Abstract
The construction of exact exponential-time algorithms for NP-complete problems has for some time been a very active research area. Unfortunately, there is a lack of general methods for studying and comparing the time complexity of algorithms for such problems. We propose such a method based on clone theory and demonstrate it on the SAT problem. Schaefer has completely classified the complexity of SAT with respect to the set of allowed relations and proved that this parameterized problem exhibits a dichotomy: it is either in P or is NP-complete. We show that there is a certain partial order on the NP-complete SAT problems with a close connection to their worst-case time complexities; if a problem SAT(S) is below a problem SAT(S′) in this partial order, then SAT(S′) cannot be solved strictly faster than SAT(S). By using this order, we identify a relation R such that SAT({R}) is the computationally easiest NP-complete SAT(S) problem. This result may be interesting when investigating the borderline between P and NP since one appealing way of studying this borderline is to identify problems that, in some sense, are situated close to it (such as a ‘very hard’ problem in P or a ‘very easy’ NP-complete problem). We strengthen the result by showing that SAT({R})-2 (i.e. SAT({R}) restricted to instances where no variable appears more than twice) is NP-complete, too. This is in contrast to, for example, 1-in-3-SAT (or even CNF-SAT), which is in P under the same restriction. We then relate SAT({R})-2 to the exponential-time hypothesis (ETH) and show that ETH holds if and only if SAT({R})-2 is not sub-exponential. This constitutes a strong connection between ETH and the SAT problem under both severe relational and severe structural restrictions, and it may thus serve as a tool for studying the borderline between sub-exponential and exponential problems. In the process, we also prove a stronger version of Impagliazzo et al.’s sparsification lemma for k-SAT; namely that all finite Boolean constraint languages S and S’ such that SAT(·) is NP-complete can be sparsified into each other. This should be compared with Santhanam and Srinivasan’s recent negative result which states that the same does not hold for all infinite Boolean constraint languages.

1 Introduction
This paper is concerned with the SAT(S) class of problems: given a finite set of Boolean relations S, decide whether a conjunction of constraints (where only relations from S are used) is satisfiable or not. This class of problems is very rich and contains many problems that are highly relevant both theoretically and in practice. Since Schaefer’s seminal dichotomy result \[27\], the computational complexity of SAT(S) is completely known: we know for which S that SAT(S) is polynomial-time solvable and for which it is NP-complete, and these are the only possible cases. On the other hand, judging from the running times of the many algorithms that have been proposed for different NP-complete SAT(S) problems, it seems that the computational complexity varies greatly for different S. As an example, 3-SAT (where S consists of all clauses of length at most 3) is only known to be solvable in time \[O(1.3334^n)\] \[22\] (where n is the number of variables), and so it seems to be a much harder problem than, for instance, positive 1-in-3-SAT (where S consists only of the relation \[\{(0,0,1),(0,1,0),(1,0,0)\}\], which can be solved in time \[O(1.0984^n)\] \[32\]. It is fair to say that we have a very vague understanding of the time complexity of NP-complete problems, and this fact is clearly expressed in Cygan et al. \[6\].

Complexity of SAT Problems, Clone Theory and the Exponential Time Hypothesis
Peter Jonsson* Victor Lagerkvist† Gustav Nordh‡ Bruno Zanuttini§
What the field of exponential-time algorithms sorely lacks is a complexity-theoretic framework for showing running time lower bounds.

In this paper, we initiate a systematic study of the relationships between the worst-case complexity of different SAT(S) problems, where we measure the complexity as a function of the number of variables. Ultimately, one would like to have a ‘table’ that for each NP-complete SAT(S) problem contains a number c such that SAT(S) can be solved in $\Theta(c^n)$ time. It seems that we are very far from this goal, unfortunately. Let us imagine a weaker qualitative approach: construct a table that for every two problems SAT(S) and SAT(S’) tells us whether SAT(S) and SAT(S’) can be solved equally fast, whether SAT(S) can be solved strictly faster than SAT(S’), or vice versa. That is, we have access to the underlying total order on running times but we cannot say anything about the exact figures. Not surprisingly, we are far from this goal, too. However, this table can, in a sense, be approximated: there are non-trivial lattices that satisfy this property whenever $S$ and $S'$ are comparable to each other in the lattice. To obtain such lattices, we exploit clone theory \cite{5,18,20}. This theory has proven to be very powerful when studying the complexity of SAT(S) and its multi-valued generalization known as constraint satisfaction problems (CSP) \cite{2}. However, it is not clear how this theory can be used for studying the worst-case running times for algorithms. We show how to use it for this purpose in Section 3 and our basic observation is that the lattice of partial clones \cite{22,23} has the required properties. We would like to emphasize that this approach can be generalized in different ways; it is not restricted to Boolean problems and it is applicable to other computational problems such as counting and enumeration.

As a concrete application of this method, we identify the computationally easiest NP-complete SAT(S) problem in Section 3 by ‘computationally easiest’, we mean that if any NP-complete SAT(S) problem can be solved in $O(c^n)$ time, then the easiest problem can be solved in $O(c^n)$ time, too. This easiest NP-complete SAT(S) problem is surprisingly simple: $S$ consists of a single 6-ary relation $R_{2/3}^{\pm\pm\pm}$ which contains the three tuples $(1, 0, 0, 0, 1, 1)$, $(0, 0, 1, 1, 0, 0)$, and $(0, 0, 1, 1, 1, 1)$. This result is obtained by making use of Schmoor and Schnoor’s machinery for constructing weak bases. We note that there has been an interest in identifying extremely easy NP-complete problems before. For instance, van Rooij et al. \cite{21} have shown that the Partition into Triangles problem restricted to graphs of maximum degree four can be solved in $O(1.02445^n)$ time. They argue that practical algorithms may arise from this kind of studies, and the very same observation has been made by, for instance, Woeginger \cite{23}. It is important to note that our results give much more information than just the mere fact that SAT($(R_{2/3}^{\pm\pm\pm})$) is easy to solve; they also tell us how this problem is related to all other problems within the large and diverse class of SAT(S) problems. This is one of the major advantages in using the clone-theoretical approach when studying these kind of questions. Another reason to study such problems is that, in some sense, are ‘close’ to the borderline between problems in NP that are not complete and NP-complete problems (here we tacitly assume that P ≠ NP). The structure of this borderline has been studied with many different aims and many different methods; two well-known examples are the articles by Ladner \cite{17} and Schöning \cite{29}.

We continue by studying the complexity of SAT($(R_{2/3}^{\pm\pm\pm})$) and general SAT problems in greater detail by relating them to the EXPONENTIAL TIME HYPOTHESIS (ETH) \cite{13}, i.e. the hypothesis that $k$-SAT cannot be solved in sub-exponential time for $k \geq 3$. The ETH has recently gained popularity when studying the computational complexity of combinatorial problems, cf. the survey by Lokshin et al. \cite{19}.

We first note (in Section 3) that SAT($(R_{2/3}^{\pm\pm\pm})$) restricted to instances where no variable appears more than twice (the SAT($(R_{2/3}^{\pm\pm\pm})$)-2 problem) is still NP-complete (in contrast to, for instance, positive 1-in-3-SAT or CNF-SAT, which is in P under the same restriction). We prove this by using results by Dalmau and Ford \cite{8} combined with the fact that $R_{2/3}^{\pm\pm\pm}$ is not a $\Delta$-matroid relation. We then show (in Section 6) that the exponential-time hypothesis holds if and only if SAT($(R_{2/3}^{\pm\pm\pm})$)-2 cannot be solved in sub-exponential time. By using this result, we show the following consequence: if ETH does not hold, then SAT(S)-B is sub-exponential for every $B$ whenever $S$ is finite. Impagliazzo et al. \cite{13} have proved that many NP-complete problems in SNP (which contains the SAT problems) are sub-exponential if and only if $k$-SAT is sub-exponential. Thus, we strengthen this result when restricted to SAT problems. In the process, we also prove a stronger version of Impagliazzo et al.’s \cite{13} sparsification lemma for $k$-SAT; namely that all finite Boolean constraint languages $S$ and $S'$ such that SAT($S$) is NP-complete can be sparsified into each other. This can be contrasted with Santhanam’s and Srinivasan’s \cite{26} recent negative result which states that the same does not hold for the unrestricted SAT problem and, consequently, not for all infinite Boolean constraint languages.
2 The Boolean SAT problem
We begin by introducing the notation and basic results that will be used in the rest of this paper. The set of all $k$-tuples over $\{0, 1\}$ is denoted by $\{0, 1\}^k$. Any subset of $\{0, 1\}^k$ is called an $k$-ary relation on $\{0, 1\}$. The set of all finitary relations over $\{0, 1\}$ is denoted by $BR$. A constraint language over $\{0, 1\}$ is a finite set $S \subset BR$.

