Existence versus Exploitation: The Opacity of Backdoors and Backbones

Lane A. Hemaspaandra[*]
Department of Computer Science
University of Rochester
Rochester, NY 14627, USA

David E. Narváez[†]
College of Computing and Inf. Sciences
Rochester Institute of Technology
Rochester, NY 14623, USA

October 16, 2020

Abstract

Boolean formulas often have what are known as hidden structures. We study the complexity of whether such structures (of certain sizes) exist in a formula, the complexity of getting one's hands on the structure's information, and whether even when in hand the information can be efficiently exploited. In particular, backdoors and backbones of Boolean formulas are important hidden structural properties. A natural goal, already in part realized, is that solver algorithms seek better performance by exploiting these structures.

However, the present paper is not intended to improve the performance of SAT solvers, but rather is a cautionary tale. The theme of this paper is that there is a potential chasm between the existence of such structures in the Boolean formula and being able to effectively exploit them.

This does not mean that these structures are not useful to solvers. It does mean that one must be very careful not to assume that it is computationally easy to go from the existence of information to being able to get one's hands on it and/or being able to exploit it. We construct backdoor- and backbone-based cases where, if $\mathrm{P} \neq \mathrm{NP}$, such assumptions fail. For example, if $\mathrm{P} \neq \mathrm{NP}$ then (a) there are easily recognizable families of Boolean formulas with strong backdoors that are easy to find, yet for which it is hard to determine whether the formulas are satisfiable, and (b) there are easily recognizable sets of Boolean formulas for which it is hard to determine whether they have a large backbone.

Key words: backdoors; backbones; structural complexity; hidden structures; SAT solvers.

[*]Orcid ID: 0000-0003-0659-5204. Supported in part by NSF grant CCF-2006496, a Renewed Research Stay award from the Alexander von Humboldt Foundation, and sabbatical visit support from ETH Zürich's Department of Computer Science.

[†]Corresponding author. Orcid ID: 0000-0003-3704-1060. Email: david.narvaez@computer.org.

1 Introduction

Many algorithms for the Boolean satisfiability problem exploit hidden structural properties of formulas in order to find a satisfying assignment or prove that no such assignment exists. These structural properties are called hidden because they are not explicit in the input formula; rather, they are information that is implicit. A natural question that arises then is what is the computational complexity associated with these hidden structures. In this paper we focus on two hidden structures: backbones and strong backdoors [37].

The complexity of decision problems associated with backdoors and backbones has been studied by Nishimura, Ragde, and Szeider [28], Kilby, Slaney, Thiébaux, and Walsh [25], and Dilkina, Gomes, and Sabharwal [11], among others. In particular, the work of Dilkina, Gomes, and Sabharwal [11] provides concrete examples of how backdoors can be found and used in real-world domains. Many other papers have studied, usually in more applied ways, the use of several types of backdoors in SAT solving [15, 33, 26], and also various other hidden structures of Boolean formulas have been studied in the literature [13, 1, 7]. SAT solvers are a tremendously powerful and increasingly important tool throughout AI and beyond, and understanding the extent to which hidden structures of SAT can be found and exploited is an important focus of study within AI (as for example reflected in the venues in which the papers cited in this paragraph appear).

In the present paper, we show that, under the assumption that P $\neq$ NP, there are easily recognizable families of formulas with strong backdoors that are easy to find, yet the problem of determining whether these formulas are satisfiable remains hard (in fact, NP-complete).

Hemaspaandra and Narváez [20] showed, under the (rather strong) assumption that P $\neq$ NP $\cap$ coNP, a separation between the complexity of finding backbones and that of finding the values to which the backbone variables must be set. In the present paper, we also add to that line of research by showing that, under the (less demanding) assumption that P $\neq$ NP, there are families of formulas that are easy to recognize (i.e., they can be recognized by polynomial-time algorithms) yet no polynomial-time algorithm can, given a formula from the family, decide whether the formula has a large backbone (doing so is NP-complete).

Far from being a paper that is intended to speed up SAT solvers, this is a paper trying to get a better sense of the (potential lack of) connection between properties existing and being able to get one's hands on the variables or variable settings that are the ones expressing the property's existence. That is, the paper's point is that there is a potential gap between on one hand the existence of small backdoors and large backbones, and on the other hand using those to find satisfying assignments. Indeed, the paper establishes not just that (if P $\neq$ NP) such gaps exist, but even rigorously proves that if any NP set exists that is frequently hard (with respect to polynomial-time heuristics), then sets of our sort exist that are essentially just as frequently hard; we in effect prove an inheritance of frequency-of-hardness result, under which our sets are guaranteed to be essentially as frequently hard as any set in NP is. The results mentioned in this paragraph are this paper's key contributions to the understanding of hidden properties of SAT, and thus to AI.

Our results admittedly are theoretical results, but they speak both to the importance of not viewing backdoors or backbones as magically transparent—we prove that they are in some cases rather opaque—and to the fact that the behavior we mention likely happens on quite dense sets; and, further, since we tie this to whether any set is densely hard, these SAT-solver issues due to this paper have now become inextricably linked to the extremely important, long-open question (which we will comment on a bit more as the final paragraph of this section) of how resistant to polynomial-time heuristics the hardest sets in NP can be. We are claiming that these important hidden properties—backdoors and backbones—have some rather challenging behaviors that one must at least be aware of. Indeed, what is most interesting about this paper is likely not the theoretical constructions themselves, but rather the behaviors that those constructions prove must exist unless P = NP. Knowing that those behaviors cannot be avoided unless P = NP is of important to AI, and beyond. Additionally, the behavior in one of our results is closely connected to the deterministic time complexity of SAT; in our result (Theorem 3.5) about easy-to-find hard-to-assign-values-to backdoors, we show that the backdoor size bound in our theorem cannot be improved even slightly unless NP is contained in subexponential time.

This work is somewhat surprisingly both related and unrelated to a key stream of inquiry in computer science, namely, the relationship between search and decision. Or, to be more precise, it is related to that stream, and yet expands it in a rather new way, showing the existing stream to exist within an even broader setting. Let us explain what we mean. Our paper's central existence versus exploitation point is somewhat related to the extremely important search-versus-decision issue that is an ongoing flaw in the field: Much of CS is framed in terms of decision problems (because in the early days, search and decision were so tightly related for so many problems that people thought that doing so was not a problem), but a handful of papers have started to show that even when decision is easy search may be hard. Borodin, Demers, and Valiant ([4, 36], see also [16, 35, 30]) started this research theme in the 1970s, but it has turned out to be a very difficult theme on which to make progress in natural domains of application. An exception is that in the 1990s Bellare and Goldwasser [2] showed a lovely search-vs-decision gap in the world of cryptography. However, that gap's existence required an assumption of separation of double-exponential time classes—which is an extraordinarily strong assumption. Finally, more than a quarter of a century after the work of Borodin, Demers, and Valiant, search-versus-decision separations based on the very reasonable, standard assumption that integer factoring is hard were found in the relatively natural domains of voting problems [17] and, in a precursor of this paper, backbones for SAT [20] (see relatedly the survey paper [18]). Each example of search being hard when decision is easy is an example of separating exploitation from existence, and so all those earlier works are cases of separating exploitation from existence. However, the present paper's work is in part giving separations of exploitation from existence that are *not* of the search-versus-decision sort. Most centrally, Theorem 3.5, our main result about backdoors, gives a case where the existence (decision) issue for backdoors is made trivially easy, but it also ensures that finding those backdoors (search) is easy. The particular exploitation that is being precluded there is not about finding a backdoor, but rather about it being impossible to *use* it. Our results about backbones also are not a search-versus-decision separation. In short, this paper's work expands beyond the case of search-versus-decision the study of separating exploitation from existence.

