Example of a decision tree for judging loan applicants with binary features. The decision path for a concrete applicant, Jane Doe, is highlighted in blue.

the key difference is that with the direct formulation at intermediate states of the search the solver will hold partial assignments that do not contain multiple occurrences of a number in a row, column, or box, while *also* not containing multiple occurrences of a value in any cell since this doesn't really help with satisfying other clauses. The inverse formulation however, can maintain throughout large periods of the search a partial assignment that contains multiple occurrences of a value in different cells of a given column (or row/box), which will be wasted time by backtracking until the solver learns more explicit clauses implying the column-wise AtMostOne constraint. Wouldn't it be nice to have tooling that allowed us to make such an explanation more rigorous, falsifiable, and grounded in data?

Proposed Project 1. I propose to develop a framework to better understand the impact of encodings, as well as solver parameters, on the practical performance of state-of-the-art solvers. As a first step, I would like to modify the CaDiCaL solver to expose via an API a large amount of data concerning the solver's internal state without too much overhead. Then, the plan is to develop a visualization tool for easier interpretability of the solver's behavior based on such data. After this, we would use methodologies and techniques borrowed from the interpretable AI literature to compute higher-level abstractions from the data that are suitable for direct human interpretation. A concrete interpretability technique I am interested in applying is that of *causal interventions* [GIZ+25], which in the context of SAT solvers could be performed by triggering specific sequences of decision variables through CaDiCaL's existing API and studying their effects on different encodings.

Automated (Re)encodings

We have seen already in this Sudoku example that encodings matter in practice. However, designing effective encodings is still a difficult task that requires combinatorial expertise. This can limit the

adoption of SAT solvers both in academia and in industry; in my domain of interest, SAT for mathematics, I have seen several times mathematicians attempt SAT-solving on a problem, and give up prematurely due to poor results that can be attributed to suboptimal encodings (which I know since I was then able to obtain much better results myself). This motivates the design of *automated reencoding* tools that can take a CNF formula, and produce a better encoding for it by analyzing its structure. The best example is that of *Bounded Variable Addition* (BVA) [MHB12], a powerful preprocessing technique which introduces convenient auxiliary variables in order to minimize the number of clauses. A successor of BVA, called SBVA, was the key ingredient for the winner solver of the 2023 SAT Competition [BHI+23], and factor, a new implementation by Armin Biere, has been incorporated into state-of-the-art solvers CaDiCaL and Kissat. In our work submitted to SAT 2026 [PSH26], we made substantial progress in understanding the effectiveness of BVA from a theoretical perspective, and leveraged algorithms for biclique partitions [KPSMS25] to develop a new prototype implementation, BiVA, that significantly outperforms other implementations on random dense formulas.

Proposed Project 2. I propose to further develop automated reencoding tools, by incorporating *structural* information of the instance formulas (as in [HGH23]) to improve performance on non-random formulas. Furthermore, I think it is important to develop tooling for exposing to the user what the introduced auxiliary variables represent, since that can allow for designing better encodings in the first place. For instance, the encoding that finally allowed us to solve the 20-year-old problem on the *packing chromatic number* of $\mathbb{Z}^2$ [SH23b] was inspired by reverse-engineering BVA variables and then manually designing them in a more structured manner. On the more theoretical side, I would like to understand better the reencoding capabilities of current implementations of BVA on random graphs; this comes down to a concrete, but hard, purely graph-theoretic question. The reason this is interesting is that current implementations of BVA are based on greedy heuristics, that turn out to perform remarkably well, so I see this question as a concrete angle of attack towards understanding the success of modern solvers by isolating one component at a time. Separately, we have started to study the power of BVA from a proof-complexity perspective, and have partial results showing that it leads to exponentially shorter proofs than resolution in many formulas. However, our current proof relies on a form of BVA which is not performed by our BiVA tool. I would like to understand whether this gap is essential, and whether it can guide us to design even better reencoding tools.

Back to the Theory

We have shown, through the bad performance of the inverse encoding,[3] that clause-minimization is not to be followed as a unilateral goal, but rather as a guiding light towards uncovering better encodings. In fact, several encodings that have allowed us to get speedups of multiple orders of magnitude on combinatorial problems have arisen from thinking about reducing the number of clauses. I hope this example, without requiring more specific mathematical background, has illustrated some of what the "applied" side of my research looks like.

[3]The impact of these "flipped" formulations, which I have observed in other problems as well, always reminds me of the following old joke: Two monks at a monastery like to smoke. The first one goes to his superior and asks: "Father, would it be okay if I smoke while praying?" to which the superior replies "No, that would be highly disrespectful". The second monk goes to the same superior and asks: "Father, would it be okay if I pray while smoking?", to which the superior replies "Of course, any opportunity for prayer is good".

Now, I will discuss some more theoretical ideas, to convey as well that aspect and show it can be impactful for the theory community. Let us consider now more directly the question of what the minimum number of clauses to encode a Sudoku puzzle is, and we will see that we can do much better in this metric than what I have shown thus far. First, let me take this opportunity to showcase a recent result we proved. Our smallest encoding thus far, the inverse one, still includes AtMostOne constraints, and thus smaller encodings for AtMostOne will lead to a smaller encoding overall. In his 2010 paper, Chen conjectured that his product encoding used the fewest number of clauses, and we recently managed to refute this conjecture. Our improved encoding is actually quite simple, but it requires a change of perspective on Chen's encoding.

First, as a conceptual move, instead of thinking of our base variables as cells on a grid, we identify them with the edges of a complete bipartite graph as illustrated in Figure 5. We say that an edge is *selected* if the corresponding input variable is true, and we say that a vertex is *selected* if it is incident to a selected edge. Now, the key idea of the product encoding can be rephrased as follows: at most one edge is selected if and only if (a) at most one vertex in the left part is selected and (b) at most one vertex in the right part is selected. Of course, the grid interpretation and bipartite interpretation are equivalent, but the bipartite interpretation provides a nice conceptual lens for designing new encodings for AtMostOne.

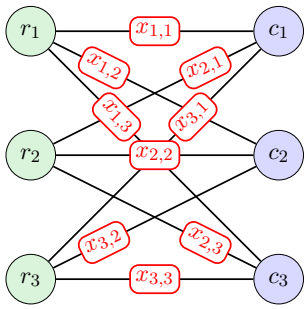


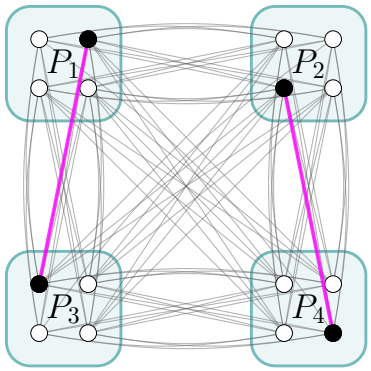
Figure 5: Bipartite interpretation.

