Electronic Colloquium on Computational Complexity, Report No. 93 (2022)

More Verifier Efficient Interactive Protocols for Bounded Space

Joshua Cook*

June 27, 2022

Abstract

Let $\mathbf{TISP}[T,S]$, $\mathbf{BPTISP}[T,S]$, $\mathbf{NTISP}[T,S]$, and $\mathbf{CoNTISP}[T,S]$ be the set of languages recognized by deterministic, randomized, nondeterminsitic, and co-nondeterministic algorithms, respectively, running in time T and space S. Let $\mathbf{ITIME}[T_V]$ be the set of languages recognized by an interactive protocol where the verifier runs in time T_V. We prove:

$$\mathbf{TISP}[T,S] \cup \mathbf{BPTISP}[T,S] \subseteq \mathbf{ITIME}[\tilde{O}(\log(T)S+n)] \quad (1)$$

$$\mathbf{NTISP}[T,S] \cup \mathbf{CoNTISP}[T,S] \subseteq \mathbf{ITIME}[\tilde{O}(\log(T)^2 S+n)]. \quad (2)$$

The prior most verifier time efficient interactive protocol for **TISP** uses ideas from Goldwasser, Kalai and Rothblum [GKR15], which gives

$$\mathbf{NTISP}[T,S] \subseteq \mathbf{ITIME}[\tilde{O}(\log(T)S^2+n)].$$

1 Introduction

One of the most celebrated results of computer science is the proof that $\mathbf{IP} = \mathbf{PSPACE}$ [Sha92; Lun+90]. Any language computable in polynomial space can be verified in polynomial time by a verifier with access to randomness and an untrusted, computationally unbounded prover.

Interactive proofs were used to prove circuit lower bounds for $\mathbf{MA}/1$ [San07], and for **NQP** [MW18]. More verifier time efficient **PCP**s [MC22] improved the results of [San07]. Some pseudo random generators [CT22] use interactive proofs [GKR15] as an important component.

The previous best interactive proof for an algorithm running in time T and space S has a verifier running in time $\tilde{O}(\log(T)S^2)$. We improve this by making the quadratic dependence on S linear. If $T = \mathbf{poly}(S)$, our protocol improves the verifier time from $\tilde{O}(S^2)$ to $\tilde{O}(S)$. If $T = 2^{O(S)}$, our protocol improves the verifier time from $\tilde{O}(S^3)$ to $\tilde{O}(S^2)$.

*jac22855@utexas.edu. Department of Computer Science, UT Austin. This material is based upon work supported by the National Science Foundation under grant number 1705028.

ISSN 1433-8092

Our results prove a more direct, more efficient, and (in our opinion) simpler protocol than that given in [Sha92] using many of the same ideas from [Lun+90]. We then apply this protocol to the special cases of deterministic, randomized, and nondeterministic algorithms.

1.1 Results

Let $\mathbf{ITIME}[T_V, T_P]$ be the class of languages computed by an interactive protocol where the verifier runs in time T_V and the prover runs in time T_P. Similarly, let $\mathbf{ITIME}^1[T_V, T_P]$ be the same with perfect completeness. Let $\mathbf{TISP}[T, S]$ be the class of languages computable in simultaneous time T and space S. Our first result is:

Theorem 1 (Efficient Interactive Protocol For **TISP**)**.** *Let S and T be computable in time $\tilde{O}(\log(T)S + n)$. Then*

$$\mathbf{TISP}[T, S] \subseteq \mathbf{ITIME}^1[\tilde{O}(\log(T)S + n), 2^{O(S)}].$$

Our protocol has several other desirable properties. The verifier only needs space $\tilde{O}(S)$. This protocol is also public coin, non adaptive, and unambiguous (as described in [RRR16]). This protocol can also verify an $O(S)$ bit output, not just membership in a language.

We note that $L \in \mathbf{ITIME}[T_V, T_P]$ implies that $L \in \mathbf{SPACE}[T_V]$, since a prover can find an optimal prover strategy in a space efficient way. Thus our dependence on S is essentially optimal. It is still open whether one can remove the $\log(T)$ factor.

A corollary of Theorem 1 is that

$$\mathbf{ITIME}[T] \subseteq \mathbf{SPACE}[O(T)] \subseteq \mathbf{ITIME}[\tilde{O}(T^2)].$$

This gives a hierarchy theorem: $\mathbf{ITIME}[\tilde{O}(T^2)] \not\subset \mathbf{ITIME}[O(T)]$ by using the space hierarchy theorem. Not many hierarchies for semantic complexity classes are known. For instance, there isn't a time hierarchy theorem known for **BPP**. This improves the polynomial gaps in the interactive time hierarchy to only quadratic gaps.

Using Nisan's **PRG** for bounded space [Nis90], we extend Theorem 1 to get a similar result for randomized bounded space algorithms. Let $\mathbf{BPTISP}[T, S]$ be the class of languages computable in simultaneous randomized time T and space S.

Theorem 2 (Efficient Interactive Protocol For **BPTISP**)**.** *Let S and T be computable in time $\tilde{O}(\log(T)S + n)$. Then*

$$\mathbf{BPTISP}[T, S] \subseteq \mathbf{ITIME}[\tilde{O}(\log(T)S + n), 2^{O(S)}].$$

A conventional way to get interactive protocols for **BPSPACE** is to use Saks and Zhou's result [SZ99] to put $\mathbf{BPSPACE}[S] \subseteq \mathbf{SPACE}[O(S^{3/2})]$, then

use Theorem 1 to get $\mathbf{BPSPACE}[S] \subseteq \mathbf{ITIME}[\tilde{O}(S^3), 2^{O(S^{3/2})}]$. We show directly that $\mathbf{BPSPACE}[S] \subseteq \mathbf{ITIME}[\tilde{O}(S^2), 2^{O(S)}]$.

The protocol used for Theorem 1 internally counts the number of accepting paths in a nondeterministic algorithm, modulo a prime. By appropriately sampling a prime, we can get an efficient interactive protocol for **NTISP** and **CoNTISP**.

Theorem 3 (Efficient Interactive Verifier Protocol For **NTISP**)**.** *Let S and T be computable in time $\tilde{O}(\log(T)^2 S + n)$. Then*

$$\mathbf{NTISP}[T, S] \cup \mathbf{CoNTISP}[T, S] \subseteq \mathbf{ITIME}^1[\tilde{O}(\log(T)^2 S + n), 2^{O(S)}].$$

1.2 Proof Idea

1.2.1 Number Of Paths Mod Prime

We use a sum check from LFKN [Lun+90], similar to Shamir [Sha92], but without reducing to a quantified boolean formula first. We focus on computation graphs. For an algorithm A running in space S on input x, it's computation graph, G, has as vertices S bit states and as edges the state transitions for A on input x. That is, there is an edge from state s_0 to s_1 if and only if when A on input x is in state s_0, after one step, A can have state s_1.

Consider G's adjacency matrix, M. If A runs in T steps starting at state s_s, then A halts at state s_e if and only if $(M^T)_{s_s,s_e}$ is non zero. A sum check reduces a statement about M^2 to a statement about M in $O(S)$ field operations. By repeated squaring, we perform this reduction $O(\log(T))$ times to convert a statement about $(M^T)_{s_s,s_e}$ to one about M.

Since we consider nondeterministic algorithms, $(M^T)_{s_s,s_e}$ will contain the number of computation paths ending in s_e. Since we work in a finite field with characteristic p, entry $(M^T)_{s_s,s_e}$ will contain the number of computation paths mod p.