We do not believe that labeling each paper as being in a single area of CS, or drawing bright lines between areas, is possible, constructive, or important. In the best of cases, research can be not just relevant to more than one area but even can have each area—variously via contributing problems, tools, or implications—help or advance the other. That is, the helping can be a two-way street. We feel that the present paper's work on backdoors and backbones has that nature. (As a different example of such a two-way street of help between a subarea of theory and an area primarily viewed as part of AI, we mention that there is a survey that makes that case regarding computational social choice [18].) As mentioned earlier, the domain of the present paper, hidden structures of SAT and the degree to which they can or cannot be found and exploited, is an important stream of work at the central AI venues, as one can see for example by looking at the bibliography of the present paper, and the degree to which the key papers related to this work have appeared in AAAI and IJCAI. However, our study of the area certainly takes a highly theoretical approach, both in the rigor of the definition and theorem statements, and in using such proof techniques/tools as padding, one-to-one reductions, and density transfer. So a theory-influenced sensibility is helping us in our study of this area. But, as to the other direction, the study of this area is helping theory. Namely, as just mentioned we expand the "search versus decision" issue important in theory to a broader particular, "exploitation versus existence" that arguably has not been previously studied, and also our Theorems 3.7 and 4.3 (or, to be more specific, their contrapositives) provide new, indirect ways to—from what may hold regarding backdoors and backbones of SAT—obtain results about a sweepingly important and foundational open question in theory (and AI), namely, what level of resistance NP sets can have relative to heuristics. In brief, the study of backbones and backdoors of SAT on one hand, and the study of (so-called "structural") complexity on the other hand, are intertwining in this paper in a way that, we feel, is a two-way street as to support and benefits.

The rest of this paper is organized as follows. Section 2 defines the notation we will use throughout this paper. Sections 3 and 4 contain our results related to backdoors and backbones, respectively. Finally, Section 5 adds some concluding remarks.

Before going on, we briefly return, for those interested, to comment a bit more on the issue mentioned two paragraphs ago of the long-open question of how resistant to polynomial-time heuristics the hardest sets in NP can be. We mention two lines of work on this question. The first is that there are relativized worlds (aka black-box models) in which NP sets exist for which all polynomial-time heuristics are asymptotically wrong half the time [24]; heuristics basically do no better than one would do by flipping a coin to give one's answer. Indeed, that is known to hold with probability one relative to a random oracle, i.e., it holds in all but a measure zero set of possible worlds [24]. Although many researchers suspect that the same holds in the real world, proving that would separate NP from P in an extraordinarily strong way, and currently even proving that P and NP differ is viewed as likely being decades (or worse) away [14]. The second, and also related, line of work argues that SAT cannot have good heuristics unless SAT in fact is *exactly* solvable unexpectedly fast. This long line of work—whose origins date back to a 1978 paper of Berman [3]—shows for various failure rates that having a polynomial-time heuristic algorithm for SAT with that failure rate would have earthshaking consequences, such as, depending on the failure rate, surprisingly implying P=NP or surprisingly putting all of NP into subexponential time ([3, 27, 32, 5, 6]; for a survey of this, and a discussion of the extent to which it can be squared with the observations, impressions, and experiments indicating that heuristic algorithms seem to do very well in practice on real-world instances, see [23]).

2 Definitions and Notations

For a Boolean formula F, we denote by $V(F)$ the set of variables appearing in F. Adopting the notations of Williams, Gomes, and Selman [37], we use the following. A partial assignment of F is a function $a_S : S \to \{\textsc{True}, \textsc{False}\}$ that assigns Boolean values to the variables in a set $S \subseteq V(F)$ (as is usual, $S = V(F)$ is a legal case, namely, the case of a complete assignment, for which one wants the notation also to work). For a Boolean value $v \in \{\textsc{True}, \textsc{False}\}$ and a variable $x \in V(F)$, the notation $F[x/v]$ denotes the formula F after replacing every occurrence of x by v and simplifying. This extends to partial assignments, e.g., to $F[a_S]$, in the natural way; by "in the natural way," we mean simultaneously setting each variable in S to whatever a_S maps it to, and then simplifying. We just used "simplifying" twice rather casually, but to be clear and concrete, we need to in a completely unambiguous way define the exact order/process/algorithm we use for simplifying a propositional Boolean formula that, in addition to the usual aspects, has had a number of substitutions done to variables, resulting in some number of True and False atoms being part of the formula. We do that in the following paragraph.

It is important to be concrete as to what we mean by "simplifying," both for the reason mentioned in the preceding paragraph (i.e., so that all discussed algorithms/formulas are unambiguously specified), and also so that the reader knows we are not expecting simplifying to itself reshape the formulas in computationally expensive ways. What we mean by "simplifying" is that one repeatedly (until either all instances of True and False have been removed, or the formula has been simplified to be exactly True or to be exactly False) takes whatever the then-currently-leftmost instance of True or False is in the formula, and simplifies it relative to whichever logical operation or parenthesizing is the immediately controlling action within the standard parsing of the formula. So, for example, (True) simplifies to True; $\overline{\textsc{True}}$ simplifies to False; the OR of True with FOO simplifies to True; and the AND of True with FOO simplifies to FOO. Although the following does not apply in Section 3 since there all the formulas are assumed to be CNF formulas, for settings (such as Section 4) in which binary logical operators not mentioned above are allowed, the obvious analogous simplifications hold (e.g., the NXOR of True with FOO simplifies to FOO). False instances are similarly handled as to simplification, except of course using the natural simplifications that apply to that case. Note that the process of simplifying a formula in this way is mechanical, can be done in polynomial time, and after the simplification one is left with a formula that has no occurrences of True or False, or with a formula that has simplified to True, or with a formula that has simplified to False. (We will soon speak of a set-like notation for CNF formulas, but that is merely a notational issue as to representing formulas, and does not interfere with the issue of simplification.)

Example 2.1. *Consider the CNF formula* $F = (x_1 \vee \overline{x_2} \vee \overline{x_3} \vee x_5) \wedge (x_1 \vee x_2 \vee x_4 \vee x_5) \wedge (x_3 \vee \overline{x_4}) \wedge (\overline{x_1} \vee x_2 \vee x_3 \vee x_5)$*. Suppose we set* x_1 *to* False *and* x_5 *to* False*. We have* $F[x_1/\text{False}, x_5/\text{False}] = (\overline{x_2} \vee \overline{x_3}) \wedge (x_2 \vee x_4) \wedge (x_3 \vee \overline{x_4})$*.*

On the other hand, if we instead set x_3 *to* False *and* x_4 *to* True*, it is not hard to see that, by the above process,* $F[x_3/\text{False}, x_4/\text{True}]$ *simplifies to* False*.*

For a finite set A, $\|A\|$ denotes A's cardinality. For any string x, $|x|$ denotes the length of (number of characters of) x. The empty string is denoted by ϵ. For any set A, A^* denotes the Kleene closure of A, i.e., $\{\epsilon\} \cup \{y \mid (\exists n \in \{1, 2, 3, \ldots\})(\exists w_1, \ldots, w_n)[(\forall i : 1 \leq i \leq n)[w_i \in A] \wedge y = w_1 w_2 \cdots w_n]\}$. For each set T and each natural number n, $T^{\leq n}$ denotes the set of all strings in T whose length is less than or equal to n. In particular, $(\Sigma^*)^{\leq n}$ denotes the strings of length at most n, over the alphabet Σ.

We quickly review the standard reduction, hardness, and completeness notions that we will use in this paper (see, e.g., [29, 22]). A set A over an alphabet Σ *many-one polynomial-time reduces* to a set B if there is a polynomial-time function f such that, for each string $x \in \Sigma^*$, it holds that

$$x \in A \iff f(x) \in B,$$

A set B is said to be *NP-hard* if $B \in$ NP and, for each set $A \in$ NP, it holds that A many-one polynomial-time reduces to a B. A set B is said to be *NP-complete* if $B \in$ NP and B is NP-hard. A particularly important NP-complete set is SAT, the set of all propositional Boolean formulas that are satisfiable.

For the sake of being self-contained, we give here the definition of backbones as presented by Williams, Gomes, and Selman [37]. We restrict ourselves to the Boolean domain, since we only deal with Boolean formulas in this paper.

Definition 2.2 (Backbone [37])**.** *For a Boolean formula* F*, a subset* S *of its variables is a* backbone *if there is a unique partial assignment* a_S *such that* $F[a_S]$ *is satisfiable.*

The *size* of a backbone S is the number of variables in S. One can readily see from Definition 2.2 that all satisfiable formulas have at least one backbone, namely, the empty set. This backbone is called the *trivial* backbone, while backbones of size at least one are called *nontrivial* backbones. It follows from Definition 2.2 that unsatisfiable formulas do not have backbones. Note also that some satisfiable formulas have no nontrivial backbones, e.g., $x_1 \vee x_2 \vee x_3$ is satisfiable but has no nontrivial backbone.