Definition 2.1. The Boolean satisfiability problem over the constraint language $S \subset BR$, denoted by $SAT(S)$, is defined to be the decision problem with instance $(V, C)$, where $V$ is a set of Boolean variables, and $C$ is a set of constraints $\{C_1, \dots, C_q\}$, in which each constraint $C_i$ is a pair $(s_i, R_i)$ with $s_i$ a list of variables of length $k_i$, called the constraint scope, and $R_i$ an $k_i$-ary relation over the set $\{0, 1\}$, belonging to $S$, called the constraint relation.

The question is whether there exists a solution to $(V, C)$, that is, a function from $V$ to $\{0, 1\}$ such that, for each constraint in $C$, the image of the constraint scope is a member of the constraint relation.

Example. Let $R_{\text{NAE}}$ be the following ternary relation on $\{0, 1\}$: $R_{\text{NAE}} = \{0, 1\}^3 \setminus \{(0, 0, 0), (1, 1, 1)\}$. It is easy to see that the well known NP-complete problem Not-All-Equal 3-Sat can be expressed as $SAT(\{R_{\text{NAE}}\})$.

Constraint languages where negation is normally used needs some extra care: let the sign pattern of a constraint $\gamma(x_1, \ldots, x_k)$ be the tuple $(s_1, \ldots, s_k)$, where $s_i = +$ if $x_i$ is unnegated, and $s_i = -$ if $x_i$ is negated. For each sign pattern we can then associate a relation that captures the satisfying assignments of the constraint. For example, the sign pattern of $R_{\text{NAE}}(x, \neg y, \neg z)$ is the tuple $(+, -, -)$, and its associated relation is $R_{\text{NAE}}^{+, -, -} = \{0, 1\}^3 \setminus \{(0, 1, 1), (1, 0, 0)\}$. More generally, we write $\Gamma_{\text{NAE}}^k$ for the corresponding constraint language of not-all-equal relations (with all possible sign patterns) of length $k$. If $\phi$ is a SAT($\Gamma_{\text{NAE}}^k$) instance we write $\gamma_{\text{NAE}}^k(x_1, \ldots, x_k)$ for a constraint in $\phi$, where each $x_i$ is unnegated or negated. In the same manner we write $\Gamma_{\text{SAT}}^k$ for the constraint language consisting of all $k$-SAT relations of length $k$.

When explicitly defining relations, we often use the standard matrix representation where the rows of the matrix are the tuples in the relation. For example, $R_{\text{NAE}}$:

$$R_{\text{NAE}} = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix}$$

Note that the relative order of the columns in the matrix representation does not matter since this only corresponds to a different order of the variables in a constraint.

3 Partial clones and the complexity of SAT
We will now show that the time complexity of $SAT(S)$ is determined by the so-called partial polymorphisms of $S$. For a more in-depth background on SAT and algebraic techniques, we refer the reader to Böhler et al. [4] and Lau [18], respectively. Even though most of the results in this section hold for arbitrary finite domains we present everything in the Boolean setting. We first note that any $k$-ary operation $f$ on $\{0, 1\}$ can be extended in a standard way to an operation on tuples over $\{0, 1\}$, as follows: for any collection of $k$ tuples, $t_1, t_2, \ldots, t_k \in R$, the $n$-tuple $f(t_1, t_2, \ldots, t_k)$ is defined as follows:

$$f(t_1, t_2, \ldots, t_k) = (f(t_1[1], t_2[1], \ldots, t_k[1]), f(t_1[2], t_2[2], \ldots, t_k[2]), \ldots, f(t_1[n], t_2[n], \ldots, t_k[n]),$$

where $t_j[i]$ is the $i$-th component in tuple $t_j$. We are now ready to define the concept of polymorphisms.

Definition 3.1. Let $S$ be a Boolean constraint language and $R$ an arbitrary relation from $S$. If $f$ is an operation such that for all $t_1, t_2, \ldots, t_k \in R$ it holds that $f(t_1, t_2, \ldots, t_k) \in R$, then $R$ is closed (invariant) under $f$. If all relations in $S$ are closed under $f$ then $S$ is closed under $f$. An operation $f$ such that $S$ is closed under $f$ is called a polymorphism of $S$. The set of all polymorphisms of $S$ is denoted by $\text{Pol}(S)$. Given a set of operations $F$, the set of all relations that are closed under all the operations in $F$ is denoted by $\text{Inv}(F)$.

Example. The ternary majority operation $f$ over the Boolean domain is the operation satisfying $f(a, b, a) = f(b, a, a) = a$ for $a, b \in \{0, 1\}$. Let

$$R = \{(0, 0, 1), (1, 0, 0), (0, 1, 1), (1, 0, 1)\}.$$

It is then easy to verify that for every triple of tuples, $x, y, z \in R$, we have $f(x, y, z) \in R$. For example, if $x = (0, 0, 1), y = (0, 1, 1)$ and $z = (1, 0, 1)$ then

$$f(x, y, z) = (f(x[1], y[1], z[1]), f(x[2], y[2], z[2]), f(x[3], y[3], z[3])) = (f(0, 0, 1), f(0, 1, 0), f(1, 1, 1)) = (0, 0, 1) \in R.$$
Schaefer’s classification of SAT(S) follows more or less directly from this result together with Post’s lattice of clones. It is worth noting that out of the countably infinite number of Boolean clones, there are just two that corresponds to NP-complete SAT(S) problems. These are the clone \( I_2 \) consisting of all projections (i.e. the operations of the form \( f^k_i(x_1, \ldots, x_k) = x_i \)), and the clone \( N_2 \) consisting of all projections together with the unary negation function \( n(0) = 1, n(1) = 0 \). It is easy to realize that \( Inv(I_2) \) is the set of all Boolean relations (i.e., \( BR \)) and we denote \( Inv(N_2) \) by \( IN_2 \).

Theorem 3.1 is not very useful for studying the complexity of SAT problems in terms of their worst-case complexity as a function of the number of variables. The reason is that the reductions do not preserve instance sizes and may introduce large numbers of new variables. It also seems that the lattice of clones is not fine grained enough for this purpose. For example, 1-in-3-SAT and \( k \)-SAT (for \( k \geq 3 \)) both correspond to the same clone \( I_2 \).

One way to get a more refined framework is to consider partial operations in Definition 3.1. That is, we say that \( R \) is closed under the (partial) operation \( f \) if \( f \) applied componentwise to the tuples of \( R \) always results in a tuple from \( R \) or an undefined result (i.e., \( f \) is undefined on at least one of the components). The set of all (partial) operations preserving the relations in \( S \), i.e., the partial polymorphisms of \( S \) is denoted by \( pPol(S) \) and forms a partial clone.

Example. Consider again the relation \( R \) and the affine operation \( g \) from Example 3 and let \( p \) be the partial operation defined as \( p(x, y, z) = g(x, y, z) \) except that it is undefined for \((1, 1, 0), (1, 0, 1), (0, 1, 1) \) and \((1, 1, 1) \). Now it can be verified that \( p \) is a partial polymorphism of \( R \).

