THEORY OF COMPUTING, Volume 9 (22), 2013, pp. 685–702
www.theoryofcomputing.org

Hamming Approximation of NP Witnesses

Daniel Sheldon[*] Neal E. Young[†]

Received August 13, 2012; Revised April 24, 2013; Published August 4, 2013

Abstract: *Given a satisfiable 3-SAT formula, how hard is it to find an assignment to the variables that has Hamming distance at most $n/2$ to a satisfying assignment?* More generally, consider any polynomial-time verifier for any NP-complete language. A $d(n)$*-Hamming-approximation algorithm* for the verifier is one that, given any member x of the language, outputs in polynomial time a string a with Hamming distance at most $d(n)$ to some witness w, where (x,w) is accepted by the verifier. Previous results have shown that, if P≠NP, every NP-complete language has a verifier for which there is no $(n/2 - n^{2/3+\delta})$-Hamming-approximation algorithm, for various constants $\delta \geq 0$.

Our main result is that, if P≠NP, then every paddable NP-complete language has a verifier that admits no $(n/2 + O(\sqrt{n \log n}))$-Hamming-approximation algorithm. That is, one can't get even *half* the bits right. We also consider *natural* verifiers for various well-known NP-complete problems. They do have $n/2$-Hamming-approximation algorithms, but, if P≠NP, have no $(n/2 - n^{\varepsilon})$-Hamming-approximation algorithms for any constant $\varepsilon > 0$.

We show similar results for randomized algorithms.

ACM Classification: F.1.3

AMS Classification: 68Q17

Key words and phrases: complexity theory, inapproximability, approximation algorithms, Hamming distance

1 Introduction

Consider the discrete tomography problem. An instance is specified by numerous two-dimensional x-ray images, each formed by x-raying a three-dimensional object at some angle. A solution is a description of

[*]Supported by NSF Award IIS-1125098.

[†]This material is based upon work supported by the National Science Foundation under Grant No. 1117954.

DOI: 10.4086/toc.2013.v009a022

the internal structure of the object, specifying the density of matter (0 or 1) within each voxel in the object. Given sufficiently many x-ray images, the internal structure can be determined uniquely, yet computing it exactly is (in general) NP-complete. What form of approximate solution is appropriate? One standard measure of an approximate solution would be the extent to which it yields approximately the same x-ray images. By this measure, a "good" approximate solution can have a very different internal structure than any feasible solution. If the goal is to discover the internal structure of the object, then the *Hamming distance* to a feasible solution is a more appropriate metric. This paper is about the computational complexity of computing (possibly infeasible) solutions that have small Hamming distance to feasible solutions, for various NP-complete problems.

1.1 Previous results

Definition of Hamming approximation Here we follow [2, 8] conceptually, but with different terms. By definition, every NP language L has a *verifier* V, such that $L = \{x : V(x,w) \text{ accepts for some } w\}$, where V runs in time polynomial in $|x|$. A string w such that $V(x,w)$ accepts is a *witness* for $x \in L$.

A *$d(n)$-Hamming-approximation algorithm* for a verifier V (or just a *$d(n)$-approximation algorithm*, since this is the only form of approximation considered here) is an algorithm that, given any input (x,n), outputs an n-bit string a that has Hamming distance at most $d(n)$ to some n-bit witness w such that $V(x,w)$ accepts, as long as there is such a witness. If randomized, the algorithm is a $d(n)$-approximation algorithm for V *with probability* $p(n)$ provided it outputs such an a with probability at least $p(n)$.

Arbitrary verifiers are hard to approximate within $n/2 - n^{2/3+\varepsilon}$ Kumar and Sivakumar were the first to study the hardness of Hamming approximation [8]. They showed that if P≠NP every NP-complete language has *some* verifier for which there is no $d(n)$-approximation algorithm, where $d(n) = n/2 - n^{4/5+\delta}$ for some $\delta > 0$. Their verifiers do not use the "natural" witness w, but instead use an error-correcting encoding of w, so that even if nearly half the bits of the encoding are corrupted w can still be recovered.

The hardness threshold $d(n)$ was increased to $n/2 - n^{3/4+\varepsilon}$ for any $\varepsilon > 0$ by Guruswami and Sudan [6], then increased further to $n/2 - n^{2/3+\varepsilon}$ for any $\varepsilon > 0$ by Guruswami and Rudra [5] via the construction of stronger codes. Guruswami and Rudra also showed that list-decodable codes natural cannot prove a hardness threshold above $n/2 - \Theta(\sqrt{n \log n})$. (Note: [4, 5] cite an unpublished draft of the current paper from 2003 [10] that contained the core theorem from the current paper.)

Many natural verifiers are hard to approximate within $n/2 - n^{1-\delta}$ Feige et al. [2] considered the hardness of approximating natural verifiers for specific well-known NP-complete problems. (As an example of a "natural" verifier, the natural verifier for 3-SAT uses witnesses that are n-bit strings, each bit encoding the truth value of one of n variables.) A motivating application was to give evidence that a SAT algorithm by Schöning couldn't be sped up in a particular way. Feige et al. leverage [8], using amplification arguments to extend the results to natural verifiers. Assuming P≠NP, for some small $\delta > 0$, for many standard NP-complete problems, they showed that the natural verifier has no $(n/2 - n^{1-\delta})$-approximation algorithm (even one that works with probability $1/n^c$, assuming RP≠NP).

This result is weaker than previous results in that the hardness threshold is lower, but stronger in that it holds for natural verifiers.

Other related work Gál et al. [3] studied self-reductions of NP-complete problems to their variants that require computing a partial witness. For example, their Theorem 1 says that, given a SAT instance ψ on n variables, one can construct another SAT instance ψ' on $N = n^{O(1)}$ variables such that (with high probability), given the values of any $N^{1/2+\varepsilon}$ of the N variables in a satisfying assignment for ψ', one can compute in polynomial time a satisfying assignment for ψ. (Such results imply that, if RP$\neq$NP, then, given a SAT instance ψ on n variables, *one cannot compute in polynomial time a partial assignment, to any $n^{1/2+\varepsilon}$ of the variables, that has an extension to a satisfying assignment.* However, an easy direct argument shows a stronger result: assuming P$\neq$NP, given any SAT instance, *one cannot compute in polynomial time a partial assignment, to even any single variable, that has an extension to a satisfying assignment.* This is simply because, by fixing the value of that one variable correctly, simplifying the formula and iterating, one could compute a satisfying assignment.)

All of the previous works above rely directly or indirectly on error-correcting codes.

1.2 New results

The "universal" NP verifier is hard to approximate within $n/2 + O(\sqrt{n \log n})$ Our core theorem (Theorem 2.2) concerns the hardness of approximating the standard verifier $V_{\mathcal{U}}$ for the "universal" NP-complete language $\mathcal{U}$ (to be defined shortly). It gives the first hardness result *above* the $n/2$ barrier:

> *Fix any constant $\alpha > 0$. If the verifier $V_{\mathcal{U}}$ has an $(n/2+\sqrt{\alpha n \log n})$-Hamming-approximation algorithm $A_{\mathcal{U}}$, then P=NP. If the verifier has a randomized $(n/2+\sqrt{\alpha n \log n})$-Hamming approximation algorithm that works with probability $1-O(1/(n^{4\alpha+1}\sqrt{\alpha \ln n}))$, then RP=NP.*

The basic idea for the proof is as follows. Given a potential instance of $\mathcal{U}$, to decide in polynomial time whether the instance is in $\mathcal{U}$, run $A_{\mathcal{U}}$ to find a string a such that, if any witness exists, then some witness must be "close" to a. Eliminate from the universe of potential witnesses all strings that are too far from a. This is a polynomial fraction of the universe (Lemma 2.1). Repeat, each time eliminating a polynomial fraction of the remaining candidates. After polynomially many iterations, the universe of remaining candidates will have polynomial size. Use the verifier on each to check if it's a witness.