Example 2.3. *Consider the formula* $F = x_1 \wedge (x_1 \leftrightarrow \overline{x_2}) \wedge (x_2 \leftrightarrow x_3) \wedge (x_2 \vee x_4 \vee x_5)$*. Any satisfying assignment of* F *must have* x_1 *set to* True*, which in turn constrains* x_2 *and* x_3*.*

Then $\{x_1, x_2, x_3\}$ is a backbone of F, as is any subset of this backbone. It is also easy to see that $\{x_1, x_2, x_3\}$ is the largest backbone of this formula since the truth values of x_4 and x_5 are not entirely constrained in F (since F in effect is—once one applies the just-mentioned forced assignments—$x_4 \vee x_5$).

The definition of backdoors is tied to the definition/notion of a subsolver, and we cover both those definitions, with examples, at the start of the next section. We locate those there to help avoid confusion about which parts of the paper are focused on CNF formulas and which are focused on general formulas.

3 Results on Backdoors to CNF Formulas

This section contains our results on backdoors to CNF formulas.

3.1 Subsolvers and Strong Backdoors

Throughout Section 3 our focus will be on Boolean formulas in conjunctive normal form (CNF). A CNF formula is a conjunction of disjunctions, and the disjunctions are called the *clauses* of the formula. Following Dilkina, Gomes, and Sabharwal [11], we define satisfiability of CNF formulas using the language of set theory. This is done by formalizing the intuition that, in order for an assignment to satisfy a CNF formula, it must set at least one literal in every clause to True. One can then define a CNF formula F to be a collection of clauses, each clause being a set of literals. $F \in \mathrm{SAT}$ if and only if there exists an assignment $a_{V(F)}$ such that for all clauses $C \in F$ there exists a literal $l \in C$ such that $a_{V(F)}$ maps l to True. Under this formalization, to be in harmony with the standard conventions that the truth value of the empty conjunctive (resp., disjunctive) formula is True (resp., False), F must be taken to be in SAT if F is empty (since the empty CNF formula must be taken to be True as a consequence of the fact that the empty conjunctive formula is taken to be True) and F must be taken to be in $\overline{\mathrm{SAT}}$ if $\emptyset \in F$ (since that empty clause is an empty disjunctive formula and so by convention is False, and thus F evaluates to False); these two cases are called, respectively, F being trivially True and F being trivially False (as the conventions as just mentioned put these cases not just in SAT and $\overline{\mathrm{SAT}}$ but fix the truth values of the represented formulas to be True and False). (For completeness, let us say that when converting to set notation the atomic, degenerate formula True it becomes $\{\}$, and that when converting to set notation the atomic, degenerate formula False it becomes $\{\emptyset\}$.)

Example 3.1. *Let us return to the CNF formula, F, from Example 2.1. Recall that*

$$F = (x_1 \vee \overline{x_2} \vee \overline{x_3} \vee x_5) \wedge (x_1 \vee x_2 \vee x_4 \vee x_5) \wedge (x_3 \vee \overline{x_4}) \wedge (\overline{x_1} \vee x_2 \vee x_3 \vee x_5).$$

We can express this formula in our set theory notation as $F = \{\{x_1, \overline{x_2}, \overline{x_3}, x_5\}, \{x_1, x_2, x_4, x_5\}, \{x_3, \overline{x_4}\}, \{\overline{x_1}, x_2, x_3, x_5\}\}$.

In Example 2.1, we already commented that $F[x_3/\textsc{False}, x_4/\textsc{True}] = \textsc{False}$. However, to give another example of the set notation, we mention here the (weaker) fact that if one were to merely locally simplify within each clause in the ways induced by that assignment to those two variables, and to then eliminate the clauses made true by that, the clause set one would end up with would be $\{\emptyset, \{\overline{x_1}, x_2, x_5\}\}$, which of course is unsatisfiable because it contains the empty set as one of its clauses.

Since CNF-SAT (the satisfiability problem restricted to CNF formulas) is well-known to be NP-complete, a polynomial-time algorithm to determine the satisfiability of CNF formulas is unlikely to exist. Nevertheless, there are several restrictions of CNF formulas for which satisfiability can be decided in polynomial time, e.g., 2-CNFSAT [8] and Horn formulas [12] (see also Schaefer's dichotomy theorem [31]). When a formula does not belong to any of these restrictions, it may have a set of variables that, once the formula is simplified over a partial assignment of these variables, the resulting formula belongs to one of these tractable restrictions. A formalization of this idea is the concept of backdoors.

Definition 3.2 (Subsolver [37]). *A polynomial-time algorithm A is a* subsolver *if, for each input formula F, A satisfies the following conditions.*

1. *A either rejects the input F (this indicates that it declines to make a statement as to whether F is satisfiable) or* determines *F (i.e., A returns a satisfying assignment if F is satisfiable and A proclaims F's unsatisfiability if F is unsatisfiable).*
2. *If F is trivially* True *A determines F, and if F is trivially* False *A determines F.*[1]
3. *If A determines F, then for each variable x and each value v, A determines $F[x/v]$.*

Definition 3.3 (Strong Backdoor [37]). *For a Boolean formula F, a nonempty subset S of its variables is a* strong backdoor *for a subsolver A if, for all partial assignments a_S, A determines $F[a_S]$ (i.e., if $F[a_S]$ is satisfiable A returns a satisfying assignment and if $F[a_S]$ is unsatisfiable A proclaims its unsatisfiability).*

Many examples of subsolvers can be found in the literature (for instance, in Table 1 of [11]). The subsolver that is of particular relevance to this paper is the *unit propagation subsolver*, which focuses on *unit clauses*. Unit clauses are clauses with just one literal. They play an important role in the process of finding models (i.e., satisfying assignments) because the literal in that clause must be set to True in order to find a satisfying assignment. The process of finding a model by searching for a unit clause (for specificity and to ensure that it runs in polynomial time, let us say that our unit propagation subsolver always focuses on the unit clause in the current formula whose encoding is the lexicographically least among the encodings of all unit clauses in the current formula), fixing the value of the variable in the unit clause, and simplifying the formula resulting from that assignment is known in the satisfiability literature as unit propagation. Unit propagation is an important

[1]Recall that if F is unsatisfiable, then "A determines F" means, perhaps somewhat confusingly as regards the plain English meaning of "determines," that A proclaims F's unsatisfiability.

building block in the seminal DPLL algorithm for SAT [10, 9]. Notice that the CNF formulas whose satisfiability can be decided by just applying unit propagation iteratively constitute a tractable restriction of SAT. The unit propagation subsolver attempts to decide the satisfiability of an input formula by using only unit propagation and empty clause detection. If satisfiability cannot be decided this way, the subsolver rejects the input formula. Szeider [34] has classified the parameterized complexity of finding backdoors with respect to the unit propagation subsolver.

As an illustration of strong backdoors under the unit propagation subsolver, Example 3.4 will show that for the formula F from Example 3.1 the three-element set $\{x_1, x_3, x_5\}$ is the unique lowest-cardinality strong backdoor of F with respect to the unit propagation subsolver.

Example 3.4. *Consider the formula F from Example 3.1. We will show that $\{x_1, x_3, x_5\}$ is the smallest strong backdoor of F with respect to the unit propagation subsolver. In fact, we will show that this three-variable subset of $V(F)$ is a strong backdoor of F with respect to the unit propagation subsolver, that no other three-variable subset of $V(F)$ is a strong backdoor of F with respect to the unit propagation subsolver, and that no two-variable subset of $V(F)$ (and thus also that no one-variable subset of $V(F)$) is a strong backdoor of F with respect to the unit propagation subsolver.*