Unlike the lattice of Boolean clones, the lattice of partial Boolean clones consists of an uncountable infinite number of partial clones, and despite being a well-studied mathematical object [18], its structure is far from being completely understood.

Before we show that the lattice of partial clones is fine-grained enough to capture the complexity of SAT(S) problems (in terms of the worst-case complexity as a function of the number of variables) we need to present a Galois connection between sets of relations and sets of (partial) functions.

Definition 3.2. For any set \( S \subseteq BR \), the set \( \langle S \rangle \) consists of all relations that can be expressed (or implemented) using relations from \( S \cup \{ = \} \) (where \( = \) denotes the equality relation on \( \{0, 1\} \)), conjunction, and existential quantification. We call such implementations primitive positive (p.p.) implementations. Similarly,
for any set \( S \subseteq B \) the set \( \langle S \rangle_2 \) consists of all relations that can be expressed using relations from \( S \cup \{=\} \) and conjunction. We call such implementations quantifier-free primitive positive (q.p.p.) implementations.

Sets of relations of the form \( \langle S \rangle_2 \) and \( \langle S \rangle_3 \) are referred to as relational clones (or co-clones) and partial relational clones, respectively. The lattice of Boolean co-clones is visualized in Figure 2. There is a Galois connection between (partial) clones and (partial) relational clones given by the following result.

**Theorem 3.2.** ([3], [4], [5], [6]) Let \( S_1 \) and \( S_2 \) be constraint languages. Then \( S_1 \subseteq \langle S_2 \rangle_2 \) if and only if \( \text{Pol}(S_2) \subseteq \text{Pol}(S_1) \), and \( S_1 \subseteq \langle S_2 \rangle_3 \) if and only if \( \text{pPol}(S_2) \subseteq \text{pPol}(S_1) \).

Finally, we show that the complexity of \( \text{SAT}(S) \) is determined by the lattice of partial clones.

**Theorem 3.3.** Let \( S_1 \) and \( S_2 \) be finite non-empty sets of Boolean relations. If \( \text{pPol}(S_2) \subseteq \text{pPol}(S_1) \) and \( \text{SAT}(S_2) \) is solvable in time \( O(c^n) \), then \( \text{SAT}(S_1) \) is solvable in time \( O(c^n) \).

**Proof.** Given an instance \( I \) of \( \text{SAT}(S_1) \) on \( n \) variables we transform \( I \) into an equivalent instance \( I' \) of \( \text{SAT}(S_2) \) on at most \( n \) variables. Since \( S_1 \) is fixed and finite we can assume that the quantifier-free primitive positive implementation of each relation in \( S_1 \) by relations in \( S_2 \) has been precomputed and stored in a table (of fixed constant size). Every constraint \( R(x_1, \ldots, x_k) \) in \( I \) can be represented as

\[
R_1(x_{11}, \ldots, x_{1k_1}) \land \cdots \land R_l(x_{l1}, \ldots, x_{lk_l})
\]

where \( R_1, \ldots, R_l \in S_2 \cup \{=\} \) and \( x_{11}, \ldots, x_{lk_l} \in \{x_1, x_2, \ldots, x_k\} \). Replace the constraint \( R(x_1, \ldots, x_k) \) with the constraints \( R_1, \ldots, R_l \). If we repeat the same reduction for every constraint in \( I \), it results in an equivalent instance of \( \text{SAT}(S_2 \cup \{=\}) \) having at most \( n \) variables. For each equality constraint \( x_i = x_j \) we replace all occurrences of \( x_i \) with \( x_j \) and remove the equality constraint. The resulting instance \( I' \) is an equivalent instance of \( \text{SAT}(S_2) \) having at most \( n \) variables, hence it can be solved in time \( O(c^n) \). Finally, since \( S_1 \) is finite, there cannot be more than \( O(n^p) \) constraints in \( I \), where \( p \) is the highest arity of a relation in \( S_1 \). Hence computing \( I' \) from \( I \) can be done in time \( O(n^p) \), and we conclude that \( \text{SAT}(S_1) \) is solvable in time \( O(n^p + c^n) = O(c^n) \).

\[\square\]

4 The easiest NP-complete \( \text{SAT}(S) \) problem

In this section we will use the theory and results presented in the previous section to determine the easiest NP-complete \( \text{SAT}(S) \) problem. Recall that by easiest we mean that if any NP-complete \( \text{SAT}(S) \) problem can be solved in \( O(c^n) \) time, then the easiest problem can be solved in \( O(c^n) \) time, too. Following this lead we say that \( \text{SAT}(S) \) is at least as easy as (or not harder than) \( \text{SAT}(S') \) if \( \text{SAT}(S') \) is solvable in \( O(c^n) \) time whenever \( \text{SAT}(S') \) is solvable in \( O(c^n) \) time. A crucial step for doing this is the explicit construction of Schwoor and Schwoor [28] that, for each relational clone \( S \), gives a relation \( R \) such that \( S = \langle \{R\} \rangle \) and \( R \) has a q.p.p. implementation in every constraint language \( S \) such that \( \langle S \rangle = X \). Essentially, this construction gives the bottom element of the interval of partial relational clones contained in each relational clone.

Let \( R = \{r_1, \ldots, r_n\} \) be a \( k \)-ary relation. The \((k + 2^n)\)-ary relation \( R^+ \) is built from \( R \) by viewing it as a \( n \times k \)-matrix and adding the \( 2^n \) Boolean tuples as columns of this matrix, in an arbitrary order (say, lexicographic). In the sequel, we use \( b_1 \ldots b_k \) as a shorthand for the tuple \( (b_1, \ldots, b_k) \). Define the relation \( R_{1/3} = \{001, 010, 100\} \) and note that \( \text{SAT}(\{R_{1/3}\}) \) corresponds to the 1-in-3-SAT problem. From \( R_{1/3} \) we then get the following 11-ary relations with 3 tuples:

\[
R_{1/3}^+ = \begin{cases}
001 & 0001111, \\
010 & 0011001, \\
100 & 01010101
\end{cases}
\]
Then the extension of \( R \), denoted by \( R^{[\text{ext}]} \), is the relation \( \bigcap_{R \in \{R_1, R_2, \ldots, R_n\}} R \), or, equivalently, the closure of \( R^+ \) under the polymorphisms of \( R \).

**Theorem 4.1.** (Schnoor and Schnoor [28]) Let \( R \) be a Boolean relation. Then \( R^{[\text{ext}]} \) has a q.p.p. implementation in \( \{R\} \). Furthermore, any constraint language \( S \) such that \( \langle S \rangle = \{\{R\}\} \) implements \( R^{[\text{ext}]} \) via a q.p.p. implementation.

We are now in the position to define the easiest NP-complete SAT problem. The relation \( R^{3\times3}_{1/3} = \{(001110, 010101, 100001)\} \) is formed by taking \( R_{1/3} \) and adding the negation of each of the columns, i.e., \( R^{3\times3}_{1/3}(x_1, x_2, x_3, x_4, x_5, x_6) \) can be defined as \( R_{1/3}(x_1, x_2, x_3) \land (x_1 \neq x_4) \land (x_2 \neq x_5) \land (x_3 \neq x_6) \).

**Lemma 4.1.** Let \( S \) be a constraint language such that \( \langle S \rangle = BR \). Then SAT(\( R^{3\times3}_{1/3} \)) is not harder than SAT(\( S \)).

*Proof.* Since \( \langle S \rangle = BR = \{(R_{1/3})\} \) we first construct the extension \( R^{[\text{ext}]}_{1/3} \) of \( R_{1/3} \) with the aim of showing that \( S \) implements \( R^{[\text{ext}]}_{1/3} \) with a q.p.p. implementation, and then, with a size-preserving reduction, prove that SAT(\( \{R^{[\text{ext}]}_{1/3}\}\)) is not harder than SAT(\( \{R_{1/3}\}\)). Since the arity of \( R_{1/3} \) is 3, we augment its matrix representation with the binary numbers from 0 to 7 as columns, and close the resulting relation under every polymorphism of \( R_{1/3} \).

