False Positives and NP-Hard Sets

Christian Glaßer*, A. Pavan†, and Stephen Travers‡

Abstract

We study the effects of faulty data on NP-hard sets. We consider hard sets for several polynomial time reductions, add corrupt data and then analyze whether the resulting sets are still hard for NP. We explain that our results are related to a deterministic variant of the notion of program self-correction by Blum, Luby, and Rubinfeld. Among other results, we prove that m-complete sets for NP are nonadaptively weakly deterministically self-correctable while btt-complete sets for NP are weakly deterministically self-correctable. Our results can also be applied to the study of Yesha's p-closeness. In particular, we strengthen a result by Ogiwara and Fu.

In the second part, we investigate a different setting where we add large amounts of false data to NP-hard sets. In this setting however, we do not allow the false data to have an arbitrary structure. Instead, the set of false positives that we add must also be NP-hard. This part is motivated by the longstanding open question whether the union of disjoint NP-complete sets always is NP-complete. We present improved necessary and sufficient conditions for an affirmative answer to this question.

1 Introduction

Even small amounts of faulty data can obscure reasonable information. For instance, by filling more and more whitespaces of a printed text with arbitrary letters, it can become quite difficult to understand the original meaning of the text.

The same holds true for NP-complete sets. Take for instance SAT, the set of all satisfiable formulas. By adding *false positives* to SAT, i.e., some unsatisfiable formulas, we can actually lose information: If we overdo it, we end up with $\mathrm{SAT} \cup \overline{\mathrm{SAT}} = \Sigma^*$, and by this definitely lose NP-completeness. But how much false positive data can NP-hard sets handle, i.e., how many false positives can we add such that the resulting set stays NP-hard? Alternatively, how much effort is needed to extract the original information?

In this paper, we investigate how polynomial time reductions can cope with false positives. More precisely, we consider NP-hard sets for several polynomial time reductions and add false positives to the sets.

Moreover, we study the effects of more general kinds of faulty data. We investigate how polynomial time reductions can handle combinations of both, false positives and *false negatives*. This

*Theoretische Informatik, Universität Würzburg, Germany, glasser@informatik.uni-wuerzburg.de.

†Department of Computer Science, Iowa State University, USA, pavan@cs.iastate.edu. Research supported in part by NSF grant CCF-0430807.

‡Theoretische Informatik, Universität Würzburg, Germany, travers@informatik.uni-wuerzburg.de.

relates our research to the notion of *program self-correction* which was introduced by Blum, Luby, and Rubinfeld [BLR93]. That notion addresses a fundamental question regarding software reliability: Can one increase the reliability of existing software without understanding the way it works? More precisely, let P be a program that is designed to solve a problem L. However, we do not know whether P is correct. Is it possible to write an auxiliary program M that uses P such that if P errs only on a small fraction of the inputs, then with high probability M corrects the errors made by P? So M has to find the right answer with high probability by calling P on several inputs.

Our investigations of the consequences of faulty data are related to a *deterministic* variant of self-correction. In this case, the error probability of the wrapping machine M must be 0, i.e., M must achieve certainty about the question of whether the input belongs to L. As in the original definition, we also demand that M runs in polynomial time.

In the first part of this paper, we investigate the setting where sparse parts of NP-hard sets can be corrupt. We prove that

- the symmetric difference of m-hard sets and arbitrary sparse sets always is tt-hard. This implies that m-complete sets for NP are *nonadaptively weakly deterministically self-correctable.*
- the symmetric difference of btt-hard sets and arbitrary sparse sets always is T-hard. This implies that btt-complete sets are *weakly deterministically self-correctable.*
- the union of dtt-hard sets and arbitrary sparse sets always is T-hard

These results show that $\leq^{\mathrm{p}}_{\mathrm{m}}$-hard, $\leq^{\mathrm{p}}_{\mathrm{btt}}$-hard, and $\leq^{\mathrm{p}}_{\mathrm{dtt}}$-hard sets do not become too easy when false positives are added (as they stay NP-hard with respect to more general reducibilities). On the other hand, we show that unless P = NP, there exist sparse sets S_1, S_2 such that $\mathrm{SAT} \cup S_1$ is not $\leq^{\mathrm{p}}_{\mathrm{btt}}$-hard for NP, and $\mathrm{SAT} \cup S_2$ is not $\leq^{\mathrm{p}}_{\mathrm{dtt}}$-hard for NP.

Furthermore, we explain that one of our results about btt-reducibility is related to the notion of *p-closeness* which was introduced by Yesha [Yes83]. We show that no $\leq^{\mathrm{p}}_{\mathrm{btt}}$-hard set for NP is p-close to P, unless P = NP. This strengthens a result by Ogiwara [Ogi91] and Fu [Fu93] who proved that no $\leq^{\mathrm{p}}_{\mathrm{m}}$-hard set for NP is p-close to P, unless P = NP.

In the second part, we investigate a different setting where we add large amounts of false positives to NP-hard sets. In this setting however, we do not allow the false data to have an arbitrary structure. Instead, the set of false positives that we add must also be NP-hard. This part of the paper is motivated by the longstanding open question [Sel88] whether the union of two disjoint m-complete sets for NP always is NP-complete. Observe that this question has a negative answer if $\mathrm{P} \subsetneq \mathrm{NP} = \mathrm{coNP}$, while it is not clear what to believe in the case that $\mathrm{NP} \neq \mathrm{coNP}$. We provide improved necessary and sufficient conditions for an affirmative answer to this question.

2 Preliminaries

We recall basic notions. Σ denotes a finite alphabet with at least two letters, Σ^* denotes the set of all words, and $|w|$ denotes the length of a word w. For $n \geq 0$, Σ^n denotes the set of all

words of length n. A set $A \subseteq \Sigma^*$ is *nontrivial* if $A \neq \emptyset$ and $A \neq \Sigma^*$. A *tally* set is a subset of 0^*. The census function of a set S is defined as $\text{census}_S(n) \stackrel{df}{=} |S \cap \Sigma^n|$. A set S is *sparse* if there exists a polynomial p such that for all $n \geq 0$, $\text{census}_S(n) \leq p(n)$. The symmetric difference of sets A and B is defined as $A \triangle B = (A - B) \cup (B - A)$.

The language accepted by a machine M is denoted by $L(M)$. The characteristic function of a set A is denoted by c_A. $\overline{L}$ denotes the complement of a language L and co$\mathcal{C}$ denotes the class of complements of languages in $\mathcal{C}$. FP denotes the class of functions computable in deterministic polynomial time.

We recall standard polynomial-time reducibilities [LLS75]. A set B *many-one-reduces* to a set C (*m-reduces* for short; in notation $B \leq^{\text{p}}_{\text{m}} C$) if there exists a total, polynomial-time-computable function f such that for all strings x,

$$x \in B \;\Leftrightarrow\; f(x) \in C.$$

A set B *Turing-reduces* to a set C (*T-reduces* for short; in notation $B \leq^{\text{p}}_{\text{T}} C$) if there exists a deterministic polynomial-time-bounded oracle Turing machine M such that for all strings x,

$$x \in B \;\Leftrightarrow\; M \text{ with } C \text{ as oracle accepts the input } x.$$

A set B *truth-table-reduces* to a set C (*tt-reduces* for short; in notation $B \leq^{\text{p}}_{\text{tt}} C$) if there exists a deterministic polynomial-time-bounded oracle Turing machine M that queries nonadaptively such that for all strings x,

$$x \in B \;\Leftrightarrow\; M \text{ with } C \text{ as oracle accepts the input } x.$$

A set B *disjunctively truth-table-reduces to a set* C (dtt-reduces for short; in notation $B \leq^{\text{p}}_{\text{dtt}} C$) if there exists a total, polynomial-time-computable function $f : \Sigma^* \to \mathcal{P}(\Sigma^*)$ such that for all strings x,

$$x \in B \;\Leftrightarrow\; f(x) \cap C \neq \emptyset.$$

A set B *conjunctively truth-table-reduces to a set* C (ctt-reduces for short; in notation $B \leq^{\text{p}}_{\text{ctt}} C$) if there exists a total, polynomial-time-computable function $f : \Sigma^* \to \mathcal{P}(\Sigma^*)$ such that for all strings x,

$$x \in B \;\Leftrightarrow\; f(x) \subseteq C.$$

A set B *bounded truth-table-reduces to a set* C (btt-reduces for short; in notation $B \leq^{\text{p}}_{\text{btt}} C$) if there exists a $k \geq 1$, a k-ary Boolean function α, and $g_1, \ldots, g_k \in \text{FP}$ such that for all x

$$x \in B \Leftrightarrow \alpha(c_C(g_1(x)), c_C(g_2(x)), \ldots, c_C(g_k(x))) = 1.$$

A set B is *many-one-hard* (*m-hard* for short) for a complexity class $\mathcal{C}$ if every $B \in \mathcal{C}$ m-reduces to B. If additionally $B \in \mathcal{C}$, then we say that B is *many-one-complete* (*m-complete* for short) for $\mathcal{C}$. Similarly, we define hardness and completeness for other reducibilities. We use the term $\mathcal{C}$-complete as an abbreviation for m-complete for $\mathcal{C}$.

A set L is *paddable* [BH77] if there exists $f(\cdot,\cdot)$, a polynomial-time computable, injective polynomial-time invertible function such that for all x and y,

$$x \in L \iff f(x, y) \in L.$$

2.1 Weak Deterministic Self-Correction

We introduce the notion of weak deterministic self-correction which is a deterministic variant of self-correction [BLR93] (*weak* because the set of errors is sparse).

Definition 2.1 *L is* weakly deterministically self-correctable *if for every polynomial q there exists a polynomial-time machine M such that $L \leq_{\mathrm{T}}^{\mathrm{p}} P$ via M whenever the census of $L \triangle P$ is bounded by q. If M queries nonadaptively, then L is* nonadaptively weakly deterministically self-correctable.

The set P in the definition formalizes a program for L that errs on at most $q(n)$ inputs of length n. So L is weakly deterministically self-correctable if there exists an auxiliary machine M that corrects *all* programs that err on at most $q(n)$ inputs of length n. The next theorem shows that such a universal M surprisingly exists already if the single programs can be corrected with possibly different machines. This establishes the connection between weak deterministic self-correction and the robustness against false positives.