Our first step will be to show that $\{x_1, x_3, x_5\}$ is a strong backdoor of F with respect to the unit propagation subsolver, by analyzing all the possible assignments of these three variables. (Those familiar with backdoors may wish to skip the rest of this paragraph and simply look at Table 1, which gives the analysis compactly in a table. Those less familiar with backdoors may wish to read the rest of this paragraph, which covers the argument in a more explanatory fashion.) Suppose x_1 is set to True *and notice that $F[x_1/\text{True}] = \{\{x_3, \overline{x_4}\}, \{x_2, x_3, x_5\}\}$. From there it is easy to see that if x_3 is set to* True*, the resulting formula after simplification is trivially satisfiable. If x_3 is set to* False*, setting x_5 to* True *yields the formula $\{\{\overline{x_4}\}\}$ after simplification and the satisfiability of this formula can be determined by the unit propagation subsolver. Setting x_5 to* False *yields a formula with two unit clauses, $\{\{\overline{x_4}\}, \{x_2\}\}$. The unit propagation subsolver will pick the unit clause $\{x_2\}$,*[2] *set x_2 to* True *and simplify, and will then pick the (sole) remaining unit clause, $\{\overline{x_4}\}$, and set x_4 to* False *and simplify to obtain a trivially satisfiable formula. Now suppose x_1 is set to* False *and notice that $F[x_1/\text{False}] = \{\{\overline{x_2}, \overline{x_3}, x_5\}, \{x_2, x_4, x_5\}, \{x_3, \overline{x_4}\}\}$. If we now set x_3 to* True*, notice that $F[x_1/\text{False}, x_3/\text{True}] = \{\{\overline{x_2}, x_5\}, \{x_2, x_4, x_5\}\}$. If we set x_5 to* True *F simplifies to a trivially satisfiable formula. If we set x_5 to* False*, the formula simplifies to $\{\{\overline{x_2}\}, \{x_2, x_4\}\}$. The unit propagation subsolver will pick the unit clause $\{\overline{x_2}\}$, set x_2 to* False*, and the resulting formula after simplification will be $\{\{x_4\}\}$ whose satisfiability can be determined by the unit propagation subsolver. If we set x_3 to* False*, notice that $F[x_1/\text{False}, x_3/\text{False}] = \{\{x_2, x_4, x_5\}, \{\overline{x_4}\}\}$. If we now set x_5 to* True *and simplify, the resulting formula would be $\{\{\overline{x_4}\}\}$ whose satisfiability can be determined by the unit propagation subsolver. If we set x_5 to* False *and simplify, the resulting formula*

[2]Here we assume that a clause $\{x\}$ precedes a clause $\{y\}$ in lexicographical order if x precedes y in lexicographical order.

Assignment $a_{\{x_1,x_3,x_5\}}$			Unit Propagation on $F[a_{\{x_1,x_3,x_5\}}]$		
x_1	x_3	x_5	Step 1	Step 2	Step 3
FALSE	FALSE	FALSE	$\{\{x_2, x_4\}, \{\overline{\mathbf{x_4}}\}\}$	$\{\{\mathbf{x_2}\}\}$	$\{\}$
FALSE	FALSE	TRUE	$\{\{\overline{\mathbf{x_4}}\}\}$	$\{\}$	
FALSE	TRUE	FALSE	$\{\{\overline{\mathbf{x_2}}\}, \{x_2, x_4\}\}$	$\{\{\mathbf{x_4}\}\}$	$\{\}$
FALSE	TRUE	TRUE	$\{\}$		
TRUE	FALSE	FALSE	$\{\{\overline{x_4}\}, \{\mathbf{x_2}\}\}$	$\{\{\overline{\mathbf{x_4}}\}\}$	$\{\}$
TRUE	FALSE	TRUE	$\{\{\mathbf{x_4}\}\}$	$\{\}$	
TRUE	TRUE	FALSE	$\{\}$		
TRUE	TRUE	TRUE	$\{\}$		

Table 1: Let $F = \{\{x_1, \overline{x_2}, \overline{x_3}, x_5\}, \{x_1, x_2, x_4, x_5\}, \{x_3, \overline{x_4}\}, \{\overline{x_1}, x_2, x_3, x_5\}\}$ be the CNF formula from Example 3.1. All assignments $a_{\{x_1,x_3,x_5\}}$ lead to formulas $F[a_{\{x_1,x_3,x_5\}}]$ whose satisfiability can be determined by the unit propagation subsolver. More specifically, for every assignment $a_{\{x_1,x_3,x_5\}}$, unit propagation on $F[a_{\{x_1,x_3,x_5\}}]$ leads to the trivially-satisfiable CNF formula $\{\}$ in at most 3 steps. The table shows the CNF formulas obtained at each step of the execution of the unit propagation subsolver on the formulas $F[a_{\{x_1,x_3,x_5\}}]$. The unit clauses picked by the unit propagation subsolver are typeset in boldface. Here we assume that a clause $\{x\}$ precedes a clause $\{y\}$ in lexicographical order if x precedes y in lexicographical order.

would contain the unit clause $\{\overline{x_4}\}$. The unit propagation subsolver would then set the value of x_4 to FALSE *and simplify, yielding the formula $\{\{x_2\}\}$, whose satisfiability can also be determined by the unit propagation subsolver. As mentioned earlier, Table 1 summarizes the above analysis.*

It should be clear from the case analysis above that just setting the values of x_1 and x_3 is not enough for the unit propagation subsolver to always be able to determine the satisfiability of the resulting formula. In particular, note that when x_1 is set FALSE *and x_3 is set* TRUE*, F simplifies to $\{\{\overline{x_2}, x_5\}, \{x_2, x_4, x_5\}\}$, which is a formula on which the unit propagation subsolver rejects (i.e., fails to reach a determination as to whether F is satisfiable).*

Furthermore, a similar analysis done on each two-element subset of $V(F)$ shows that each strong backdoor of F with respect to the unit propagation subsolver must have cardinality at least three. In particular, Table 2 shows that setting any pair of variables in $V(F)$ to FALSE *leads to a formula whose satisfiability cannot be determined by the unit propagation subsolver.*

Finally, Table 3 shows that for every cardinality-three subset S of the variables in $V(F)$, except $\{x_1, x_3, x_5\}$, there is at least one assignment a_S for which the unit propagation subsolver cannot determine the satisfiability of the CNF formula $F[a_S]$, and so S is not a strong backdoor of F with respect to the unit propagation subsolver.

S	Unit Propagation on $F[a_S]$, where a_S is the function that sets each of the variables in S to False Step 1	 Step 2
$\{x_1, x_2\}$	$\{\{x_4, x_5\}, \{x_3, \overline{x_4}\}\}$	
$\{x_1, x_3\}$	$\{\{x_2, x_4, x_5\}, \{\mathbf{\overline{x_4}}\}\}$	$\{\{x_2, x_5\}\}$
$\{x_1, x_4\}$	$\{\{\overline{x_2}, \overline{x_3}, x_5\}, \{x_2, x_5\}\}$	
$\{x_1, x_5\}$	$\{\{\overline{x_2}, \overline{x_3}\}, \{x_2, x_4\}, \{x_3, \overline{x_4}\}\}$	
$\{x_2, x_3\}$	$\{\{x_1, x_4, x_5\}, \{\mathbf{\overline{x_4}}\}, \{\overline{x_1}, x_5\}\}$	$\{\{x_1, x_5\}, \{\overline{x_1}, x_5\}\}$
$\{x_2, x_4\}$	$\{\{x_1, x_5\}, \{\overline{x_1}, x_3, x_5\}\}$	
$\{x_2, x_5\}$	$\{\{x_1, x_4\}, \{x_3, \overline{x_4}\}, \{\overline{x_1}, x_3\}\}$	
$\{x_3, x_4\}$	$\{\{x_1, x_2, x_5\}, \{\overline{x_1}, x_2, x_5\}\}$	
$\{x_3, x_5\}$	$\{\{x_1, x_2, x_4\}, \{\mathbf{\overline{x_4}}\}, \{\overline{x_1}, x_2\}\}$	$\{\{x_1, x_2\}, \{\overline{x_1}, x_2\}\}$
$\{x_4, x_5\}$	$\{\{x_1, \overline{x_2}, \overline{x_3}\}, \{x_1, x_2\}, \{\overline{x_1}, x_2, x_3\}\}$	

Table 2: Let $F = \{\{x_1, \overline{x_2}, \overline{x_3}, x_5\}, \{x_1, x_2, x_4, x_5\}, \{x_3, \overline{x_4}\}, \{\overline{x_1}, x_2, x_3, x_5\}\}$ be the CNF formula from Example 3.1. In each case here, setting the variables in S to False leads to a formula for which the unit propagation subsolver cannot determine their satisfiability. The unit clauses picked by the unit propagation subsolver are typeset in boldface.