Given a graph G, our goal is to encode AtMostOne$(E(G))$, the constraint asserting that at most one edge is selected. Let the input variables be $x_{\{i,j\}}$ for each $\{i,j\} \in E(G)$. As in the product encoding, we introduce an auxiliary variable v_i for each vertex $i \in V(G)$, and we spend $2|E|$ clauses to make each edge imply its endpoints: $\overline{x_{i,j}} \vee v_i$ and $\overline{x_{i,j}} \vee v_j$. Then, it remains to use the auxiliary variables associated with the vertices to encode that at most one edge is selected; how we do this depends on the choice of G. The efficiency of the encoding (i.e., how many clauses it has as a function of $|E|$) depends on the edge density of the graph and how succinctly we can use the vertex variables to encode that at most one edge is selected. As we will show next, a suitable multipartite graph allows us to prove the following theorem, which improves on the second order term from the product encoding.

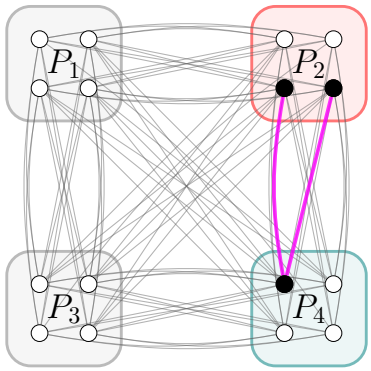
Theorem 4 (Multipartite Encoding)**.** *There is a propagation-complete encoding of the* AtMostOne$(x_1, \ldots, x_n)$ *constraint using* $2n + 2\sqrt{2n} + O(\sqrt[3]{n})$ *clauses and* $\sqrt{2n} + O(\sqrt[3]{n})$ *auxiliary variables.*

The multipartite encoding. We now describe an encoding for AtMostOne$(x_1, \ldots, x_n)$ with $2n + 2\sqrt{2n} + O(\sqrt[3]{n})$ clauses and $\sqrt{2n} + O(\sqrt[3]{n})$ auxiliary variables, thus proving Theorem 4. We use the graph-theoretic strategy just described, taking G to be a complete multipartite graph with $\Theta(\sqrt[6]{n})$ parts and $\Theta(\sqrt[3]{n})$ vertices within each part; we therefore call our encoding the *multipartite encoding*. The constants are chosen so that G has $\sim\sqrt{2n}$ vertices and $\sim n$ edges. These parameters turn out to be a good choice for two reasons. First, G has a high edge density, which allows us to assign more input variables to the edges of G. Second, we have a succinct way to use the vertex variables to encode that at most one edge is selected:

Key insight: At most one edge is selected if and only if (a) at most one vertex from each part is selected and (b) at most two parts contain a selected vertex (see Figure 6).



(a) AtMostTwo-parts constraint is violated.



(b) $\mathsf{AtMostOne}'(\_; z_2)$-vertex constraint is violated.

Figure 6: Illustration of the multipartite encoding. Parts violating the AtMostOne-vertex constraint are shaded red. Parts for which z_k is true are shaded teal.

To materialize this encoding, we first require an intermediate construction, which is a simple modification of Chen's product encoding. Let $\mathsf{AtMostOne}'(x_1, \ldots, x_n; z)$ be the constraint asserting that at most one of the variables $x_1, \ldots, x_n$ is true and that x_i implies z for each $i \in [n]$; that is $\mathsf{AtMostOne}'(x_1, \ldots, x_n; z) \iff \mathsf{AtMostOne}(x_1, \ldots, x_n) \wedge \bigwedge_{i \in [n]} (\overline{x_i} \vee z)$.

Lemma 5. *There is a propagation-complete encoding of the* $\mathsf{AtMostOne}'(x_1, \ldots, x_n; z)$ *constraint using* $2n + O(\sqrt{n})$ *clauses and* $O(\sqrt{n})$ *auxiliary variables.*

Proof sketch. It suffices to use Chen's product encoding and add one clause $(\overline{r_i} \vee z)$ per row r_i. Then, each input variable implies z indirectly through its row. □

Now we can sketch the proof of Theorem 4.

Proof sketch of Theorem 4. Let G be the complete p-partite graph with q vertices within each part for $p := \lceil \sqrt[6]{n} \rceil + 1$ and $q := \lceil \sqrt{2} \cdot \sqrt[3]{n} \rceil$. This way, $|E(G)| = \binom{p}{2} q^2 \geq n$. Let $P_1, \ldots, P_p$ be the parts of G. Assign the variables $x_1, \ldots, x_n$ to distinct edges of G, renaming the variables so that $x_{\{i,j\}}$ is the variable assigned to the edge $\{i, j\}$. Let E be the set of edges of G to which some variable is assigned. Discard any vertices of G not incident to an edge from E. Introduce auxiliary variables v_i for each $i \in V(G)$ and z_k for each $k \in [p]$. Our encoding is as follows:

$$\mathsf{ME}(\{x_{\{i,j\}} \mid \{i,j\} \in E\}) := \left(\bigwedge_{\{i,j\} \in E} (\overline{x_{\{i,j\}}} \vee v_i) \wedge (\overline{x_{\{i,j\}}} \vee v_j) \right) \wedge \left(\bigwedge_{k \in [p]} \mathsf{AtMostOne}'(\{v_i \mid i \in P_k\}; z_k) \right) \wedge \mathsf{AtMost}_2(z_1, \ldots, z_p),$$

where we use the *generalized product encoding* for $\mathsf{AtMost}_2(z_1, \ldots, z_p)$ (the boolean function that returns $\top$ if at most two of its inputs are $\top$) which is propagation complete and uses $3p + O(p^{2/3})$ clauses and $O(p^{2/3})$ auxiliary variables [FG10]. Both correctness and propagation completeness are straightforward to show.

Finally, the number of clauses is $2n + p \cdot (2q + O(\sqrt{q})) + (3p + O(p^{2/3})) = 2n + 2\sqrt{2n} + O(\sqrt[3]{n})$, and the number of auxiliary variables is $pq + p \cdot O(\sqrt{q}) + O(p^{2/3}) = \sqrt{2n} + O(\sqrt[3]{n})$. □