Of course, calling A with the *same* input repeatedly won't eliminate additional candidates. Instead we modify the verifier each iteration, mapping all u remaining candidate strings to a more "compact" set: the u lexicographically smallest bit strings (among those with $\lceil \log_2 u \rceil$ bits), and recursing on the compacted universe of candidates.

Definition of the universal NP-complete language $\mathcal{U}$ Before discussing further results, here are the definitions of the universal NP-complete language $\mathcal{U}$ and its natural verifier, and three relevant properties.

Language $\mathcal{U}$ contains the triples $(V, x, 1^t)$ where V is (the encoding of) any verifier, x is a string, and 1^t is a "padding" string of t ones, such that, for some *witness* w of length at most t, $V(x,w)$ accepts within t steps. The verifier $V_{\mathcal{U}}$ takes as input any pair $((V,x,1^t),w)$ and accepts if $V(x,w)$ accepts within t steps. The following facts are well known:

Fact 1.1. *$\mathcal{U}$ is NP-complete and $V_{\mathcal{U}}$ is a verifier for $\mathcal{U}$.*

It is easy to see that $V_{\mathcal{U}}$ is also as hard to Hamming approximate as any other NP verifier:

Fact 1.2. *If there is a $d(n)$-Hamming-approximation algorithm (with probability $p(n)$) for $V_{\mathcal{U}}$, then, for any verifier V for any language in NP, there is a $d(n)$-Hamming approximation algorithm (with probability $p(n)$) for V.*

Fact 1.2 follows directly from the fact that V accepts (x,w) if and only if $V_{\mathcal{U}}$ accepts $((V,x,1^t),w)$, for an appropriately chosen t. So, if a string a approximates a witness w such that $V_{\mathcal{U}}((V,x,1^t),w)$ accepts, then a equally well approximates a witness (the same one) such that $V(x,w)$ accepts.

A converse of Fact 1.2 holds for *paddable* NP-complete languages. (In 1977, Berman and Hartmanis observed that all known natural NP-complete languages are paddable and conjectured that all NP-complete languages are [1]. The conjecture is still open. Formally, language L is *paddable* if there are polynomial-time computable functions $\text{pad}(x,p)$ and $\text{extractPad}(x)$ such that, for all strings x and p, (1) $\text{pad}(x,p) \in L$ iff $x \in L$, and (2) $p = \text{extractPad}(\text{pad}(x,p))$.)

Here is the partial converse of Fact 1.2. The reader may safely skip the proof, which is a standard exercise given only for completeness.

Fact 1.3. *For every paddable NP-complete language L, there is a verifier V that is as hard to approximate as $V_{\mathcal{U}}$; that is, if V has a $d(n)$-Hamming-approximation algorithm (working with probability $p(n)$) then $V_{\mathcal{U}}$ has a $d(n)$-Hamming-approximation algorithm (working with probability $p(n)$).*

Proof. Fix L. By the Berman-Hartmanis isomorphism theorem [1, p.312], since both L and $\mathcal{U}$ are paddable, there exists a polynomial-time isomorphism ψ from strings over the alphabet of $\mathcal{U}$ to strings over the alphabet of L, that is, a bijection such that $I \in \mathcal{U}$ iff $\psi(I) \in L$, where ψ and its inverse are computable in polynomial time.

Use ψ and verifier $V_{\mathcal{U}}$ to create the verifier V for L as follows: *$V(x,w)$ accepts iff $V_{\mathcal{U}}(\psi^{-1}(x),w)$ accepts.* This V is a polynomial-time verifier for L, because $x \in L$ iff $\psi^{-1}(x) \in \mathcal{U}$.

Now, let A be any approximation algorithm for verifier V. Use A to create the approximation algorithm $A_{\mathcal{U}}$ for $V_{\mathcal{U}}$: *$A_{\mathcal{U}}(I,n)$ simply returns $A(\psi(I),n)$.* It is easy to verify that $A_{\mathcal{U}}$ runs in polynomial time, and that (for any $I \in \mathcal{U}$ and appropriate n) the string a returned by $A_{\mathcal{U}}(I,n)$ approximates a witness w such that $V(\psi(I),w)$ accepts, and that (by the choice of V) this w is also accepted by $V_{\mathcal{U}}(I,w)$. Thus, $A_{\mathcal{U}}$ approximates $V_{\mathcal{U}}$ just as well as A approximates V. □

With the definitions and properties related to $\mathcal{U}$ noted, we return to discussing new results.

Arbitrary verifiers are hard to approximate within $n/2+O(\sqrt{n\log n})$ By Fact 1.3, the hardness of approximating $V_{\mathcal{U}}$ (Theorem 2.2) extends immediately to every paddable NP-complete language L, for *some* verifier V for L:

Corollary 1.4. *Fix any $\alpha > 0$. For any paddable NP-complete language L, there exists a verifier V for L with the following properties. If P≠NP, verifier V has no $(n/2+\sqrt{\alpha n\log n})$-Hamming-approximation algorithm. If RP≠NP, verifier V has no $(n/2+\sqrt{\alpha n\log n})$-Hamming-approximation algorithm that works with probability $1-O(1/(n^{4\alpha+1}\sqrt{\alpha\log n}))$.*

Previous works proved a hardness threshold of $n/2 - n^{2/3+\varepsilon}$, strictly below $n/2$, so in this sense Corollary 1.4 strengthens those results. Corollary 1.4 is weaker in that previous works did not require paddability of L, but recall that to date no NP-complete language is known to be not paddable.

Many natural verifiers can be approximated within $n/2$ Does a hardness threshold above $n/2$ hold for more natural verifiers? For many natural NP-complete problems the answer is no: their natural verifiers do have $n/2$-approximation algorithms. Observation 3.1 describes trivial $n/2$-approximation algorithms for the decision problems for many unweighted NP optimization problems over sets, including the decision problems for the unweighted variants of Set Cover, Vertex Cover, Independent Set, and Clique. Theorem 3.2 gives combinatorial $n/2$-approximation algorithms for Vertex Cover, Independent Set, and Clique as *optimization problems*. (E. g., *given a graph, compute a minimum-weight vertex cover*. Approximation for optimization problems is a-priori harder than for decision problems, as the budget for the objective function is not given. Enumerating possible budgets doesn't reduce optimization to decision, because successful approximation is not easily verified.)

Many natural verifiers are hard to approximate within $n/2 - n^{\varepsilon}$ For many natural verifiers, recall that the best previous hardness results are due to Feige et al., who show a hardness threshold of $n/2 - n^{1-\delta}$ for a fixed small $\delta > 0$. Our last set of results (Theorems 4.2, 4.3, 4.4) increases this threshold to $n/2 - n^{\varepsilon}$, reducing the exponent $1 - \delta$ to any constant $\varepsilon > 0$, for many of the verifiers they considered. Unlike the previous works, these proofs do not rely on error correcting codes; they use only simple amplification arguments similar to those of Feige et al.

Approximating arbitrary verifiers (upper bounds) The hardness result for randomized approximation algorithms (Corollary 1.4 part (ii)) is roughly tight in the following sense. For any verifier V of any NP-complete language and any fixed $\alpha > 0$, the naive randomized algorithm (guess a random n-bit string) is an $(n/2 + \sqrt{\alpha n \log n})$-approximation algorithm with probability $1 - O(1/(n^{4\alpha}\sqrt{\alpha \log n}))$. (Note that the exponent here is 4α, compared to $4\alpha + 1$ in the hardness result.)