Variables (b, c, d) in S	$(a_S(b), a_S(c), a_S(d))$	$F[a_S]$
(x_1, x_2, x_3)	(False, False, True)	$\{\{x_4, x_5\}\}$
(x_1, x_2, x_4)	(False, True, False)	$\{\{\overline{x_3}, x_5\}\}$
(x_1, x_2, x_5)	(False, False, True)	$\{\{x_3, \overline{x_4}\}\}$
(x_1, x_3, x_4)	(False, False, False)	$\{\{x_2, x_5\}\}$
(x_1, x_4, x_5)	(True, False, False)	$\{\{x_2, x_3\}\}$
(x_2, x_3, x_4)	(False, False, False)	$\{\{x_1, x_5\}, \{\overline{x_1}, x_5\}\}$
(x_2, x_3, x_5)	(False, True, False)	$\{\{x_1, x_4\}\}$
(x_2, x_4, x_5)	(True, False, False)	$\{\{x_1, \overline{x_3}\}\}$
(x_3, x_4, x_5)	(False, False, False)	$\{\{x_1, x_2\}, \{\overline{x_1}, x_2\}\}$

Table 3: Let $F = \{\{x_1, \overline{x_2}, \overline{x_3}, x_5\}, \{x_1, x_2, x_4, x_5\}, \{x_3, \overline{x_4}\}, \{\overline{x_1}, x_2, x_3, x_5\}\}$ be the CNF formula from Example 3.1. For every 3-set S of variables in $V(F)$ with the exception of $\{x_1, x_3, x_5\}$ there is a partial assignment a_S such that the satisfiability of $F[a_S]$ cannot be determined by the unit propagation subsolver. Thus this table shows that no 3-set of variables in $V(F)$ is a candidate for a strong backdoor of F with respect to the unit propagation subsolver, except possibly for $\{x_1, x_3, x_5\}$. We show in Example 3.4 that $\{x_1, x_3, x_5\}$ is in fact a strong backdoor with resect to the unit propagation subsolver.

3.2 Backdoor Results

We are now ready to prove our main result about backdoors: Under the assumption that P $\neq$ NP, there are families of Boolean formulas that are easy to recognize and have strong unit propagation backdoors that are easy to find, yet deciding whether the formulas in these families are satisfiable remains NP-complete.

Theorem 3.5. *If* P $\neq$ NP*, for each* $k \in \{1, 2, 3, \ldots\}$ *there is a set* A *of Boolean formulas such that all the following hold.*

1. $A \in \mathrm{P}$ *and* $A \cap \mathrm{SAT}$ *is* NP*-complete.*
2. *Each formula* G *in* A *has a strong backdoor* S *with respect to the unit propagation subsolver, with* $\|S\| \leq \|V(G)\|^{\frac{1}{k}}$*.*
3. *There is a polynomial-time algorithm that, given* $G \in A$*, finds a strong backdoor having the property stated in item 2 of this theorem.*

Proof. For $k = 1$ the theorem is trivial, so we henceforward consider just the case where $k \in \{2, 3, \ldots\}$. Consider (since in the following set definition F is specified as being in CNF, we can safely start the following with "$F\wedge$" rather than for example "$(F)\wedge$") $A \in \mathrm{P}$ defined by $A = \{F \wedge (\mathtt{new}_1 \wedge \cdots \wedge \mathtt{new}_{\|V(F)\|^k - \|V(F)\|}) \mid F \text{ is a CNF formula}\}$, where $\mathtt{new}_i$ is the ith (in lexicographical order) legal variable name that does not appear in F. For instance, if F contains literals $\overline{x_1}$, x_2, x_3, and $\overline{x_3}$, and if our legal variable universe is $x_1, x_2, x_3, x_4, \ldots$, then $\mathtt{new}_1$ would be x_4. The backdoor is the set of variables of F, which can be found in polynomial time by parsing. It is clear that the formula resulting from simplification after assigning values to all the variables of F only has unit clauses and potentially an empty clause, so satisfiability for this formula can be decided by the unit propagation subsolver. Finally, it is easy to see that $F \wedge (\mathtt{new}_1 \wedge \cdots \wedge \mathtt{new}_{\|V(F)\|^k - \|V(F)\|}) \in \mathrm{SAT} \Leftrightarrow F \in \mathrm{SAT}$ so, since the formula-part that is being postpended to F can easily be polynomial-time constructed given F, under the assumption that P $\neq$ NP deciding satisfiability for the formulas in A is hard. $\square$

We mention in passing that one can change "Boolean" to "Boolean CNF" in Theorem 3.5's statement, via adjusting appropriately the use of parentheses in the proof's definition of A to ensure that A itself is in CNF whenever F is.

Let us address two natural worries the reader might have regarding Theorem 3.5. First, the reader might worry that the hardness spoken of in the theorem occurs very infrequently (e.g., perhaps except for just one string at every double-exponentially spaced length everything is easy). That is, are we giving a worst-case result that deceptively hides a low typical-case complexity? We are not (unless all of NP has easy typical-case complexity): we show that if any set in NP is frequently hard with respect to polynomial-time heuristics, then a set of our sort is almost as frequently hard with respect to polynomial-time heuristics. We will show this as Theorem 3.7.

But first let us address a different worry. Perhaps some readers will feel that the fact that Theorem 3.5 speaks of backdoors of size bounded by a fixed kth root in size is a weakness, and that it is disappointing that the theorem does not establish its same result for a stronger bound such as "constant-sized backdoors," or if not that then polylogarithmic-sized, or if not that then at least ensuring that not just each fixed root is handled in a separate construction/set but that a single construction/set should assert/achieve the case of a growth rate that is asymptotically less than every root. Those are all fair and natural to wonder about. However, we claim that not one of those improvements of Theorem 3.5 can be proven without revolutionizing the deterministic speed of SAT. In particular, the following result holds, showing that those three cases would respectively put NP into P, quasipolynomial time, and subexponential time.

Theorem 3.6. 1. *[Constant case] Suppose there is a* $k \in \{1, 2, 3, \ldots\}$ *and a set* A *of Boolean formulas such that all the following hold: (a)* $A \in \mathrm{P}$ *and* $A \cap \mathrm{SAT}$ *is* NP*-complete; (b) each formula* G *in* A *has a strong backdoor* S *with respect to the unit propagation subsolver, with* $\|S\| \leq k$*; and (c) there is a polynomial-time algorithm that, given* $G \in A$*, finds a strong backdoor having the property stated in item (b). Then* $\mathrm{P} = \mathrm{NP}$.

2. *[Polylogarithmic case] Suppose there is a function* $s(n)$*, with* $s(n) = (\log n)^{\mathcal{O}(1)}$*, and a set* A *of Boolean formulas such that all the following hold: (a)* $A \in \mathrm{P}$ *and* $A \cap \mathrm{SAT}$ *is* NP*-complete; (b) each formula* G *in* A *has a strong backdoor* S *with respect to the unit propagation subsolver, with* $\|S\| \leq s(\|V(G)\|)$*; and (c) there is a polynomial-time algorithm that, given* $G \in A$*, finds a strong backdoor having the property stated in item (b). Then* NP *is in quasipolynomial time, i.e.,* $\mathrm{NP} \subseteq \bigcup_{c>0} \mathrm{DTIME}[2^{(\log n)^c}]$.

3. *[Subpolynomial case] Suppose there is a polynomial-time computable function* r *and a set* A *of Boolean formulas such that all the following hold: (a) for each* $k \in \{1, 2, 3, \ldots\}$*,* $r(0^n) = \mathcal{O}(n^{\frac{1}{k}})$*; (b)* $A \in \mathrm{P}$ *and* $A \cap \mathrm{SAT}$ *is* NP*-complete; (c) each formula* G *in* A *has a strong backdoor* S *with respect to the unit propagation subsolver, with* $\|S\| \leq r(0^{\|V(G)\|})$*; and (d) there is a polynomial-time algorithm that, given* $G \in A$*, finds a strong backdoor having the property stated in item (c). Then* NP *is in subexponential time, i.e.,* $\mathrm{NP} \subseteq \bigcap_{\epsilon>0} \mathrm{DTIME}[2^{n^{\epsilon}}]$.

Here is a brief proof sketch. Consider the first part of the theorem, i.e., the "Constant case." Let k be the constant of that part. Then there are at most $\binom{N}{k}$ ways of choosing k of the variables of a given Boolean formula of N bits (and thus of at most N variables). And for each of those ways, we can try all 2^k possible ways of setting those variables. This is $\mathcal{O}(N^k)$ items to test—a polynomial number of items. If the formula is satisfiable, then via unit propagation one of these must yield a satisfying assignment (in polynomial time). Yet the set $A \cap \mathrm{SAT}$ was NP-complete by the first condition of the theorem. So we have that $\mathrm{P} = \mathrm{NP}$, since we just gave a polynomial-time algorithm for $A \cap \mathrm{SAT}$. The proofs of the theorem's remaining two cases are analogous (except in the final case, we in the theorem needed to put in the indicated polynomial-time constraint on the bounding function r since otherwise it could be badly behaved; that issue does not affect the second part of the theorem since even a badly behaved function s of the second part is bounded above by a simple-to-compute function s' satisfying $s'(n) = (\log n)^{\mathcal{O}(1)}$ and we can use s' in place of s in the proof).