However, the projection functions are the only polymorphisms of \( BR = \{(R_{1/3})\} \), so the relation is left unchanged and we get

\[
R^{[\text{ext}]}_{1/3} = \begin{pmatrix}
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\
0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\
1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1
\end{pmatrix}
\]

From \( \{(R_{1/3})\} = BR \) and Theorem 4.1 it follows that \( S \) implements \( R^{[\text{ext}]}_{1/3} \) via a q.p.p. implementation and hence, by Theorem 4.3, that SAT(\( \{R^{[\text{ext}]}_{1/3}\}\)) is not harder than SAT(\( S \)).

Now observe that \( R^{[\text{ext}]}_{1/3} \) is nothing else than \( R^{3\times3}_{1/3} \) with some columns duplicated and two constant columns adjoined. Hence, there is a reduction from SAT(\( \{R^{3\times3}_{1/3}\}\)) to SAT(\( \{R^{[\text{ext}]}_{1/3}\}\)) where each constraint \( R^{3\times3}_{1/3}(x_1, \ldots, x_6) \) is replaced with

\[
R^{[\text{ext}]}_{1/3}(x_1, x_2, x_3, F, x_1, x_2, x_6, x_3, x_5, x_4, T)
\]

with \( F, T \) being two variables common to all constraints. Since the number of variables is augmented only by 2, this indeed shows that SAT(\( \{R^{3\times3}_{1/3}\}\)) is not harder than SAT(\( \{R_{1/3}\}\)), which is not harder than SAT(\( S \)). \( \square \)

We are left with the relational clone \( IN_2 \) and need to make sure that the bottom partial relational clone in \( IN_2 \) is not (strictly) easier than \( R^{3\times3}_{1/3} \). We proceed in an analogous manner to the derivation of \( R^{3\times3}_{1/3} \) and define a maximal extended relation which is then pruned of superfluous columns. Analogously to the relation \( R^{3\times3}_{1/3} \) in \( BR \), we consider the relation \( R^{3\times3}_{2/4} \)

\[
\{(001110, 010101, 100101, 101010, 1100001, 1010101, 0110100, 1100101, 01101001)\} \in IN_2.
\]

**Lemma 4.2.** Let \( S \) be a constraint language such that \( \langle S \rangle = IN_2 \). Then \( S \) implements \( R^{3\times3}_{2/4} \) via a q.p.p. implementation.

*Proof.* Let \( R_{NAE} \) be not-all-equal-SAT as defined in Section 2 and recall that \( \langle \{R_{NAE}\}\rangle = IN_2 \). By Theorem 4.1 it is sufficient to show that \( R^{[\text{ext}]}_{2/4} \) can implement \( R^{3\times3}_{2/4} \) with a q.p.p. implementation. Note that the cardinality of \( R_{NAE} \) is 6 and its arity is 3, \( R_{NAE} \) will have arity \( 3 + 2^6 = 67 \) and consist of \( 6 + 6 = 12 \) tuples. Let the irredundant core \( R^{irr} \) (see [28]) of a relation \( R \) be obtained by collapsing duplicate columns in the matrix representation of \( R \). Clearly, \( R^{irr}_{NAE} \) has a q.p.p. implementation in \( R^{[\text{ext}]}_{2/4} \).

For example, if column 1 is identical with column 15 in \( R^{[\text{ext}]}_{2/4} \) we can define an equivalent relation \( R^{[\text{ext}]}_{NAE} \) by identifying the corresponding variables \( x_1 \) and \( x_{15} \) with each other in the implementation:

\[
R^{[\text{ext}]}_{NAE}(x_1, \ldots, x_{14}, x_{16}, \ldots, x_{67}) \equiv R^{[\text{ext}]}_{2/4}(x_1, \ldots, x_{14}, x_{16}, \ldots, x_{67})
\]

Hence the core \( R^{irr}_{NAE} \) consists of 12 tuples where the columns of the six first tuples are the binary numbers from 0 to 63, and the six last tuples the complements of the six first. The matrix representation is therefore as follows.

\[
R^{irr}_{NAE} = \\
\begin{pmatrix}
0 & 0 & \cdots & 1 & 1 \\
0 & 0 & \cdots & 1 & 1 \\
0 & 0 & \cdots & 1 & 1 \\
0 & 0 & \cdots & 1 & 1 \\
0 & 0 & \cdots & 1 & 1 \\
1 & 1 & \cdots & 0 & 0 \\
\vdots & \vdots & \ddots & \vdots & \vdots \\
1 & 0 & \cdots & 1 & 0
\end{pmatrix}
\]

Let \( c_1, \ldots, c_{64} \) denote the columns in the matrix representation of \( R^{irr}_{NAE} \). We must now prove that \( R^{irr}_{NAE} \) can implement \( R^{3\times3}_{2/4} \) with a q.p.p. implementation. Note that \( c_1 \) and \( c_2 \) only differ in one row (and its complement). If we identified \( c_1 \) and \( c_2 \) we would therefore obtain a relation where those two tuples were removed. But since \( R^{3\times3}_{2/4} \) only has 6 tuples we want to identify two columns that differ in 6 positions.
Let \( R \) be a \( SAT \) problem. We get a \( SAT \) problem, which is satisfiable if and only if \( \phi \) is satisfiable. Since \( SAT \) is NP-complete if and only if \( \phi \) is \( SAT \) problem, then \( SAT \) problem, where the degree of each variable is at most \( B \). This restriction is of particular interest since, for all constraint languages \( S \) such that \( SAT \) is NP-complete, \( SAT \) is NP-complete for some \( B \).

**Theorem 4.2.** Let \( S \) be a finite Boolean constraint language such that \( SAT \) is NP-complete. Then \( SAT \) is not harder than \( SAT \).

5 **SAT problems with bounded degree**

In this section, we investigate the \( SAT \) problem where restrictions are placed on the number of occurrences per variable. If \( x \) occurs in \( B \) constraints then we say that the degree of \( x \) is \( B \). If \( S \) is a constraint language, then \( SAT \) is the \( SAT \) problem where the degree of each variable is at most \( B \). This restriction is of particular interest since, for all constraint languages \( S \) such that \( SAT \) is NP-complete, \( SAT \) is NP-complete for some \( B \).

**Theorem 5.1.** (Jonsson et al. [16]) For any fixed \( S \) such that \( CSP(S) \) is NP-complete, there is an integer \( B \) such that \( CSP(S) \) is NP-complete.

The most interesting case is when \( B \) is the smallest \( B \) such that \( SAT \) is NP-complete. These values are already known for 1-in-3-SAT: for \( B = 2 \) it can be reduced to the problem of finding a perfect matching in a graph [14], but for \( B = 3 \) it is NP-complete even for planar instances [21]. It might be expected that the same holds for \( SAT \) since it is as easy as \( SAT \) since it is as easy as \( SAT \). Contrary to intuition this is however not the case: \( SAT \) is NP-complete even for \( B = 2 \). To prove this we first show that \( SAT \) is not a \( \Delta \) matroid relation.

**Definition 5.1.** (\( \Delta \)-matroid relation) Let \( R \) be a Boolean relation and \( x, y, x' \) be Boolean tuples of the same arity. Let \( d(x, y) \) be the binary difference function between \( x \) and \( y \). Then \( x' \) is a step from \( x \) to \( y \) if \( d(x, x') = 1 \) and \( d(x, y) = d(x', y) = d(x, x') \). \( R \) is a \( \Delta \)-matroid relation if it satisfies the following axiom: \( \forall x, y \in R \forall x' (x' \text{ is a step from } x \text{ to } y) \rightarrow (x' \in R \lor \exists x'' \in R \text{ which is a step from } x'. \to y). \)

**Lemma 5.1.** \( R_{i/3} \) is not a \( \Delta \)-matroid relation.

**Proof.** Let \( x = 00110 \) and \( y = 010101 \). These are both elements in \( R_{i/3} \). Let \( x' = 000110 \). Then \( d(x, x') = 1 \) and \( d(x, y) = d(x, x') + d(x', y) = 1 + 3 = 4 \), so \( x' \) is a step from \( x \) to \( y \). For \( R_{i/3} \) to be a \( \Delta \)-matroid relation either \( x' \in R_{i/3} \), or there exists a \( x'' \) which is a step
from $x'$ to $y$. Since neither of the disjuncts are true, it follows that $R_{1/3}^{x'x}$ is not a $\Delta$-matroid relation.