Theorem 2.2 *L is weakly deterministically self-correctable $\Leftrightarrow$ $L \leq_{\mathrm{T}}^{\mathrm{p}} L \triangle S$ for all sparse S.*

Proof $\Rightarrow$: This is a direct consequence of Definition 2.1.

$\Leftarrow$: Assume that L is not weakly deterministically self-correctable. So there exists a polynomial q such that

$$\forall \text{polynomial-time machine } M,\ \exists T \subseteq \Sigma^* \ [\mathrm{census}_T \leq q \text{ and } L \neq L(M^{L \triangle T})]. \tag{1}$$

We construct a sparse S such that $L \not\leq_{\mathrm{T}}^{\mathrm{p}} L \triangle S$. The construction is stagewise where in step i we construct a finite set S_i such that $S_1 \subseteq S_2 \subseteq \cdots$ and $S \stackrel{df}{=} \bigcup_{i \geq 1} S_i$. Let $M_1, M_2, \ldots$ be an enumeration of all deterministic, polynomial-time Turing machines such that M_i runs in time $n^i + i$. Let $S_0 = \emptyset$. For $i \geq 1$, the set S_i is constructed as follows:

Choose n large enough such that $S_{i-1} \subseteq \Sigma^{<n}$ and changing the oracle with respect to words of length $\geq n$ will not affect the computations that were simulated in earlier steps. Choose a finite $T_i \subseteq \Sigma^{\geq n}$ and an $x_i \in \Sigma^*$ such that $\mathrm{census}_{T_i} \leq q$ and

$$x_i \in L \Leftrightarrow x_i \notin L(M_i^{L \triangle (S_{i-1} \cup T_i)}). \tag{2}$$

Let $S_i \stackrel{df}{=} S_{i-1} \cup T_i$.

We argue that the choice of T_i is possible. If not, then for all finite $T_i \subseteq \Sigma^{\geq n}$ where $\mathrm{census}_{T_i} \leq q$ and all $x_i \in \Sigma^*$ it holds that

$$x_i \in L \Leftrightarrow x_i \in L(M_i^{L \triangle (S_{i-1} \cup T_i)}).$$

Let M be the polynomial-time machine obtained from M_i when queries of length $< n$ are answered according to $(L \triangle S_{i-1}) \cap \Sigma^{<n}$ (which is a finite set). So for all T where $\mathrm{census}_T \leq q$ and all $x_i \in \Sigma^*$ it holds that

$$x_i \in L(M^{L \triangle T}) \Leftrightarrow x_i \in L(M_i^{L \triangle (S_{i-1} \cup (T \cap \Sigma^{\geq n}))}) \Leftrightarrow x_i \in L(M_i^{L \triangle (S_{i-1} \cup T')}) \Leftrightarrow x_i \in L,$$

where $T' = T \cap \Sigma^{\geq n} \cap \Sigma^{\leq |x_i|^i + i}$. Hence $L = L(M^{L \triangle T})$ for all T where $\text{census}_T \leq q$. So M contradicts (1). It follows that the choice of T_i is possible and hence also the construction of S.

The equivalence (2) makes sure that

$$\forall i \geq 1 \, [x_i \in L \Leftrightarrow x_i \notin L(M_i^{L \triangle S})]$$

and hence $L \not\leq_{\mathrm{T}}^{\mathrm{p}} L \triangle S$. □

Corollary 2.3 *L is nonadaptively weakly deterministically self-correctable $\Leftrightarrow$ $L \leq_{\mathrm{tt}}^{\mathrm{p}} L \triangle S$ for all sparse S.*

Proof This is shown with the same proof as Theorem 2.2, except that all machines query nonadaptively. □

3 Partly corrupt NP-hard Sets

We investigate how polynomial reductions can cope with sparse amounts of false data in sets that are hard for NP with respect to various reducibilities. In section 3.1 we show that altering sparse information in m-hard sets results in sets that are at least tt-hard. In particular, all m-complete sets are nonadaptively weakly deterministically self-correctable. Similarly, in section 3.2 we obtain that btt-hardness softens at most to T-hardness, if sparse information is altered. In particular, all btt-complete sets are weakly deterministically self-correctable. Moreover, we improve results by Ogiwara [Ogi91] and Fu [Fu93], and show that no btt-hard set is p-close to P, unless P = NP. In section 3.3 we prove that adding a sparse amount of false positives to dtt-hard sets results in sets that are at least T-hard. However, it remains open whether dtt-complete sets are weakly deterministically self-correctable. At the end of section 3.3, we give evidence that this open problem is rather difficult to solve.

Finally, in subsection 3.4 we show that many-one reductions, bounded truth-table reductions, and disjunctive truth-table reductions are provably too weak to handle false positives in SAT.

3.1 Many-One Reductions

Here we alter sparse information in m-hard sets for NP. Under the assumption P $\neq$ NP, the resulting sets are still ctt-hard. Without the assumption, we can show that the resulting sets are at least tt-hard. On the technical side we extend an idea from [GPSZ06] which shows how many-one queries to NP-hard sets can be reformulated. In this way, for a given query we can generate a polynomial number of different, but equivalent queries (Lemma 3.1). From this we easily obtain the conditional ctt-hardness and the unconditional tt-hardness of the altered NP-hard set. As a corollary, all m-complete sets for NP are nonadaptively weakly deterministically self-correctable.

Lemma 3.1 *Let L be $\leq_{\mathrm{m}}^{\mathrm{p}}$-hard for* NP *and let $B \in$ NP. Then there exists a polynomial r such that for every polynomial q there is a polynomial-time algorithm $\mathcal{A}$ such that $\mathcal{A}$ on input x,*

- *either correctly decides the membership of x in B*
- *or outputs $k = q(r(|x|))$ pairwise disjoint $y_1, \ldots, y_k \in \Sigma^{\leq r(|x|)}$ such that for all $i \in [1, k]$,*

$$x \in B \Leftrightarrow y_i \in L.$$

Proof Choose $R \in \mathrm{P}$ and a polynomial p such that $x \in B$ if and only if there exists a $w \in \Sigma^{p(|x|)}$ such that $(x, w) \in R$. For $x \in B$, let w_x be the lexicographically greatest such witness. The following set is in NP.

$$\mathrm{Left}(B) = \{(x, y) \,\big|\, x \in B, |y| = p(|x|), y \leq w_x\}.$$

So there is a many-one reduction f from $\mathrm{Left}(B)$ to L. In particular, there exists a polynomial r such that for all $x \in \Sigma^*$ and all $y \in \Sigma^{p(|x|)}$, $|f(x, y)| \leq r(|x|)$. Choose a polynomial q. We now describe the algorithm $\mathcal{A}$.

```
1  // input x, |x| = n
2  m := p(n)
3  if (x, 1^m) ∈ R then accept
4  l := 0^m
5  if f(x, l) = f(x, 1^m) then reject
6  Q = {f(x, l)}
7  while |Q| ≤ q(r(n)) do
8     choose a ∈ Σ^m such that l ≤ a ≤ 1^m, f(x, a) ∈ Q, and f(x, a + 1) ∉ Q
9     l := a + 1
10    if (x, a) ∈ R then accept
11    if f(x, l) = f(x, 1^m) then reject
12    Q = Q ∪ {f(x, l)}
13 end while
14 output Q
```

Observe that the algorithm places a string $f(x, l)$ in Q only if $f(x, l) \neq f(x, 1^m)$. Thus $f(x, 1^m)$ is never placed in Q. So in step 8, $f(x, l) \in Q$ and $f(x, 1^m) \notin Q$. Therefore, with binary search we find the desired a in polynomial time. Every iteration of the while loop adds a new string to Q or decides the membership of x in B. Thus the algorithm works in polynomial time and when it outputs some Q, then $|Q| = q(r(|x|))$ and words in Q have lengths $\leq r(n)$.

Claim 3.2 *If the algorithm outputs some Q, then for all $y \in Q$, $x \in B \Leftrightarrow y \in L$.*

Proof If $x \notin B$, then for all $c \in [0^m, 1^m]$, $(x, c) \notin \mathrm{Left}(B)$. Observe that the algorithm places a string y in Q only if $y = f(x, a)$ where $a \in [0^m, 1^m]$. Since f is a many-one reduction from $\mathrm{Left}(B)$ to L, no string from Q belongs to L.

From now on we assume $x \in B$. We prove the claim by induction. Initially, $Q = \{f(x, 0^m)\}$. Clearly, $x \in B \Leftrightarrow (x, 0^m) \in \mathrm{Left}(B)$. Since f is a many-one reduction from $\mathrm{Left}(B)$ to L, the claim holds initially. Assume that the claim holds before an iteration of the while loop. The while loop finds a node a such that $f(x, a) \in Q$, but $f(x, a + 1) \notin Q$. From $f(x, a) \in Q$ and $x \in B$ it follows (by induction hypothesis) that $f(x, a) \in L$. Thus $(x, a) \in \mathrm{Left}(B)$ which implies

$a \leq w_x$. At this point the algorithm checks whether a is a witness of x. If so, then it accepts and halts. Otherwise, we have $a+1 \leq w_x$. Thus $(x, a+1) \in \text{Left}(B)$ and $f(x, a+1) \in L$. So the claim also holds after the iteration of the while loop. □

Claim 3.3 *If the algorithm accepts x (resp., rejects x), then $x \in B$ (resp., $x \notin B$).*

Proof The algorithm accepts x only if it finds a witness of x. Thus if the algorithm accepts, then $x \in B$. The algorithm rejects only if $f(x,l) = f(x,1^m)$. Note that $f(x,l) \in Q$, so by the previous claim, $x \in B \Leftrightarrow f(x,l) \in L$. Observe that $(x,1^m) \notin \text{Left}(B)$. Thus $f(x,l) = f(x,1^m) \notin L$ and hence $x \notin B$. □

This finishes the proof of the lemma. □

Theorem 3.4 *The following statements are equivalent.*

1. $\text{P} \neq \text{NP}$
2. *If L is $\leq^{\text{p}}_{\text{m}}$-hard for* NP *and S is sparse, then $L \cup S$ is $\leq^{\text{p}}_{\text{ctt}}$-hard for* NP.

Proof 2 $\Rightarrow$ 1: If $\text{P} = \text{NP}$, then $L = \Sigma^* - \{0\}$ and $S = \{0\}$ are counter examples for 2.