For a space S and time T non deterministic algorithm A with input x, and a field $\mathbb{F}$ with characteristic p, we define a function $\phi : \{0,1\}^S \times \{0,1\}^S \times [T] \to \mathbb{F}$ that takes two states, s_0 and s_1, and a time, t, and outputs how many computation paths of length t in A on input x start at state s_0 and end at state s_1, mod p. We define $\hat{\phi} : \mathbb{F}^S \times \mathbb{F}^S \times [T] \to \mathbb{F}$ to be the multilinear extension of ϕ in the two states. Then we get the recursive relationship that

$$\hat{\phi}(s_0, s_1, t_0 + t_1) = \sum_{s' \{0,1\}^S} \hat{\phi}(s_0, s', t_0) \cdot \hat{\phi}(s', s_1, t_1).$$

See the RHS is degree 1 in variables from s_0 and s_1, and is consistent with ϕ, so actually is $\hat{\phi}$. Note $\hat{\phi}(s_0, s', t_0) \cdot \hat{\phi}(s', s_1, t_1)$ only has degree 2 in variables from s', allowing us to perform a sum check efficiently.

Let $T = 2^t$. We recursively define $\hat{\phi}(s_0, s_1, T)$ with only $\log(T)$ steps like

$$\hat{\phi}_0(s_0, s_1) = \hat{\phi}(s_0, s_1, 1)$$
$$\hat{\phi}_i(s_0, s_1) = \hat{\phi}(s_0, s_1, 2^i) = \sum_{s' \{0,1\}^S} \hat{\phi}_{i-1}(s_0, s') \cdot \hat{\phi}_{i-1}(s', s_1).$$

Suppose that A starts at state s_s and ends at state s_e if it accepts. Now we want to know $\phi_t(s_s, s_e)$. We first use a sum check to reduce this to a claim that $v' = \hat{\phi}_{t-1}(s_s, s') \cdot \hat{\phi}_{t-1}(s', s_e)$ for some s' and v'. Now we need to convert this to a claim about a product of a univariate polynomial evaluated at few points. Define $\phi'_i : \mathbb{F} \to \mathbb{F}$ by

$$\phi'_i(x) = \hat{\phi}_i((1-x)s_s + xs', (1-x)s' + xs_e).$$

Then our new claim becomes $v' = \phi'_{t-1}(0) \cdot \phi'_{t-1}(1)$, which we can reduce to a claim that $v = \phi'_{t-1}(x)$ for some v and x in the standard way. Note $\phi'_{t-1}(x)$ is just $\hat{\phi}_{t-1}$ evaluated at some point. By performing this operation t times, our verifier reduces to a statement about $\hat{\phi}_0$, which the verifier can directly calculate.

1.2.2 Randomized and Non Deterministic Algorithms

To handle randomized algorithms, we use Nissan's Pseudo Random Generator (**PRG**) for bounded space computation. This **PRG** has seed length $O(\log(T)S)$ and can be calculated in time $\mathbf{poly}(S)$ and space $O(S)$. After choosing a seed, this **PRG** gives us a length $n + O(\log(T)S)$ input for a deterministic algorithm running in space $O(S)$ and time $\mathbf{poly}(T)$ which agrees with the randomized algorithm with high probability. Now we can run our deterministic protocol on this input.

To handle nondeterministic algorithms, we choose a random prime p. The protocol works if p does not divide the number of accepting computation paths. Let Q be the set of primes between $m = 100T$ and $2m = 200T$. If $w \leq 2^T$ is the number of accepting paths, then the number of prime factors of w in Q is at most $\frac{T}{\log(m)}$. By the prime number theorem, for large enough T, set Q should contain at least $\frac{0.5m}{\ln(m)}$ elements. If we can randomly sample one p from Q and $w \neq 0$, the probability $w \mod p = 0$ is at most $\frac{1}{10}$.

1.3 Related Work

This work builds on techniques used by Lund, Fortnow, Karloff and Nisan [Lun+90] to prove that $\#\mathbf{P} \in \mathbf{IP}$ and extended by Shamir [Sha92] to show that $\mathbf{PSPACE} = \mathbf{IP}$. Shamir proved:

Theorem 4 (Shamir's Protocol)**.** *Let S and T be time $\tilde{O}(\log(T)^2 S^2 + n)$ computable. Then*

$$NTISP[T, S] \subseteq \mathbf{ITIME}^1[\tilde{O}(\log(T)^2 S^2 + n), 2^{O(\log(T)S)}].$$

Shamir used Savitch's theorem [Sav70] to reduce space bounded computation to a totally quantified boolean formula, and then gave a sum check similar to [Lun+90] to verify it.

Similar techniques were also used by Babai, Fortnow and Lund [BFL90] to prove that **MIP** = **NEXP**. This line of work is foundational to many **PCP** results [AS98; Aro+98].

In a very influential paper, Goldwasser, Kalai and Rothblum gave doubly efficient interactive proofs for depth bounded computation [GKR15]. Doubly efficient proofs are proofs where the prover runs in time polynomial in the algorithm it wishes to prove.

Theorem 5 (GKR For Depth)**.** *Let L be a language computed by a family of $O(\log(w))$-space uniform boolean circuits of width w and depth d where w and d are computable in time $(n+d)\mathbf{polylog}(w)$. Then*

$$L \in \mathbf{ITIME}^1[(n+d)\mathbf{polylog}(w), \mathbf{poly}(wd)].$$

Similar to what we do in this paper, a space S and a time T non-deterministic algorithm, A, can be converted to a width $2^{O(S)}$ and depth $O(\log(T)S)$ circuit, C, using repeated squaring on the adjacency matrix of A's computation graph. Using Theorem 5 with our circuit above, we get a protocol for bounded space. The circuit is very uniform, so the $\mathbf{polylog}(w) = \mathbf{poly}(S)$ term can be made $\tilde{O}(S)$.

Theorem 6 (GKR for **NSPACE**)**.** *Let S and T be time $\tilde{O}(\log(T)S^2 + n)$ computable. Then*

$$\mathbf{NTISP}[T, S] \subseteq \mathbf{ITIME}^1[\tilde{O}(\log(T)S^2 + n), 2^{O(S)}].$$

The GKR protocol, as well as ours in this paper, only have polynomial time provers when $T = 2^{\Omega(S)}$. Reingold, Rothblum and Rothblum [RRR16] gave a doubly efficient protocol for any time T with constantly many rounds of communication, but is only efficient for the verifier when T is polynomial.

Theorem 7 (RRR Protocol)**.** *For any constant $\delta > 0$, and integers S and T computable in time $T^{O(\delta)}S^2$, and $T = \Omega(n)$ we have*

$$\mathbf{TISP}[T, S] \subseteq \mathbf{ITIME}^1[O(n\mathbf{polylog}(T) + T^{O(\delta)}S^2), T^{1+O(\delta)}\mathbf{poly}(S)].$$

Further the prover only sends $\left(\frac{1}{\delta}\right)^{O(1/\delta)}$ messages to the verifier.

The specific, S^2 power in the verifier time comes from a note by Goldreich [Gol18], confirmed by the authors of [RRR16]. Our result gives a more efficient verifier ($\log(T)S$ vs $T^\delta S^2$), but a less efficient prover ($2^{O(S)}$ vs $\mathbf{poly}(T)$). We note the result in the [RRR16] paper allows sub constant δ, but is complex and can not give a verifier with time $\mathbf{poly}(\log(T)S)$.