The previous encoding is interesting for several reasons. First, it leverages width-3 clauses to get a smaller encoding for AtMostOne which is a 2-CNF function. More importantly, our encoding has direct consequences to circuit complexity. As we discuss in [KPS26], the same construction can be adapted to obtain a circuit for the *threshold-2* function T_2 (the complement of AtMostOne) with $2n+\sqrt{2n}+O(\sqrt[3]{n})$ gates, which improves on the prior best construction of Adleman (see [Blo79]). Sergeev [Ser20] proved that every monotone boolean circuit for $\mathsf{T}_2(x_1,\ldots,x_n)$ has at least $2n+\sqrt{(2n-4)/3}-19/6$ gates, so our improved circuit is almost optimal. Furthermore, there is an even more striking consequence of our result I discuss next. A monotone boolean circuit is said to be *single level* if every path from an input variable to the output gate goes through at most one $\wedge$ gate. Sergeev showed that every *single-level* monotone boolean circuit for $\mathsf{T}_2(x_1,\ldots,x_n)$ has at least $2n+2\sqrt{n+11}-10$ gates.[4] Thus, Adleman's construction is essentially optimal for single-level circuits, and a corollary of our improved circuit is that the smallest monotone boolean circuits for $\mathsf{T}_2(x_1,\ldots,x_n)$ cannot be single level. This answers a 47-year-old open question from Bloniarz [Blo79, p. 158], and it should be contrasted with a result of Krichevskii [Kri64] stating that single-level monotone boolean *formulas* are optimal for $\mathsf{T}_2(x_1,\ldots,x_n)$. It was a long-standing open problem whether there exists a quadratic boolean function (i.e., a disjunction of cubes of the form $x_i \wedge x_j$) whose single-level monotone circuit complexity is strictly greater than its monotone circuit complexity; the negation of this statement was sometimes called the *single-level conjecture*. The problem appears to originate with Bloniarz [Blo79, p. 158] and was further studied by Lenz and Wegener [LW91] and several other authors (see, e.g., [Bub86, MS92, AM06]). The conjecture was finally disproved by Jukna [Juk06] using a carefully constructed quadratic boolean function. Thus, our results show that, surprisingly, the conjecture already fails for $\mathsf{T}_2(x_1,\ldots,x_n)$, the simplest quadratic boolean function of them all.

Closing on Sudokus. So where does this leave us in terms of the minimum number of clauses to encode Sudoku puzzles? Still at $\Theta(n^3)$. It turns out that we can get an encoding with only $O(n^2\lg^2 n)$ clauses by resorting to heavier machinery, and using base variables $x_{i,j,b}$ with $b\in[\lg n+1]$ stating that the b-th bit of cell (i,j) is 1. The main insight was presented at the very beginning of this proposal, when I explained the rules of Sudokus by saying that numbers in each row, column, and box should be a *permutation* of $\{1,\ldots,n\}$. Now, how do we check if a sequence of integers is a permutation of $\{1,\ldots,n\}$? A standard idea is to *sort* such permutation and then simply compare the result to the sequence $(1,2,3,\ldots,n)$. To this effect, the celebrated AKS sorting network of Ajtai, Komlós, and Szemerédi [AKS83] provides a circuit that sorts n input gates into n output gates with only $O(n\lg n)$ gates. Such a circuit yields, through the Tseitin transformation, a CNF encoding with $O(n\lg^2 n)$ clauses that "sorts" n numbers represented in binary by $O(\lg n)$ base variables each, into $O(n\lg n)$ auxiliary variables representing the n integers in ascending order. We can thus obtain an encoding for Sudoku that uses one sorting network for each row, column, and box, and then use an $O(n\lg n)$ "comparison" for each sorting network to enforce its output coincides with $(1,\ldots,n)$. I will spare the details, but it should be clear that since there are $O(n)$ rows/columns/boxes, this strategy yields an encoding with $O(n^2\lg^2 n)$ clauses overall. While the AKS sorting network has constant factors that are too large for practical purposes, more practical sorting networks such as Batcher's [Bat68] are known, and useful for practical SAT encodings at the cost of adding an additional $\lg n$ factor to the number of clauses [ES06]. More refined sorting networks and *comparison networks* for SAT encodings have also been developed since [ANORC11, Kar19], and thus the presented encoding for Sudoku could perhaps be practical for very large values of n.

[4] In [Ser20], the bound is stated as $2n+2\sqrt{n+27}-14$. Sergeev told us that the proof contains a mistake that, when corrected, yields the (improved) bound $2n+2\sqrt{n+11}-10$.

Encoding Size and Propagation Completeness

Now that we are done talking about Sudokus, and several ideas about the art of encodings have been presented, I would like to consider again the more general problem of encoding an arbitrary boolean function f with as few clauses as possible. I view this as a central question of my thesis. I will first focus on truly arbitrary boolean functions, and then the concrete case of the T_2 function, which in the SAT literature is named $\mathsf{AtLeast}_2$. Both of these questions will naturally lead us to the proposed projects 3 and 4.

First, for some contrast, it is a well-known result of Shannon [Sha49] that every boolean function on n inputs can be computed by a circuit with $O\left(\frac{2^n}{n}\right)$ gates, and that this bound is information-theoretically tight. In comparison, we have:

Theorem 6. *For a boolean function f, let us denote by $\mathsf{enc}(f)$ the minimum number of clauses in a CNF encoding of f, regardless of the auxiliary variables used, and let $\mathsf{enc}(n)$ be the maximum of $\mathsf{enc}(f)$ over all functions f on n bits. Then, $\mathsf{enc}(n) = \Theta(\sqrt{2^n})$.*

The proof of the lower bound is also information theoretic, but it crucially relies on the following fact:

Proposition 7. *If a boolean function f can be encoded with m clauses, then it can be encoded with m clauses while not using more than $m-1$ auxiliary variables.*

Proposition 7 has a nice graph-theoretic proof, very similar in spirit to that of Aharoni and Linial for minimal unsatisfiable formulas [AL86]. We can easily prove the lower bound with it: since we can assume at most $n+m-1$ variables for encodings with at most $m > n$ clauses, there are at most $2(n+m)$ literals, and thus at most $2^{2(n+m)}$ possible clauses. Then, the number of ways of selecting at most m of these clauses is at most

$$\sum_{i=1}^{m} \binom{2^{2(n+m)}}{m} \leq \binom{2^{2(n+m)}}{m+1} \leq \left(\frac{e \cdot 2^{2(n+m)}}{m+1}\right)^{m+1} \leq 2^{2(n+m)(m+1)}.$$

But there are 2^{2^n} boolean functions on n inputs, so if there were all encodable with at most m clauses, we would have $2^n \leq 2(n+m)(m+1)$, from where it follows that $m = \Omega(\sqrt{2^n})$. Let us now prove the upper bound.

Proof of Theorem 6 (Upper bound). Consider the set $G := \{\mathbf{x} \in \{\bot, \top\}^n \mid f(\mathbf{x}) = \top\}$ of "good" assignments we want to allow. Assume for simplicity that n is even, and note that then each good assignment $\mathbf{x} \in G$ can be written as the concatenation of a prefix $\mathbf{p} \in \{\bot, \top\}^{n/2}$ and a suffix $\mathbf{s} \in \{\bot, \top\}^{n/2}$. Let P be the set of all such prefixes of good assignments, and S be the set of all such suffixes of good assignments. Next, we will construct auxiliary variables $y_{\mathbf{p}}$, for $\mathbf{p} \in P$, which will imply that the first $n/2$ base variables of an assignment match $\mathbf{p}$, and similarly $z_{\mathbf{s}}$, for $\mathbf{s} \in S$ for the last $n/2$ variables. Then, we will add a wide clause $\bigvee_{\mathbf{p} \in P} y_{\mathbf{p}}$ to state that some good prefix must be matched. Now, if we let $G_{\mathbf{p}} := \{\mathbf{s} \in S \mid (\mathbf{p}, \mathbf{s}) \in G\}$ be the set of suffixes compatible with prefix $\mathbf{p}$, then for every $\mathbf{p} \in P$ we add a clause

$$\overline{y_{\mathbf{p}}} \vee \bigvee_{\mathbf{s} \in G_{\mathbf{p}}} z_{\mathbf{s}}.$$