Regarding deterministic approximation algorithms, any verifier has a trivial deterministic $(n-c)$-approximation algorithm for any fixed $c > 0$. This is a weak upper bound, but it is the best possible for any so-called *black-box* algorithm. See Section 5.

2 The universal verifier is hard to approximate within $n/2 + O(\sqrt{n \log n})$

This section proves our core result: the hardness of $(n/2 + \sqrt{\alpha n \ln n})$-approximating the natural verifier $V_{\mathcal{U}}$ for the universal NP-complete language $\mathcal{U}$. The proof is based on the fact that, for any fixed α and n-bit string a, the number of n-bit strings that are *not* within Hamming distance $n/2 + \sqrt{\alpha n \ln n}$ from a is at least a polynomial fraction of all 2^n strings. We prove a particular form of this fact next. The reader can safely skip the proof, which is a standard calculation.

To simplify notation define utility functions $H(n, \alpha) = \sqrt{\alpha n \ln n}$ and $P(n, \alpha) = (n^{4\alpha}\sqrt{\alpha \ln n})$.

Lemma 2.1. *For any constant $\alpha > 0$, the number of $(n-1)$-bit strings that have more than $n/2 + H(n, \alpha)$ ones is $\Omega(2^n/P(n, \alpha))$.*

Proof. Assume for simplicity of notation that n is even. (The case when n is odd is similar.) We first prove a weaker claim: *the number of n-bit strings that have more than $n/2+H(n,\alpha)$ ones is $\Omega(2^n/P(n,\alpha))$.* Then we extend the proof to prove the lemma.

Let $H = 1+\lfloor H(n,\alpha)\rfloor = \Theta(\sqrt{n\log n})$. Let $p_i = \binom{n}{n/2+i}$. The number in question in the claim equals $\sum_{i=H}^{n/2} p_i$. We lower-bound the value of this sum. Consider the ratio between any two successive terms:

$$\frac{p_{i+1}}{p_i} = \frac{1-2i/n}{1+2(i+1)/n}.$$

For $i \le H+n/H = O(H)$, the ratio is at least $1-O(H/n)$, so $p_{(H+n/H)}$ (considering the product of the first $n/H+1$ ratios, for $i \in \{H, H+1, \ldots, H+n/H\}$) is at least $p_H(1-O(H/n))^{O(n/H)} = \Omega(p_H)$. Thus, the first n/H terms in the sum total $\Omega((n/H)p_H)$. A calculation (details below) shows that $p_H = \Omega(2^n/n^{4\alpha+1/2})$. Thus, the sum is $\Omega((n/H)2^n/n^{4\alpha+1/2})$. Plugging in the definitions of H and P and simplifying, this value is $\Omega(2^n/P(n,\alpha))$, proving the claim.

To prove the lemma, first observe that, following the above calculations, the number of n-bit strings with at least $n/2+H+1$ ones (i. e., $\sum_{i=H+1}^{n/2} p_i$) is also $\Omega((n/H)p_H) = \Omega(2^n/P(n,\alpha))$. Deleting the first bit from any such string w yields an $(n-1)$-bit string w' that must have at least $n/2+H$. Each such string w' is obtained from at most two strings w in this manner. Thus, the number of $(n-1)$-bit strings with $n/2+H$ or more ones is at least half the number of n-bit strings with $n/2+H+1$ or more ones. Thus, both numbers are $\Omega(2^n/P(n,\alpha))$, proving the lemma.

Here are the promised details of the calculation that $p_H = \Omega(n^{-4\alpha-1/2})$:

$$\frac{2^n}{p_H\sqrt{n}} = \frac{2^n}{\binom{n}{n/2+H}\sqrt{n}} = O\Big((1-2H/n)^{n/2-H}(1+2H/n)^{n/2+H}\Big) = O\left(\exp(4H^2/n)\right) = O\left(n^{4\alpha}\right).$$

The second step uses Stirling's approximation, $k! = \Theta((k/e)^k\sqrt{k})$, to simplify $\binom{n}{n/2+H}$, followed by a careful simplification of terms. The third step uses $(1+a)^b \le \exp(ab)$ when $|a|<1$. □

We remark without proof that the bounds in Lemma 2.1 are tight up to constant factors: a random $n-1$-bit string has more than $n/2+H(n,\alpha)$ ones with probability $\Theta(1/P(n,\alpha))$.

Here is the core theorem. Recall $H(n,\alpha) = \sqrt{\alpha n\ln n}$, $P(n,\alpha) = (n^{4\alpha}\sqrt{\alpha\ln n})$, and $V_{\mathcal{U}}$ is the verifier for the universal NP-compete language $\mathcal{U}$.

Theorem 2.2. *Fix constant $\alpha > 0$.*

(i) If there is an $(n/2+H(n,\alpha))$-Hamming-approximation algorithm $A_{\mathcal{U}}$ for $V_{\mathcal{U}}$, then P=NP.

(ii) If there is an $(n/2+H(n,\alpha))$-Hamming-approximation algorithm $A_{\mathcal{U}}$ for $V_{\mathcal{U}}$ that works with probability $1-O(1/(nP(n,\alpha)))$, then RP=NP.

Proof. (i) Assume that there exists $A_{\mathcal{U}}$ as in the theorem. We show that $A_{\mathcal{U}}$ can be used to decide $\mathcal{U}$ in polynomial time. Since $\mathcal{U}$ is NP-complete, it follows that P=NP.

The basic idea is the following. Given $(V,x,1^t)$, run $A_{\mathcal{U}}$ to find a string a such that, if any witness exists, then some witness must be "close" to a. Eliminate from the universe of candidates all strings that are too far from a. By Lemma 2.1, this is a polynomial fraction of the universe. Repeat, each time eliminating a polynomial fraction of the remaining candidates. After polynomially many iterations, the universe of possible candidates will have polynomial size. Use the verifier on each one to check if it's a witness.

Of course, calling A with the *same* input repeatedly would not eliminate additional candidates. Instead we modify the verifier each iteration. Here are the details. Let $(V,x,1^t)$ be the given possible member of $\mathcal{U}$. As the algorithm iterates, it will modify V and t, and it will keep an additional variable u, which is the current candidate universe size (initially 2^n). Given the current universe size u, the corresponding universe of candidates will be the set containing the u lexicographically smallest n_u-bit strings, where n_u is $\lceil \log_2 u \rceil$. Abusing notation, let $[u]$ denote this set. (Initially $[u]$ is $[2^n]$, that is, all n-bit strings.) Given any n_u-bit string a, define $\mathcal{N}(a)$, the *neighborhood* of a, to contain all n_u-bit strings whose Hamming distance to a is at most $n_u/2+H(n_u,\alpha)$. (This may include some strings not in the universe $[u]$.) The algorithm to decide membership of $(V,x,1^t)$ in $\mathcal{U}$ is in Figure 1.