There has been work on other notions of verifier efficiency in interactive protocols. Goldwasser, Gutfreund, Healy, Kaufman and Rothblum [Gol+07]

studied the computation depth required by verifiers. They showed that for any k round interactive proof, there is a $k + O(1)$ round interactive proof where the computation of the verifier during each round is in $\mathbf{NC}^0$. But the total time to evaluate the new verifier (or the total verifier circuit size) is greater than the verifier time of the original protocol.

2 Preliminaries

We use RAM algorithms as our model of computation, although these results should still hold for many other common models of computation, like multi-tape Turing machines. We may assume that on accepting or rejecting, our machines instantly clear their states to some canonical accept or reject state. We also assume that $S = \Omega(\log(n))$, otherwise our algorithm can't read it's entire input.

We use $\tilde{O}$ to suppress poly logarithmic factors.

Definition 8 ($\tilde{O}$)**.** *For $f, g : \mathbb{N} \to \mathbb{N}$, we say $f(n) = \tilde{O}(g(n))$ if and only if for some constant k, $f(n) = O(g(n)\log(g(n))^k)$.*

We focus on languages with simultaneous time and space constraints.

Definition 9 (**TISP**)**.** *For functions $T, S : \mathbb{N} \to \mathbb{N}$, we say language L is in $\mathbf{TISP}[T, S]$ if there is an algorithm, A, running in time T and space S that recognizes L.*

Since we are working with space bounded computation, we define access to randomness or nondeterminism through a read once input.

Definition 10 (Read Once Input)**.** *We say algorithm A uses read once input W if there is a special instruction in A that, for all i, on the ith time being called returns the ith symbol of W.*

If A uses an input x and a read once input W, we define $A(x, W)$ as the output of A when run with input x and read once input W. Note: the read once input W is not considered part of the input when discussing nondeterministic or randomized algorithms and loading any input from W on a special instruction is considered a valid transition.

Note if A runs in time T, we can upper bound the size of W as $|W| \leq T$.

Now we define **BPTISP** and **NTISP**.

Definition 11 (**BPTISP**)**.** *Let L be a language, A be an algorithm with read once input, and $c > 0$. If for all x we have $\Pr[A(x, U) = 1_{x \in L}] \geq \frac{2}{3}$, where U is the uniform distribution, then we say A is a randomized algorithm for L.*

If for $T, S : \mathbb{N} \to \mathbb{N}$ algorithm A runs in time T and space S, then $L \in \mathbf{BPTISP}[T, S]$.

Definition 12 (**NTISP** and **CoNTISP**)**.** *For language L and RAM algorithm A, if*

$$\forall x \in L : \exists W : A(x, W) = 1$$
$$\forall x \notin L : \forall W : A(x, W) = 0$$

then we say A is a nondeterministic algorithm for L.

*If for $T, S : \mathbb{N} \to \mathbb{N}$ algorithm A runs in time T and space S, then $L \in$ **NTISP**$[T, S]$. We say $L \in$ **CoNTISP**$[T, S]$ if the complement of L is in* **NTISP**$[T, S]$.

In an interactive protocol for some function f, we want our verifier to output $f(x)$ with high probability if the prover is honest, and output the wrong answer with low probability regardless of the prover. Our verifier can either reject, or output some value. This is a more general formulation of interactive proofs than is standard.

Let us formally define the interaction of a protocol.

Definition 13 (Interaction Between Verifier and Prover (Int))**.** *Let Σ be a alphabet and $\perp$ be a symbol not in Σ. Let V be a RAM machine with access to randomness, that can make oracle queries, and V outputs one of $\Sigma \cup \{\perp\}$. Let P' be any function, and x be an input.*

Now we define the interaction of V and P' on input x. For all i, define y_i to be V's ith oracle query given its first $i-1$ queries were answered with $z_1, \ldots, z_{i-1}$ and define $z_i = P(x, y_1, \ldots, y_i)$.

Define the output of V when interacting with P', $Int(V, P, x)$, as the output of V on input x when it's oracle queries are answered by $z_1, z_2, \ldots$.

Now we define interactive time.

Definition 14 (Interactive Time (**ITIME**))**.** *Let Σ be an alphabet. If for function $f : \{0,1\}^* \to \Sigma$, soundness $s \in [0,1]$, completeness $c \in [0,1]$, verifier V and prover P we have*

Completeness: $\Pr[Int(V, P, x) = f(x)] \geq c$, *and*

Soundness: *for any function P' we have* $\Pr[Int(V, P', x) \notin \{f(x), \perp\}] \leq s$,

then we say V and P are an interactive protocol for f with soundness s and completeness c.

If in addition L is a language with $f(x) = 1_{x \in L}$, verifier V runs in time T_V, soundness $s < \frac{1}{3}$, and completeness $c > \frac{2}{3}$, then

$$L \in \textbf{ITIME}[T_V].$$

If P is also computable by an algorithm running in time T_P, we say

$$L \in \textbf{ITIME}[T_V, T_P].$$

Finally, if completeness $c = 1$, then we say the protocol has perfect completeness and

$$L \in \textbf{ITIME}^1[T_V, T_P].$$

We assume the reader is familiar with low degree polynomials. For any function $f : \{0,1\}^n \to \mathbb{F}$, its multilinear extension is the unique degree 1 polynomial $\hat{f} : \mathbb{F}^n \to \mathbb{F}$ such that for any $x \in \{0,1\}^n$, we have $f(x) = \hat{f}(x)$.

Our interactive protocol crucially uses that for any RAM algorithm A on any input x, if ϕ is the function that checks if A changes from one state to another in one step, then the multilinear extension of ϕ can be computed efficiently.

Lemma 15 (Algorithm Arithmetization)**.** *For any nondeterministic RAM algorithm A running in space S and time T on length n inputs, let $\phi_0 : \{0,1\}^n \times \{0,1\}^S \times \{0,1\}^S \to \{0,1\}$ be the function that takes an input x and two states, s_0 and s_1, and outputs whether A running on input x starting with state s_0 has s_1 as a valid transition.*

Then for any finite field $\mathbb{F}$ the multilinear polynomial $\hat{\phi_0} : \{0,1\}^n \times \mathbb{F}^S \times \mathbb{F}^S \to \mathbb{F}$ consistent on boolean inputs with ϕ_0 that can be computed in $\tilde{O}(\log(|\mathbb{F}|)(n+S))$ time.

Proof. (Sketch) Sum over all potential locations of the instruction pointer in the program of the multilinear polynomial identifying that unique instruction being the current instruction, times the multilinear polynomial of that instruction being performed. Since A is constant, there are only constantly many instructions we could be on.

For load and store, just sum over the locations in memory of the multilinear polynomial of that being the address to load or store from, times the equality polynomial of the register being changed with that value in memory. There are only $S + n$ locations in memory plus the input, so this only takes time $\tilde{O}(\log(|\mathbb{F}|)(n + S))$.

For standard register register operations like moves, addition, bit shifts, conditionals, these only act on $O(\log(S + n))$ bits and are generally straight forward to find efficient enough ways to calculate their multilinear extensions. □

For primality, we use the Rabin-Miller [Mil75; Rab80] primality test. It is well known using the fast fourier transform for multiplication we can get the following result:

Theorem 16 (Miller-Rabin Primality Test)**.** *There is a randomized algorithm, A, that given an n bit number a and $\epsilon > 0$ runs in time $\tilde{O}(\log(\frac{1}{\epsilon})n^2)$ such that if a is prime, A outputs 1, and if a is not prime, A outputs 1 with probability at most ϵ.*

Then by choosing $n = O(\log(m))$ random numbers between m and $2m$, by the prime number theorem, we select a prime number with high probability.