Even the final part of the above theorem, which is the part that has the weakest hypothesis, implies that NP is in subexponential time. However, it is widely suspected that the NP-complete sets lack subexponential-time algorithms. And so we have established that $n^{1/k}$ growth, which we *do* prove in Theorem 3.5, is the smallest bound in part 2 of that result that one can reasonably hope to prove Theorem 3.5 for. Doing better would as a side effect put NP into a deterministic time class so small that we would have a revolutionarily fast deterministic algorithm for SAT.

Moving on, we now, as promised above, address the frequency of hardness of the sets we define in Theorem 3.5, and show that if any set in NP is frequently hard then a set of our type is almost-as-frequently hard. (Recall that, when n's universe is the naturals as it is in the following theorem, "for almost every n" means "for all but at most a finite number of natural numbers n.") We will say that a (decision) algorithm errs with respect to B on an input x if the algorithm disagrees with B on x, i.e., if the algorithm accepts x yet $x \notin B$ or the algorithm rejects x yet $x \in B$.

Theorem 3.7. *If h is any nondecreasing function and for some set $B \in \mathrm{NP}$ it holds that each polynomial-time algorithm errs with respect to B, at infinitely many lengths n (resp., for almost every length n), on at least $h(n)$ of the inputs up to that length, then there will exist an $\epsilon > 0$ and a set $A \in \mathrm{P}$ of Boolean formulas satisfying the conditions of Theorem 3.5, yet being such that each polynomial-time algorithm g, at infinitely many lengths n (resp., for almost every length n), will fail to determine membership in $A \cap \mathrm{SAT}$ for at least $h(n^{\epsilon})$ inputs of length at most n.*

Before getting to the proof of this theorem, let us give concrete examples that give a sense about what the theorem is saying about density transference. It follows from Theorem 3.7 that if there exists even one NP set such that each polynomial-time heuristic algorithm asymptotically errs exponentially often up to each length (i.e., has $2^{n^{\Omega(1)}}$ errors), then there are sets of our form that in the same sense fool each polynomial-time heuristic algorithm exponentially often. As a second example, it follows from Theorem 3.7 that if there exists even one NP set such that each polynomial-time heuristic algorithm asymptotically errs quasipolynomially often up to each length (i.e., has $n^{(\log n)^{\Omega(1)}}$ errors), then there are sets of our form that in the same sense fool each polynomial-time heuristic algorithm quasipolynomially often. Since almost everyone suspects that some NP sets are quasipolynomially and indeed even exponentially densely hard, one must with equal strength of belief suspect that there are sets of our form that are exponentially densely hard.

Proof of Theorem 3.7. For conciseness and to avoid repetition, we build this proof on top of a proof (namely, of Theorem 4.3) that we will give later in the paper. That later proof does not rely directly or indirectly on the present theorem/proof, so there is no circularity at issue here. However, readers wishing to read the present proof should probably delay doing that until after they have first read that later proof.

We define r_B as in the proof of Theorem 4.3 (the r_B given there draws on a construction from Appendix A of [19], and due to that construction's properties outputs only conjunctive normal form formulas). For a given k, we define $A = \{r_B(x) \wedge (\mathtt{new}_1 \wedge \cdots \wedge \mathtt{new}_{\|V(r_B(x))\|^k - \|V(r_B(x))\|}) \mid x \in \Sigma^*\}$, and since $r_B(x) \wedge (\mathtt{new}_1 \wedge \cdots \wedge \mathtt{new}_{\|V(r_B(x))\|^k - \|V(r_B(x))\|}) \in \mathrm{SAT} \Leftrightarrow r_B(x) \in \mathrm{SAT}$ and $r_B(x) \in \mathrm{SAT} \Leftrightarrow x \in B$, we can now proceed as in the proof of Theorem 4.3, since here too the tail's length is polynomially bounded. □

4 Results on Backbones

Our first result on backbones states that if $\mathrm{P} \neq \mathrm{NP}$ then there are families of Boolean formulas that are easy to recognize, with the property that deciding whether a formula in these families has a large backbone is NP-complete (and so is hard). As a corollary to its proof, we have that if $\mathrm{P} \neq \mathrm{NP}$ then there are families of Boolean formulas that are easy to recognize, with the property that deciding whether a formula in these families has a nontrivial backbone is NP-complete (and so is hard).[3]

Theorem 4.1. *For any real number $0 < \beta < 1$, there is a set $A \in \mathrm{P}$ of Boolean formulas such that the language $L_A = \{F \mid F \in A$ and F has a backbone S with $\|S\| \geq \beta\|V(F)\|\}$ is NP-complete (and so if $\mathrm{P} \neq \mathrm{NP}$ then L_A is not in P).*

Corollary (to the Proof) 4.2. *There is a set $A \in \mathrm{P}$ of Boolean formulas such that the language $L_A = \{F \mid F \in A$* and F has a nontrivial backbone $S\}$ *is NP-complete (and so if $\mathrm{P} \neq \mathrm{NP}$ then L_A is not in P).*

Proof of Theorem 4.1 and Corollary 4.2. We will first prove Theorem 4.1, and then will note that Corollary 4.2 follows easily as a corollary to the proof/construction.

[3]We have not been able to find Corollary (to the Proof) 4.2 in the literature. Certainly, two things that on their surface might seem to be the claim we are making in Corollary (to the Proof) 4.2 are either trivially true or are in the literature. However, upon closer inspection they turn out to be quite different from our claim.

In particular, if one removes the word "nontrivial" from Corollary (to the Proof) 4.2's statement, and one is in the model in which every satisfiable formula is considered to have the empty collection of variables as a backbone and every unsatisfiable formula is considered to have no backbones, then the thus-altered version of Corollary (to the Proof) 4.2 is clearly true, since if one with those changes takes A to be the set of all Boolean formulas, then the theorem degenerates to the statement that if $\mathrm{P} \neq \mathrm{NP}$, then SAT is (NP-complete, and) not in P.

Also, it is stated in Kilby et al. [25] that finding a backbone of CNF formulas is "NP-hard." However, though this might seem to be our result, their claim and model differ from ours in many ways, making this a quite different issue. First, their hardness refers to Turing reductions (and in contrast our paper is about many-one reductions and many-one completeness). Second, they are not even speaking of NP-Turing-hardness—much less NP-Turing-completeness—in the standard sense since their model is assuming a function reply from the oracle rather than having a set as the oracle. Third, even their notion of backbones is quite different as it (unlike the influential Williams, Gomes, and Selman 2003 paper [37] and our paper) in effect requires that the function-oracle gives back both a variable *and its setting*. Fourth, our claim is about *nontrivial* backbones.

So fix a β from Theorem 4.1's statement. For each Boolean formula G, let $q(G) = \left\lceil \frac{\beta\|V(G)\|}{1-\beta} \right\rceil$. Define $A = \{(G) \wedge (\texttt{new}_1 \wedge \texttt{new}_2 \wedge \cdots \wedge \texttt{new}_{q(G)}) \mid G$ is a Boolean formula having at least one variable$\}$, where, as in the proof of Theorem 3.5, we define $\texttt{new}_i$ as the ith variable that does not appear in G. Note that $\texttt{new}_1 \wedge \texttt{new}_2 \wedge \cdots \wedge \texttt{new}_{q(G)}$ is a backbone if and only if $G \in \mathrm{SAT}$, thus under the assumption that $\mathrm{P} \neq \mathrm{NP}$ and keeping in mind that for zero-variable formulas satisfiability is easy to decide, it follows that no polynomial-time algorithm can decide L_A, since the size of this backbone is $q(G) > 0$, which by our definition of q will satisfy the condition $\|S\| \geq \beta\|V(F)\|$. Why does it satisfy that condition? $\|S\|$ here is $q(G)$. And $\|V(F)\|$ here, since F is the formula $(G) \wedge (\texttt{new}_1 \wedge \texttt{new}_2 \wedge \cdots \wedge \texttt{new}_{q(G)})$, equals $\|V(G)\| + q(G)$. So the condition is claiming that $q(G) \geq \beta(\|V(G)\| + q(G))$, i.e., that $q(G) \geq \frac{\beta}{(1-\beta)}\|V(G)\|$, which indeed holds in light of the definition of q. And why do we claim that no polynomial-time algorithm can decide L_A? Well, note that SAT many-one polynomial-time reduces to L_A via the reduction $g(H)$ that equals some fixed string in L_A if H is in SAT and H has zero variables and that equals some fixed string in $\overline{L_A}$ if H is not in SAT and H has zero variables (these two cases are included merely to handle degenerate things such as $\textsc{True} \vee \textsc{False}$ that can occur if we allow $\textsc{True}$ and $\textsc{False}$ as atoms in our propositional formulas), and that equals $(H) \wedge (\texttt{new}_1 \wedge \texttt{new}_2 \wedge \cdots \wedge \texttt{new}_{q(H)})$ otherwise (the above formula is H conjoined with a large number of new variables). Thus L_A is NP-hard, and since $\mathrm{P} \neq \mathrm{NP}$ was part of the theorem's hypothesis, L_A cannot be in P.