alg$_{\mathcal{U}}$(verifier V, string x, string 1^t): *Accepts iff* $(V,x,1^t) \in \mathcal{U}$ *(that is, iff* $\exists w.\ (V(x,w)$ *accepts in* t *steps).)*

```
1.   For each $n = 0,1,2,\ldots,t$:                          Try each possible witness size n.
1.1    Define verifier $W_n(x,w)$ to do the following:        Modify V for precondition in line 3.
           "Accept iff $w \in [2^n]$ and $V(x,w)$ accepts within $t$ steps."
1.2    Take $T$ polynomially larger than $t$, such that $W_n$ necessarily halts within $T$ steps.
1.3    Run check$_{\mathcal{U}}(n, 2^n, (W_n, x, 1^T))$.       check_U is defined below.
2.   Accept $(V,x,1^t)$ if any call to check$_{\mathcal{U}}$ returned True, else reject.
```

check$_{\mathcal{U}}$(integer n, integer u, $(V,x,1^t)$): *If precondition met, returns whether* $\exists w.\ V(x,w)$ *accepts.*

```
3. precondition: $V$ halts within $t$ steps on any input, and $\{w \mid V(x,w)$ accepts$\}$ is a subset of $[u]$.
4.   if $u$ is bounded by a polynomial in $n$:                 Base case.
4.1    Return True if for some $w \in [u]$, $V(x,w)$ accepts. Otherwise return False.
5.   else:
6.1    Let $a \leftarrow A_{\mathcal{U}}((V,x,1^t), n_u = \lceil \log_2 u \rceil)$.   Compute approximate witness, if any witness exists.
6.2    Let $u' \leftarrow |[u] \cap \mathcal{N}(a)|$.          Restrict to smaller universe $[u] \cap \mathcal{N}(a)$.
6.3    Define bijection $\phi$ to map the $i$th element of $[u']$ to the $i$th element of $\mathcal{N}(a) \cap [u]$.
6.4    Define verifier $V_a(x,w')$ to do the following: "Accept iff $w' \in [u']$ and $V(x,\phi(w'))$ accepts."
6.5    Take $t'$ polynomially larger than $t$ (details in proof) such that $V_a$ necessarily halts within $t'$ steps.
6.6    Return check$_{\mathcal{U}}(n, u', (V_a, x, 1^{t'}))$.  Recurse on universe $[u']$.
```

Figure 1: Definition of algorithm alg$_{\mathcal{U}}$, which uses $A_{\mathcal{U}}$ to decide membership in $\mathcal{U}$.

Correctness. First we show that, assuming the precondition in line 3 is met, check$_{\mathcal{U}}(n,u,(V,x,1^t))$ is correct. That is, check$_{\mathcal{U}}$ returns True iff there is a string w such that $V(x,w)$ accepts.

We call such a string w a *witness for* V. In contrast, by a *witness for* $V_{\mathcal{U}}$, we mean a string w such that $V_{\mathcal{U}}((V,x,1^t),w)$ accepts, which by definition is a w such that $V(x,w)$ accepts *within* t *steps*. But note that, by the precondition on t in line 3, w is a witness for V if and only if w is a witness for $V_{\mathcal{U}}$.

By inspection, if $\text{check}_{\mathcal{U}}$ returns on line 4.1, it is correct. Otherwise, lines 6.1-6.6 are executed, so

	V has a witness		
$\Leftrightarrow$	V has an n_u-bit witness		*By precondition on* $[u]$.
$\Leftrightarrow$	$V_{\mathcal{U}}$ has an n_u-bit witness		*By precondition on* t.
$\Leftrightarrow$	$V_{\mathcal{U}}$ has a witness w in $\mathcal{N}(a)$.	*(Line 6.1)*	*By def. of* $A_{\mathcal{U}}$.
$\Leftrightarrow$	V has a witness w in $\mathcal{N}(a)$.		*By precondition on* t.
$\Leftrightarrow$	V has a witness w in $[u] \cap \mathcal{N}(a)$		*By precondition on* $[u]$.
$\Leftrightarrow$	V_a has a witness w'	*(Line 6.4)*	*By def. of* V_a. *Note* $w' = \phi^{-1}(w)$.
$\Leftrightarrow$	$\text{check}_{\mathcal{U}}(n, u', (V_a, x, 1^{t'}))$ accepts	*(Line 6.6)*	*By induction (as precondition is met by* $(V_a, x, 1^{t'})$*).*
$\Leftrightarrow$	$\text{check}_{\mathcal{U}}(n, u, (V, x, 1^t))$ accepts.	*(Line 6.6)*	*By inspection.*

Thus, $\text{check}_{\mathcal{U}}$ is correct. Correctness of the algorithm $\text{alg}_{\mathcal{U}}$ follows just by inspecting $\text{alg}_{\mathcal{U}}$. Namely there is a witness such that $V(x,w)$ accepts within t steps iff there is such a witness with n bits for some $n \le t$. Further, $V(x,w)$ accepts some n-bit witness w within t steps iff $W_n(x,w)$ accepts some w. Finally, the input $(n, 2^n, (W_n, x, 1^T))$ meets the precondition for the call to $\text{check}_{\mathcal{U}}$, so that call correctly determines whether there is a w such that $W_n(x,w)$ accepts.

Running time. To disambiguate, let $(V_0, x, 1^{t_0})$ denote the original input to $\text{alg}_{\mathcal{U}}$. Below, by "polynomial," we mean polynomial in the length of this input.

We first show that, for any n, when $\text{alg}_{\mathcal{U}}$ calls $\text{check}_{\mathcal{U}}(n, 2^n, (W_n, x, 1^T))$, it results in only polynomially many recursive calls. Consider an execution of lines 6.1-6.6 of $\text{check}_{\mathcal{U}}$. Let $d = n/2 + H(n_u, \alpha)$. By Lemma 2.1, the number of strings in $[2^{n_u-1}]$ that have distance more than d from the string 0^{n_u-1} is $\Omega(2^{n_u}/P(n_u,\alpha))$. By symmetry, the number of strings in $[2^{n_u-1}]$ have distance more than d from the last $n_u - 1$ bits of a is $\Omega(2^{n_u}/P(n_u,\alpha))$ (using here that $[2^{n_u-1}]$ contains exactly all (n_u-1)-bit strings). By definition of n_u, we have $u > 2^{n_u-1}$, so, for each such string z, the n_u-bit string $0z$ is in $[u]$ and has distance at least d from a. Thus, $\Omega(2^{n_u}/P(n_u,\alpha))$ strings in $[u]$ have distance more than d from a, so $u - u' = \Omega(2^{n_u}/P(n_u,\alpha))$. This implies that $\text{check}_{\mathcal{U}}$ recurses $O(P(n_u,\alpha))) = O(P(n,\alpha))$ times before n_u decreases by 1. Since $1 \le n \le n_u$ throughout, this in turn implies that $\text{check}_{\mathcal{U}}$ recurses $O(nP(n,\alpha))$ times total.

Next consider how the encoding size of the arguments $(n, u, (V, x, 1^t))$ grows as the computation progresses. By inspection of $\text{alg}_{\mathcal{U}}$ each triple $(W_n, x, 1^T)$ has polynomial encoding size. With each recursive call made by $\text{check}_{\mathcal{U}}$, the size of the encoding of the arguments remains polynomial, because u and n_u do not increase and, in line 6.4, the encoding of the verifier V_a is only an additive polynomial larger than the encoding of V (the verifier V_a has encoded in it V, along with u' and a, which each have $n_u \le n \le t_0$ bits). We verify below that, when the verifier V_a runs, the check "$w' \in [u']$" and the computation of $\phi(w')$ from w' can be done in polynomial time, so that t' is only an additive polynomial larger than t. Thus, with each recursive call, the encoding size of the arguments grows by an additive polynomial. Since there are polynomially many calls, the encoding size remains polynomial.