Theorem 17 (Miller-Rabin Probably Prime Generation)**.** *There is a randomized algorithm, A, that when given a number m (with $n = \log(m)$) and $\epsilon > 0$, algorithm A runs in time $\tilde{O}(\mathbf{polylog}(\frac{1}{\epsilon})n^3)$ and with probability $1 - \epsilon$ outputs a uniform prime between m and $2m$.*

We use Nissan's **PRG** for space bounded computation [Nis90]. First, we define a **PRG**.

Definition 18 (Pseudo Random Generator (**PRG**)). *For integers n and s, we say that a function $G : \{0,1\}^s \to \{0,1\}^n$, is a Pseudo Random Generator (**PRG**) with seed length s that ϵ fools function $F : \{0,1\}^n \to \{0.1\}$ if*

$$|\mathbb{E}[F(G(U_s)) - F(U_n)]| \leq \epsilon$$

where U_s and U_n are the uniform distribution over s and n bits respectively.

Specifically, we say G fools randomized algorithm A if for all x we have

$$|\mathbb{E}[A(x, G(U_s)) - A(x, U_n)]| \leq \epsilon.$$

Then Nissan gives an efficient **PRG** which fools algorithms that use small space.

Theorem 19 (Nissan's **PRG**). *For any space S, time T and error $\epsilon > 0$, there exists a **PRG** with seed length $O(S \log(T/\epsilon))$ computed by an algorithm running in time $\mathbf{poly}(S)$ and space $O(S)$ that ϵ fools randomized algorithms running in time T and space S.*

3 Efficient IP for TISP

We first give a protocol that outputs the number of accepting paths in a non deterministic algorithm, mod a prime. We start with a protocol to reduce a statement about a matrix squared to a statement about the matrix itself. Applying this many times gives our protocol.

Lemma 20 (Matrix Squared to Matrix Protocol). *Let S be an integer and p be a prime. Suppose $\phi_0 : \{0,1\}^S \times \{0,1\}^S \to \mathbb{F}_p$ encodes a $2^S \times 2^S$ matrix, M, containing values in $\mathbb{F}_p$ given by $M_{i,j} = \phi(i,j)$ where i and j are encoded in binary. Similarly, let $\phi_1 : \{0,1\}^S \times \{0,1\}^S \to \mathbb{F}_p$ encode M^2 so that for any $s_s, s_e \in \{0,1\}^S$*

$$\phi_1(s_s, s_e) = \sum_{s' \in \{0,1\}^S} \phi_0(s_s, s')\phi_0(s', s_e).$$

Let $\mathbb{F}$ be a field with characteristic p and $|\mathbb{F}| > \max\{3, 2S+1\}$. Let $\hat{\phi}_0 : \mathbb{F}^S \times \mathbb{F}^S \to \mathbb{F}$ be a multilinear extension of ϕ_0, and $\hat{\phi}_1 : \mathbb{F}^S \times \mathbb{F}^S \to \mathbb{F}$ be a multilinear extension of ϕ_1.

There is a interactive protocol with verifier V and prover P such that on input $s_s, s_e \in \mathbb{F}^S$ and $v \in \mathbb{F}$ behaves in the following way:

Completeness: *If $\hat{\phi}_1(s_s, s_e) = v$, then when V interacts with P, verifier V outputs some $s'_s, s'_e \in \mathbb{F}^S$ and $v' \in \mathbb{F}$ such that $\hat{\phi}_0(s'_s, s'_e) = v'$.*

Soundness: *If $\hat{\phi}_1(s_s, s_e) \neq v$, then for any prover P', when V interacts with P', with probability at most $\frac{4S}{|\mathbb{F}|}$ will V output $s'_s, s'_e \in \mathbb{F}^S$ and $v' \in \mathbb{F}$ such that $\hat{\phi}_0(s'_s, s'_e) = v'$.*

Verifier Time: *V runs in time $\tilde{O}(\log(|\mathbb{F}|))O(S)$.*

Verifier Space: *V runs in space $O(\log(|\mathbb{F}|)S)$.*

Prover Time: *P runs in time $\tilde{O}(\log(|\mathbb{F}|))2^S$ when given oracle access to $\hat{\phi}_0$.*

Proof. We start by explaining the protocol. The verifier starts with the claim that $\hat{\phi}_1(s_s, s_e) = v$. We claim that

$$\hat{\phi}_1(s_s, s_e) = \sum_{s' \in \{0,1\}^S} \hat{\phi}_0(s_s, s')\hat{\phi}_0(s', s_e).$$

See that the right hand side of the above equation is linear in every variable, thus is multilinear. And when restricted to boolean inputs, are equal. Since the multilinear extension is unique, the right hand side is the multilinear extension of ϕ_1, thus is $\hat{\phi}_1$.

Now we define a sequence of low degree polynomials between $\hat{\phi}_1$ and $\hat{\phi}_0$. Define $\psi_0 : \mathbb{F}^S \times \mathbb{F}^S \times \mathbb{F}^S \to \mathbb{F}$ by $\psi_0(s_s, s_e, s') = \hat{\phi}_0(s_s, s')\hat{\phi}_0(s', s_e)$. Then for each $i \in [S]$, define $\psi_i : \mathbb{F}^S \times \mathbb{F}^S \times \mathbb{F}^{S-i} \to \mathbb{F}$ by

$$\begin{aligned}\psi_i(s_s, s_e, s_l) &= \sum_{s_r \in \{0,1\}^i} \hat{\phi}_0(s_s, (s_l, s_r))\hat{\phi}_0((s_l, s_r), s_e) \\ &= \sum_{j \in \{0,1\}} \psi_{i-1}(s_s, s_e, (s_l, j)).\end{aligned}$$

See that $\psi_S = \hat{\phi}_1$. Since ψ_0 is degree at most 2 in every variable, so is every ψ_i.

The idea is for every $i \in [S]$ to reduce a statement about ψ_i to a statement about ψ_{i-1}. Then to reduce a statement about ψ_0 to one about $\hat{\phi}_0$. Since $|\mathbb{F}| > 3$, let $a \in \mathbb{F}^3$ be any arbitrary, distinct 3 elements of $\mathbb{F}$. Then the verifier protocol is:

1. Set $s'_S = ()$, the empty tuple.
2. Set $v_S = v$.
3. For i from S to 1:
   (a) For $k \in [3]$, ask the prover for $\psi_{i-1}(s_s, s_e, (s', a_k))$. Let $w \in \mathbb{F}^3$ be the prover's response. Let $g_i : \mathbb{F} \to \mathbb{F}$ be the degree 2 polynomial so that for each $k \in [3]$, we have $g_i(a_k) = w_k$.
   (b) If $g_i(0) + g_i(1) \neq v_i$, then reject.
   (c) Choose some $s_{i-1} \in \mathbb{F}$ and send it to the prover.
   (d) Set $s'_{i-1} = (s'_i, s_{i-1})$.
   (e) Set $v_{i-1} = g_i(s_{i-1})$.
4. Define $\psi'_0 : \mathbb{F} \to \mathbb{F}$ so that $\psi_0(s_s, s_e, s'_0) = \psi'_0(0)\psi'_0(1)$. Specifically, let

$$\psi'_0(x) = \hat{\phi}_0((1-x)s_s + xs'_0, (1-x)s'_0 + xs_e).$$

5. Let $b \in \mathbb{F}^{2S+1}$ be any arbitrary, distinct elements of $\mathbb{F}$. Then for $k \in [2S+1]$, ask the prover for $\psi'_0(b_k)$. Let $w \in \mathbb{F}^S$ be the prover's response. Let $g_0 : \mathbb{F} \to \mathbb{F}$ be the degree $2S$ polynomial so that for each $k \in [2S+1]$, we have $g_0(b_k) = w_k$.