However, the statement of Theorem 4.1 asserts that L_A is NP-complete, and so in addition to showing (as was just done) NP-hardness, we must also establish L_A's membership in NP. In fact, if one looks just at the definition of L_A, one might worry that L_A might have only $\mathrm{NP}^{\mathrm{NP}}$ as an obvious upper bound. However, as noted above our particular choice of A ensures that $\texttt{new}_1 \wedge \texttt{new}_2 \wedge \cdots \wedge \texttt{new}_{q(H)}$ is a backbone of $(H) \wedge (\texttt{new}_1 \wedge \texttt{new}_2 \wedge \cdots \wedge \texttt{new}_{q(H)})$ if and only if $H \in \mathrm{SAT}$; and that makes clear that our set is indeed in NP.

The above proof establishes Theorem 4.1. Corollary 4.2 follows immediately from the proof/construction of Theorem 4.1. Why? The set A from the proof of Theorem 4.1 is constructed in such a way that each of its potential members $(G) \wedge (\texttt{new}_1 \wedge \texttt{new}_2 \wedge \cdots \wedge \texttt{new}_{q(G)})$ (where G is a Boolean formula having at least one variable) either has no nontrivial backbone (indeed, no backbone) or has a backbone of size at least $\beta(\|V(G)\|)$. Thus the issue of backbones that are nontrivial but smaller than $\beta(\|V(F)\|)$, where F is $(G) \wedge (\texttt{new}_1 \wedge \texttt{new}_2 \wedge \cdots \wedge \texttt{new}_{q(G)})$, does not cause a problem under the construction. That is, our A (which itself is dependent on the value of β one is interested in) is such that we have ensured that $\{F \mid F \in A$ and F has a nontrivial backbone $S\} = \{F \mid F \in A$ and F has a backbone S with $\|S\| \geq \beta\|V(F)\|\}$. $\square$

We now address the potential concern that the hard instances for the decision problems we just introduced may be so infrequent that the relevance of Theorem 4.1 and Corollary 4.2 is undercut. The following theorem argues against that possibility by proving that, unless not a single NP set is frequently hard (in the sense made rigorous in the theorem's statement), there exist sets of our form that are frequently hard. (This result is making for backbones a point analogous to the one our Theorem 3.7 makes for backdoors. Hemaspaandra and Narváez [20] looks at frequency of hardness result for backbones, but with results focused on $\mathrm{NP} \cap \mathrm{coNP}$ rather than NP.)

Theorem 4.3. *If h is any nondecreasing function and for some set $B \in \mathrm{NP}$ it holds that each polynomial-time algorithm errs with respect to B, at infinitely many lengths n (resp., for almost every length n), on at least $h(n)$ of the inputs up to that length, then there will exist an $\epsilon > 0$ and a set $A \in \mathrm{P}$ of Boolean formulas satisfying the conditions of Theorem 4.1, yet being such that each polynomial-time algorithm g, at infinitely many lengths n (resp., for almost every length n), will fail to correctly determine membership in L_A for at least $h(n^\epsilon)$ inputs of length at most n.*

The same claim also holds for Corollary 4.2.

Proof. We will prove the theorem's statement regarding Theorem 4.1. It is not hard to also then see that the analogous claim holds regarding Corollary 4.2.

$B \in \mathrm{NP}$ and SAT is NP-complete. So let r_B be a polynomial-time function, transforming strings into Boolean formulas, such that (a) $r_B(x) \in \mathrm{SAT} \Leftrightarrow x \in B$, and (b) r_B is one-to-one. (A construction of such a function is given in Appendix A of [19], and let us assume that that construction is used.) As in the proof of Theorem 4.1, if F is a Boolean formula we define $q(F) = \left\lceil \frac{\beta \|V(F)\|}{1-\beta} \right\rceil$.

Without loss of generality, we assume that r_B outputs only formulas having at least one variable. Note that throughout this proof, q is applied only to outputs of r_B. Thus we have ensured that none of the logarithms in this proof have a zero as their argument.

Set $A = \{(r_B(x)) \land (\mathtt{new}_1 \land \mathtt{new}_2 \land \cdots \land \mathtt{new}_{q(r_B(x))}) \mid x \in \Sigma^*\}$. Because r_B is computable in polynomial time, there is a polynomial b such that for every input x of length at most n, the length of $r_B(x)$ is at most $b(n)$. Fix some such polynomial b, and let k denote its degree. In order to find a bound for the length of the added "tail" $\mathtt{new}_1 \land \mathtt{new}_2 \land \cdots \land \mathtt{new}_{q(r_B(x))}$ in terms of $b(n)$, notice that the length of the tail is less than some constant (that holds over all x and n, $|x| \leq n$) times $q(r_B(x)) \log q(r_B(x))$. Since $q(r_B(x)) = \left\lceil \frac{\beta \|V(F)\|}{1-\beta} \right\rceil$ and the length of $r_B(x)$ is at least a constant times the number of its variables, our assumption that $|r_B(x)| \leq b(n)$ implies the existence of a constant c such that, for all x and n, $|x| \leq n$, we have $q(r_B(x)) \leq c \cdot b(n)$. Taken together, the two previous sentences imply the existence of a constant d such that, for all x and n, $|x| \leq n$, we have that the length of $\mathtt{new}_1 \land \mathtt{new}_2 \land \cdots \land \mathtt{new}_{q(r_B(x))}$ is at most $d \cdot b(n) \log(b(n))$, and so certainly is less than $d \cdot b^2(n)$. Let N be a natural number such that, for all $n \geq N$ and all x, $|x| \leq n$ implies that $|(r_B(x)) \land (\mathtt{new}_1 \land \mathtt{new}_2 \land \cdots \land \mathtt{new}_{q(r_B(x))})| \leq n^{2k+1}$; by the previous sentence and the fact that b is of degree k, such an N will exist. Let g be a polynomial-time heuristic for L_A. Notice that $g \circ r_B$—i.e., $g(r_B(\cdot))$—is a polynomial-time heuristic for B, since $(r_B(x)) \land (\mathtt{new}_1 \land \mathtt{new}_2 \land \cdots \land \mathtt{new}_{q(r_B(x))}) \in L_A \Leftrightarrow r_B(x) \in \mathrm{SAT}$ and $r_B(x) \in \mathrm{SAT} \Leftrightarrow x \in B$. Let $n_B \geq N$ be such that there is a set of strings $S_{n_B} \subseteq (\Sigma^*)^{\leq n_B}$, $\|S_{n_B}\| \geq h(n_B)$, having the property that for all $x \in S_{n_B}$, $g \circ r_B$ fails to correctly determine the membership of x in B. Consequently, there is a set of strings $T_{n_B} \subseteq (\Sigma^*)^{\leq (n_B)^{2k+1}}$, $\|T_{n_B}\| \geq h(n_B)$, such that for all $x \in T_{n_B}$, g fails to correctly determine the membership of x in L_A; in particular the set $T_{n_B} = \{(r_B(x)) \land (\mathtt{new}_1 \land \mathtt{new}_2 \land \cdots \land \mathtt{new}_{q(r_B(x))}) \mid x \in S_{n_B}\}$ has this property.