The hardness result then follows from Theorem 3 in Dalmau and Ford [5], which states that SAT($S$)-2 is NP-complete if $S$ contains a relation that is not $\Delta$-matroid.

**Theorem 5.2.** SAT($\{R_{1/3}^{x'x}\}$)-2 is NP-complete.

### 6 The exponential-time hypothesis

Even though SAT($\{R_{1/3}^{x'x}\}$) is the easiest NP-complete SAT($\cdot$) problem, we cannot hope to prove or disprove that SAT($\{R_{1/3}^{x'x}\}$) or SAT($\{R_{1/3}^{x'x}\}$)-2 is solvable in polynomial time since this would settle the $P = \text{NP}$ question. A more assailable question is if the problem can be solved in sub-exponential time (see Definition 6.1). If yes, then we are none the wiser about $P \neq \text{NP}$; but if no, then $P \neq \text{NP}$. As a tool for studying sub-exponential problems, Impagliazzo et al. [13] proved a sparsification lemma for $k$-SAT. Intuitively, the process of sparsification means that a SAT($S$) instance with a large number of constraints can be expressed as a disjunction of instances with a comparably small number of constraints. We prove that sparsification is possible not only for $k$-SAT, but between all finite constraint languages $S$ and $S'$ for which SAT($S$) and SAT($S'$) are NP-complete, and use this to prove that SAT($\{R_{1/3}^{x'x}\}$)-2 is sub-exponential if and only if the exponential-time hypothesis is false. Due to sparsification we can also prove that all such SAT($S$) problems are sub-exponential if and only if one of them is sub-exponential (and that this holds also in the degree-bounded case), which is a significant refinement of Impagliazzo et al.’s result when restricted to finite Boolean constraint languages $S$.

#### 6.1 Sub-exponential problems

There has been a stride in constructing faster exponential algorithms for NP-complete problems. A natural question to ask is whether there exists a constant $c$ such that a problem is solvable in $O(c^n)$, but not for any $c'$ smaller than $c$. Problems without such a sharp limit are said to be sub-exponential.

**Definition 6.1.** A constraint language $S$ is **sub-exponential** if SAT($S$) is solvable in $O(2^{\epsilon n})$ for all $\epsilon > 0$.

We now need a class of reductions that relates constraint languages based on their sub-exponential complexity. Reductions based on q.p.p. definitions are too precise to fully encompass this since they preserve exact complexity — a reduction should be able to introduce new variables as long as the resulting instance can be solved in sub-exponential time. We introduce **linear variable reductions**, which should be compared to the more complex but general class of SERF-reductions by Impagliazzo et al. [13].

**Definition 6.2.** Let $S$ and $S'$ be two finite constraint languages and $\phi$ a SAT($S$) instance with $n$ variables. A total function $f$ from SAT($S$) to SAT($S'$) is a many-one linear variable reduction, or an LV-reduction, if:

1. $\phi$ is satisfiable if and only if $f(\phi)$ is satisfiable,
2. the number of variables in $f(\phi)$, $n'$, is only increased by a linear amount, i.e. there exists a universal constant $C$ such that $n' \leq Cn$, and
3. $f(\phi)$ can be computed in $O(poly(n))$ time.

The point of the definition is that an LV-reduction between two constraint languages preserves sub-exponentiality. Hence, if SAT($S$) is sub-exponential and we know that SAT($S'$) is LV-reducible to SAT($S$), then SAT($S'$) is sub-exponential as well.

**Lemma 6.1.** Let $S$ and $S'$ be two finite constraint languages such that SAT($S$) is sub-exponential. If there exists an LV-reduction from SAT($S'$) to SAT($S$), then SAT($S'$) is sub-exponential.

**Proof.** Assume that SAT($S'$) can be solved in time $O(c^n)$ but not in $O(c^\epsilon n)$ for any $0 < \epsilon < 1$. Since SAT($S$) is sub-exponential it can be solved in time $O(c^\epsilon n)$ for all $\epsilon > 0$. Assume that the LV-reduction from SAT($S'$) to SAT($S$) implies that the resulting instance contains at most $C \cdot n$ variables where $C$ is a constant. This will make SAT($S'$) solvable in time $O(c^{C \cdot n} \epsilon r)$ for all $\epsilon > 0$ which contradicts the assumption. \(\Box\)

Let $S$ and $S'$ be two finite constraint languages such that $S \subseteq (S')$. We can then reduce SAT($S$) to SAT($S'$) by replacing each constraint from $S$ by its equivalent implementation in $S'$. Such a reduction would need $C \cdot m$ new variables, where $C$ is a constant that only depends on $S'$, and $m$ the number of constraints in the instance. If $m$ is large compared to the number of variables, $n$, this would however require more than a linear amount of new variables. We can therefore only prove that LV-reductions exist for classes of problems where $m$ is linearly bounded by the number of variables.

**Lemma 6.2.** Let $S$ and $S'$ be two finite constraint languages. If $S' \subseteq (S)$, then SAT($S'$)-B is LV-reducible to SAT($S$).

**Proof.** Let $\phi$ be a SAT($S'$)-B-instance with $n$ variables. Since each variable can occur in at most $B$ constraints there cannot be more than $n \cdot B$ constraints in total.
Each such constraint is of the form \( R(x_1, \ldots, x_k) \) where \( R \in S' \). By assumption \( R \) can then be expressed as a conjunction of constraints over \( S \cup \{=\} \) with a set of existentially quantified variables: \( \exists y_1, \ldots, y_l. \psi(Y) \), where each \( \psi \in S \cup \{=\} \) and \( Y \subseteq \{x_1, \ldots, x_k\} \cup \{y_1, \ldots, y_l\} \).

Hence the number of extra variables for each constraint depends on the relations from \( S' \). Let \( t \) denote the largest amount of variables that is required for implementing a constraint. In the worst case the total amount of new variables in the reduction is then \((n \cdot B) t\), which is linear with respect to \( n \) since \( B \) and \( t \) are fixed values.

Since the reduction only increases the amount of variables with a linear factor it is indeed an LV-reduction, which concludes the lemma. \( \square \)

This does not imply that there exists LV-reductions between SAT(S) and SAT(S') for all constraint languages \( S \) and \( S' \), since these problems are not degree-bounded, but it is a useful tool in the sparsification process.

**Definition 6.3.** Let \( S \) and \( S' \) be two finite constraint languages. We say that \( S \) is sparsifiable into \( S' \) if, for all \( \epsilon > 0 \) and for all SAT(S) instances \( \phi \) (with \( n \) variables), \( \phi \) can be expressed by a disjunctive formula \( \bigvee_{i=1}^{t} \phi_i \), where:

1. \( \phi \) is satisfiable if and only if at least one \( \phi_i \) is satisfiable,
2. \( B \) is a constant that only depends on \( \epsilon \), \( S \) and \( S' \),
3. \( \phi_i \) is a SAT(S')-B instance,
4. \( t \leq 2^{\epsilon n} \), and
5. \( \bigvee_{i=1}^{t} \phi_i \) can be computed in \( O(poly(n) \cdot 2^{\epsilon n}) \) time.

Note that nothing in the definition says that \( S \) and \( S' \) cannot be the same constraint language. If so, we simply say that \( S \) is sparsifiable. Impagliazzo et al. \[13\] prove the following for k-SAT.