6. If $v_0 \neq g_0(0) \cdot g_0(1)$, then reject.

7. Choose $x \in \mathbb{F}$ and send it to the prover.

8. Output $s'_s = (1-x)s_s + xs'_0$ and $s'_e = (1-x)s'_0 + xs_e$ with claimed value $v' = g_0(x)$.

The honest prover responds with the value the verifier requests.

Completeness: Suppose $\hat{\phi}_1(s_s, s_e) = v$. Then for $i \in [S]$, define $\psi'_i : \mathbb{F} \to \mathbb{F}$ to be

$$\begin{aligned}\psi'_i(x) =& \psi_{i-1}(s_s, s_e, (s'_i, x)) \\ =& \sum_{s_r \in \{0,1\}^{i-1}} \hat{\phi}_0(s_s, (s'_i, x, s_r))\hat{\phi}_0((s'_i, x, s_r), s_e).\end{aligned}$$

See that ψ'_i is a degree 2 polynomial since $\hat{\phi}_0$ is multilinear. Then for each $k \in [3]$, $\psi'_i(a_k) = g_i(a_k)$. Since g_i and ψ'_i are degree 2 and agree in three places, they are equal. By assumption,

$$\begin{aligned}v_S =& v \\ =& \hat{\phi}_1(s_s, s_e) \\ =& \psi_S(s_s, s_e) \\ =& \psi_S(s_s, s_e, s'_S).\end{aligned}$$

Now by induction we get for all i from S to 0:

$$\begin{aligned}g_i(0) + g_i(1) =& \psi'_i(0) + \psi'_i(1) \\ =& \psi_{i-1}(s_s, s_e, (s'_i, 0)) + \psi_{i-1}(s_s, s_e, (s'_i, 1)) \\ =& \psi_i(s_s, s_e, s'_i) \\ =& v_i\end{aligned}$$

and

$$\begin{aligned}v_{i-1} =& \psi'_i(s_{i-1}) \\ =& \psi_{i-1}(s_s, s_e, s'_{i-1}),\end{aligned}$$

so we don't reject on the rest of the loop. Further: $v_0 = \psi_0(s_s, s_e, s'_0)$.

See that ψ'_0 is degree $2S$, since $\hat{\phi}_0$ is degree $2S$ and ψ'_0 is just a degree one function composed with ψ'_0. For an honest prover, since ψ'_0 agrees

with g_0 on $2S+1$ places, and both are degree $2S$, they are equal. Since $\psi_0(s_s, s_e, s'_0) = \psi'_0(0)\psi'_0(1)$, we have

$$\begin{aligned} v_0 =& \psi_0(s_s, s_e, s'_0) \\ =& \psi'_0(0)\psi'_0(1) \\ =& g_0(0)g_0(1). \end{aligned}$$

So the verifier does not reject. Finally

$$\begin{aligned} v' =& g_0(x) \\ =& \psi'_0(x) \\ =& \hat{\phi}_0((1-x)s_s + xs'_0, (1-x)s'_0 + xs_e) \\ =& \hat{\phi}_0(s'_s, s'_e) \end{aligned}$$

as we wanted.

Soundness: Suppose $\hat{\phi}_2(s_s, s_e) \neq v$.

For any $i \in [S]$, We claim if $v_i \neq \psi_i(s_s, s_e, s'_i)$, then with probability at most $\frac{2}{|\mathbb{F}|}$ will the verifier not reject and $v_{i-1} = \psi_{i-1}(s_s, s_e, s'_{i-1})$. To show this, suppose $v_i \neq \psi_i(s_s, s_e, s'_i)$. See that if $g_i = \psi'_i$, and the verifier doesn't reject in step i, then $v_i = \psi'_i(0) + \psi'_i(1) = \psi_i(s_s, s_e, s'_i)$, a contradiction. So consider when $g_i \neq \psi'_i$. Then since both are degree 2, the probability they agree at s_i is at most $\frac{2}{|\mathbb{F}|}$ by the Schwartz-Zippel lemma. If $g_i(s_i) \neq \psi'_i(s_i)$, then $v_{i-1} \neq \psi_{i-1}(s_s, s_e, s'_{i-1})$.

Then by a union bound, the probability the verifier does not reject and $v_0 = \psi_0(s_s, s_e, s'_0)$ is at most $\frac{2S}{|\mathbb{F}|}$. So suppose $v_0 \neq \psi_0(s_s, s_e, s'_0)$. See that ψ'_0 is degree at most $2S$. Then if $\psi'_0 = g_0$, we will reject, since $\psi'_0(0)\psi'_0(1) = \psi_0(s_s, s_e, s'_0) \neq v_0$. Otherwise, the probability that we choose an x such that $\psi'_0(x) = g_0(x)$ is at most $\frac{2S}{|\mathbb{F}|}$. If $g_0(x) \neq \psi'_0(x)$, then $v' \neq \phi(s'_s, s'_e)$.

So by a union bound, the probability that the verifier does not reject and $v' = \phi(s'_s, s'_e)$ is at most $\frac{4S}{|\mathbb{F}|}$.

Verifier Time: See that each iteration on the inside of the first loop only takes a constant number of field operations. Since there are S iterations of the loop, the first loop takes time $O(S)$ field operations. Then the verifier must receive $O(S)$ values from the prover, and calculate a polynomial going through all these points. This can also be done in $O(S)$ field operations. Finally, the verifier uses $O(S)$ field operations to calculate s'_s and s'_e. Since a field operation only takes time $\tilde{O}(\log(|\mathbb{F}|))$, the verifier only takes time $\tilde{O}(\log(|\mathbb{F}|))O(S)$ overall.

Verifier Space: The algorithm only requires storing $O(S)$ field elements, and each field element only has size $O(\log(|\mathbb{F}|))$. So the verifier only uses space $O(\log(|\mathbb{F}|)S)$.

Prover Time: See each ψ_0 can be computed in one field operation and two oracle queries to $\hat{\phi}_0$. Every other ψ_i is just a sum of up to 2^S terms of ψ_0, so can be calculated in $O(2^S)$ field operations. Function ψ_0' is a single valuation of $\hat{\phi}_0$, and its argument only takes $O(S)$ field operations to compute. So any prover query only takes time $\tilde{O}(\log(|\mathbb{F}|))2^S$ to compute, given oracle access to $\hat{\phi}_0$.

□

Now we give our protocol for counting the number of accepting paths in a computation.