1 $\Rightarrow$ 2: Assume $\text{P} \neq \text{NP}$ and let L and S be as in statement 2. If $\overline{L}$ is sparse, then there exist sparse coNP-hard sets and hence $\text{P} = \text{NP}$ [For79]. So it follows that $\overline{L}$ is not sparse and $L \cup S \neq \Sigma^*$. Hence there exist elements $x_0 \notin L \cup S$ and $x_1 \in L \cup S$.

Let $B \in \text{NP}$; we show $B \leq^{\text{p}}_{\text{ctt}} L \cup S$. First, choose the polynomial r according to Lemma 3.1. Let q be a polynomial such that $|S \cap \Sigma^{\leq n}| < q(n)$. Lemma 3.1 provides an algorithm $\mathcal{A}$ that on input x either correctly decides the membership of x in B, or outputs $k = q(r(|x|))$ pairwise disjoint $y_1, \ldots, y_k \in \Sigma^{\leq r(|x|)}$ such that for all $i \in [1,k]$, $(x \in B \Leftrightarrow y_i \in L)$. Define the following polynomial-time-computable function.

$$g(x) \stackrel{df}{=} \begin{cases} x_0 & : \text{ if } \mathcal{A}(x) \text{ rejects} \\ x_1 & : \text{ if } \mathcal{A}(x) \text{ accepts} \\ (y_1, \ldots, y_k) & : \text{ if } \mathcal{A}(x) \text{ returns } Q = \{y_1, \ldots, y_k\} \end{cases}$$

Note that in the last case, $k = q(r(|x|))$ and the y_i have lengths $\leq r(|x|)$. So at least one of the y_i does not belong to S. From $\mathcal{A}$'s properties stated in Lemma 3.1 it follows that $B \leq^{\text{p}}_{\text{ctt}} L \cup S$ via g. □

Theorem 3.5 *If L is $\leq^{\text{p}}_{\text{m}}$-hard for* NP *and S is sparse, then $L \triangle S$ is $\leq^{\text{p}}_{\text{tt}}$-hard for* NP.

Proof For $B \in \text{NP}$ we show $B \leq^{\text{p}}_{\text{tt}} L \triangle S$. First, choose the polynomial r according to Lemma 3.1. Let q be a polynomial such that $2 \cdot |S \cap \Sigma^{\leq n}| < q(n)$. Lemma 3.1 provides an algorithm $\mathcal{A}$ that

on input x either correctly decides the membership of x in B, or outputs $k = q(r(|x|))$ pairwise disjoint $y_1, \ldots, y_k \in \Sigma^{\leq r(|x|)}$ such that for all $i \in [1,k]$, $(x \in B \Leftrightarrow y_i \in L)$. We describe a polynomial-time oracle machine M on input x: If $\mathcal{A}(x)$ accepts, then M accepts. If $\mathcal{A}(x)$ rejects, then M rejects. Otherwise, $\mathcal{A}(x)$ returns elements $y_1, \ldots, y_k \in \Sigma^{\leq r(|x|)}$. M queries all these elements and accepts if and only if at least $k/2$ of the answers were positive.

Clearly, if $\mathcal{A}(x)$ accepts or rejects, then $(x \in B \Leftrightarrow M(x)$ accepts). So assume that $\mathcal{A}(x)$ returns elements y_i. S contains less than $q(r(|x|))/2 = k/2$ words of length $\leq r(|x|)$. So more than $k/2$ of the y_i do not belong to S. Hence, for more than $k/2$ of the y_i it holds that

$$x \in B \quad \Leftrightarrow \quad y_i \in L \quad \Leftrightarrow \quad y_i \in L \triangle S.$$

Therefore, x belongs to B if and only if at least $k/2$ of the y_i belong to $L \triangle S$. This shows that $B \leq^{\mathrm{p}}_{\mathrm{tt}} L \triangle S$ via M. □

Corollary 3.6 *If L is $\leq^{\mathrm{p}}_{\mathrm{m}}$-hard for* NP *and S is sparse, then $L \cup S$ is $\leq^{\mathrm{p}}_{\mathrm{tt}}$-hard for* NP.

Proof Note that $S' \stackrel{df}{=} S - L$ is sparse. By Theorem 3.5, $L \triangle S' = L \cup S$ is $\leq^{\mathrm{p}}_{\mathrm{tt}}$-hard for NP. □

Corollary 3.7 *All $\leq^{\mathrm{p}}_{\mathrm{m}}$-complete sets for* NP *are nonadaptively weakly deterministically self-correctable.*

Proof Let L be $\leq^{\mathrm{p}}_{\mathrm{m}}$-complete for NP. By Corollary 3.5, for all sparse S, $L \leq^{\mathrm{p}}_{\mathrm{tt}} L \triangle S$. By Corollary 2.3, L is nonadaptively weakly deterministically self-correctable. □

3.2 Bounded Truth-Table Reductions

We show that altering sparse information in btt-hard sets for NP results in sets that are still T-hard for NP. Our proof builds on the left-set technique by Ogihara and Watanabe [OW91]. First, in Lemma 3.8 we isolate the combinatorial argument for the case that a Turing machine has oracle access to the symmetric difference of a btt-hard set B and a sparse set S. Then, with this argument at hand, we perform an Ogihara-Watanabe-tree-pruning in the computation tree of a given NP-machine. Finally this shows that the acceptance of the latter machine can be determined in polynomial time with access to the oracle $B \triangle S$. As a corollary we obtain that all btt-complete sets in NP are weakly deterministically self-correctable. Moreover, we obtain the following improvement of results by Ogiwara [Ogi91] and Fu [Fu93]: No btt-hard set for NP is p-close to P, unless P = NP.

For our combinatorial argument we need to define the following polynomials r_k for $k \geq 0$.

$$\begin{aligned} r_0(n) &\stackrel{df}{=} 2 \\ r_k(n) &\stackrel{df}{=} 2^k(2kn+2)(r_{k-1}(n))^k \quad \text{for } k \geq 1 \end{aligned}$$

Lemma 3.8 *For every $k \geq 0$ there exists a polynomial-time oracle transducer M_k with the following properties: For every input $(0^n, V)$ where $V = (v_{i,j}) \in (\Sigma^{\leq n})^{k \times r_k(n)}$ and for all sets $B, S \subseteq \Sigma^{\leq n}$ where $|S| \leq n$ the computation $M_k^{B \triangle S}(0^n, V)$ outputs some $b \in (1, r_k(n)]$ such that*

$$\exists a, c \in [1, r_k(n)] \text{ such that } a < b \leq c \text{ and } \forall i \in [1,k], (v_{i,a} \in B \Leftrightarrow v_{i,c} \in B). \tag{3}$$

Proof If $k = 0$, then we are done by defining M_0 as the transducer that always outputs 2. So assume $k \geq 1$. We describe the oracle transducer M_k^O on input $(0^n, V)$.

- **Case 1:** V contains a row m in which some word w appears at least $s \stackrel{df}{=} r_{k-1}(n)$ times. So there exist columns $j_1 \leq j_2 \leq \cdots \leq j_s$ such that $v_{m,j_i} = w$ for $i \in [1,s]$. Let V' be the matrix that consists of the columns $j_1, j_2, \ldots, j_s$ of V where the m-th row is deleted, i.e.,

$$V' \stackrel{df}{=} \begin{pmatrix} v_{1,j_1} & v_{1,j_2} & \cdots & v_{1,j_s} \\ v_{2,j_1} & v_{2,j_2} & \cdots & v_{2,j_s} \\ \vdots & \vdots & & \vdots \\ v_{m-1,j_1} & v_{m-1,j_2} & \cdots & v_{m-1,j_s} \\ v_{m+1,j_1} & v_{m+1,j_2} & \cdots & v_{m+1,j_s} \\ \vdots & \vdots & & \vdots \\ v_{k,j_1} & v_{k,j_2} & \cdots & v_{k,j_s} \end{pmatrix}.$$

  V' is a matrix of dimension $(k-1) \times r_{k-1}(n)$. Let $b' \stackrel{df}{=} M_{k-1}^O(0^n, V')$ and return $b \stackrel{df}{=} j_{b'}$.

- **Case 2:** V does not contain rows in which a word appears at least $r_{k-1}(n)$ times. We will hide several columns in V and will finally define b as the number of some unhidden column. First, we hide columns in V such that the remaining matrix has no rows in which a word appears more than once. More precisely, we hide column $j \geq 2$ in V if and only if

$$\exists i \in [1,k] \exists j' \in [1, j-1], (v_{i,j} = v_{i,j'}).$$

  We will see that at least $2^k(2kn+2)$ columns remain unhidden. In a second step, for each unhidden column j, we query the words $v_{1,j}, v_{2,j}, \ldots, v_{k,j}$ and obtain the vector of answers $a_j = (a_{1,j}, a_{2,j}, \ldots, a_{k,j})$. There are at most 2^k different such vectors. Let $a = (a_1, a_2, \ldots, a_k)$ be the vector that appears most often and note that at least $2kn+2$ unhidden columns $j_1 < j_2 < \cdots < j_{2kn+2}$ share this vector. Return $b \stackrel{df}{=} j_{kn+2}$.

This finishes the description of the oracle transducer M_k.

An induction on $k \geq 0$ shows that M_k works in polynomial time. So it remains to show that M_k has the properties stated in the lemma. This is done by induction on $k \geq 0$. For $k = 0$, M_0 always outputs $b = 2$ and hence (3) is satisfied trivially by choosing $a = 1$ and $c = 2$.

For the induction step, let $k \geq 1$. Let $(0^n, V)$, B, and S be as in the lemma and let b denote the output of the computation $M_k^{B \triangle S}(0^n, V)$. We show that b satisfies condition (3).