Next we consider the running time of each call $\text{check}_{\mathcal{U}}(n, u, (V, x, 1^t))$. First consider the case that lines 6.1-6.6 execute. Line 6.1 executes in polynomial time by the assumption on $A_{\mathcal{U}}$ (using that the encodings of V and t are polynomial, as shown above). Implement line 6.2 in polynomial time as follows. Let $N(s)$ denote the number of n_u-bit binary strings having string s as a prefix and having Hamming distance at most $d = \lfloor n_u/2 + H(n_u,\alpha) \rfloor$ to a. The first $|s|$ bits of such a string agree with s; the remaining $n_u - |s|$ bits differ in up to $d - \ell$ places from a, where ℓ is the Hamming distance between s and the first $|s|$ bits of a. Thus, $N(s) = \sum_{j=0}^{d-\ell} \binom{n_u - |s|}{j}$. Thus, given s, $N(s)$ can be computed in polynomial time. Now, compute u' in line 6.2 in polynomial time via the identity

$$u' = \sum_{\ell : b_\ell = 1} N(b_1 b_2 \cdots b_{\ell-1} 0)\,,$$

where b is the binary representation of u (with $b_1 = 1$ being the most significant bit). (Each string w in $\mathcal{N}(a) \cap [u]$ is lexicographically less than b, so is counted by the term for ℓ in the sum where $\ell = \min\{i \mid b_i = 1, w_i = 0\}$.) Each term is computable in polynomial time as described in the previous paragraph, so the sum is computable in polynomial time.

Likewise, in line 6.4, the verifier V_a can compute $w = \phi(w')$ given any $w' \in [u']$ in polynomial time as follows. Abusing notation for just this paragraph, for any binary string z, let $[z]$ denote the set of $|z|$-bit binary strings that are lexicographically no larger than z. Compute $i = |[w']|$, the (lexicographic) rank of w' in $[u']$ (to do so, use the fact that w' is the binary representation of $i-1$). Then the w we want (by definition of ϕ) is the one with rank i within $\mathcal{N}(a) \cap [u]$. In other words, w is the string such that $|\mathcal{N}(a) \cap [w]| = i$. For any string z, using the previous paragraph, we can compute $|\mathcal{N}(a) \cap [z]|$ in polynomial time, so we can use binary search over $z \in [u]$ (checking $|\mathcal{N}(a) \cap [z]| \le i$) to find w in polynomial time.

Finally, when the base case is reached (line 4.1 is executed) the verifier V is actually run on (polynomially many) inputs (x, w). Since V halts within t steps, and (as argued previously) t is polynomial, this step takes polynomial time. This completes the proof for part (i).

Proof of part (ii). Assume that $A_{\mathcal{U}}$ exists as in part (ii) of the theorem statement. Consider using that $A_{\mathcal{U}}$ directly in the algorithm $\mathrm{alg}_{\mathcal{U}}$ described in the proof for part (i). First consider the case that the resulting algorithm is given a positive instance of $\mathcal{U}$, that is, an instance that has some witness w. Let n be the length of the witness, and consider the corresponding call to $\mathrm{check}_{\mathcal{U}}$ by $\mathrm{alg}_{\mathcal{U}}$. As observed in analyzing the number of iterations, for each $n_u \in \{\lfloor c \log n \rfloor, \ldots, n\}$, this call results in $O(P(n_u,\alpha))$ recursive calls to $\mathrm{check}_{\mathcal{U}}$ with that n_u. Each recursive call calls $A_{\mathcal{U}}$ once. Since $A_{\mathcal{U}}$ is randomized and (by assumption) has probability $O(1/n_u P(n_u,\alpha))$ of failure on any input $((V, x, 1^t), n_u)$, the probability that none of these calls to $A_{\mathcal{U}}$ fails is at least

$$\prod_{n'=c\log n}^{n} \left(1 - O\Big(\frac{1}{n' P(n',\alpha)}\Big)\right)^{O(1/P(n',\alpha))} \le \exp\left(-O\Big(\sum_{n'=1}^{n} \frac{1}{n'}\Big)\right) = \exp\big(-O(\log n)\big) = \frac{1}{n^{O(1)}}\,.$$

Thus, the algorithm has probability $1/n^{O(1)}$ of accepting the input. Since the algorithm never has false positives, this shows that $\mathcal{U} \in \mathsf{RP}$. Since $\mathcal{U}$ is NP-complete, the result follows. □

3 Many natural verifiers can be approximated within $n/2$

By Corollary 1.4 to Theorem 2.2, every (paddable) NP-complete problem has *some* verifier that is hard to approximate within $n/2 + O(\sqrt{n \log n})$. These hard verifiers are not natural verifiers for the problems in question. Next we observe that that natural verifiers can be easier to approximate: for many NP-complete problems, there are $n/2$-approximation algorithms for a natural verifier.

As examples of natural verifiers, the natural verifier for 3-SAT uses witnesses that are n-bit strings, where the ith bit is 1 if the ith variable is assigned True. For Vertex Cover (and other optimization problems over subsets), the natural verifier uses witnesses that are n-bit strings, where the ith bit is 1 if the ith vertex is in the cover.

For a few problems, $n/2$-approximation algorithms follow trivially by symmetry. For example, for any instance of not-all-equal 3-SAT (NAE-3SAT), for any satisfying assignment, complementing all the variables also gives a satisfying assignment. Thus, the all-False assignment gives an $n/2$-approximation.

Next focus on decision versions of optimization problems. For intuition, consider the decision version of unweighted Vertex Cover: *Given a graph $G = (V,E)$ and an integer $k \leq |V|$, does there exist a vertex cover (a subset of vertices incident to every edge) of size at most k?* Let $n = |V|$ be the number of vertices.

The natural verifier V_{VC}, on input $((G,k),C)$, where $C \subseteq V$, accepts iff $|C| \leq k$ and C is a vertex cover. Here is a trivial $n/2$-approximation algorithm A_{VC}: *On input $((G = (V,E),k),n)$, if $k \leq n/2$, output the empty set, otherwise output V.* We assume here the natural encoding of the set C as an n-bit string, whose ith bit determines whether C contains the ith vertex in V.

To see that A_{VC} is an $n/2$-approximation algorithm, suppose that G does have a vertex cover C of size at most k. Then there also exists a vertex cover C' of size exactly k. If $k \leq n/2$, then $|C'| \leq n/2$, so C' has Hamming distance at most $n/2$ from the empty set. Otherwise $k > n/2$ and $|C'| > n/2$, so C' has Hamming distance at most $n/2$ from V.

The same idea gives $n/2$-approximation algorithms for the unweighted versions of similarly structured NP-complete problems such as Set Cover, Independent Set, and Clique. (The latter two are maximization problems, but the idea is the same.)

Observation 3.1. Consider any NP verifier V_1 that, on input $((I,k),w)$, accepts only if w is the natural encoding of a subset S of some n-element universe $U(I)$ such that $|S| \leq k$, or any NP verifier V_2 that, on input $((I,k),w)$, accepts only if w is the natural encoding of a subset S of some n-element universe $U(I)$ such that $|S| \geq k$.

Any such verifier V_1 or V_2 has an $(n/2)$-Hamming-approximation algorithm.

(Note that "only if" in the observation is not "if and only if.")

Observation 3.1 applies to many problems (unweighted Vertex Cover, Independent Set, Clique, Set Cover, Feedback Vertex Set, and, e. g., Hamiltonian Path if the witness is encoded as an edge set), but it is somewhat unsatisfying in that it is an artifact of the fact that the problems are presented as *decision* problems, and not in their more natural *optimization* form. For example, the optimization form of unweighted Vertex Cover is, *given a graph G, compute a Vertex Cover C of minimum size*. In this context, by a *$d(n)$-approximation algorithm*, we mean a polynomial-time algorithm that, given just G and n (but not the size of the desired cover!) outputs a subset S of the n vertices, where S has Hamming distance at most $d(n)$ from some *minimum-size* vertex cover.

Theorem 3.2. *There are $n/2$-Hamming-approximation algorithms for the optimization versions of unweighted Vertex Cover, Independent Set, and Clique.*

Proof. The algorithms for Independent Set and Clique work by standard reductions to Vertex Cover. The algorithm for Vertex Cover is based on a classic result of Nemhauser and Trotter.