Theorem 21 (Number of Accepting Paths Mod P)**.** *Suppose A is a nondeterministic algorithm running in space S, and time T where S and T are time $O(\log(T)S)$ computable. Then there is an interactive protocol with verifier V and prover P such that, when given input x, state s_e, error bound $\epsilon > 0$ and prime p behaves the following:*

Completeness: *When V interacts with prover P on input x, V outputs the number of computation paths of A on input x ending at s_e, mod p.*

Soundness: *Given any prover P', when V interacts with prover P', V outputs the incorrect number of computation paths of A on input x ending at s_e, mod p, with probability at most ϵ.*

Verifier Time: *V runs in time $\tilde{O}\left(\log(pS/\epsilon)\right) O(\log(T)S)$.*

Verifier Space: *V runs in space $O\left(\log(pS/\epsilon)S\right)$.*

Prover Time: *P runs in time* $\mathbf{polylog}(pS/\epsilon)2^{O(S)}$.

Proof. We start by outlining how to convert this number of computation paths to a matrix problem, then we show how to apply Lemma 20 to solve that.

Take T to be a power of two so that $T = 2^t$. If T is not a power of 2, we can just take T to be the smallest power of two greater than the original T. Let k be the smallest integer so that $p^k > \frac{4\log(T)S}{\epsilon}$ and $p^k \geq 5$. Let $q = p^k$ and $\mathbb{F}$ be the field with q elements.

Let $\phi_0 : \{0,1\}^S \times \{0,1\}^S \to \mathbb{F}$ be the function that takes two states, s_0 and s_1, and outputs 1 if A on input x starting at state s_0 can be s_1 after one step, and 0 otherwise. That is, if M is the adjacency matrix for the computation graph of A on input x, then $M_{i,j} = \phi_0(i,j)$ if i and j are memory states.

Similarly, for $i \in [t]$, we define $\phi_i : \{0,1\}^S \times \{0,1\}^S \to \mathbb{F}$ so that $\phi_i(s_s, s_e) = (M^{2^i})_{s_s,s_e}$. Since all the elements of M are 0 or 1, they are in $\mathbb{F}_p$. Since $\mathbb{F}_p$ is closed under addition and multiplication, ϕ_i also only contains elmenents in $\mathbb{F}_p$.

Observe that $M^{2^i}_{s_s,s_e}$ is just the number of computation paths of length 2^i from s_s to s_e, mod p, since M is an adjacency matrix with entries in $\mathbb{F}_p$. Thus our verifier just needs to output $\phi_t(s_s, s_e)$. Next we show how to apply Lemma 20.

For $i \in [0,t]$, define $\hat{\phi}_i : \mathbb{F}^S \times \mathbb{F}^S \to \mathbb{F}$ to be the multilinear extension of ϕ_i. By Lemma 15, the multilinear extension of ϕ_0, function $\hat{\phi}_0 : \mathbb{F}^S \times \mathbb{F}^S \to \mathbb{F}$, can be computed in time $\tilde{O}(\log(|\mathbb{F}|)(n+S))$. For each $i \in [t]$, functions $\hat{\phi}_i$ and $\hat{\phi}_{i-1}$ satisfy the assumptions of Lemma 20. So we apply it t times then the verifier checks $\hat{\phi}_0$ itself.

Now we more formally describe the protocol. Let s_s be the starting state of A. The verifier starts by asking the prover for the number of computation paths from s_s to s_e, and the prover responds with v_t. The honest prover responds with $v_t = \hat{\phi}_t(s_s, s_e)$.

Let $s_s^t = s_s$ and $s_e^t = s_e$. Then for any $i \in [t]$, see that $\hat{\phi}_i$ and $\hat{\phi}_{i-1}$ satisfy the requirements of Lemma 20. So we use it to reduce the claim that $v_i = \hat{\phi}_i(s_s^i, s_e^i)$ to the claim that $v_{i-1} = \hat{\phi}_{i-1}(s_s^{i-1}, s_e^{i-1})$.

Finally, since the verifier can calculate $\hat{\phi}_0$ directly, it performs the final check whether $v_0 = \hat{\phi}_0(s_s^0, s_e^0)$. If every step succeeds, the verifier outputs v_t. Otherwise, it rejects.

Completeness: For an honest prover, indeed $v_t = \hat{\phi}_t(s_s, s_e)$. By induction and completeness of Lemma 20, for every $i \in [0,t]$ we have $v_i = \hat{\phi}_i(s_s^i, s_e^i)$. Thus $v_0 = \hat{\phi}_0(s_s^0, s_e^0)$. Thus the verifier check whether $v_0 = \hat{\phi}_0(s_s^0, s_e^0)$ succeeds, and the verifier outputs v_t.

Soundness: If $v_t = \hat{\phi}_t(s_s, s_e)$, the verifier either outputs v_t or rejects, either satisfies our assumption. So suppose $v_t \neq \hat{\phi}_t(s_s, s_e)$. Then for any $i \in [t]$, by the soundness of Lemma 20, if the verifier hasn't rejected and $v_i \neq \hat{\phi}_i(s_s^i, s_e^i)$, the probability the verifier does not reject and $v_{t-1} = \hat{\phi}_{t-1}(s_s^{t-1}, s_e^{t-1})$ is at most $\frac{4S}{|\mathbb{F}|}$. By a union bound, the probability that the verifier does not reject and for any i we have $v_i = \hat{\phi}_i(s_s^i, s_e^i)$ is at most

$$\frac{4S\log(T)}{|\mathbb{F}|} \leq \epsilon.$$

In particular, the probability the verifier does not reject and $v_0 = \hat{\phi}_0(s_s^0, s_e^0)$ is at most ϵ. See that if $v_0 \neq \hat{\phi}_0(s_s^0, s_e^0)$, then the verifier rejects. So the probability the verifier does not reject is at most ϵ.

Verifier Time: This verifier takes time $\tilde{O}(\log(|\mathbb{F}|))S\log(T)$ to run Lemma 20 t times plus $\tilde{O}(\log(|\mathbb{F}|))O(S)$ to calculate $\hat{\phi}_0(s_s^0, s_e^0)$ (using Lemma 15), so in total takes time

$$\tilde{O}(\log(|\mathbb{F}|))S\log(T) = \tilde{O}(\log(pS/\epsilon))O(\log(T)S).$$

Verifier Space: Between subsequent applications of Lemma 20, the verifier only needs to store i, s_s^i, s_e^i, and v_i, which only takes space $O(\log(|\mathbb{F}|)S)$. Each call to Lemma 20 also only needs space $O(\log(|\mathbb{F}|)S)$. Finally, the verifier only needs space $O(\log(|\mathbb{F}|)S)$ to calculate $\hat{\phi}_0(s_s^0, s_e^0)$. So the overall verifier only requires space

$$O(\log(|\mathbb{F}|)S) = O(\log(pS/\epsilon)\log(T)S).$$

Prover Time: First, the prover calculates M^{2^i} in time $\tilde{O}(\log(|\mathbb{F}|))S2^{3S}$ for every $i \in [t]$. This is equivalent to calculating each input to each ϕ_i. Then we can calculate the multilinear extension of any individual ϕ_i, which is $\hat{\phi}_i$, in time $\tilde{O}(\log(|\mathbb{F}|))2^{2S}$. Finally, given these, the prover in Lemma 20 only takes time $\tilde{O}(\log(|\mathbb{F}|))2^S$.

By just checking the transitions of algorithm A for each of the 2^{2S} entries, matrix $M = M^{2^0}$ can be calculated in time $O(S2^{2S}) = O(2^{3S})$. Matrix multiplication can be performed with $O(2^{3S})$ field operations. So by induction, for each i, matrix M^{2^i} can be calculated with $(1+i)O(2^{3S})$ field operations. Thus in $O(S2^{3S})$ field operations, or time $\tilde{O}(\log(|\mathbb{F}|))S2^{3S}$, for each $i \in [0, t]$, matrix M^{2^i} can be calculated.