1. Assume that $M_k^{B \triangle S}(0^n, V)$ computes the output according to Case 1. Let $v'_{i,j}$ be the element in row i and column j of V', i.e., $V' = (v'_{i,j})$. So $v_{i,j}$ and $v'_{i,j}$ translate to each other as follows.

$$\begin{aligned} v'_{d,e} &= v_{d,j_e} \quad \text{for } d \in [1, m-1] \\ v'_{d,e} &= v_{d+1,j_e} \quad \text{for } d \in [m, k-1] \end{aligned}$$

M_k defines $b' \stackrel{df}{=} M_{k-1}^{B\triangle S}(0^n, V')$. By induction hypothesis,

$$\exists a', c' \in [1, r_{k-1}(n)] \text{ such that } a' < b' \leq c' \text{ and } \forall i \in [1, k-1], (v'_{i,a'} \in B \Leftrightarrow v'_{i,c'} \in B).$$

The translation of the $v'_{i,j}$ to the corresponding $v_{i,j}$ yields:

$$\exists a', c' \in [1, r_{k-1}(n)] \text{ such that } a' < b' \leq c', \forall i \in [1, m-1], (v_{i,j_{a'}} \in B \Leftrightarrow v_{i,j_{c'}} \in B) \text{ and } \forall i \in [m, k-1], (v_{i+1,j_{a'}} \in B \Leftrightarrow v_{i+1,j_{c'}} \in B)$$

By the definition of M_k, $b = j_{b'}$. So with $a \stackrel{df}{=} j_{a'}$ and $c \stackrel{df}{=} j_{c'}$ it holds that $a < b \leq c$ and

$$\forall i \in [1, k] - \{m\}, (v_{i,a} \in B \Leftrightarrow v_{i,c} \in B).$$

Together with $v_{m,j_{a'}} = v_{m,j_{c'}} = w$ we obtain

$$\forall i \in [1, k], (v_{i,a} \in B \Leftrightarrow v_{i,c} \in B).$$

This proves (3) if the output is made according to Case 1.

2. Assume now that $M_k^{B\triangle S}(0^n, v)$ computes the output according to Case 2. So V does not contain rows in which a word appears at least $r_{k-1}(n)$ times. If we hide repeated words in row 1, then at least $r_k(n)/r_{k-1}(n)$ columns remain unhidden. If we additionally hide repeated words in row 2, then at least $r_k(n)/(r_{k-1}(n))^2$ columns remain unhidden. If we treat the remaining rows in the same way, then at least $r_k(n)/(r_{k-1}(n))^k = 2^k(2kn+2)$ columns remain unhidden. After querying the oracle, the algorithm finds unhidden columns $j_1 < j_2 < \cdots < j_{2kn+2}$ that share the same vector of answers. Let V_a be the matrix that consists of the columns $j_1, \ldots, j_{kn+1}$ and let V_c be the matrix that consists of the columns $j_{kn+2}, \ldots, j_{2kn+2}$. Since these columns are unhidden, both matrices, V_a and V_c, have no rows with multiple occurrences of words. From $|S| \leq n$ it follows that at most n elements of a row (of V_a or V_c) belong to S. So if we delete in V_a (resp., V_c) all columns that contain words from S, then we delete at most kn columns and so at least one column survives. This means that there exist $a' \in [1, kn+1]$ and $c' \in [kn+2, 2kn+2]$ such that the columns $j_{a'}$ and $j_{c'}$ in V do not contain words from S. We already know that both columns share the same vector of answers, i.e.,

$$\forall i \in [1, k], (v_{i,j_{a'}} \in B\triangle S \Leftrightarrow v_{i,j_{c'}} \in B\triangle S).$$

Since none of these words belongs to S,

$$\forall i \in [1, k], (v_{i,j_{a'}} \in B \Leftrightarrow v_{i,j_{c'}} \in B).$$

By the definition of M_k, $b = j_{kn+2}$. So with $a \stackrel{df}{=} j_{a'}$ and $c \stackrel{df}{=} j_{c'}$ it holds that $a < b \leq c$ and

$$\forall i \in [1, k], (v_{i,a} \in B \Leftrightarrow v_{i,c} \in B).$$

This proves (3) if the output is made according to Case 2. □

Theorem 3.9 *If B is $\leq_{\text{btt}}^{\text{p}}$-hard for* NP *and S is sparse, then $B\triangle S$ is $\leq_{\text{T}}^{\text{p}}$-hard for* NP.

Proof The proof uses the left-set technique by Ogihara and Watanabe [OW91] where Lemma 3.8 provides the argument for deleting a single node in the left tree.

Let q be a polynomial such that $|S \cap \Sigma^{\leq n}| \leq q(n)$. Let L be a $\leq^{\mathrm{p}}_{\mathrm{m}}$-complete set for NP that is accepted by the nondeterministic Turing machine N in time p. Define the left set of L as

$$\mathrm{Left}(L) \stackrel{df}{=} \{(x,y) \mid |y| \leq p(|x|) \text{ and } \exists z \in \Sigma^{p(|x|)}, z \leq y1^{p(|x|)-|y|}, N(x) \text{ accepts along path } z\}.$$

$\mathrm{Left}(L)$ belongs to NP and so $\mathrm{Left}(L) \leq^{\mathrm{p}}_{\mathrm{btt}} B$. Hence there exists a $k \geq 1$, a k-ary Boolean function α, and $g_1, \ldots, g_k \in \mathrm{FP}$ such that for all x, y

$$(x,y) \in \mathrm{Left}(L) \Leftrightarrow \alpha(c_B(g_1(x,y)), c_B(g_2(x,y)), \ldots, c_B(g_k(x,y))) = 1.$$

Let p' be a polynomial such that for all $i \in [1,k]$, $|g_i(x,y)| \leq p'(|xy|)$. The following algorithm uses the oracle $B \triangle S$ and decides L on input x.

```
1  m := p'(|x| + p(|x|))        // maximal length of queries of the btt-reduction
2  n := q(m)        // maximal number of words in S ∩ Σ^{≤m}
3  T_0 := (ε)       // list containing the empty word
4  for l = 0 to p(|x|)        // for all stages of the left-tree
5     T_l := list obtained from T_{l-1} by replacing every y ∈ T_{l-1} by y0, y1
6     while(T_l contains at least r_k(n) elements)
7        let y_1, ..., y_{r_k(n)} be the first elements on T_l
8        V := (v_{i,j}) ∈ (Σ^{≤m})^{k×r_k(n)} where v_{i,j} := g_i(x, y_j) for i ∈ [1,k], j ∈ [1, r_k(n)]
9        determine some b ∈ (1, r_k(n)] such that
         ∃a, c ∈ [1, r_k(n)], a < b ≤ c, ∀i ∈ [1,k], (v_{i,a} ∈ B ⇔ v_{i,c} ∈ B)
10       delete y_b from T_l
11    end while
12 next l
13 if T_{p(|x|)} contains an accepting path of N(x) then accept else reject
```

Step 9 needs further explanation, since here we need the polynomial-time oracle transducer M_k constructed in Lemma 3.8. Let $B' \stackrel{df}{=} B \cap \Sigma^{\leq m}$ and $S' \stackrel{df}{=} S \cap \Sigma^{\leq m}$. Note that $B', S' \subseteq \Sigma^{\leq n}$, $|S'| \leq n$, and $V \in (\Sigma^{\leq n})^{k \times r_k(n)}$. By Lemma 3.8, $M_k^{B' \triangle S'}(0^n, V)$ outputs some $b \in (1, r_k(n)]$ such that

$$\exists a, c \in [1, r_k(n)], a < b \leq c, \forall i \in [1,k], (v_{i,a} \in B' \Leftrightarrow v_{i,c} \in B').$$

It is easy to simulate access to the oracle $B' \triangle S' = (B \triangle S) \cap \Sigma^{\leq m}$, since the algorithm presented above has access to the oracle $B \triangle S$. So we can simulate the computation $M_k^{B' \triangle S'}(0^n, V)$ in polynomial time. From $B' = B \cap \Sigma^{\leq m}$ and $|v_{i,j}| \leq m$ it follows that $v_{i,j} \in B' \Leftrightarrow v_{i,j} \in B$. Hence

$$\exists a, c \in [1, r_k(n)], a < b \leq c, \forall i \in [1,k], (v_{i,a} \in B \Leftrightarrow v_{i,c} \in B),$$

which shows that in polynomial time we find the b claimed in step 9.

Observe that for all $i \in [1, p(|x|)]$, T_i is always a sorted list of at most $2r_k(n)$ pairwise disjoint words of length i. (The loop 6–11 deletes elements from T_i until there are no more than $r_k(n)$; so the next iteration of the loop 4–12 starts with a list T_{i+1} that contains at most $2r_k(n)$ elements.) This shows that the presented algorithm works in polynomial time.

Claim 3.10 *If $x \in L$ and $l \in [0, p(|x|)]$, then after the l-th pass of the loop 4–12, the list T_l contains an element y that is a prefix of the left-most accepting path of $N(x)$.*

Proof We show this by induction on $l \geq 0$. The induction base, the case $l = 0$, is trivial. For the induction step let $l \geq 1$ and let $z \in \Sigma^{p(|x|)}$ denote the left-most accepting path of $N(x)$. By induction hypothesis, T_{l-1} contains a prefix of z. So after step 5, T_l contains a prefix of z. Let $y_1, y_2, \ldots$ be the elements of T_l (remember that they are lexicographically ordered and pairwise disjoint). By line 9, we only delete an element y_b from T_l, if there exist $a, c \in [1, r_k(n)]$ such that $a < b \leq c$ and

$$\forall i \in [1,k], (v_{i,a} \in B \Leftrightarrow v_{i,c} \in B). \tag{4}$$

Assume such a y_b is a prefix of z. So $(x, y_a) \notin \mathrm{Left}(L)$ and $(x, y_b) \in \mathrm{Left}(L)$. By (4),

$$(c_B(v_{1,a}), c_B(v_{2,a}), \ldots, c_B(v_{k,a})) = (c_B(v_{1,c}), c_B(v_{2,c}), \ldots, c_B(v_{k,c}))$$

and hence

$$\alpha(c_B(v_{1,a}), c_B(v_{2,a}), \ldots, c_B(v_{k,a})) = \alpha(c_B(v_{1,c}), c_B(v_{2,c}), \ldots, c_B(v_{k,c})).$$

The left-hand side is 1 if and only if $(x, y_a) \in \mathrm{Left}(L)$; the right-hand side is 1 if and only if $(x, y_c) \in \mathrm{Left}(L)$. So $(x, y_a) \in \mathrm{Left}(L) \Leftrightarrow (x, y_c) \in \mathrm{Left}(L)$ and hence $(x, y_c) \notin \mathrm{Left}(L)$. However, $(x, y_b) \in \mathrm{Left}(L)$ although $y_b \leq y_c$. This is a contradiction and hence, from the list T_l, we do not delete prefixes of z. This proves Claim 3.10. □

Now consider line 13. If $x \in L$, then by Claim 3.10, $T_{p(|x|)}$ contains the left-most accepting path of $N(x)$ and so the algorithms accepts. If $x \notin L$, then the algorithm rejects. So the algorithm demonstrates that $L \leq^{\mathrm{p}}_{\mathrm{T}} B \triangle S$. □

Corollary 3.11 *If L is $\leq^{\mathrm{p}}_{\mathrm{btt}}$-hard for* NP *and S is sparse, then $L \cup S$ is $\leq^{\mathrm{p}}_{\mathrm{T}}$-hard for* NP.