Proposition 3.3 ([9, 7]). *Fix any instance I of Vertex Cover. Let y be any (minimum cost) basic feasible solution to the linear program relaxation of the standard integer linear program for I. Then, for each vertex v, the variable y_v has value in $\{0,1/2,1\}$, and there exists an optimal vertex cover C^* that has the following property. For each vertex v, if $y_v = 0$, then $v \notin C^*$, while if $y_v = 1$, then $v \in C^*$.*

The $n/2$-approximation algorithm for the optimization form of unweighted Vertex Cover is as follows: *Given the instance G, compute the minimum-cost basic feasible solution y, then output the set of vertices $S = \{v \in V \mid y_v > 0\}$.*

Since the basic feasible solution y can be computed in polynomial time, the algorithm clearly runs in polynomial time. Next we prove the approximation guarantee. Let C^* be the optimal vertex cover from Proposition 3.3. Since there is a feasible solution of cost $|C^*|$ to the linear program, while y is a minimum-cost solution, it follows that $|C^*| \geq \sum_i y_i$. The choice of S implies $\sum_i y_i \geq |S|/2$. Thus, $|C^*| \geq |S|/2$.

The choice of S also implies that $C^* \subseteq S$. This and $|C^*| \geq |S|/2$ imply that C is within Hamming distance $|S|/2$ (which is at most $n/2$) from C^*.

This proves the approximation guarantee for Vertex Cover. The other problems follow, because the standard reductions between these problems preserve Hamming approximation. Here are the details:

The $n/2$-approximation algorithm for Independent Set is as follows: *Given the instance G, run the $n/2$-approximation algorithm for Vertex Cover. Let S be the output. Return the complement of S, i. e., $\overline{S} = V - S$.* The output is an $n/2$-approximation because a vertex set $I^* \subseteq V$ is a maximum independent set if and only if its complement $\overline{I}^* = V - I^*$ is a minimum vertex cover.

The $n/2$-approximation algorithm for Clique is as follows: *Given the instance $G = (V,E)$, compute the complement graph $G' = (V,\overline{E})$ with the same vertex set but whose edge set is the complement of E. Run the $n/2$-approximation algorithm for Independent Set on G'. Let S be the output. Return S.* The output is an $n/2$-approximation because a vertex set is a maximum clique in G if and only if it is a maximum independent set in $\overline{G}$. □

4 Many natural verifiers are hard to approximate within $n/2 - n^{\varepsilon}$

This section describes how to strengthen many of the hardness results of Feige et al., to show that, for many of the natural verifiers that they consider, (for any $\varepsilon > 0$, assuming P≠NP) there are no $(n/2 - n^{\varepsilon})$-approximation algorithms. The proofs here are elementary padding arguments, similar in spirit to the those of Feige et al.

The proofs use the following standard observation:

Observation 4.1.

(i) If the following problem has a polynomial-time algorithm, then P=NP: Given a 3-SAT formula, find a *feasible value* for the first variable in the formula, if one exists.

(ii) If there is a randomized polynomial-time algorithm that solves any n-variable instance of the above problem with probability $1/2 + 1/n^c$ for any fixed $c > 0$, then RP=NP.

If the formula is satisfiable, a *feasible* value for the variable is one that is consistent with some satisfying assignment. (If the formula is not satisfiable, any value can be found.)

Part (i) holds by standard arguments. To see why part (ii) is true, note that the randomized algorithm could be used to find a satisfying assignment of a given formula with high probability: to determine the likely value of the first variable, run the randomized algorithm, say, n^{c+2} times, then take the majority value—this standard amplification trick boosts the probability of finding a feasible value to at least $1-1/n^2$. Then substitute the likely feasible value for the variable, simplify the formula, then recurse. This would find a full satisfying assignment with probability at least $1-O(1/n)$ in polynomial time, showing that RP=NP.

We start by showing hardness of approximating the natural 3-SAT verifier:

Theorem 4.2. *Fix any constants* $\varepsilon, c > 0$.

(i) If the natural verifier for 3-SAT has an $(n/2-n^\varepsilon)$*-Hamming-approximation algorithm A, then P=NP.*

(ii) If that verifier has a randomized $(n/2-n^\varepsilon)$*-Hamming-approximation algorithm A working with probability* $1/2+1/n^c$*, then RP=NP.*

Proof. (i) The proof is an elementary amplification argument.

Assume without loss of generality that $1/\varepsilon$ is an integer (otherwise replace ε by $1/\lceil 1/\varepsilon \rceil$).

Assume the algorithm A in the statement of Theorem 4.2 exists. Given any 3-SAT formula ψ, we compute, in polynomial time, a feasible value for the first variable z as follows.

To ψ, add $N = n^{1/\varepsilon}$ copies of z. Specifically, add new clauses $(z^1 = z) \wedge (z^2 = z) \wedge \cdots \wedge (z^N = z)$ (where $a = b$ is shorthand for $(a \vee \overline{b}) \wedge (\overline{a} \vee b)$, and $z^1, z^2, \ldots, z^N$ are new variables). This gives a formula ψ' with $n' = n + n^{1/\varepsilon}$ variables, essentially preserving any satisfying assignments, but forcing the $n^{1/\varepsilon}$ added variables to take the same value as z in any satisfying assignment.

Run the approximation algorithm A on ψ', and let a' be the returned value. If ψ' is satisfiable, then a' achieves Hamming distance at most $n'/2-(n')^\varepsilon$ to an assignment w' satisfying ψ'. That is, a' agrees with w' on at least $n'/2+(n')^\varepsilon > n^{1/\varepsilon}/2+n$ variables. To do so, even if x agrees with w' on *all* n of the original variables, it would still have to agree with w' on more than *half* $(n^{1/\varepsilon}/2)$ of the duplicates of z. Thus, the majority value of the duplicate variables in a' (true or false, whichever a' assigns to more duplicates) must be the value that w' assigns to z. This value must also be a feasible value for z in ψ (if one exists).

Thus, if A exists, then one can compute a feasible value for z in ψ (if one exists) in polynomial time. By Observation 4.1, then, P=NP.

(ii) Assume the algorithm A exists, and use it as in the proof for part (i). Given the formula ψ with n variables, call A on the formula ψ' with n' variables, where $n' = n + n^{1/\varepsilon}$. By the properties of A assumed in the theorem, the probability that the call succeeds in finding a feasible value for z (if one exists) is $1/2+1/(n')^c \geq 1/2+1/n^{O(c/\varepsilon)}$. Thus, by Observation 4.1 (b), RP would equal NP. □

Next we sketch how the same idea applies to other problems.

Theorem 4.3. *Fix any constant* $\varepsilon > 0$.

(i) If the natural verifier for unweighted Vertex Cover, Independent Set, or Clique has an $(n/2-n^{\varepsilon})$-Hamming-approximation algorithm, then P=NP.

(ii) If any of these verifiers has a randomized $(n/2-n^{\varepsilon})$-Hamming-approximation algorithm working with probability $1/2+1/n^c$, then RP=NP.

Proof. (i) We sketch the proof for Vertex Cover. The rest follow via standard reductions, from Vertex Cover to Independent Set and from Independent Set to Clique, as these reductions preserve Hamming approximation.

Suppose such an algorithm A exists. Given a graph $G=(V,E)$ with n vertices, a non-isolated vertex $v\in V$, and the minimum size, k, of any vertex cover in G, we will use A to determine in polynomial time either (a) that G has a size-k vertex cover containing v, or (b) that G has a size-k vertex cover not containing v. By standard arguments, if this can be done in polynomial time, then P=NP.