Now for any $s_s, s_e \in \mathbb{F}^S$, our prover can calculate $\hat{\phi}_i(s_s, s_e)$ in $O(2^{2S})$ field operations by computing the multilinear extension of M^{2^i}. This takes time $\tilde{O}(\log(|\mathbb{F}|))2^{2S}$.

On the first query to get v_t, the prover just returns the already computed $\phi_t(s_s, s_e) = (M^T)_{s_s, s_e}$. To answer the prover queries during the ith call to Lemma 20 just takes the time of that protocol, times the time to compute $\hat{\phi}_i$. This is

$$\tilde{O}(\log(|\mathbb{F}|))2^{2S}\tilde{O}(\log(|\mathbb{F}|))2^{2S} = \mathbf{poly}\left(\log(pS/\epsilon)^2\right)2^{3S}.$$

Thus the prover runs in time $\mathbf{poly}\left(\log(pS/\epsilon)^2\right)2^{O(S)}$

□

As an immediate corollary, since deterministic algorithms always have either one accepting computation path or zero, by using $p = 2$, and setting s_e to be some canonical end state, we have Theorem 1. We can extend this into multi bit outputs by storing the output in the final end state and asking the prover for the end state first.

4 Efficient IP for BPTISP

Our protocol for randomized algorithms first uses a **PRG** for bounded space to convert our randomized algorithm into a deterministic algorithm, then applies our deterministic protocol. Note our **PRG** uses $O(\log(T)S)$ random bits, so these can't be stored in the algorithms state without incurring an extra $\log(T)$ factor in the verifier run time. This is fine since our **PRG** only uses $O(S)$ bits of working space and our protocol works for small space.

Theorem 2 (Efficient Interactive Protocol For **BPTISP**)**.** *Let S and T be computable in time $\tilde{O}(\log(T)S + n)$. Then*

$$\mathbf{BPTISP}[T, S] \subseteq \mathbf{ITIME}[\tilde{O}(\log(T)S + n), 2^{O(S)}].$$

Proof. Suppose $L \in \mathbf{BPTISP}[T, S]$ is computed by randomized algorithm A running in space S and time T. For x of length n, we want to verify if $x \in L$ or not. We will construct a new input, x', of length $n + O(S \log(T))$ decided by deterministic algorithm A' running in time $\mathbf{poly}(T)$ and space $O(S)$ such that if $x \in L$, then with high probability $A'(x') = 1$, and if $x \notin L$, then with high probability $A'(x') = 0$. Then we use Theorem 1 on A' and x' to verify whether $A'(x') = 1$, which with high probability is equivalent to whether $x \in L$.

As a technical detail, to maintain soundness and completeness, we will first need to amplify A to get a new protocol, A^*, by repeating it a constant number of times and taking the majority output. Similarly, we repeat the interactive protocol.

Since A is a randomized algorithm, $\Pr[M(x, U) = 1_{x \in L}] \geq \frac{2}{3}$. Let A^* be the algorithm which runs A three times and outputs the majority. Algorithm A^* runs in time $O(T)$, uses space $S + O(1)$, and $\Pr[A'(x, U) = 1_{x \in L}] \geq \frac{20}{27}$.

Recall that Nissan's **PRG** (Theorem 19) gives a function G with seed length $l = O(\log(T)S)$ computable in space $O(S)$ and time $\mathbf{poly}(S)$ that $\frac{1}{27}$ fools A^*. That is, $|\mathbb{E}[A^*(x, G(U)) - A^*(x, U)]| < \frac{1}{27}$. Let $A'(x, s) = A^*(x, G(s))$ and L' be the language accepted by A'. Then by a triangle inequality,

$$\begin{aligned}\Pr_s[A'(x, s) \neq 1_{x \in L}] &= \left|\mathbb{E}_s[A'(x, s) - A^*(x, U) + A^*(x, U) - 1_{x \in L}]\right| \\ &< \frac{1}{27} + \frac{7}{27} \\ &< \frac{8}{27}.\end{aligned}$$

See that A' runs in time $T' = \mathbf{poly}(T)$ and space $S' = O(S)$.

In our interactive protocol, the verifier first chooses $l = O(\log(T)S)$ bits, s, for our **PRG** and sends them to the prover. Let $x' = (x, s)$, so our new input length is $m = n + l = O(\log(T)S + n)$. By Theorem 1, there is an interactive protocol for whether $x' \in L'$ with perfect completeness and soundness $\frac{1}{3}$ where the verifier, V', runs in time $\tilde{O}(\log(T)S + n)$ and the prover, P', runs in time $2^{O(S)}$.

Our final protocol repeats the above protocol three times, and outputs that $x \in L$ if the prover proves $x' \in L'$ three times, or outputs that $x \notin L$ if the prover proves $x' \notin L'$ three times, and rejects otherwise.

Completeness If $x \in L$, with probability $\frac{19}{27} > \frac{2}{3}$ we have $x' \in L'$. If $x' \in L'$, by completeness of our deterministic **IP**, our prover will convince our verifier $x' \in L$. Thus the verifier outputs $x \in L$ with probability at least $\frac{2}{3}$. Similarly for $x \notin L$.

Soundness If $x \in L$, then with probability at most $\frac{8}{27}$ will we have $x' \notin L'$. If $x' \in L'$, by soundness of our deterministic **IP**, the probability any prover convinces V' that $x' \notin L'$ is at most $\frac{1}{3}$. So the probability it convinces V' that $x' \notin L'$ three times, and thus convincing V to output $x \notin L$, is at most $\frac{1}{27}$. So by a union bound, the probability V outputs that $x \notin L$ is at most $\frac{1}{3}$. Similarly for $x \notin L$.

Time The final verifier spends $O(\log(T)S)$ time choosing s, then runs V' three times, which takes time $\tilde{O}(\log(T)S+n)$. The final prover is just P', which takes time $2^{O(S)}$.

□

5 Efficient IP for NTISP

The non deterministic algorithm uses Theorem 21, but some care must be taken to choose p so that the number of accepting paths is not 0 mod p. In general, the number of accepting paths may be an adversarial number, so we choose p randomly. For instance, it could be the product of every number less than $\frac{T}{\log(T)}$. Thus this strategy may need $p = \Omega(\frac{T}{\log(T)})$.

First we show such a prime p can be found with high probability.

Lemma 22 (Find Non-Divisor WHP)**.** *There is an algorithm A taking integer W, and constant $\epsilon > 0$ running in time $\tilde{O}(\mathbf{polylog}(\frac{1}{\epsilon})\log(W)^3)$ such that for any $w \leq 2^W$ with probability at least $1-\epsilon$, algorithm A outputs prime $p = O(W/\epsilon)$ and $w \mod p \neq 0$.*

Proof. First, for any integer m, let k_m be the number of prime numbers dividing w greater than m. Then we have

$$\begin{aligned} m^{k_m} \leq & w \\ \leq & 2^W \\ k_m \leq & \frac{W}{\log(m)} \\ \leq & \frac{W}{\ln(m)}. \end{aligned}$$

By the prime number theorem, for large enough W, for any $m \geq W$, the number of primes less than m are at most $\frac{1.25m}{\ln(m)}$, and the number of primes less than $2m$ are at least $\frac{1.75m}{\ln(m)}$. Thus there are at least $\frac{0.5m}{\ln(m)}$ primes between m and $2m$. If W is too small, just hard code some large, constant prime.