Proof Note that $S' \stackrel{df}{=} S - L$ is sparse. By Theorem 3.9, $L \triangle S' = L \cup S$ is $\leq^{\mathrm{p}}_{\mathrm{T}}$-hard for NP. □

Corollary 3.12 *All $\leq^{\mathrm{p}}_{\mathrm{btt}}$-complete sets for* NP *are weakly deterministically self-correctable.*

Proof Let L be $\leq^{\mathrm{p}}_{\mathrm{btt}}$-complete for NP. By Theorem 3.9, for all sparse S, $L \leq^{\mathrm{p}}_{\mathrm{T}} L \triangle S$. By Theorem 2.2, L is weakly deterministically self-correctable. □

Yesha [Yes83] defined two sets A and B to be close if the census of their symmetric difference, $A \triangle B$, is a slowly increasing function. Accordingly, A and B are p-close, if the census of $A \triangle B$ is polynomially bounded. A is p-close to a complexity class $\mathcal{C}$, if there exists some $B \in \mathcal{C}$ such that A and B are p-close. Yesha [Yes83] poses the question of whether $\leq^{\mathrm{p}}_{\mathrm{m}}$- or $\leq^{\mathrm{p}}_{\mathrm{T}}$-hard sets for NP can be p-close to P (assuming P $\neq$ NP). Schöning [Sch86] showed that no $\leq^{\mathrm{p}}_{\mathrm{T}}$-hard set for NP is p-close to P, unless PH $= \Delta^{\mathrm{P}}_2$. Ogiwara [Ogi91] and Fu [Fu93] proved that no $\leq^{\mathrm{p}}_{\mathrm{m}}$-hard set for NP is p-close to P, unless P = NP. We can strengthen the latter result as follows.

Corollary 3.13 *No $\leq^{\mathrm{p}}_{\mathrm{btt}}$-hard set for* NP *is p-close to* P*, unless* P = NP.

Proof Assume that B is $\leq^{\mathrm{p}}_{\mathrm{btt}}$-hard for NP and B is p-close to some $A \in$ P. So $S \stackrel{df}{=} A \triangle B$ is sparse and it holds that $A = B \triangle S$. By Theorem 3.9, A is $\leq^{\mathrm{p}}_{\mathrm{T}}$-hard for NP and hence P = NP. □

3.3 Disjunctive Truth-Table Reductions

In this section we analyze how disjunctive truth-table reductions can handle false positives. We show that the union of dtt-hard sets with arbitrary sparse sets is always T-hard.

Theorem 3.14 *Let L be $\leq^{\mathrm{p}}_{\mathrm{dtt}}$-hard for* NP*, and let S be a sparse set. Then $L \cup S$ is $\leq^{\mathrm{p}}_{\mathrm{T}}$-hard for* NP*.*

Proof Let $L \subseteq \Sigma^*$ and $S \subset \Sigma^*$ be as above, and let M be a nondeterministic Turing-machine whose running time on input x is bounded by polynomial p. Without loss of generality, we assume that on input x, M develops precisely $2^{p(|x|)}$ nondeterministic computation paths. Each path can hence be identified by a word $z \in \{0,1\}^{p(|x|)}$. For a path $z \in \{0,1\}^{p(|x|)}, z \neq 0^{p(|x|)}$, we denote the path on the left of z with $z' - 1$.

Let A be the language accepted by M. We will show that $A \leq^{\mathrm{p}}_{\mathrm{T}} L \cup S$. The left-set of A is defined as

$$\mathrm{Left}(A) \stackrel{df}{=} \{(x,y) \,\big|\, \text{ there exists a } z \geq_{\mathrm{lex}} y \text{ such that } M \text{ accepts } x \text{ along } z\}.$$

From $A \in$ NP it follows that $\mathrm{Left}(A) \in$ NP. Since L is $\leq^{\mathrm{p}}_{\mathrm{dtt}}$-hard for NP, there exists a function f such that $\mathrm{Left}(A) \leq^{\mathrm{p}}_{\mathrm{dtt}} L$ via $f : \Sigma^* \to \mathcal{P}(\Sigma^*)$, $f \in$ FP. By the definition of $\leq^{\mathrm{p}}_{\mathrm{dtt}}$ it holds that $(x,y) \in \mathrm{Left}(A) \Leftrightarrow f(x,y) \cap L \neq \emptyset$.

Without loss of generality, we assume that M does neither accept on its first computation path nor on its last path. Furthermore, we define

$$f_+(x,y) \quad \stackrel{df}{=} \quad f(x,y) \cap (L \cup S)$$

Let q be a polynomial such that for all $x \in \Sigma^*$ and for all $y \in \{0,1\}^{p(|x|)}$ it holds that

$$q(|x|) > \mathrm{census}_S(|f(x,y)|)$$

We will construct a deterministic polynomial time oracle machine N such that the following holds for all x:

$$\begin{aligned} x \in A &\Leftrightarrow \exists y \in \{0,1\}^{p(|x|)} \big((x,y) \in \mathrm{Left}(A)\big) \\ &\Leftrightarrow \exists y \in \{0,1\}^{p(|x|)} (f(x,y) \cap L \neq \emptyset) \\ &\Leftrightarrow N^{L \cup S} \text{ accepts } x. \end{aligned}$$

We describe how N works on input $x \in \Sigma^*$.

```
1. i := 0
2. z_i := 1^{p(|x|)}      //current position in the tree, start with rightmost path
3. F_i := f_+(x, z_i)      //positively answered oracle queries
4. while i < q(|x|)
5.     if f_+(x, 0^{p(|x|)}) − F_i = ∅ then reject
```

```
 6.    determine z_{i+1} ∈ {0,1}^{p(|x|)}, z_{i+1} <_lex z_i such that (f_+(x, z_{i+1}) − F_i) ≠ ∅ and
       (f_+(x, z_{i+1} + 1) − F_i) = ∅
 7.    if M accepts along z_{i+1} then accept
 8.    F_{i+1} := F_i ∪ f_+(x, z_{i+1})     //cull new element from S − L
 9.    i := i + 1
10. end while
11. reject    //this statement is never reached
```

We show that N is a polynomial time machine: As the number of passes of the while loop is bound by a polynomial, it suffices to argue that step 6 can be performed in polynomial time. Note that N can compute the set $f_+(x,z)$ by querying the oracle $L \cup S$ for all elements in $f(x,z)$. Step 6 is an easy binary search: Start with $z_1 := 0^{p(|x|)}$ and $z_2 := 1^{p(|x|)}$. Let z' be the middle element between z_1 and z_2. If $\big(f_+(x,z') - F_i\big) = \emptyset$ then $z_2 := z'$ (i.e., the binary search continues on the left) else $z_1 := z'$ (i.e., the binary search continues on the right). Choose the middle element between z_1 and z_2 and repeat the above steps until a suitable path is found. Consequently, we obtain that N runs in polynomial time.

We now argue that the algorithm is correct, i.e., N accepts x if and only if $x \in A$.

For the only-if part, let us assume that N accepts x. If N accepts in line 7 then it has found an accepting path of M on input x. Hence, $x \in A$. This proves the only-if part.

We now prove the if-part. Let $x \in A$, so there exists a rightmost accepting path of M on input x, say z_{right}. As M does neither accept on the leftmost nor on the rightmost path, it holds that $0^{p(|x|)} <_{\text{lex}} z_{\text{right}} <_{\text{lex}} 1^{p(|x|)}$.

We explain that during the execution of the while loop, the accepting path z_{right} is found.

Claim 3.15 *For $0 \le i \le q(|x|)$, if z_{right} was not found during the first i iterations of the while loop, then the following holds after i iterations:*

1. $\#F_i \ge i$
2. $z_i >_{\text{lex}} z_{\text{right}}$
3. $F_i \subseteq S - L$
4. $f_+(x, 0^{p(|x|)}) - F_i \neq \emptyset$

Proof We prove the claim by induction over i. Let $i = 0$. Since M does not accept on its rightmost path, it follows that $F_0 \cap L = \emptyset$ and hence $F_0 = f_+(x, 1^{p(|x|)}) \subseteq S - L$. Moreover, $z_0 = 1^{p(|x|)} >_{\text{lex}} z_{\text{right}}$. As $x \in L$, it follows that $f_+(x, 0^{p(|x|)}) \cap L \neq \emptyset$. Hence $f_+(x, 0^{p(|x|)}) - F_0 \neq \emptyset$.

Assume the claim does hold for an $i \in \{0, \ldots, q(|x|) - 1\}$. So z_{right} was not found during the first i iterations of the while loop.

Observe that since M accepts on path z_{right}, it holds for all $z' \in \{0,1\}^{p(|x|)}$ that

- $z' \le_{\text{lex}} z_{\text{right}} \Rightarrow f(x,z') \cap L \neq \emptyset \Rightarrow f_+(x,z') \cap L \neq \emptyset$ and

- $z' >_{\text{lex}} z_{\text{right}} \Rightarrow f(x,z') \cap L = \emptyset \Rightarrow f_+(x,z') \cap L = \emptyset$.

Since $i < q(|x|)$, the algorithm proceeds with the i+1-th iteration. By the induction hypothesis, it holds that $f_+(x, 0^{p(|x|)}) - F_i \neq \emptyset$, so the condition in line 5 is not satisfied, hence the while-loop is not left prematurely.

N then determines z_{i+1} such that $z_{i+1} <_{\text{lex}} z_i$, $f_+(x, z_{i+1}) - F_i \neq \emptyset$, and $f_+(x, z_{i+1}+1) - F_i = \emptyset$.

Clearly, such a z_{i+1} must exist since $f_+(x, z_{\text{right}})$ (which is on the left of z_i) contains at least one element from L which cannot have been culled before because $F_i \subseteq S - L$ by the induction hypothesis. The same holds true for all $f_+(x, z')$ where $z' <_{\text{lex}} z_{\text{right}}$. If $z_{i+1} = z_{\text{right}}$, this means that the algorithm has found z_{right} in the $i+1$-th iteration of the while loop. In this case, we are done.