So far, we have $|P| + 1 = O\big(2^{n/2}\big)$ clauses. It remains to add clauses that enforce the intended semantics of the y and z variables. For this, we start by creating variables $y_\bot$ and $y_\top$, and clauses

for $y_\top \to x_1$, and $y_\bot \to \overline{x_1}$. Then, variables $y_{\bot\bot}, y_{\bot\top}, y_{\top\bot}, y_{\top\top}$ and clauses for $y_{\bot\bot} \to (y_\bot \wedge \overline{x_2})$, $y_{\bot\top} \to (y_\bot \wedge x_2)$, and so on. More generally, for $\mathbf{x} \in \{\bot, \top\}^k$, with $1 \leq k < n/2$, we will have clauses

$$(\overline{y_{\mathbf{x}\bot}} \vee y_{\mathbf{x}}) \wedge (\overline{y_{\mathbf{x}\top}} \vee y_{\mathbf{x}}) \wedge (\overline{y_{\mathbf{x}\bot}} \vee \overline{x_{k+1}}) \wedge (\overline{y_{\mathbf{x}\top}} \vee x_{k+1}),$$

which end up enforcing that indeed $y_{\mathbf{p}}$ implies that the first $n/2$ base variables match $\mathbf{p}$. We proceed analogously for the z variables. To conclude the proof, note that the number of clauses introduced to enforce the desired semantics is at most

$$2\sum_{k=1}^{n/2-1} 4 \cdot 2^k = 8\left(2^{n/2} - 2\right) = O\left(2^{n/2}\right). \qquad \square$$

This encoding, however, is not propagation complete; if variables $x_2, \dots x_n$ have been assigned by a partial assignment τ, and there is no satisfying assignment that extends τ, the previous encoding won't detect this until x_1 is assigned, since otherwise the prefix variables cannot be assigned. Nonetheless, we can do a light modification to obtain propagation completeness, at the cost of using $2^{0.793n}$ clauses instead of $2^{0.5n}$. The idea is now to have variables $y_{\mathbf{t}}$ with $\mathbf{t} \in \{\bot, \top, *\}^{n/2}$, where '$*$' denotes an unassigned variable. For instance, at $n = 12$, we will have a variable

$$y_{*\bot\top\top*\bot}$$

that represents that $x_2 = \bot, x_3 = \top, x_4 = \top, x_6 = \bot$ while the rest are unassigned. The clauses to give them semantics are essentially the same as before, but now for each $y_{\mathbf{t}}$ where $\mathbf{t}$ has k occurrences of $*$, we will have $2k$ clauses regarding the possible extensions of $\mathbf{t}$. Continuing with the previous example, we would have

$$(\overline{y_{*\bot\top\top*\bot}} \vee x_1 \vee y_{\bot\bot\top\top*\bot}) \wedge (\overline{y_{*\bot\top\top*\bot}} \vee \overline{x_1} \vee y_{\top\bot\top\top*\bot}),$$

as well as $n/2 - k$ clauses ensuring that $y_{\mathbf{t}}$ implies every $y_{\mathbf{t}'}$ where $\mathbf{t}'$ has one fewer $*$ than $\mathbf{t}$ and matches $\mathbf{t}$ on all non-$*$ positions. That is,

$$(\overline{y_{*\bot\top\top*\bot}} \vee y_{**\top\top*\bot}) \wedge (\overline{y_{*\bot\top\top*\bot}} \vee y_{*\bot*\top*\bot}) \wedge \dots$$

In general, for each of the $3^{n/2}$ possibilities for $\mathbf{t}$, we will have $n/2+k$ clauses, where k is the number of $*$'s in $\mathbf{t}$, and the same for the z variables. Thus, the total number of clauses is proportional to

$$\sum_{k=0}^{n/2} \binom{n/2}{k} 2^{n/2-k}(n/2+k) \leq n\sum_{k=0}^{n/2} \binom{n/2}{k} 2^{n/2-k} = n\sum_{k=0}^{n/2} \binom{n/2}{k} 2^k = n3^{n/2} = n2^{(n/2)\cdot\lg 3}.$$

Since $\lg 3 \approx 1.58$, this is $O(n2^{0.792n})$, which is $O(2^{0.793n})$. I will spare a proof of propagation completeness, but hopefully it is reasonably clear that it will hold essentially by construction, since our auxiliary variables will carry the precise information of what has been partially assigned.

Interestingly, I do not know how to prove a lower bound close to $O(2^{0.793n})$; information theory does not seem to capture propagation concerns nicely, and thus we will need new tools for this kind of analysis. Being able to prove such lower bounds (or better upper bounds!) would be the ideal outcome for the proposed project 3.

Furthermore, it is worth mentioning that in our work on cardinality constraints [KPS26], we show how non-propagation-complete encodings can actually be practical in certain problems, thus challenging the traditional view around this property, which has been described as "a must" [Kar19, p. 95]. In fact, a better understanding of the empirical impact of propagation properties is tightly related to the proposed project 1 on the usage of explainable AI for analyzing encodings. The more contextualized quote of Karpinski's PhD thesis on "CNF Encodings of Cardinality Constraints Based on Comparator Networks" is:

> "Possibly the biggest mystery we can leave the reader with is: why encoding cardinality constraints using comparator networks is so efficient? It is in fact a very fundamental question. We can only see the empirical evidence as shown in this thesis, as well as other papers. We can compare two networks using various measures like the number of comparators, depth, the number of variables and clauses the encoding generates, or variables to clauses ratio. But all of those measures do not seem to give a conclusive answer to our question, especially if we want to collate our encodings with the ones not based on comparator networks. [Propagation completeness] is important, but even with it we still cannot decisively distinguish between the top used encodings. In fact, being [propagation complete] is a must if an encoding is to be competitive. *It looks like another, more complex propagation property is needed in order to theoretically prove superiority of encodings based on comparator networks. Such property may have not been discovered yet.*"

My emphasis on the last part reflects what I find most interesting from the applied perspective: propagation completeness (and tightly related definitions such as *generalized arc-consistency*) are binary properties that an encoding has or lacks, but it seems that making finer distinctions will be helpful for theoretical work that aims to predict which encodings will perform better in practice. For instance, analyzing the average length of "propagation chains" that lead to a deduction might provide a more fine-grained way of distinguishing between propagation complete encodings. A line of work in this direction that I take as a compelling starting point is the work of Gwynne and Kullmann [GK13] appropriately titled *"Towards a theory of good SAT representations"*, where a hierarchy of propagation completeness, PC_k is defined. The base level PC_0, corresponding to propagation completeness in general, can be defined by requiring that semantic entailments are derived by unit propagation alone without having to branch on any variables, whereas PC_k states that semantic entailments are derived by branching with depth at most k (i.e., k successive correct branching decisions). Similarly, the constraint programming community has also studied different notions of propagation strength [AGMES16], which also sets a precedent for further study of CNF encodings as done by Kučera and Savický [KS22]. More generally, I plan to continue studying the minimum size of encodings for different functions and classes of functions, and the tradeoff between propagation strength and encoding size.