**Lemma 6.3.** (sparsification lemma for k-SAT) k-SAT is sparsifiable.

### 6.2 General sparsification

Recall from Section \[2\] that we use \( \Gamma_{\text{SAT}}^k \) and \( \Gamma_{\text{NAR}}^k \) to denote the constraint languages of k-SAT and NAE-k-SAT respectively. In order to prove the general sparsification result, we first prove that \( \Gamma_{\text{NAR}}^k \) is sparsifiable, and then that all constraint languages \( S \) in \( BR \) and \( IN_2 \) for which SAT(S) is NP-complete can be sparsified by reducing them to either \( \Gamma_{\text{SAT}}^k \) or \( \Gamma_{\text{NAR}}^k \).

**Lemma 6.4.** (sparsification lemma for NAE-k-SAT) \( \Gamma_{\text{NAR}}^k \) is sparsifiable.

**Proof.** Let \( \phi \) be a SAT(\( \Gamma_{\text{NAR}}^k \)) instance with \( n \) variables. If \( \gamma_{\text{NAR}}(x_1, \ldots, x_k) \) is a constraint from \( \phi \), then it be verified that it is satisfiable if and only if \( \gamma_{\text{SAT}}(x_1, \ldots, x_k) \wedge \gamma_{\text{NAE}}(x_1, \ldots, x_k) \) is satisfiable. We can therefore form an equivalent SAT(\( \Gamma_{\text{SAT}}^k \)) instance \( \psi \) by adding the complement of every \( \gamma_{\text{NAR}} \)-constraint. By the sparsification lemma for k-SAT, it then follows that \( \phi \) can be sparsified into the disjunctive formula \( \bigvee_{i=1}^{t} \psi_i \). We must now prove that each SAT(\( \Gamma_{\text{SAT}}^k \))-B instance \( \psi_i \) is reducible to an equivalent SAT(\( \Gamma_{\text{NAR}}^k \))-B' instance for some constant \( B' \).

For simplicity, we shall first reduce each disjunct to SAT(\( \Gamma_{\text{NAR}}^{k+1} \)). For each constraint \( \gamma_{\text{NAR}}(x_1, \ldots, x_k) \in \psi_i \), we let \( \gamma_{\text{NAR}}^t(x_1, \ldots, x_k, X) \) be the corresponding \( \gamma_{\text{NAR}}^{k+1} \)-constraint, where \( X \) is a fresh variable common to all constraints. Let \( \psi'_i \) be the resulting SAT(\( \Gamma_{\text{NAR}}^{k+1} \)) instance. Then \( \psi_i \) is satisfiable if and only if \( \psi'_i \) is satisfiable: if \( \psi_i \) is satisfiable, then \( \psi'_i \) is satisfiable with \( X = 0 \); if \( \psi'_i \) is satisfiable we may assume that \( X = 0 \) since the complement of each valid assignment is also a valid assignment. But then each constraint has at least one literal that is not 0, by which it follows that \( \psi_i \) must be satisfiable.

Since \( \psi \) was sparsified, the degree of the variables in \( \psi_i \) is bounded by the constant \( B \). Hence \( X \) cannot occur in more than \( B \cdot n \) constraints. We now prove that the degree of \( X \) can be reduced to a constant value. Since \( \Gamma_{\text{NAR}}^{k+1} = IN_2 \) we can implement an equality relation that has the form \( Eq(x, y) \equiv \exists z_1, \ldots, z_r \theta \), where \( \theta \) is a conjunction of constraints over \( \Gamma_{\text{NAR}}^{k+1} \). Let \( V \) denote the highest degree of any variable in \( \theta \). We may without loss of generality assume that \( 2V < B \) since we can otherwise adjust the \( \epsilon \)-parameter in the sparsification process.

To decrease the degree of \( X \) we introduce the fresh variables \( X'_1, \ldots, X'_W \) in place of \( X \) and the following chain of equality constraints:

\[
Eq(X, X'_1) \land Eq(X'_1, X'_2) \land \ldots \land Eq(X'_{W-1}, X'_W).
\]

Let the resulting formula be \( \psi''_i \). Then \( \psi''_i \) is satisfiable if and only if \( \psi''_i \) is satisfiable since \( f(X) = f(X'_1) = \ldots = f(X'_W) \) for all satisfying assignments \( f \). Then \( W = \frac{B \cdot n}{2 \cdot 2^V} \) new variables are needed since each \( X'_j \) can occur in \( B - 2V \) additional constraints. Since each equality constraint requires \( T \) variables the whole equality chain requires \( \frac{B \cdot n}{2 \cdot 2^V} T \) variables which is linear with respect to \( n \) since \( T, B \) and \( V \) are constants.

But since \( \Gamma_{\text{NAR}}^{k+1} = (\Gamma_{\text{NAR}}^{k+1}) = IN_2 \) and no variable occurs more than \( B \) times we can use Lemma 6.2 and LV-reduce \( \psi''_i \) to an equivalent SAT(\( \Gamma_{\text{NAR}}^k \)) instance \( \phi_i \).
Since all variables in \( \psi'' \) are degree bounded by \( B \), there exists a constant \( B' \) determined by \( B \) and \( \Gamma_{\text{NAE}}^k \) such that no variable in \( \phi_i \) occurs in more than \( B' \) constraints, i.e. \( \phi_i \) is an instance of \( \text{SAT}(\Gamma_{\text{NAE}}^k)^B \). It now follows that \( \phi \) is satisfiable if and only if at least one of the disjuncts \( \phi_i \) is satisfiable. Hence \( \Gamma_{\text{NAE}}^k \) is sparsifiable.

The following auxiliary lemma establishes that for any constraint language such that \( S \subset BR \) or \( S \subset IN_2 \), it holds that \( \text{SAT}(S) \) is \( LV \)-reducible to either \( k\)-SAT or \( \text{NAE}-k\)-SAT.

**Lemma 6.5.** Let \( S \) be a finite constraint language such that \( S \subset IN_2 \) and \( S' \) be a finite constraint language such that \( S' \subset BR \). Then, \( \text{SAT}(S) \) is \( LV \)-reducible to \( \text{SAT}(\Gamma_{\text{NAE}}^k) \), for some \( k \) dependent on \( S \), and \( \text{SAT}(S') \) is \( LV \)-reducible to \( \text{SAT}(\Gamma_{\text{NAE}}^k) \), for some \( k' \) dependent on \( S' \).

**Proof.** Let \( \phi \) be an instance of \( \text{SAT}(S) \) with \( n \) variables, and \( R \in S \) be a relation with arity \( k \). By definition, \( R = \{0,1\}^k \setminus E \), where \( E \) is a set of \( k \)-ary tuples over \{0,1\} that describes the excluded tuples in the relation. Since all relations in \( S \) must be closed under complement, we can partition \( E \) into \( E_1 \) and \( E_2 \) where each tuple in \( E_2 \) is the complement of a tuple in \( E_1 \).

Let \( |E_1| = |E_2| = N \) and \( e_1, \ldots, e_N \) be an enumeration of the elements in \( E_1 \). Let \( e_i = (b_{i,1}, \ldots, b_{i,k}), b_{i,j} \in \{0,1\} \).

If \( R(x_1, \ldots, x_k) \) is a constraint in \( \phi \), then it can be expressed by the \( \text{SAT}(\Gamma_{\text{NAE}}^k) \) formula \( \psi_1 \land \ldots \land \psi_N \), where each \( \psi_i = \gamma_{\text{NAE}}^k(y_1, \ldots, y_k) \), and \( y_j = x_j \) if \( b_{i,j} = 0 \), and \( y_j = \overline{x_j} \) if \( b_{i,j} = 1 \). Each such constraint represents one of the excluded tuples in \( E_1 \) and one of the excluded tuples in \( E_2 \), and as such the formula as a whole is satisfiable if and only if \( R(x_1, \ldots, x_k) \) is satisfiable. The same procedure can be repeated for all the other relations in \( S \). Moreover, since no extra variables are introduced and the number of new constraints is linear in the size of \( \phi \) (because \( S \) is fixed and finite), the reduction is an \( LV \)-reduction.