So let us assume for the sake of contradiction that $z_{i+1} <_{\text{lex}} z_{\text{right}}$. Then $f_+(x, z_{i+1}+1) \cap L \neq \emptyset$. This is a contradiction because $f_+(x, z_{i+1}+1) \subseteq F_i \subseteq S - L$. For this reason, z_{i+1} cannot be the path chosen in the i+1-th iteration. It follows that $z_{i+1} >_{\text{lex}} z_{\text{right}}$. This implies $f_+(x, z_{i+1}) \subseteq S - L$.

Recall that $f_+(x, z_{i+1}) - F_i \neq \emptyset$. This means that $f_+(x, z_{i+1})$ contains an element from $S - L$ that has not been culled before, i.e., an element which is not in F_i. It follows that $\#F_{i+1} \geq \#F_i + 1 \geq i + 1$. Finally, $f_+(x, z_{i+1}) \cap L = \emptyset$ implies that $f_+(x, 0^{p(|x|)}) - F_{i+1} \neq \emptyset$. This proves the claim. □

By Claim 3.15 either z_{right} is found during the first $q(|x|)$ iterations of the while loop or $F_{q(|x|)}$ contains at least $q(|x|)$ elements from $S - L$. Together with $q(|x|) > \text{census}_S(|f(x,y)|)$, we obtain that $S - L$ cannot contain this many elements. We conclude that $z_{\text{right}}$is found during the first $q(|x|)$ iterations of the while loop. This proves the theorem. □

Contrary to sections 3.1 and 3.2, we do not know how dtt-reductions react towards false negatives. For that reason, we cannot deduce that dtt-complete sets are weakly deterministically self-correctable. We can provide evidence that the question is indeed difficult. We explain that it is related to the longstanding open question [HOW92] of whether the existence of sparse dtt-complete sets implies P = NP.

Corollary 3.16 *If dtt-complete sets for* NP *are weakly deterministically self-correctable, then the existence of sparse dtt-complete sets for* NP *implies* P = NP.

Proof We assume that dtt-complete sets for NP are weakly deterministically self-correctable and that there exists a sparse set L such that L is dtt-complete for NP. Since L is weakly deterministically self-correctable, it follows from Theorem 2.2 that for all sparse sets S, $L \leq^{\text{p}}_{\text{T}} L \triangle S$. It follows that $L \leq^{\text{p}}_{\text{T}} L \triangle L$ and hence $L \leq^{\text{p}}_{\text{T}} \emptyset$. This implies P = NP. □

3.4 Non-Robustness against sparse sets of False Positives

So far we concentrated on reductions strong enough to manage partly corrupt NP-hard sets. Now we ask for reductions that are provably too weak to handle such corrupt information. Under

the assumption P $\neq$ NP we show that many-one reductions, bounded truth-table reductions, and disjunctive truth-table reductions are weak in this sense. More precisely, altering sparse information in SAT can result in sets that are not $\leq^{\mathrm{p}}_{\mathrm{m}}$-hard, not $\leq^{\mathrm{p}}_{\mathrm{btt}}$-hard, and not $\leq^{\mathrm{p}}_{\mathrm{dtt}}$-hard for NP. On the other hand, Corollary 3.19 shows that similar results for $\leq^{\mathrm{p}}_{\mathrm{ctt}}$, $\leq^{\mathrm{p}}_{\mathrm{tt}}$, and $\leq^{\mathrm{p}}_{\mathrm{T}}$ would imply the existence of NP-complete sets that are not paddable. This explains that such results are hard to obtain.

Theorem 3.17 *The following statements are equivalent.*

1. P $\neq$ NP
2. *There exists a sparse* S *such that* SAT $\cup S$ *is not* $\leq^{\mathrm{p}}_{\mathrm{btt}}$*-hard for* NP.
3. *There exists a sparse* S *such that* SAT $\cup S$ *is not* $\leq^{\mathrm{p}}_{\mathrm{dtt}}$*-hard for* NP.

Proof 1 $\Rightarrow$ 2: Assume P $\neq$ NP and let $M_1, M_2, \ldots$ be an enumeration of polynomial-time oracle Turing machines such that M_i runs in time $n^i + i$ and queries at most i strings (so the machines represent all $\leq^{\mathrm{p}}_{\mathrm{btt}}$-reduction functions). We construct an increasing chain of sets $S_1 \subseteq S_2 \subseteq \cdots$ and finally let $S \stackrel{df}{=} \bigcup_{i \geq 1} S_i$. Let $S_0 \stackrel{df}{=} \{\varepsilon\}$ and define S_k for $k \geq 1$ as follows:

1. let n be greater than k and greater than the length of the longest word in S_{k-1}
2. let $T \stackrel{df}{=} (\mathrm{SAT} \cap \Sigma^{\leq n}) \cup S_{k-1} \cup \Sigma^{>n}$
3. choose a word x such that $M_k^T(x)$ accepts if and only if $x \notin$ SAT
4. let Q be the set of words that are queried by $M_k^T(x)$ and that are longer than n
5. let $S_k \stackrel{df}{=} S_{k-1} \cup Q$

We first observe that the x in step 3 exists: If not, then $L(M_k^T) = \mathrm{SAT}$ and T is cofinite. Hence SAT $\in$ P which is not true by assumption. So the described construction is possible.

If a word w of length j is added to S in step k (i.e., $w \in S_k - S_{k-1}$), then in all further steps, no words of length j are added to S (i.e., for all $i > k$, $S_i \cap \Sigma^j = S_k \cap \Sigma^j$). In the definition of S_k it holds that $|Q| \leq k \leq n$. So in step 5, at most n words are added to S and these words are of length greater than n. Therefore, for all $i \geq 0$, $|S \cap \Sigma^i| \leq i$ and hence S is sparse.

Assume SAT$\cup S$ is $\leq^{\mathrm{p}}_{\mathrm{btt}}$-hard for NP. So there exists a $k \geq 1$ such that SAT$\leq^{\mathrm{p}}_{\mathrm{btt}}SAT\cup S$ via M_k. Consider the construction of S_k and let n, T, x, and Q be the corresponding variables. In all steps $i \geq k$, S will be only changed with respect to words of lengths greater than n. Therefore, $S \cap \Sigma^{\leq n} = S_{k-1}$ and hence

$$\forall w \in \Sigma^{\leq n}, (w \in \mathrm{SAT} \cup S \;\Leftrightarrow\; w \in T). \tag{5}$$

If q is an oracle query of $M_k^T(x)$ that is longer than n, then $q \in Q$ and hence $q \in S_k \subseteq S$. So $q \in$ SAT $\cup S$ and $q \in T$. Together with (5) this shows that the computations $M_k^T(x)$ and $M_k^{\mathrm{SAT} \cup S}(x)$ are equivalent. From step 3 it follows that $M_k^{\mathrm{SAT} \cup S}(x)$ accepts if and only if $x \notin$ SAT. This contradicts the assumption that M_k reduces SAT to SAT $\cup S$. Hence SAT $\cup S$ is not $\leq^{\mathrm{p}}_{\mathrm{btt}}$-hard for NP.

2 ⇒ 1: If P = NP, then for all sparse S, SAT $\cup S$ is trivially $\leq^{\mathrm{p}}_{\mathrm{m}}$-complete for NP.

1 ⇔ 3: Analogous to the equivalence of 1 and 2; we only sketch the differences. We use an enumeration of $\leq^{\mathrm{p}}_{\mathrm{dtt}}$-reduction machines (i.e., machines that nonadaptively query an arbitrary number of strings and that accept if at least one query is answered positively). Moreover, we change the definition of S_k in step 5 such that

$$S_k \stackrel{df}{=} \begin{cases} S_{k-1} & : \text{ if } Q = \emptyset \\ S_{k-1} \cup \{q\} & : \text{ if } Q \neq \emptyset, \text{ where } q = \max(Q). \end{cases}$$

This makes sure that S is sparse.

Assume SAT$\leq^{\mathrm{p}}_{\mathrm{dtt}}$SAT $\cup S$ via M_k. If no query of $M_k^T(x)$ is longer than n, then $M_k^T(x)$ and $M_k^{\mathrm{SAT}\cup S}(x)$ are equivalent computations and hence $L(M_k^{\mathrm{SAT}\cup S}) \neq$ SAT by step 3. Otherwise, there exists a query that is longer than n. Let q be the greatest such query and note that $q \in S_k \subseteq S$. This query gets a positive answer in the computation $M_k^T(x)$. So the computation accepts and by step 3, $x \notin$ SAT. In the computation $M_k^{\mathrm{SAT}\cup S}(x)$, the query q also obtains a positive answer and hence the computation accepts. So also in this case, $L(M_k^{\mathrm{SAT}\cup S}) \neq$ SAT. This shows that SAT $\cup S$ is not $\leq^{\mathrm{p}}_{\mathrm{dtt}}$-hard for NP. □

This tells us that while $\leq^{\mathrm{p}}_{\mathrm{m}}$-hard, $\leq^{\mathrm{p}}_{\mathrm{btt}}$-hard, and $\leq^{\mathrm{p}}_{\mathrm{dtt}}$-hard sets do not become too easy when false positives are added (as they stay NP-hard with respect to more general reducibilities, confer sections 3.1, 3.2, and section 3.3), they are not robust against sparse sets of false positives. The next result says that this is different for hard sets which are paddable.

Proposition 3.18 *Let L be paddable and let S be sparse.*

1. *If L is $\leq^{\mathrm{p}}_{\mathrm{tt}}$-hard for* NP*, then $L \cup S$ is $\leq^{\mathrm{p}}_{\mathrm{tt}}$-hard for* NP*.*
2. *If L is $\leq^{\mathrm{p}}_{\mathrm{T}}$-hard for* NP*, then $L \cup S$ is $\leq^{\mathrm{p}}_{\mathrm{T}}$-hard for* NP*.*
3. *If L is $\leq^{\mathrm{p}}_{\mathrm{ctt}}$-hard for* NP*, then $L \cup S$ is $\leq^{\mathrm{p}}_{\mathrm{ctt}}$-hard for* NP*.*

Proof We start with the first statement. Let M be a polynomial-time oracle Turing machine that witnesses SAT$\leq^{\mathrm{p}}_{\mathrm{tt}}L$ and let p be a polynomial bounding the running time of M. Without loss of generality we may assume that the words queried by $M(x)$ are pairwise different. Let f be a padding function for L and let q be a polynomial bounding both, the computation time for f and the census of S.