Proposed Project 3. I propose to further study the encoding size measure $\mathsf{enc}(f)$, both for specific functions of interest as well as classes of functions. This study shall also take into account other metrics that can inform better predictions of practical performance, such as the number of auxiliary variables, and different notions of propagation strength (both existing ones and potentially new ones that I aim to discover through empirical evaluations). As mentioned above, even the tradeoff between encoding size and propagation completeness remains unclear for the broadest class of all boolean functions on n bits. We have an unpublished proof of $\mathsf{enc}(f) = o(n^k/\lg n)$ for all functions f that can be expressed in k-CNF, which we expect to polish and publish in the next month. This encoding is not propagation complete, whereas for propagation-complete encodings the best we can do at the moment is an $O(n^k/\lg n)$ upper bound; proving a separation result here would be very interesting. In the same spirit, Kučera and Savický [KS22] proved separation results between two different strengths of propagation: propagation completeness and the weaker *unit-refutation completeness* property. However, their separation applies to the setting without auxiliary variables, and they leave as an important open question whether one can prove a similar separation for general encodings.

In the remaining couple of pages of this proposal, I will show the depth of this question with a concrete function, which will also motivate the proposed project 4.

CNF encodings for the $\mathsf{AtLeast}_2$ (T_2) function.

Recall that $\mathsf{AtLeast}_2(x_1, \ldots, x_n)$ is the boolean function that returns $\top$ whenever at least two of its inputs are $\top$. This function admits a nice and simple encoding with n clauses:

$$\bigwedge_{i=1}^{n} \left(\bigvee_{j \neq i} x_j \right).$$

For instance, for $n = 5$, we would have

$$(x_2 \vee x_3 \vee x_4 \vee x_5) \wedge (x_1 \vee x_3 \vee x_4 \vee x_5) \wedge (x_1 \vee x_2 \vee x_4 \vee x_5) \wedge (x_1 \vee x_2 \vee x_3 \vee x_5) \wedge (x_1 \vee x_2 \vee x_3 \vee x_4),$$

which is correct since each clause is missing a different positive base variable, so the only way to satisfy them all is by having at least two positive base variables. It turns out that we can use a logarithmic number of clauses only, as first proved by Ben-Haim, Ivrii, Margalit and Matsliah [BHIMM12]. The main idea is that, if we have at least two base variables assigned to $\top$, say, x_i and x_j, then there is some bit index $b \in [\lg n + 1]$ for which we have both a base variable with 1 on its b-th bit and another one with 0 on its b-th bit; this must hold since i and j are different numbers and thus must differ on some bit. More concretely, we introduce auxiliary variables y_b for $b \in [\lg n + 1]$, and using notation $i_2[b]$ for the b-th bit of i, we introduce clauses:

$$\left(\overline{y_b} \vee \bigvee_{i \text{ s.t. } i_2[b]=1} x_i \right) \wedge \left(\overline{y_b} \vee \bigvee_{i \text{ s.t. } i_2[b]=0} x_i \right), \quad \forall b \in [\lg n + 1],$$

which state that if y_b is true, then there are both a true base variable whose b-th bit is 1 and a true base variable whose b-th bit is 0. The encoding is then completed by adding a "master clause" $\left(\bigvee_{b \in [\lg n+1]} y_b\right)$ stating that the condition is met for some bit index b. This encoding uses $\sim 2 \lg n$ clauses, and [BHIMM12] generalizes it for $\mathsf{AtLeast}_k$ for small k by using perfect hash families.

Once again: *can we do better?* Yes, slightly; by using base 3 instead of base 2. Let $t := \lceil \lg_3(n)+1 \rceil$, and once again we will have a master clause $\bigvee_{b=1}^{t} y_b$. Now, for each symbol $s \in \{0, 1, 2\}$, and bit index $b \in [t]$, we add a clause

$$C_{b,s} := \left(\overline{y_b} \vee \bigvee_{i \text{ s.t. } i_3[b] \neq s} x_i \right),$$

which in conjunction mean that y_b implies that for each symbol $s \in \{0, 1, 2\}$, there is some i with x_i assigned to true which differs from s in its b-th 'trit' (base-3 digit). If at least two variables are true, then there is some trit b in which they differ (e.g., one takes 0, the other takes 2) which implies that each $s \in \{0, 1, 2\}$ has some other value $s' \in \{0, 1, 2\} \setminus \{s\}$ which is taken by one of the true base variables in their b-th trit. Soundness is also straightforward to prove. The number of clauses is thus $\sim 3 \lg_3 n = \frac{3}{\lg 3} \cdot \lg n \approx 1.893 \lg n$. Interestingly, we cannot do better by using base 4 instead of 3; the general form of the encoding using base q has $\sim q \lg_q(n) = \frac{q}{\lg q} \lg n$ clauses, and the minimum value of $\frac{q}{\lg q}$ is achieved at $q = e \approx 2.7172\ldots$, which makes 3 the best integral option.

In fact, we can prove the following lower bound.

Proposition 8. $\mathsf{enc}(\mathit{AtLeast}_2(x_1, \ldots, x_n)) \geq \lg(n)$.

Proof. Let φ be a CNF encoding for $\mathsf{AtLeast}_2(x_1, \ldots, x_n)$, with a set Y of auxiliary variables. Then, for each $i \in [n]$, let $L_i := \{C \in \varphi \mid x_i \in C\}$ be the set of clauses containing variable x_i positively.

Now, we claim that for every pair of distinct indices $i, j \in [n]$, we cannot have $L_i \subseteq L_j$ nor $L_j \subseteq L_i$. Indeed, assume for a contradiction we had $L_i \subseteq L_j$ for some $i \neq j$. Then, if we consider the partial assignment τ which assigns $x_i \mapsto \top, x_j \mapsto \top$ and all other x_k variables to $\bot$, by definition of $\mathsf{AtLeast}_2$ there is an extension τ' of τ which assigns Y so that $\tau' \models \varphi$. But now, if we change τ' into τ'' by flipping x_i to $\bot$, we still have $\tau'' \models \varphi$, since the only clauses that could lose satisfaction are those in L_i, which by the assumption $L_i \subseteq L_j$ also contain x_j, which is still assigned to $\top$. Therefore, we conclude that $L_i \not\subseteq L_j$ for any distinct i, j. This implies that, if φ has m clauses, then $\{L_1, \ldots, L_n\}$ is a *Sperner family* of size n over m elements, which by Sperner's theorem [Spe28] implies

$$n \leq \binom{m}{\lfloor m/2 \rfloor} \leq \sum_{i=0}^{m} \binom{m}{i} = 2^m,$$

so the result follows. □

We have seen three different encodings for the $\mathsf{AtLeast}_2$ function: a *direct* one with n clauses, a *binary* encoding with roughly $2 \lg n$ clauses and a *ternary* encoding with roughly $1.89 \lg n$ clauses. It turns out that out of these, only the direct encoding is propagation complete. I will spare the proofs, but more in general, I have the following conjecture that I plan to investigate as part of the proposed project 3:

Conjecture 1. Every propagation-complete encoding for $\mathsf{AtLeast}_2(x_1, \ldots, x_n)$ has $\Omega(n)$ clauses.