Determine (a) or (b) as follows. Construct graph G' from G by adding a copy v' of v (with edges to all neighbors of v), and a path P (of new vertices) connecting v to v', so that $|P|$ (the number of edges in P) is even and roughly equals $n^{1/\varepsilon}$. Let $n'=n+|P|$ be the number of vertices in G', and let $k'=k+|P|/2$. Denote the successive vertices on path P as $v=v_0,v_1,v_2,\ldots,v_{|P|}=v'$. Let P_0 contain the "even" vertices $v_2,\ldots,v_{|P|}=v'$ (not including v). Let P_1 contain the "odd" vertices $v_1,v_3,\ldots,v_{|P|-1}$.

Run A on the instance $\langle G',k'\rangle$ and let C_A (interpreted as a vertex set) be the output. Define the Hamming distance of C_A to P_i ($i\in\{0,1\}$) to be the number of vertices $v\in P$ that are in exactly one of the two sets C_A, P_i. Return *(a) G has a size-k vertex cover containing v* if the Hamming distance from C_A to P_0 is less than the Hamming distance from C_A to P_1. Otherwise, return *(b) G has a size-k vertex cover not containing v.*

To finish the proof we sketch why this procedure is correct. Let C' be any minimum-size vertex cover of G'. By standard arguments, C' has size k' and one of two cases holds:

Case (1) $C'=C\cup P_0$ where C is a size-k vertex cover of G and $v\in C$, or

Case (2) $C'=C\cup P_1$ where C is a size-k vertex cover of G and $v\notin C$.

By assumption, C_A has Hamming distance at most $n'/2-(n')^{\varepsilon}$ to some such C'. That is, C_A agrees with C' on at least $n'/2+(n')^{\varepsilon}>n^{1/\varepsilon}/2+n$ vertices. Thus, focusing just on vertices in P, C_A agrees with C' on more than half of the vertices in P.

If Case (1) above occurs for C', then the Hamming distance from C' to P_0 must be less than $|P|/2$, so the Hamming distance from C' to P_1 must be more than $|P|/2$, so the algorithm returns *(a) G has a size-k vertex cover containing v.* This is correct, given that Case (1) occurs.

By a similar argument, if Case (2) occurs, then the algorithm returns *(b) G has a size-k vertex cover not containing v*, which is correct in this case.

This proves part (i). The proof for part (ii) follows just as part (ii) of Theorem 4.2 follows from part (i) in the proof of Theorem 4.2. □

Theorem 4.4. *Fix any constants $\varepsilon,c>0$.*

(i) Suppose that, for unweighted, directed Hamiltonian Cycle, the verifier that uses edge-subsets for witnesses has an $(n/2-n^{\varepsilon})$-Hamming-approximation algorithm. Then P=NP.

(ii) Suppose that verifier has a randomized $(n/2-n^{\varepsilon})$-Hamming-approximation algorithm that works with probability $1/2+1/n^c$. Then RP=NP.

Proof. (i) Consider any polynomial-time reduction from 3-SAT to unweighted, directed Hamiltonian Cycle. By definition, such a reduction works as follows. Given a 3-SAT formula ψ, the reduction produces, in polynomial time, a directed graph $G=(V,E)$ such that G has a Hamiltonian cycle if and only if ψ is satisfiable; further, given any Hamiltonian cycle C in G, the reduction describes how to compute an assignment $A(C)$ satisfying ψ in polynomial time.

There exist well known reductions (e. g., see [11]) such that ψ, G, and $A(\cdot)$ have the following further properties. For any variable z in ψ, there are a pair of edges (u,v) and (v,u) such that, for any Hamiltonian cycle C, either C contains (u,v) and $A(C)$ assigns $z=$ True, or C contains (v,u) and $A(C)$ assigns $z=$ False.

Assume the algorithm A from the observation exists. We describe below how to modify any reduction with the above properties so as to solve the following problem in polynomial time: *Given a 3-SAT formula ψ, determine a feasible value for the first variable in ψ (if any exists)*. By Observation 4.1, this is enough to prove P=NP.

Given ψ, apply the reduction with the above properties to compute the graph $G=(V,E)$. Then, for the first variable, say, z in ψ, let (v,v') and (v',v) be the two edges in G for z as described above. Replace the edges (v,v') and (v,v'), respectively, with paths $P_0=(v=v_0,v_1,v_2,\ldots,v_k,v_{k+1}=v')$ and $P_1=(v,v_k,v_{k-1},\ldots,v_1,v')$, where $v_1,v_2,\ldots,v_k$ are new vertices and $k=n^{1/\varepsilon}/2$, where $n=|E|$ is the number of edges in G. Say that each edge (v_i,v_{i+1}) is a *duplicate* of (v,v'), and that each edge (v_{i+1},v_i) is a duplicate of (v',v). Let $P=P_0\cup P_1$ be the set of all duplicate edges. Call the resulting graph G', and let $n'=n+n^{1/\varepsilon}$ be the number of edges in G'.

Run the algorithm A on G', and let P' be the output (interpreted as an edge set). Define the Hamming distance from P' to P_i (for $i=0,1$) to be the number of edges e in P that are in exactly one of the two sets P', P_i. If the Hamming distance from P' to P_0 is less than the Hamming distance from P' to P_1, then return *(a) The value True is feasible for the variable z* and otherwise return *(b) The value False is feasible for the variable z.*

To finish, we prove that this procedure determines a feasible value for z, if there is one.

Assume that there is a feasible value for z (that is, that ψ is satisfiable). The output P' of A then has Hamming distance at most $n'/2-(n')^{\varepsilon}$ to some Hamiltonian cycle C' in G'. That is, P' agrees with some C' on at least $n'/2+(n')^{\varepsilon}>n^{1/\varepsilon}/2+n$ edges. Thus, P' agrees with C' on strictly more than $n^{1/\varepsilon}/2$ (half) of the edges in P. By the properties of the reduction, one of two cases holds:

Case 1. $C'\cap P=P_0$, and True is a feasible value for z. In this case, the Hamming distance from P' to P_0 must be less than $|P|/2$, so the Hamming distance from P' to P_1 must be more than $|P|/2$, so the procedure returns *(a) The value True is feasible for the variable z*, which is correct.

Case 2. $C'\cap P=P_1$, and False is a feasible value for z. By similar reasoning, the procedure is correct in this case as well.

This proves part (i). The proof for part (ii) follows just as part (ii) of Theorem 4.2 follows from part (i) in the proof of Theorem 4.2. □

5 Black-box algorithms for approximating arbitrary verifiers

By Facts 1.2 and 1.3, the problem of approximating arbitrary NP verifiers is equivalent to the problem of approximating the verifier $V_{\mathcal{U}}$ for the universal NP-complete language $\mathcal{U}$.

Randomized algorithms. The best randomized approximation algorithm for $V_{\mathcal{U}}$ that we know of is the trivial algorithm: guess n random bits. By Lemma 2.1, this achieves $(n/2+H(n,\alpha))$-Hamming approximation with probability $1-O(1/P(n,\alpha))$. (As observed in the introduction, this nearly matches the main hardness result here for randomized approximation algorithms for $V_{\mathcal{U}}$.)

Deterministic algorithms. The best deterministic polynomial-time approximation algorithm for $V_{\mathcal{U}}$ that we know of is as follows: *Test each n-bit candidate string with c or fewer 1's. If one is a witness, return it, otherwise return* 1^n. This is an $(n-c)$-approximation algorithm (for any constant c). (Note that 1^n is within Hamming distance $n-c$ of all untested strings, and therefore within Hamming distance $n-c$ of any witness if the algorithm returns 1^n.)