Let \( R \in S' \) be a relation with arity \( k \) and \( \phi \) an instance of \( \text{SAT}(S') \) with \( n \) variables. By definition, \( R = \{0,1\}^k \setminus E \), where \( E \) is a set of \( k \)-ary tuples over \{0,1\} that describes the excluded tuples in the relation.

Let \( |E| = N \) and \( e_1, \ldots, e_N \in E \) be an enumeration of its elements. Let \( e_i = (b_{i,1}, \ldots, b_{i,k}), b_{i,j} \in \{0,1\} \).

If \( R(x_1, \ldots, x_k) \) is a constraint in \( \phi \), then it can be expressed by the \( \text{SAT}(\Gamma_{\text{NAE}}^k) \) formula \( \phi_1 \land \ldots \land \phi_N \), where each \( \phi_i = \gamma_{\text{NAE}}^k(y_1, \ldots, y_k) \), and \( y_j = x_j \) if \( b_{i,j} = 0 \), and \( y_j = \overline{x_j} \) if \( b_{i,j} = 1 \). Each constraint represents exactly one of the excluded tuples in \( R \), and as such the formula as a whole is satisfiable if and only if \( R(x_1, \ldots, x_k) \) is satisfiable. The same procedure can be repeated for all the other relations in \( S \). Moreover, since no extra variables are introduced and the number of new constraints is linear in the size of \( \phi \) (because \( S \) is fixed and finite), the reduction is an \( LV \)-reduction.

Since \( \text{SAT}(S) \) is \( NP \)-complete if and only if \( \langle S \rangle = BR \) or \( \langle S \rangle = IN_2 \), we can now prove that sparsification is possible between all finite constraint languages for which \( \text{SAT}(\cdot) \) is \( NP \)-complete.

**Theorem 6.1.** (General Sparsification) Let \( S \) and \( S' \) be two finite constraint languages such that \( \text{SAT}(S) \) and \( \text{SAT}(S') \) are \( NP \)-complete. Then, \( \text{SAT}(S) \) is sparsifiable into \( \text{SAT}(S') \).

**Proof.** There are a few different cases depending on which co-clones that are generated by \( S \) and \( S' \):

1. \( \langle S \rangle = \langle S' \rangle = IN_2 \)
2. \( \langle S \rangle = \langle S' \rangle = BR \)
3. \( \langle S \rangle = IN_2, \langle S' \rangle = BR \)
4. \( \langle S \rangle = BR, \langle S' \rangle = IN_2 \)

For case (1), assume that \( \langle S \rangle = \langle S' \rangle = IN_2 \). Let \( k \) denote the highest arity of a relation in \( S \). If \( \phi \) is a \( \text{SAT}(S) \) instance with \( n \) variables it can be reduced to a \( \text{SAT}(\Gamma_{\text{NAE}}^k) \) instance \( \phi' \) with the same number of variables by Lemma 6.3. Then, according to the sparsification lemma for \( \text{NAE}-k\)-SAT, there exists a disjunction of \( \text{SAT}(\Gamma_{\text{NAE}}^k)^B \) formulas such that \( \phi' = \bigvee_{i=1}^k \phi_i \). Since \( \langle S' \rangle = IN_2 \), each \( \phi_i \) can be implemented as a conjunction of constraints over \( S' \) with a linear amount of extra constraints and variables. Let \( \phi_i' \) denote each such implementation. Then \( \phi_i' \) is an instance of \( \text{SAT}(S')^B \), for some \( B' \) determined by \( B \) and \( S' \). Hence \( \text{SAT}(S) \) is sparsifiable into \( \text{SAT}(S') \).

Case (2) is analogous to case (1) but with \( \Gamma_{\text{NAE}}^k \) instead of \( \Gamma_{\text{NAE}}^k \). Case (3) follows from case (2) since all finite constraint languages are sparsifiable into \( \Gamma_{\text{NAE}}^k \) by Lemmas 6.3 and 6.3.

For case (4), assume that \( \langle S \rangle = BR \) and \( \langle S' \rangle = IN_2 \). Let \( k \) denote the relation with the highest arity in \( S \). If \( \phi \) is a \( \text{SAT}(S) \) instance with \( n \) variables it can be reduced to \( \text{SAT}(\Gamma_{\text{NAE}}^k) \) instance \( \phi' \) by lemma 6.5. Since \( \Gamma_{\text{NAE}}^k \) is sparsifiable there exists a disjunction such that \( \phi' = \bigvee_{i=1}^k \phi_i \). By recapitulating the steps from Lemma 6.4 we then can reduce each \( \phi_i \) to a \( \text{SAT}(\Gamma_{\text{NAE}}^{k+1})^B \) instance \( \phi_i' \). Then, since \( \langle S' \rangle = IN_2 \), each \( \phi_i' \) can be implemented as a conjunction of constraints over \( S' \) such that no variable occurs in more than \( B' \) constraints, where \( B' \) is determined by \( B \) and \( S' \).
Santhanam and Srinivasan [24] have shown that the unrestricted SAT problem (which corresponds to an infinite constraint language) does not admit sparsification to arbitrary finite constraint languages such that SAT(·) is NP-complete. Consequently, it is a necessary condition that the constraint languages in Theorem 6.1 are indeed finite.

6.3 SAT and the exponential-time hypothesis

The exponential-time hypothesis states that k-SAT is not sub-exponential [12] for $k \geq 3$. If one assumes that P ≠ NP, then this statement is plausible since it enforces a limit on the time complexity of exponential algorithms. Impagliazzo et al. proved that many NP-complete problems such as $k$-colorability, clique and vertex cover are as hard as k-SAT with respect to sub-exponential complexity. In this section we prove that both SAT($\{R_{i/3}^{x,x'}\}$) and SAT($\{R_{i/3}^{x,x'}\}$) are sub-exponential if and only if k-SAT is sub-exponential, and, as a consequence, that this is true for all finite constraint languages S for which SAT(S) is NP-complete, even for degree-bounded instances.

Theorem 6.2. The following statements are equivalent:

1. The exponential-time hypothesis is false.
2. SAT($\{R_{i/3}^{x,x'}\}$) is sub-exponential.
3. SAT($\{R_{i/3}^{x,x'}\}$) is sub-exponential.
4. For every finite constraint language S such that SAT(S) is NP-complete, SAT(S) is sub-exponential.
5. For every finite constraint language S such that SAT(S) is NP-complete, SAT(S)-B is sub-exponential for every B.
6. There exists a finite constraint language S such that SAT(S) is NP-complete and sub-exponential.
7. There exists a finite constraint language S such that SAT(S) is NP-complete and SAT(S)-B is sub-exponential for all B.

Proof. We will prove that $1 \implies \ldots \implies 7 \implies 1$ and hence that all statements are equivalent.

1 $\implies$ 2 If the exponential-time hypothesis is false then k-SAT is sub-exponential. But then $R_{i/3}^{x,x'}$ must also be sub-exponential since it is as easy as k-SAT, which immediately implies that SAT($\{R_{i/3}^{x,x'}\}$) is sub-exponential.

2 $\implies$ 3 We must prove that SAT($\{R_{i/3}^{x,x'}\}$) is sub-exponential if SAT($\{R_{i/3}^{x,x'}\}$)-B is sub-exponential. Let $\phi$ be a SAT($\{R_{i/3}^{x,x'}\}$) instance with $n$ variables. Due to Theorem 6.1 there exists a disjunction of SAT($\{R_{i/3}^{x,x'}\}$)-B instances such that $\phi = \bigvee_{i=1}^{t} \phi_i$, where $B$ is a constant that does not depend on $n$. Next assume that $x$ is a variable in $\phi_i$ that occurs in $2 < B' \leq B$ constraints. Since this is not a valid SAT($\{R_{i/3}^{x,x'}\}$)-B instance the degree of $x$ must be lowered to 2.