I have made partial progress towards proving this conjecture, showing that any encoding of the general form used by the binary and ternary encoding (i.e., relying on a master clause to enforce the constraint) must use $\Omega(n)$ clauses; the proof is in my opinion beautiful, and it leverages the celebrated Graham-Pollak theorem [GP71, GP72]: every partition of the edges of K_n into complete bipartite graphs must use at least $n - 1$ parts.

With hopes of better understanding the possibilities for $\mathsf{AtLeast}_2$ encodings, I designed a CNF encoding for searching for small CNF encodings of this constraint (*SAT for SAT*). More concretely, this *meta-encoding* takes as parameters the target number of auxiliary variables (a) and the target number of clauses (m), and its variables are of the form $\mathsf{pos}_{i,j}$ for $i \in [n + a], j \in [m]$ to represent that variable i appears positively on clause j, and similarly $\mathsf{neg}_{i,j}$ to represent that variable i appears negatively on clause j. For small values of a, we can enumerate all 2^a possibilities of assignments for the auxiliary variables and add clauses to enforce that the satisfying assignments of our meta-encoding indeed correspond to encodings of the target function ($\mathsf{AtLeast}_2$ in this case). Running this with parameters $n = 6, a = 1, m = 5$ yielded the following:

```
[2, 3, 4, 6, 7], [1, 3, 5, 6, 7], [2, 5, 6, -7], [1, 3, 4, -7], [1, 2, 4, 5, 7].
```

Each list corresponds to a clause, where the base variables $x_1, \ldots, x_6$ are represented by their index, and the single auxiliary variable is represented by 7. After thinking for a while about what is going on with this encoding, I not only understood its correctness, but also how it generalizes into the following theorem:

Theorem 9. *There is an encoding for $\mathsf{AtLeast}_2(x_1, \ldots, x_n)$ using $O(\sqrt{n})$ clauses and a single auxiliary variable.*

Proof. Place the base variables into a $\sqrt{n} \times \sqrt{n}$ grid as in Chen's product encoding, thus renaming them as $x_{i,j}$ for $i, j \in [\sqrt{n}]$. But this time, instead of creating one aux variable per row and column,

just create one global auxiliary variable R. Now, for each row $r \in [\sqrt{n}]$, create a clause

$$\mathsf{Row}_r := \left(R \vee \bigvee_{i \neq r} \bigvee_{j=1}^{\sqrt{n}} x_{i,j} \right),$$

stating that either R is true or there is at least one true base variable outside of row r. Similarly, for each column $c \in [\sqrt{n}]$ create a clause

$$\mathsf{Col}_c := \left(\overline{R} \vee \bigvee_{i=1}^{\sqrt{n}} \bigvee_{j \neq c} x_{i,j} \right)$$

stating that if R is false then there is at least one true base variable outside of column c. That is the entire encoding. Now, assume a single base variable $x_{i,j}$ is true. Then, if R is true, clause Col_j is falsified, whereas if R is false then Row_i is falsified. On the other hand, if there are at least two true variables, $x_{i,j}$ and $x_{i',j'}$, we proceed by cases. If $i \neq i'$ and $j \neq j'$, every clause Row_r and Col_c is satisfied regardless of R. If $i = i'$, then setting R to true satisfies all clauses, whereas if $j = j'$ setting R to false satisfies all clauses.

□

This experience has convinced me of the value of using SAT to discover small encodings that can show us new techniques and ideas about other encodings. Unfortunately, my current meta-encoding does not scale well. I therefore plan on the following:

> *Proposed Project 4.* Design a better meta-encoding, as well as symmetry-breaking techniques towards the automated design of compact encodings for arbitrary functions with small n. My target is $n = 8$, which may sound humble but I see as a very challenging number. Given that these functions may require up to 20 clauses, and thus by Proposition 7 up to 20 auxiliary variables, the number of potential clauses is roughly $3^{28} \approx 2.28 \times 10^{13}$. On its own, this search space is not that intimidating for modern solvers; the key issue is even after having guessed what the clauses should be, checking whether they are a correct encoding is a $\Pi_2\mathsf{P}$-complete problem. I plan to draw inspiration from the work of Peitl and Szeider [PS21] which used an efficient SAT encoding to find formulas that are hard for resolution, as well as the work of Kojevnikov, Kulikov, and Yaroslavtsev using SAT to find small circuits [KKY09].

Conclusion

I hope to have conveyed by now a good sense of what my research is about, and what kind of results and discussions will be part of my thesis. In a nutshell, the central object of my thesis are CNF encodings, and the main thesis (in the sense of "hypothesis") is that the world of CNF encodings is much richer than previously thought, with an important part of that richness coming from the interplay between three ideas: (1) choosing the right auxiliary variables, (2) leveraging wide clauses, and (3) being willing to sacrifice propagation completeness in the right circumstances. Each of these ideas challenges some folk wisdom of the field (i.e., *use fewer variables, use narrow clauses, and always strive for propagation completeness!*), and while we are not the first to question these assumptions (cf. [Bjö11]), I believe my thesis can be a fundamental contributor towards grounding the conversation in more solid theoretical and empirical grounds.

To conclude, I offer the members of my committee (except for Marijn to whom I have already revealed the answer) a question I posed during my invited talk at the recent BIRS workshop

"Theory and Practice of SAT and Combinatorial Solving": consider the three presented encodings for $\mathsf{AtLeast}_2$ (direct, binary and ternary one), how will they rank from best to worst in terms of runtime to prove UNSAT when adding an AtMostOne constraint on the same input variables? Nobody guessed it right among the BIRS participants, which I take as a symptom of our need for better tools, both theoretical and practical, that will allow us to better predict what encodings will be performant in practice. I plan to reveal the empirical answer during my proposal talk on April 8th.