Using the variable renaming $n_A = (n_B)^{2k+1}$, it is now easy to see that we have proven that every length $n_B \geq N$ at which $g \circ r_B$ (viewed as a heuristic for B) errs on at least $h(n_B)$ inputs of length up to n_B has a corresponding length n_A at which g (viewed as a heuristic for L_A) errs on at least $h((n_A)^{\frac{1}{2k+1}})$ inputs of length up to n_A. Our hypothesis guarantees the existence of infinitely many such $n_B \geq N$ (resp., almost all $n \geq N$ can take the role of n_B), each with a corresponding n_A. Setting $\epsilon = \frac{1}{2k+1}$, our theorem is now proven. $\square$

5 Conclusions

We constructed easily recognizable families of Boolean formulas that provide hard instances for decision problems related to backdoors and backbones under the assumption that P $\neq$ NP. In particular, we have shown that, under the assumption P $\neq$ NP, there exist easily recognizable families of Boolean formulas with easy-to-find strong backdoors yet for which it is hard to determine whether the formulas are satisfiable. Under the same P $\neq$ NP assumption, we have shown that there exist easily recognizable collections of Boolean formulas for which it is hard (in fact, NP-complete) to determine whether they have a backbone, and that there exist easily recognizable collections of Boolean formulas for which it is hard (in fact, NP-complete) to determine whether they have a large backbone. (These results can be taken as indicating that, under the very plausible assumption that P $\neq$ NP, search and decision shear apart in complexity for backdoors and backbones. That makes it particularly unfortunate that their definitions in the literature are framed in terms of decision rather than search, especially since when one tries to put these to work in SAT solvers, it is the search case that one typically tries to use and leverage.)

For both our backdoor and backbone results, we have shown that if *any* problem B in NP is frequently hard, then there exist families of Boolean formulas of the sort we describe that are hard almost as frequently as B.

Acknowledgments An earlier version [21] appeared in the SOFSEM 2019 conference. We are extremely grateful to the anonymous conference and journal referees for valuable comments and suggestions that much improved this paper.

References

[1] C. Ansótegui, M. Bonet, J. Giráldez-Cru, J. Levy, and L. Simon. Community structures in industrial SAT instances. *Journal of Artificial Intelligence Research*, 66:443–472, 2019.

[2] M. Bellare and S. Goldwasser. The complexity of decision versus search. *SIAM Journal on Computing*, 23(1):97–119, 1994.

[3] P. Berman. Relationship between density and deterministic complexity of NP-complete languages. In *Proceedings of the 5th International Colloquium on Automata, Languages,*

and Programming, pages 63–71. Springer-Verlag *Lecture Notes in Computer Science #62*, July 1978.

[4] A. Borodin and A. Demers. Some comments on functional self-reducibility and the NP hierarchy. Technical Report TR 76-284, Department of Computer Science, Cornell University, Ithaca, NY, July 1976.

[5] H. Buhrman and J. Hitchcock. NP-hard sets are exponentially dense unless coNP $\subseteq$ NP/poly. In *Proceedings of the 23rd Annual IEEE Conference on Computational Complexity*, pages 1–7. IEEE Computer Society Press, June 2008.

[6] J. Cai, V. Chakaravarthy, L. Hemaspaandra, and M. Ogihara. Competing provers yield improved Karp–Lipton collapse results. *Information and Computation*, 198(1):1–23, 2005.

[7] W. Chen and D. Whitley. Decomposing SAT instances with pseudo backbones. In *Proceedings of the 17th European Conference on Evolutionary Computation in Combinatorial Optimization*, pages 75–90. Springer-Verlag *Lecture Notes in Computer Science #10197*, March 2017.

[8] S. Cook. The complexity of theorem-proving procedures. In *Proceedings of the 3rd ACM Symposium on Theory of Computing*, pages 151–158. ACM Press, May 1971.

[9] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem-proving. *Communications of the ACM*, pages 394–397, July 1962.

[10] M. Davis and H. Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7(3):201–215, 1960.

[11] B. Dilkina, C. Gomes, and A. Sabharwal. Tradeoffs in the complexity of backdoors to satisfiability: dynamic sub-solvers and learning during search. *Annals of Mathematics and Artificial Intelligence*, 70(4):399–431, 2014.

[12] W. Dowling and J. Gallier. Linear-time algorithms for testing the satisfiability of propositional Horn formulae. *The Journal of Logic Programming*, 1(3):267–284, 1984.

[13] T. Friedrich, A. Krohmer, R. Rothenberger, and A. Sutton. Phase transitions for scale-free SAT formulas. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence*, pages 3893–3899. AAAI Press, February 2017.

[14] W. Gasarch. The third $\mathrm{P} =? \mathrm{NP}$ poll. *SIGACT News*, 50(1):38–59, 2019.

[15] S. Gaspers, N. Misra, S. Ordyniak, S. Szeider, and S. Živný. Backdoors into heterogeneous classes of SAT and CSP. *Journal of Computer and System Sciences*, 85:38–56, 2017.

[16] J. Hartmanis and L. Hemachandra. Complexity classes without machines: On complete languages for UP. *Theoretical Computer Science*, 58(1–3):129–142, 1988.

[17] E. Hemaspaandra, L. Hemaspaandra, and C. Menton. Search versus decision for election manipulation problems. *ACM Transactions on Computation*, 12:1–42, 2020.

[18] L. Hemaspaandra. Computational social choice and computational complexity: BFFs? In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence*, pages 7971–7977. AAAI Press, February 2018.

[19] L. Hemaspaandra and D. Narváez. The opacity of backbones. Technical Report arXiv:1606.03634 [cs.AI], Computing Research Repository, arXiv.org/corr/, June 2016. Revised, January 2017.

[20] L. Hemaspaandra and D. Narváez. The opacity of backbones. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence*, pages 3900–3906. AAAI Press, February 2017.

[21] L. Hemaspaandra and D. Narváez. Existence versus exploitation: The opacity of backbones and backdoors under a weak assumption. In *Proceedings of the 45th International Conference on Current Trends in Theory and Practice of Computer Science*, pages 247–259. Springer-Verlag Lecture Notes in Computer Science #11376, January 2019.

[22] L. Hemaspaandra and M. Ogihara. *The Complexity Theory Companion.* Springer-Verlag, 2002.

[23] L. Hemaspaandra and R. Williams. An atypical survey of typical-case heuristic algorithms. *SIGACT News*, 43(4):71–89, 2012.

[24] L. Hemaspaandra and M. Zimand. Strong self-reducibility precludes strong immunity. *Mathematical Systems Theory*, 29(5):535–548, 1996.

[25] P. Kilby, J. Slaney, S. Thiébaux, and T. Walsh. Backbones and backdoors in satisfiability. In *Proceedings of the 20th National Conference on Artificial Intelligence*, pages 1368–1373. AAAI Press, July 2005.

[26] S. Kochemazov and O. Zaikin. ALIAS: A modular tool for finding backdoors for SAT. In *Proceedings of the 21st International Conference on Theory and Applications of Satisfiability Testing*, pages 419–427. Springer-Verlag *Lecture Notes in Computer Science #10929*, June 2018.

[27] S. Mahaney. Sparse complete sets for NP: Solution of a conjecture of Berman and Hartmanis. *Journal of Computer and System Sciences*, 25(2):130–143, 1982.

[28] N. Nishimura, P. Ragde, and S. Szeider. Detecting backdoor sets with respect to Horn and binary clauses. In *Informal Proceedings of the 7th International Conference on Theory and Applications of Satisfiability Testing*, pages 96–103, May 2004.

[29] C. Papadimitriou. *Computational Complexity.* Addison-Wesley, 1994.

[30] J. Rothe. Complexity of certificates, heuristics, and counting types, with applications to cryptography and circuit theory. Habilitation thesis, Friedrich-Schiller-Universität Jena, Institut für Informatik, Jena, Germany, June 1999.

[31] T. Schaefer. The complexity of satisfiability problems. In *Proceedings of the 10th ACM Symposium on Theory of Computing*, pages 216–226. ACM Press, May 1978.

[32] U. Schöning. Complete sets and closeness to complexity classes. *Mathematical Systems Theory*, 19(1):29–42, 1986.

[33] A. Semenov, O. Zaikin, I. Otpuschennikov, S. Kochemazov, and A. Ignatiev. On cryptographic attacks using backdoors for SAT. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence*, pages 6641–6648. AAAI Press, February 2018.

[34] S. Szeider. Backdoor sets for DLL subsolvers. *Journal of Automated Reasoning*, 35(1–3):73–88, 2005.

[35] G. Tardos. Query complexity, or why is it difficult to separate $\mathrm{NP}^A \cap \mathrm{coNP}^A$ from P^A by random oracles A. *Combinatorica*, 9:385–392, 1989.

[36] L. Valiant. The relative complexity of checking and evaluating. *Information Processing Letters*, 5(1):20–23, 1976.

[37] R. Willams, C. Gomes, and B. Selman. Backdoors to typical case complexity. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence*, pages 1173–1178. Morgan Kaufmann, August 2003.