We describe a machine M' with oracle $L \cup S$ on input x: First, M' simulates $M(x)$ until the list of all queries $Q = (q_1, \ldots, q_m)$ is computed. Let $k = q(q(2p(|x|)))$ and let $Q_j = (f(q_1, j), \ldots, f(q_m, j))$ for $j \in [0, 2k]$. M' queries all words in Q_j for $j \in [0, 2k]$. Let A_j be the corresponding vectors of answers. For every $j \in [0, 2k]$, M' continues the simulation of M by answering the queries according to A_j. M' accepts if and only if the majority of the simulations accepts.

We argue that SAT$\leq^{\mathrm{p}}_{\mathrm{tt}}L \cup S$ via M'. Note that all $f(q_i, j)$ are pairwise different, since f is injective. For sufficiently large x it holds that $|q_i| \leq p(|x|)$ and $|j| \leq p(|x|)$. So $|f(q_i, j)| \leq q(2p(|x|))$ and hence at most k of the words $f(q_i, j)$ belong to S. Therefore, for the majority of all

$j \in [0, 2k]$, Q_j does not contain a word from S. From the padding property $f(q_i, j) \in L \Leftrightarrow q_i \in L$ it follows that for the majority of all $j \in [0, 2k]$, A_j equals the vector of answers occurring in the computation $M^L(x)$. Hence the majority of all simulations shows the same acceptance behavior as $M^L(x)$. This shows $\mathrm{SAT} \leq^{\mathrm{p}}_{\mathrm{tt}} L \cup S$ via M' and hence $L \cup S$ is $\leq^{\mathrm{p}}_{\mathrm{tt}}$-hard for NP. This shows the first statement.

The remaining statements are shown analogously, where in the $\leq^{\mathrm{p}}_{\mathrm{ctt}}$-case M' accepts if and only if *all* simulations accept. □

In Theorem 3.17 we have seen that $\leq^{\mathrm{p}}_{\mathrm{m}}$-complete, $\leq^{\mathrm{p}}_{\mathrm{btt}}$-complete, and $\leq^{\mathrm{p}}_{\mathrm{dtt}}$-complete sets are not robust against sparse sets of false positives. The following corollary of Proposition 3.18 explains the difficulty of showing the same for $\leq^{\mathrm{p}}_{\mathrm{ctt}}$-complete, $\leq^{\mathrm{p}}_{\mathrm{tt}}$-complete, and $\leq^{\mathrm{p}}_{\mathrm{T}}$-complete sets.

Corollary 3.19 *1. If there exists a $\leq^{\mathrm{p}}_{\mathrm{tt}}$-complete set L in* NP *and a sparse S such that $L \cup S$ is not $\leq^{\mathrm{p}}_{\mathrm{tt}}$-hard for* NP*, then there exist $\leq^{\mathrm{p}}_{\mathrm{tt}}$-complete sets in* NP *that are not paddable.*

2. If there exists a $\leq^{\mathrm{p}}_{\mathrm{T}}$-complete set L in NP *and a sparse S such that $L \cup S$ is not $\leq^{\mathrm{p}}_{\mathrm{T}}$-hard for* NP*, then there exist $\leq^{\mathrm{p}}_{\mathrm{T}}$-complete sets in* NP *that are not paddable.*

3. If there exists a $\leq^{\mathrm{p}}_{\mathrm{ctt}}$-complete set L in NP *and a sparse S such that $L \cup S$ is not $\leq^{\mathrm{p}}_{\mathrm{ctt}}$-hard for* NP*, then there exist $\leq^{\mathrm{p}}_{\mathrm{ctt}}$-complete sets in* NP *that are not paddable.*

4 Unions of Disjoint NP-complete Sets

In this section we consider what happens when the set of false positives that is added to an NP-complete set, itself is NP-complete. This is equivalent to the question of whether unions of disjoint NP-complete sets remain NP-complete. We give sufficient and necessary conditions for this statement to be true. We start with a few definitions.

A set S is *dense* if there is a constant k such that for every n, S contains a string whose length lies between n and n^k.

A set A is *$t(n)$-printable*, if there is a machine M such that for all n, $M(1^n)$ outputs the set $A \cap \Sigma^{\leq n}$ after at most $t(n)$ steps. A set A is *weakly $t(n)$-printable*, if there exists a dense $S \subseteq A$ that is $t(n)$-printable.

A set L is *$t(n)$-immune*, if L is infinite and no infinite subset of L is decidable in time $t(n)$. L is *$t(n)$-printable-immune*, if L is infinite and no infinite subset of L is $t(n)$-printable.

Hypothesis A: There is a language $L \subseteq \overline{\mathrm{SAT}}$ such that $\overline{L \cup \mathrm{SAT}}$ is not 2^{2n}-printable-immune.

Hypothesis B: For all $L \subseteq \overline{\mathrm{SAT}}$ where $L \in$ NP, there is a polynomial p such that $\overline{L \cup \mathrm{SAT}}$ is weakly $p(n)$-printable.

Informally, Hypothesis A states that no 2^{2n}-time bounded algorithm can output strings from $\overline{L \cup \mathrm{SAT}}$ infinitely often, for some L in NP that is disjoint from SAT. Whereas Hypothesis B states that for every L in NP, that is disjoint from SAT, there is a polynomial-time algorithm that outputs strings from $\overline{\mathrm{SAT} \cup L}$. Though Hypothesis A is not the converse of Hypothesis B, it is somewhat close to being converse.

We will show that if Hypothesis A is true, then there exist two disjoint NP-complete sets whose union is not NP-complete. On the other hand, we show that if the Hypothesis B is true, then unions of disjoint NP-complete sets remain NP-complete.

The following proposition is easy to prove.

Proposition 4.1 *If a set L is not $t(n)$-immune, then L has a $2^n t(n)$-printable subset.*

Theorem 4.2 *If Hypothesis A is true, then there exist two disjoint* NP*-complete sets whose union is not* NP*-complete.*

Proof Let L be a set disjoint from SAT such that $\overline{L \cup \mathrm{SAT}}$ is 2^{2n}-printable-immune. By [GPSS06, Theorem 5.9], it suffices to exhibit an NP-complete set A and disjoint set B in NP such that $A \cup B$ is not NP-complete.

Since $L \in \mathrm{NP}$, there is a constant $k \geq 1$ such that L can be decided in time 2^{n^k}. Let $t_1 = 2$, and $t_{i+1} = t_i^{k^2}$. Consider the following sets.

$$\begin{aligned} E &= \{x \mid t_i^{1/k} \leq |x| < t_i^k \text{ and } i \text{ is even}\} \\ O &= \{x \mid t_i^{1/k} \leq |x| < t_i^k \text{ and } i \text{ is odd}\} \\ J &= \{x \mid |x| = t_i \text{ and } i \text{ is even}\} \end{aligned}$$

Since $\overline{\mathrm{SAT} \cup L}$ is 2^{2n}-printable-immune, $\overline{\mathrm{SAT} \cup L}$ is infinite. Since $E \cup O = \Sigma^*$, it must be the case that at least one of $E \cap (\overline{\mathrm{SAT} \cup L})$ or $O \cap (\overline{\mathrm{SAT} \cup L})$ must be infinite. From now we will assume that $O \cap (\overline{\mathrm{SAT} \cup L})$ is infinite. If that were not the case we can interchange the roles of E and O and the proof structure remains similar.

Let $L_J = L \cap J$, $L_o = L \cap O$, and $\mathrm{SAT}_J = \mathrm{SAT} \cap J$. Since SAT is paddable, SAT_J is NP-complete. Clearly, L_J is disjoint from SAT_J. We claim that $\mathrm{SAT}_J \cup L_J$ is not NP-complete. Suppose not, then there is a polynomial-time many-one reduction f from L_o to $\mathrm{SAT}_J \cup L_J$.

We now show that $\overline{\mathrm{SAT} \cup L}$ is not 2^n-immune. By Proposition 4.1, this contradicts Hypothesis A.

Let

$$T = O \cap (\overline{\mathrm{SAT} \cup L}).$$

Recall that T is infinite.

Consider the following sets.

$$\begin{aligned} T_1 &= \{x \in T \mid f(x) \notin J\} \\ T_2 &= \{x \in T \mid f(x) \in J \text{ and } |f(x)| < |x|\} \\ T_3 &= \{x \in T \mid f(x) \in J \text{ and } |f(x)| \geq |x|\} \end{aligned}$$

Clearly $T = T_1 \cup T_2 \cup T_3$. We now show that each of T_1, T_2, and T_3 is finite. Since T is infinite, we obtain a contradiction.

Claim 4.3 *T_1 is finite.*

Proof Suppose not. Since $T \subseteq \overline{\text{SAT} \cup L}$, T_1 is an infinite subset of $\overline{\text{SAT} \cup L}$. Consider the following algorithm for T_1. On input x, if $x \notin O \cap \overline{\text{SAT}}$, then reject. Else, compute $f(x)$. If $f(x) \notin J$, then accept, else reject. Clearly this algorithm runs in time 2^n.

We now show that this algorithm correctly decides T_1. Consider a string x that is accepted by the algorithm. This happens when $x \in O \cap \overline{\text{SAT}}$ and $f(x) \notin J$. If $f(x) \notin J$, then $f(x) \notin SAT_J \cup L_J$. Thus $x \notin L_o$. Since L coincides with L_o on strings from O and $x \in O$, it follows that $x \notin L$. Thus $x \in O \cap \overline{\text{SAT}} \cap \overline{L} = T$. Since $f(x) \notin J$, $x \in T_1$.

Consider a string x that is not accepted by the algorithm. This happens when $x \notin O \cap \overline{\text{SAT}}$ or when $x \in O \cap \overline{\text{SAT}}$ and $f(x) \in J$. In the first case $x \notin T$ and so x can not belong to T_1. In the second case, since $f(x) \in J$, by definition $x \notin T_1$.