Call two variables in an $R_{i/3}^{x,x'}$ constraint complementary if one occurs in position $i = 1, 2$ or 3, and the other in position $i + 3$. It is easily verified that variables fulfilling this constraint are indeed each other’s complement. Let $C_1, \ldots, C_{B'}$ be an enumeration of the constraints that contains $x$. Without loss of generality we may assume that $x$ always occur in position 1 and that it has a single complementary variable $x'$ which occurs in position 4 in the constraints $C_1, \ldots, C_{B'}$. For each three constraints:

$$C_{i-1} = R_{i/3}^{x,x'}(x, y_{i-1}, z_{i-1}, x', y'_{i-1}, z'_{i-1}),$$
$$C_i = R_{i/3}^{x,x'}(x, y_{i-1}, z_{i-1}, x', y'_{i-1}, z'_{i-1}),$$
$$C_{i+1} = R_{i/3}^{x,x'}(x, y_{i-1}, z_{i+1}, x', y'_{i+1}, z'_{i+1}),$$

we can then lower the degree of $x$ and $x'$ by introducing the constraints $C'_{i-1}, C'_i$, and $C'_{i+1}$ which we define such that

$$C'_{i-1} = R_{i/3}^{x,x'}(x, y_{i-1}, z_{i-1}, x', y'_{i-1}, z'_{i-1}),$$
$$C'_i = R_{i/3}^{x,x'}(x, y_{i-1}, z_{i-1}, x', y'_{i-1}, z'_{i-1}),$$
$$C'_{i+1} = R_{i/3}^{x,x'}(x'', y_{i+1}, z_{i+1}, x'', y''_{i+1}, z''_{i+1}),$$

Here, $x''$ and $x'''$ are fresh variables. Since the new variables occur in the same complementary positions it follows that $f(x) = f(x'') = \neg f(x''') = \neg f(x''')$ for all satisfying assignments $f$, and that $C_{i-1}, C_i, C_{i+1}$ are satisfiable if and only if $C'_{i-1}, C'_i, C'_{i+1}$ are satisfiable. If the procedure is repeated iteratively for all $C_1, \ldots, C_{B'}$ the degree of $x$ or any newly introduced variable in $C'_1, \ldots, C'_{B'}$ is at most 2. Let $\phi_i'$ denote the formula obtained when the procedure is repeated for all variables occurring $B'$ times, $2 < B' \leq B$. The total number of variables needed is then bounded by the linear expression $B \cdot n$.

Since no variable in $\phi_i'$ occurs in more than two constraints, we can then use a sub-exponential algorithm for SAT($\{R_{i/3}^{x,x'}\}$)-2, and answer yes if and only if at least one $\phi_i'$ is satisfiable. Hence SAT($\{R_{i/3}^{x,x'}\}$) is sub-exponential if SAT($\{R_{i/3}^{x,x'}\}$)-2 is sub-exponential.
a method for SAT\((\{R^{\neq}_{i,j}\})\) and answer yes if and only if at least one of the disjuncts is satisfiable.

\[ 4 \implies 3 \text{ Trivial.} \]

\[ 5 \implies 6 \text{ Assume that SAT}(S) \text{ is NP-complete} \]
and that SAT\((S)\)-B is sub-exponential for every B. Let \(\phi\) be a SAT\((S)\) instance. Then according to Theorem 6.1, there exists a disjunction \(\bigvee_{i=1}^{\omega} \phi_i\) of SAT\((S)\)-B\(^i\) formulas, for some constant \(B^i\). Since SAT\((S)\)-B is sub-exponential for every B we can use a sub-exponential algorithm for SAT\((S)\)-B\(^i\) and answer yes if and only if at least one of the disjuncts is satisfiable.

\[ 6 \implies 7 \text{ Trivial.} \]

\[ 7 \implies 1 \text{ Assume that SAT}(S) \text{ is NP-complete} \]
and that SAT\((S)\)-B is sub-exponential for all B. According to Theorem 6.1, any instance of SAT\((\Gamma^\times_{i,j})\) can be expressed as a disjunction of SAT\((S)\)-B\(^i\) instances for some constant \(B^i\). But since SAT\((S)\)-B is sub-exponential for all B we can simply use a sub-exponential algorithm for SAT\((S)\)-B\(^i\) and answer yes if and only if at least one of the disjuncts is satisfiable. □

7 Research directions and open questions

We will now discuss some research directions and pose some open questions. After showing that SAT\((\{R^{\neq}_{i,j}\})\) is the easiest NP-complete SAT\((S)\) problem, it is tempting to try to determine bounds on its time complexity. We are currently working on a branch-and-bound algorithm for this problem and preliminary results show that this algorithm runs in \(O(\alpha^n)\) time where \(\alpha \approx 1.05\). We are fairly convinced that the time complexity can be significantly lowered by a more careful analysis of the algorithm.

Since our definition of ‘easier’ means ‘not harder than’ we have not excluded the possibility that there exists other constraint languages which are as easy as \(R^{\neq}_{i,j}\). A possible starting point would be to investigate whether the partial relational clones just above \(\langle\{R^{\neq}_{i,j}\}\rangle_\beta\) results in strictly harder SAT\((S)\) problems or whether they all are reducible to each other.

We have proved that SAT\((S)\) is sub-exponential if and only if SAT\((S)\)-B is sub-exponential for all B. This result is inconclusive since it does not rule out the possibility that a SAT\((S)\)-B problem is sub-exponential and NP-complete for some B, but that the sub-exponential property is lost for larger values. Hence, it would be interesting to (dis)prove that SAT\((S)\) is sub-exponential if and only if there exists some B such that SAT\((S)\)-B is NP-complete and sub-exponential. This holds for SAT\((\{R^{\neq}_{i,j}\})\) so it does not seem impossible that the result holds for all constraint languages. We also remark that bounding the degree of variables is not the only possible structural restriction: many attempts at establishing structurally based complexity results are based on the tree-width (or other width parameters) of some graph representation of the constraints, cf. [7, 10].

A particularly interesting example is a result that connects ETH with structural restrictions: if ETH holds, then solving the CSP problem for instances whose primal graph has treewidth \(k\) requires \(n^{\Omega(k/\log k)}\) time.

A natural continuation of this research is to generalize the methods in Section 6 to other problems. Generalizing them to constraint satisfaction problems over finite domains appears to be effortless, and such a generalization would give us a tool for studying problems such as \(k\)-colorability and its many variations. Lifting the results to infinite-domain constraints appears to be more difficult, but it may be worthwhile: Bodirsky and Grohe [1] have shown that every computational decision problem is polynomial-time equivalent to such a constraint problem. Hence, this may lead to general methods for studying the time complexity of computational problems. Another interesting generalization is to study problems that are not satisfiability problems, e.g. enumeration problems, counting problems, and non-monotonic problems such as abduction and circumscription.

As we have already mentioned, a drawback of Theorem 3.3 is that the structure of the Boolean partial clone lattice is far from well-understood (and even less well-understood when generalized to larger domains). Hence, it would be interesting to look for lattices that have a granularity somewhere in between the clone lattice and the partial clone lattice. One plausible candidate is the lattice of frozen partial clones that were introduced in Nordh and Zanuttini [23]. A frozen implementation is a primitive positive implementation where we are only allowed to existentially quantify over variables that are frozen to a constant (i.e., variables that are constant over all solutions). For more details about frozen partial clones (e.g., the Galois connection between frozen partial clones and frozen partial relational clones), we refer the reader to Nordh and Zanuttini [23].

We remark that the complexity of SAT\((S)\) is determined by the frozen partial clones and that the lattice of frozen partial clones is indeed coarser than the lattice of partial clones, as there are examples of infinite chains of partial clones that collapse to a single frozen partial clone [11, 23].

Acknowledgments

We thank Magnus Wahlström for helpful discussions on the topic of this paper.
References