References

[ABB+24] Marcelo Arenas, Pablo Barceló, Diego Bustamante, Jose Caraball, and Bernardo Subercaseaux. A uniform language to explain decision trees. In *Proceedings of the 21st International Conference on Principles of Knowledge Representation and Reasoning (KR 2024)*, pages 60–70, 2024. 2, 3

[ABK+25] Marcelo Arenas, Pablo Barcelo, Alexander Kozachinskiy, Miguel Romero, and Bernardo Subercaseaux. On computing probabilistic explanations for decision trees. *Journal of Artificial Intelligence Research*, 83, 2025. 10

[ABOS22] Marcelo Arenas, Pablo Barceló, Miguel A. Romero Orth, and Bernardo Subercaseaux. On computing probabilistic explanations for decision trees. In *Advances in Neural Information Processing Systems 35 (NeurIPS 2022)*, 2022. 2, 3

[AGMES16] Ignasi Abío, Graeme Gange, Valentin Mayer-Eichberger, and Peter J. Stuckey. On CNF Encodings of Decision Diagrams. In Claude-Guy Quimper, editor, *Integration of AI and OR Techniques in Constraint Programming*, pages 1–17. Springer International Publishing, 2016. 18

[AKS83] Miklós Ajtai, János Komlós, and Endre Szemerédi. Sorting in $c \log n$ parallel steps. *Combinatorica*, 3(1):1–19, 1983. 15

[AL86] Ron Aharoni and Nathan Linial. Minimal non-two-colorable hypergraphs and minimal unsatisfiable formulas. *Journal of Combinatorial Theory, Series A*, 43(2):196–204, 1986. 16

[AM05] Carlos Ansótegui and Felip Manyà. Mapping Problems with Finite-Domain Variables to Problems with Boolean Variables. In Holger H. Hoos and David G. Mitchell, editors, *Theory and Applications of Satisfiability Testing*, pages 1–15. Springer, 2005. 5

[AM06] Kazuyuki Amano and Akira Maruoka. The monotone circuit complexity of quadratic boolean functions. *Algorithmica*, 46(1):3–14, 2006. 15

[ANORC11] Roberto Asín, Robert Nieuwenhuis, Albert Oliveras, and Enric Rodríguez-Carbonell. Cardinality Networks: A theoretical and empirical study. *Constraints*, 16(2):195–221, 2011. 15

[Bat68] Kenneth E. Batcher. Sorting networks and their applications. In *American Federation of Information Processing Societies: AFIPS Conference Proceedings: 1968 Spring Joint Computer Conference, Atlantic City, NJ, USA, 30 April - 2 May 1968*, volume 32 of *AFIPS Conference Proceedings*, pages 307–314. Thomson Book Company, Washington D.C., 1968. 15

[BGI+26] Hideo Bannai, Keisuke Goto, Masakazu Ishihata, Shunsuke Kanda, Dominik Köppl, Takaaki Nishimoto, and Bernardo Subercaseaux. Computing NP-hard repetitiveness measures via MAX-SAT. *ACM Trans. Algorithms*, 22(2), February 2026. 2

[BHI+23] Tomas Balyo, Marijn Heule, Markus Iser, Matti Järvisalo, and Martin Suda, editors. *Proceedings of SAT Competition 2023: Solver, Benchmark and Proof Checker Descriptions*. Department of Computer Science Series of Publications B. Department of Computer Science, University of Helsinki, Finland, 2023. 12

[BHIMM12] Yael Ben-Haim, Alexander Ivrii, Oded Margalit, and Arie Matsliah. Perfect Hashing and CNF Encodings of Cardinality Constraints. In Alessandro Cimatti and Roberto Sebastiani, editors, *Theory and Applications of Satisfiability Testing – SAT 2012*, pages 397–409. Springer, 2012. 19

[Bjö11] Magnus Björk. Successful SAT Encoding Techniques. *Journal on Satisfiability, Boolean Modelling and Computation*, 7(4):189–201, 2011. 21

[BKR+25] Pablo Barceló, Alexander Kozachinskiy, Miguel Romero, Bernardo Subercaseaux, and José Verschae. Explaining k-nearest neighbors: Abductive and counterfactual explanations. *Proceedings of the ACM on Management of Data*, 3(2):97:1–97:26, 2025. Presented at PODS 2025. 2, 3

[Blo79] Peter Anthony Bloniarz. *The complexity of monotone boolean functions and an algorithm for finding shortest paths on a graph*. PhD thesis, Massachusetts Institute of Technology, 1979. 15

[BMS12] Lucas Bordeaux and Joao Marques-Silva. Knowledge Compilation with Empowerment. In Mária Bieliková, Gerhard Friedrich, Georg Gottlob, Stefan Katzenbeisser, and György Turán, editors, *SOFSEM 2012: Theory and Practice of Computer Science*, pages 612–624. Springer, 2012. 6

[Bub86] Siegfried Bublitz. Decomposition of graphs and monotone formula size of homogeneous functions. *Acta Inform.*, 23(6):689–696, 1986. 15

[Che10] Jingchao Chen. A new SAT encoding of the at-most-one constraint. In *Proc. of the Tenth Int. Workshop of Constraint Modelling and Reformulation*, 2010. 7, 8

[ES06] Niklas Eén and Niklas Sörensson. Translating pseudo-boolean constraints into SAT. *J. Satisf. Boolean Model. Comput.*, 2(1-4):1–26, 2006. 15

[FG10] Alan M. Frisch and Paul A. Giannaros. SAT encodings of the at-most-k constraint: Some old, some new, some fast, some slow. In *Proc. of the Tenth Int. Workshop of Constraint Modelling and Reformulation*, 2010. 14

[GHS24] Thomas Garrison, Marijn J. H. Heule, and Bernardo Subercaseaux. Packit!: Gamified rectangle packing. In *12th International Conference on Fun with Algorithms (FUN 2024)*, volume 291 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 14:1–14:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. 3

[GIZ+25] Atticus Geiger, Duligur Ibeling, Amir Zur, Maheep Chaudhary, Sonakshi Chauhan, Jing Huang, Aryaman Arora, Zhengxuan Wu, Noah Goodman, Christopher Potts, and Thomas Icard. Causal abstraction: A theoretical foundation for mechanistic interpretability, 2025. 4, 11

[GK13] Matthew Gwynne and Oliver Kullmann. Towards a theory of good SAT representations, 2013. 18

[GN04] Ian P. Gent and Peter Nightingale. A new encoding of AllDifferent into SAT. In *International Workshop on Modelling and Reformulating Constraint Satisfaction*, volume 3, pages 95–110, 2004. 5

[GP71] R. L. Graham and H. O. Pollak. On the addressing problem for loop switching. *The Bell System Technical Journal*, 50(8):2495–2519, 1971. 20

[GP72] R. L. Graham and H. O. Pollak. On embedding graphs in squashed cubes. In Y. Alavi, D. R. Lick, and A. T. White, editors, *Graph Theory and Applications*, pages 99–110. Springer, 1972. 20

[HGH23] Andrew Haberlandt, Harrison Green, and Marijn J. H. Heule. Effective Auxiliary Variables via Structured Reencoding. In Meena Mahajan and Friedrich Slivovsky, editors, *26th International Conference on Theory and Applications of Satisfiability Testing (SAT 2023)*, volume 271 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 11:1–11:19, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. 12

[HKB18] Marijn J. H. Heule, Oliver Kullmann, and Armin Biere. Cube-and-conquer for satisfiability. In Youssef Hamadi and Lakhdar Sais, editors, *Handbook of Parallel Constraint Reasoning*, pages 31–59. Springer, 2018. 2