Thus T_1 is an infinite subset of $\overline{\text{SAT} \cup L}$ that can be decided in time 2^n. Thus $\overline{\text{SAT} \cup L}$ is not 2^n-immune. This is a contradiction. □

Claim 4.4 *T_2 is finite.*

Proof If T_2 were infinite, then T_2 is an infinite subset of $\overline{\text{SAT} \cup L}$. Consider the following algorithm for T_2. If a string x does not belong to $O \cap \overline{\text{SAT}}$, then reject. Else, compute $f(x)$. If $f(x) \notin J$ or $|f(x)| \geq |x|$, then reject x. If $f(x) \in J$ and $|f(x)| < |x|$, then accept x if and only if If $f(x) \notin \text{SAT}_J \cup L_J$.

Checking whether $x \in O \cap \overline{\text{SAT}}$ takes 2^n time. Computing $f(x)$ and checking whether $f(x) \in E$ and $|f(x)| < |x|$ takes polynomial-time. If $x \in O$, then $t_i^{1/k} \leq |x| < t_i^k$ for an odd integer i. If $f(x) \in J$, it must be the case that $|f(x)| = t_l$ for an even l. Since $|f(x)| < |x|$, it follows that $l < i$. Observe that $t_l^k = t_{l+1}^{1/k}$, thus it follows that $|f(x)| \leq |x|^{1/k}$. Since $\text{SAT}_J \cup L_J$ can be decided in time 2^{n^k}, membership of $f(x)$ in $\text{SAT}_J \cup L_J$ can be decided in time $2^{|x|}$. Thus the above algorithm runs time 2^n.

We now claim that the algorithm correctly decides T_2. The algorithm rejects a string x if $x \notin O \cap \overline{\text{SAT}}$ or $f(x) \notin J$, or $|f(x)| \geq |x|$. In all these cases that algorithm is correct. Consider the case $x \in O \cap \overline{\text{SAT}}$, $f(x) \in J$, and $|f(x)| < |x|$. Now the algorithm accepts if and only if $f(x) \notin \text{SAT}_J \cup L_J$.

If $f(x) \notin \text{SAT}_J \cup L_J$, then it must be the case that $x \notin L_o$. Since L coincides with L_o on strings from O, and $x \in O$, it follows that $x \notin L$. Thus $x \in O \cap \overline{\text{SAT}} \cap \overline{L}$. Since $f(x) \in J$ and $|f(x)| < |x|$, $x \in T_2$. On the other hand if $f(x) \in \text{SAT}_J \cup L_j$, the $x \in L_o$. It then follows that $x \in L$. Thus $x \notin T$ and so $x \notin T_2$.

Thus T_2 is an infinite subset of $\overline{\text{SAT} \cup L}$ that be decided in time 2^n. Thus $\overline{\text{SAT} \cup L}$ is not 2^n-immune. This is a contradiction. □

We now claim that T_3 must also be finite.

Claim 4.5 *T_3 is finite.*

Proof Suppose not. Then T_3 is an infinite subset of $\overline{\mathrm{SAT} \cup L}$. Consider $f(T_3)$. Since $|f(x)| \geq |x|$, $f(T_3)$ is infinite. Since T_3 is a subset of $O \cap \overline{L}$, $T_3 \cap L_o = \emptyset$. Since f is a reduction from L_o to $\mathrm{SAT}_J \cup L_J$, $f(T_3) \cap (\mathrm{SAT}_J \cup L_J) = \emptyset$.

By definition, $f(T_3) \subseteq J$. Since $f(T_3) \cap \mathrm{SAT}_J = \emptyset$, it follows that $f(T_3) \cap \mathrm{SAT} = \emptyset$. Since L_J coincides with L on J, $f(T_3) \subseteq J$, and $f(T_3) \cap L_J = \emptyset$, it follows that $f(T_3) \cap L = \emptyset$. Thus $f(T_3)$ is an infinite subset of $\overline{\mathrm{SAT} \cup L}$.

It remains to show that $f(T_3)$ can be decided in time 2^n. Consider the following algorithm. If the input y does not belong to J, then reject. Else check if there is a string x such that $|x| \leq |y|^{1/k}$ such that $f(x) = y$. If no such string is found, then reject y. Else accept y if and only if $x \in T$.

Searching for x takes time less than $2^{|y|}$. Once such x is found, $|x| \leq |y|^{1/k}$. Since T can be decided in time 2^{n^k}, the time taken by the algorithm is bounded by $2^{|y|}$.

Observe that if a string $y \in f(T_3)$, then by definition $y \in J$ and there is a string x that is in O such that $f(x) = y$ and $|x| \leq |y|$. If $y \in J$, then $|y| = t_i$ for some even i. If $x \in O$, the $|x|$ lies between $t_l^{1/k}$ and t_l^k for some odd l. Since $|x| \leq |y|$, and $t_{l+1} = t_l^{k^2}$, it follows that $|x| \leq |y|^{1/k}$.

The above algorithm accepts y if and only if $y \in J$, there is a string x such that $x \in T$, $f(x) = y$ and $|x| \leq |y|^{1/k}$. By previous argument, the condition $|x| \leq |y|^{1/k}$ is equivalent to $|x| \leq |y|$. Thus the algorithm correctly accepts $f(T_3)$.

Thus $f(T_3)$ is an infinite subset of $\overline{\mathrm{SAT} \cup L}$ that can be decided in time 2^n. This is a contradiction. □

Thus it follows that T must be a finite set which is a contradiction. Thus f can not be a many-one reduction from L_o to $\mathrm{SAT}_J \cup L_J$. Thus $\mathrm{SAT}_J \cup L_J$ is not many-one complete. □

We will now show that if Hypothesis B is true, then NP-complete sets are closed under disjoint unions.

Glaßer et al. [GPSS06, Theorem 5.9] showed that if there exist two disjoint NP-complete sets A and B such that $A \cup B$ if not NP-complete, then there exist two disjoint NP-complete sets C and D that are isomorphic to SAT such that $C \cup D$ is not NP-complete.

Since C is isomorphic to SAT, there is a polynomial-time invertible bijection from Σ^* to Σ^* that is a reduction from C to SAT. Now consider the sets SAT and $f(D)$. Since f is polynomial-time invertible, $f(D)$ belongs to NP. Moreover $f(D)$ is disjoint from SAT. Suppose there is a reduction g from SAT to $\mathrm{SAT} \cup f(D)$, then $f^{-1}g$ becomes reduction from SAT to $C \cup D$. Thus if $\mathrm{SAT} \cup f(D)$ is NP-complete, so is $C \cup D$. Thus we have the following proposition.

Proposition 4.6 *If there exist two disjoint* NP*-complete sets whose union is not* NP*-complete, then there is a set B in* NP *that is disjoint from* SAT *such that* $\mathrm{SAT} \cup B$ *is not* NP*-complete.*

Theorem 4.7 *If Hypothesis B is true, then unions of disjoint* NP*-complete sets are* NP*-complete.*

Proof By the previous proposition, it suffices to show that if L is any set in NP that is disjoint from SAT, then $L \cup \text{SAT}$ is NP-complete.

Let *pad* be a padding function for SAT. Consider the following set

$$B = \{x \mid \exists\, y, |y| = |x|, \text{ and } pad(x, y) \in L\}.$$

Clearly $B \in NP$, and is disjoint from SAT. Thus by our hypothesis, $\overline{\text{SAT} \cup B}$ is weakly p-printable. Let M be an algorithm that on input n outputs a string whose length is between n and n^k.

Consider the following reduction from SAT to $\text{SAT} \cup L$. On input x, let y be a string of length m output by $M(1^{|x|})$. Use the padding function of SAT to obtain a string z of length m such that $x \in \text{SAT}$ if and only if $z \in \text{SAT}$. Output $pad(z, y)$.

A string x is in SAT if and only if z is in SAT. Given the property of the padding function, it follows that $x \in \text{SAT}$, if and only if $pad(z, y) \in \text{SAT}$. It remains to show that if x is not in SAT, then $pad(z, y)$ is not in L. Since $x \notin \text{SAT}$, z also does not belong to SAT. Suppose $pad(z, y)$ belongs to L. Then by the definition of B, y must belong to B. However y belongs to $\overline{\text{SAT} \cup B}$. This is a contradiction. Thus $pad(z, y)$ does not belong to L. Thus $\text{SAT} \cup L$ is NP-complete. □

References

[BH77] L. Berman and J. Hartmanis. On isomorphism and density of NP and other complete sets. *SIAM Journal on Computing*, 6:305–322, 1977.

[BLR93] M. Blum, M. Luby, and R. Rubinfeld. Self-testing/correcting with applications to numerical problems. *Journal of Computer and System Sciences*, 47(3):549–595, 1993.

[For79] S. Fortune. A note on sparse complete sets. *SIAM Journal on Computing*, 8(3):431–433, 1979.

[Fu93] B. Fu. On lower bounds of the closeness between complexity classes. *Mathematical Systems Theory*, 26(2):187–202, 1993.

[GPSS06] C. Glaßer, A. Pavan, A. L. Selman, and S. Sengupta. Properties of NP-complete sets. *SIAM Journal on Computing*, 36(2):516–542, 2006.

[GPSZ06] C. Glaßer, A. Pavan, A. L. Selman, and L. Zhang. Redundancy in complete sets. In *Proceedings 23nd Symposium on Theoretical Aspects of Computer Science*, volume 3884 of *Lecture Notes in Computer Science*, pages 444–454. Springer, 2006.

[HOW92] L. A. Hemachandra, M. Ogiwara, and O. Watanabe. How hard are sparse sets? In *Structure in Complexity Theory Conference*, pages 222–238, 1992.

[LLS75] R. E. Ladner, N. A. Lynch, and A. L. Selman. A comparison of polynomial time reducibilities. *Theoretical Computer Science*, 1:103–123, 1975.

[Ogi91] M. Ogiwara. On P-closeness of polynomial-time hard sets. manuscript, 1991.

[OW91] M. Ogiwara and O. Watanabe. On polynomial-time bounded truth-table reducibility of NP sets to sparse sets. *SIAM Journal on Computing*, 20(3):471–483, 1991.

[Sch86] U. Schöning. Complete sets and closeness to complexity classes. *Mathematical Systems Theory*, 19(1):29–41, 1986.

[Sel88] A. L. Selman. Natural self-reducible sets. *SIAM Journal on Computing*, 17(5):989–996, 1988.

[Yes83] Y. Yesha. On certain polynomial-time truth-table reducibilities of complete sets to sparse sets. *SIAM Journal on Computing*, 12(3):411–425, 1983.