Lower bound for deterministic "black-box" algorithms. An adversary argument shows that the above algorithm is near-optimal among deterministic algorithms that use the verifier $V_{\mathcal{U}}$ as a *black box* (by which we mean that, given any possible instance $I=(V,x,1^t)$, the algorithm determines information about I only by querying the NP verifier $V_{\mathcal{U}}(I,w)$ with this I and various potential witnesses w). The adversary behaves as follows: *Whenever the algorithm queries $V_{\mathcal{U}}(I,w)$ with a given choice of w, the adversary has $V_{\mathcal{U}}$ return "no."* Suppose for contradiction that the algorithm runs in $o(n^{c-1})$ time for some constant c, before it returns its answer a. There are at least $\binom{n}{n-c+1}=\Omega(n^{c-1})$ strings whose Hamming distance to a is $n-c+1$. At least one of these strings w' was not queried by the algorithm. The adversary can take w' to be the true witness, that is, it can make V be such that $V(x,w)$ accepts in t steps iff $w=w'$. Then, for this instance I, the algorithm's answer a does not achieve Hamming distance $n-c$. Thus, any "black-box" deterministic algorithm that achieves Hamming distance $n-c$ must take time $\Omega(n^{c-1})$ in the worst case.

6 Gaps between lower and upper bounds

For deterministic Hamming-approximation algorithms for $V_{\mathcal{U}}$, there is still a large gap between the lower bound $(n/2+O(\sqrt{n\log n}))$ and the upper bound $(n-O(1))$. As discussed above, the upper bound cannot be improved for "black-box" algorithms. As for the lower bound, improving it would also require a different proof technique, one that does more than use the supposed approximation algorithm $A_{\mathcal{U}}$ as a black box. (This is simply because (1) such a proof technique cannot distinguish between a deterministic algorithm and a randomized algorithm that fails with only exponentially small probability, and (2) a stronger lower bound *does not hold* for such randomized algorithms: guessing n random bits achieves Hamming distance $n/2+O(\sqrt{n\log n})$ with high probability.) Similarly, it seems likely that standard derandomization methods might yield a deterministic $\mathsf{P/POLY}$ algorithm for achieving Hamming distance $n/2+O(\sqrt{n\log n})$ for $\mathcal{U}$, in which case a stronger lower bound would have to distinguish polynomial-time algorithms from polynomial-time algorithms with polynomial advice.

Regarding the hardness of approximating *natural* verifiers, we know that there are $n/2$-approximation algorithms for natural verifiers for unweighted Vertex Cover, Clique, Independent Set, and similar problems. The gap for these problems is smaller, since the lower bound is $n/2-n^{\varepsilon}$. But the gap could still be reduced for these problems. For other problems the gap is larger, mainly because we do not yet have $n/2$-approximation algorithms. These include, for example, the weighted versions of the above problems, and 3-SAT. The following leading problem is still open, both for deterministic algorithms and for randomized algorithms that work with high probability:

> *Given a satisfiable 3-SAT formula, how hard is it to find an assignment to the variables that has Hamming distance at most $n/2$ to a satisfying assignment?*

Acknowledgments

Thanks to the anonymous referees for their careful reading and constructive suggestions, including Observation 3.1.

References

[1] LEONARD BERMAN AND JURIS HARTMANIS: On isomorphisms and density of NP and other complete sets. *SIAM J. Comput.*, 6(2):305–322, 1977. Preliminary version in STOC'76. [doi:10.1137/0206023] 688

[2] URIEL FEIGE, MICHAEL LANGBERG, AND KOBBI NISSIM: On the hardness of approximating NP witnesses. In *Proc. 3rd Internat. Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX'00)*, pp. 120–131. Springer, 2000. [doi:10.1007/3-540-44436-X_13] 686

[3] ANNA GÁL, SHAI HALEVI, RICHARD J. LIPTON, AND EREZ PETRANK: Computing from partial solutions. In *Proc. 14th IEEE Conf. on Computational Complexity (CCC'99)*, pp. 34–45. IEEE Comp. Soc. Press, 1999. [doi:10.1109/CCC.1999.766260] 687

[4] VENKATESAN GURUSWAMI: *List Decoding of Error-Correcting Codes (Winning Thesis of the 2002 ACM Doctoral Dissertation Competition)*. Volume 3282 of *Lecture Notes in Computer Science*. Springer, 2005. [doi:10.1007/b104335] 686

[5] VENKATESAN GURUSWAMI AND ATRI RUDRA: Soft decoding, dual BCH codes, and better list-decodable ε-biased codes. *IEEE Trans. Inform. Theory*, 57(2):705–717, 2011. Preliminary version in CCC'08. See also at ECCC. [doi:10.1109/TIT.2010.2095193] 686

[6] VENKATESAN GURUSWAMI AND MADHU SUDAN: List decoding algorithms for certain concatenated codes. In *Proc. 32nd STOC*, pp. 181–190. ACM Press, 2000. [doi:10.1145/335305.335327] 686

[7] SAMIR KHULLER: Algorithms Column: the Vertex Cover problem. *SIGACT News*, 33(2):31–33, 2002. [doi:10.1145/564585.564598] 695

[8] RAVI KUMAR AND D. SIVAKUMAR: Proofs, codes, and polynomial-time reducibilities. In *Proc. 14th IEEE Conf. on Computational Complexity (CCC'99)*, pp. 46–53. IEEE Comp. Soc. Press, 1999. [doi:10.1109/CCC.1999.766261] 686

[9] GEORGE L. NEMHAUSER AND LESLIE E. TROTTER, JR.: Vertex packings: Structural properties and algorithms. *Mathematical Programming*, 8(1):232–248, 1975. [doi:10.1007/BF01580444] 695

[10] DANIEL SHELDON AND NEAL E. YOUNG: Hamming approximation of NP witnesses. Unpublished manuscript, 2003. 686

[11] MICHAEL SIPSER: *Introduction to the Theory of Computation*. Thomson Course Technology, 2006. 698

AUTHORS

Daniel Sheldon
Assistant Professor
University of Massachusetts Amherst
sheldon@cs.umass.edu
http://people.cs.umass.edu/~sheldon

Neal E. Young
Professor
University of California Riverside
http://www.cs.ucr.edu/~neal

ABOUT THE AUTHORS

DANIEL SHELDON is a Five Colleges Assistant Professor in computer science at the University of Massachusetts and Mount Holyoke College. Prior to his appointment in 2012, he was a Postdoctoral Fellow in the School of EECS at Oregon State University, where he held a National Science Foundation (NSF) Fellowship in Bioinformatics. He received his Ph. D. from Cornell University in 2009 under the supervision of John Hopcroft; his thesis explored issues of manipulation in web-based reputation systems and developed new inference methods for probabilistic graphical models, inspired by the problem of modeling bird migration. His research interests include: algorithms for computational ecology and environmental science; machine learning; probabilistic modeling and inference; and optimization.

NEAL E. YOUNG is a Professor in computer science at the University of California, Riverside. In 1991 he completed a Ph. D. in Computer Science at Princeton University under Professor Robert Tarjan. From 1991 to 1994 he did postdocs at the University of Maryland, Princeton, Cornell, and AT&T Bell Laboratories. From 1995 to 1999 he was an assistant professor in the Computer Science department at Dartmouth College. From 1999 to 2003 he worked at Akamai Technologies (the world's largest Internet content-delivery network). He has been at the University of California since 2004. His research to date is on fast approximation algorithms for combinatorial optimization problems, including online problems, NP-hard problems, and linear programs. He is primarily interested in general techniques for the design and mathematical analysis of such algorithms.