[HKM16] Marijn J. H. Heule, Oliver Kullmann, and Victor W. Marek. Solving and verifying the boolean pythagorean triples problem via cube-and-conquer. In Nadia Creignou and Daniel Le Berre, editors, *Theory and Applications of Satisfiability Testing - SAT 2016 - 19th International Conference, Bordeaux, France, July 5-8, 2016, Proceedings*, volume 9710 of *LNCS*, pages 228–245. Springer, 2016. 2

[Juk06] Stasys Jukna. Disproving the single level conjecture. *SIAM J. Comput.*, 36(1):83–98, 2006. 15

[Kar19] Michał Karpiński. CNF encodings of cardinality constraints based on comparator networks, 2019. 15, 17

[KKY09] Arist Kojevnikov, Alexander S. Kulikov, and Grigory Yaroslavtsev. Finding Efficient Circuits Using SAT-Solvers. In Oliver Kullmann, editor, *Theory and Applications of Satisfiability Testing - SAT 2009*, pages 32–44. Springer Berlin Heidelberg, 2009. 21

[KPS26] Andrew Krapivin, Benjamin Przybocki, and Bernardo Subercaseaux. Near-optimal encodings of cardinality constraints, 2026. Manuscript submitted to SAT 2026. 3, 15, 17

[KPSMS25] Andrew Krapivin, Benjamin Przybocki, Nicolás Sanhueza-Matamala, and Bernardo Subercaseaux. Optimal and efficient partite decompositions of hypergraphs, 2025. 2, 3, 12

[KPSS25] Markus Kirchweger, Tomás Peitl, Bernardo Subercaseaux, and Stefan Szeider. From the finite to the infinite: Sharper asymptotic bounds on Norin's conjecture via SAT. *CoRR*, abs/2511.08386, 2025. 2, 3

[Kri64] Rafail E. Krichevskii. Complexity of contact circuits realizing a function of logical algebra. *Soviet Physics Doklady*, 8:770–772, 1964. 15

[KS22] Petr Kučera and Petr Savický. Propagation complete encodings of smooth DNNF theories. *Constraints*, 27(3):327–359, 2022. 18

[KSV19] Petr Kučera, Petr Savický, and Vojtěch Vorel. A lower bound on CNF encodings of the at-most-one constraint. *Theor. Comput. Sci.*, 762:51–73, 2019. 6

[LW91] Katja Lenz and Ingo Wegener. The conjunctive complexity of quadratic boolean functions. *Theoret. Comput. Sci.*, 81(2):257–268, 1991. 15

[MHB12] Norbert Manthey, Marijn J. H. Heule, and Armin Biere. Automated reencoding of boolean formulas. In *Haifa Verification Conference*, pages 102–117. Springer, 2012. 12

[MS92] Roland Mirwald and Claus P. Schnorr. The multiplicative complexity of quadratic boolean forms. *Theoret. Comput. Sci.*, 102(2):307–328, 1992. 15

[PS21] Tomáš Peitl and Stefan Szeider. Finding the Hardest Formulas for Resolution. *Journal of Artificial Intelligence Research*, 72:69–97, 2021. 21

[PSH26] Benjamin Przybocki, Bernardo Subercaseaux, and Marijn J. H. Heule. Automated reencoding meets graph theory, 2026. Manuscript submitted to SAT 2026. 3, 12

[QWSH25] Long Qian, Eric Wang, Bernardo Subercaseaux, and Marijn J. H. Heule. Unfolding boxes with local constraints. In *Automated Deduction – CADE 30: 30th International Conference on Automated Deduction, Stuttgart, Germany, July 28–31, 2025, Proceedings*, pages 736–754, 2025. 2, 3

[Ser20] Igor S. Sergeev. On the complexity of monotone circuits for threshold symmetric boolean functions. *Diskret. Mat.*, 32(1):81–109, 2020. 15

[SH22] Bernardo Subercaseaux and Marijn J. H. Heule. The packing chromatic number of the infinite square grid is at least 14. In *25th International Conference on Theory and Applications of Satisfiability Testing (SAT 2022)*, volume 236 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 21:1–21:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. 2, 3

[SH23a] Bernardo Subercaseaux and Marijn Heule. Toward optimal radio colorings of hypercubes via SAT-solving. In *24th International Conference on Logic for Programming, Artificial Intelligence and Reasoning (LPAR 2023)*, pages 386–404, 2023. 2, 3

[SH23b] Bernardo Subercaseaux and Marijn J. H. Heule. The packing chromatic number of the infinite square grid is 15. In Sriram Sankaranarayanan and Natasha Sharygina, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 389–406, Cham, 2023. Springer Nature Switzerland. 2, 3, 12

[Sha49] Claude E. Shannon. The synthesis of two-terminal switching circuits. *The Bell System Technical Journal*, 28(1):59–98, 1949. 16

[Sin05] Carsten Sinz. Towards an Optimal CNF Encoding of Boolean Cardinality Constraints. In Peter van Beek, editor, *Principles and Practice of Constraint Programming - CP 2005*, pages 827–831. Springer, 2005. 5

[SMHM24] Bernardo Subercaseaux, John Mackey, Marijn J. H. Heule, and Ruben Martins. Automated mathematical discovery and verification: Minimizing pentagons in the plane. In *Intelligent Computer Mathematics*, pages 21–41, 2024. CICM 2024. 2, 3

[SMQH26] Bernardo Subercaseaux, Ethan Mackey, Long Qian, and Marijn Heule. Automated symmetric constructions in discrete geometry. In *Intelligent Computer Mathematics*, pages 29–47, 2026. CICM 2025. 2, 3

[Spe28] Emanuel Sperner. Ein satz über untermengen einer endlichen menge. *Mathematische Zeitschrift*, 27(1):544–548, 1928. 20

[Sub25] Bernardo Subercaseaux. Asymptotically smaller encodings for graph problems and scheduling. *CoRR*, abs/2506.14042, 2025. 2, 3

[SW24] Naomi Saphra and Sarah Wiegreffe. Mechanistic? In Yonatan Belinkov, Najoung Kim, Jaap Jumelet, Hosein Mohebbi, Aaron Mueller, and Hanjie Chen, editors, *Proceedings of the 7th BlackboxNLP Workshop: Analyzing and Interpreting Neural Networks for NLP*, pages 480–498, Miami, Florida, US, November 2024. Association for Computational Linguistics. 4

[War98] Joost P. Warners. A linear-time transformation of linear inequalities into conjunctive normal form. *Information Processing Letters*, 68(2):63–69, October 1998. 5

[WHH14] Nathan Wetzler, Marijn J. H. Heule, and Warren A. Hunt. DRAT-trim: Efficient Checking and Trimming Using Expressive Clausal Proofs. In Carsten Sinz and Uwe Egly, editors, *Theory and Applications of Satisfiability Testing – SAT 2014*, pages 422–429. Springer International Publishing, 2014. 10