Otherwise, let $m = \frac{4W}{\epsilon}$. Then the number of primes between m and $2m$ is at least $\frac{2W}{\ln(m)\epsilon}$. Recall the total number of primes larger than m dividing w is at most $\frac{W}{\ln(m)}$. Therefore, at most $\frac{\epsilon}{2}$ fraction of the primes between m and $2m$ divide w.

Then using Miller Rabin tests on randomly chosen numbers (see Theorem 17), there is an algorithm running in time

$$\tilde{O}(\mathbf{polylog}(\frac{2}{\epsilon})\log(m)^3) = \tilde{O}(\mathbf{polylog}(\frac{1}{\epsilon})\log(W)^3)$$

outputting a uniform prime p between m and $2m$ with probability at least $\frac{\epsilon}{2}$. Then by a union bound the probability it fails to output a uniform prime p or that p divides w is at most ϵ. □

A corollary is that any w has some prime number not dividing it with size $O(\log(w))$.

Corollary 23 (Log Size Non Divisors Exist)**.** *For any integer w, there exists a prime number $p = O(\log(w))$ such that p does not divide w.*

Then using this procedure, we can with high probability find an appropriate prime, p, so that if the number of accepting paths, w, is non zero, than $w \mod p \neq 0$.

Theorem 24 (Verifier Efficient Interactive Protocol For **NTISP**)**.** *Let S, T and W be computable in time $\tilde{O}(\log(W)\log(T)S + n)$. Suppose L is recognized by a nondeterministic algorithm, A, running in time T and space S where the total number of accepting witnesses are at most 2^W. Then*

$$L \cup L^c \subseteq \mathbf{ITIME}^1[\tilde{O}(\log(W)\log(T)S + n), 2^{O(S)}]$$

where L^c is the complement of L.

Proof. We begin by describing with verifier V and honest prover P. We describe the protocol for L, the protocol for L^c is similar. Let $w = O(2^W)$ be the number of accepting paths of A on x. First P outputs whether $w = 0$ or $w \neq 0$. The protocol splits into two cases depending on what the prover claimed about w:

$w \neq 0$: Then P chooses a prime number $p = O(W)$ such that $w \mod p \neq 0$. Such a prime is guaranteed to exist by Corollary 23. Prover P gives p to the verifier V.

Then V tests if p is prime with the Miller Rabin primality test, Theorem 16, with soundness $\frac{1}{3}$ and rejects if it fails. If p passes the primality test, V performs the interactive protocol of Theorem 21 with verifier V' and honest prover P' to confirm that $w \mod p \neq 0$ with soundness $\frac{1}{3}$. Verifier V outputs $x \in L$ if V' outputs that $w \mod p \neq 0$ and rejects otherwise.

$w = 0$: Then V uses Lemma 22 to choose a prime $p = O(W/\epsilon)$ that does not divide w with probability at least $\frac{5}{6}$. If the selected p is not prime, then P proves it by sending a factorization of p. If the factorization is correct, V gives up and says $x \notin L$.

Otherwise, V performs the interactive protocol from Theorem 21 with verifier V' and honest prover P' to verify $w \mod p$ is 0 with soundness $\frac{1}{6}$. The verifier V outputs $x \notin L$ if V' outputs that $w \mod p = 0$ and rejects otherwise.

Now we prove completeness, soundness, verifier time and prover time.

Completeness First, P truthfully outputs whether $w = 0$.

$w \neq 0$**:** Suppose $x \in L$. Then for number accepting paths w, there exists a $p = O(W)$ such that $w \mod p \neq 0$ by Corollary 23. By completeness of the Miller Rabin primality test Theorem 16, V confirms p is prime. By the completeness of Theorem 21, P' convinces V' that $w \mod p \neq 0$. Thus V outputs $x \in L$.

$w = 0$**:** Suppose $x \notin L$. If V does not find a prime p, prover P proves the candidate is not prime. Thus the V outputs $x \notin L$.

If V does find prime p, by the completeness of the protocol in Theorem 21, P' convinces V' that $w \mod p = 0$. Thus V outputs $x \notin L$.

Soundness: Consider any prover $\tilde{P}$. We use two cases, depending on whether w is actually 0, or not.

$w \neq 0$**:** Suppose $x \in L$. Then if $\tilde{P}$ claims $w \neq 0$, then V either confirms $x \in L$, or rejects. So suppose $\tilde{P}$ claims $w = 0$.

Since $w \neq 0$, by soundness of Lemma 22, the probability V chooses a p so that $w \mod p \neq 0$ is at least $\frac{5}{6}$. If $w \mod p \neq 0$, then from the soundness of Theorem 21, the probability V' accepts that $w \mod p = 0$ is at most $\frac{1}{6}$. Thus by a union bound, V rejects with probability at least $\frac{2}{3}$.

$w = 0$**:** Suppose $x \notin L$. If $\tilde{P}$ claims $w = 0$, then V either confirms $x \notin L$, or rejects. So suppose $\tilde{P}$ claims $w \neq 0$.

Then for any number p provided by $\tilde{P}$, if p is not prime, by soundness of Theorem 16, V rejects with probability $\frac{2}{3}$. If p is prime, then $w \mod p = 0$, so by soundness of Theorem 21, $\tilde{P}$ can only convince V' that $w \mod p \neq 0$ with probability $\frac{1}{3}$. So V rejects with probability at least $\frac{2}{3}$.

Verifier Time: Since prime generation and testing run in time $\tilde{O}(\log(W)^3)$, and the verifier in Theorem 21 runs in time

$$\tilde{O}\left(\log(pS)\right) O(\log(T)S + n) = \tilde{O}(\log(W)\log(T)S + n),$$

the total verifier time is $\tilde{O}(\log(W)\log(T)S + n)$.

Prover Time: The number of accepting paths, w, can be calculated in time $2^{O(S)}$ by repeated squaring of the computation graph of A on input x. Given $w \neq 0$, the prover can find p such that $w \mod p \neq 0$ through exhaustive search in time $\mathbf{poly}(W) = 2^{O(S)}$. Similarly, factorizing a composite p by exhaustive search takes time $\mathbf{poly}(p) = \mathbf{poly}(W) = 2^{O(S)}$. Finally, the prover in Theorem 21 runs in time $\tilde{O}(\log(W))2^{O(S)} = 2^{O(S)}$. So the prover runs in time $2^{O(S)}$.

□

One can trivially upper bound the total number of accepting witness with $W = O(T)$. This is because at each time step, only one bit of the witness can be read. Thus the length of a witness string is only T bits, so there are only at most 2^T possible witnesses. This gives us the immediate corollary of Theorem 3.

6 Open Problems

Two directions for improvement are improving the verifier time and prover time.

1. Could one remove the dependence on $\log(T)$? Is it true that, for $S = \omega(n)$,

$$\mathbf{SPACE}[S] \subseteq \mathbf{ITIME}[\tilde{O}(S)]?$$

   This would prove an equivalence, up to polylogarithmic factors, between $\mathbf{SPACE}[S]$ and $\mathbf{ITIME}[S]$. Recent work, [MC22], showed a similar equivalence between $\mathbf{NTIME}[T]$ and languages verified by **PCP**s with $\log(T)$ time verifiers, for $\log(T) = \Omega(n)$.

2. Can interactive protocols for **NTISP** be as efficient as those for **TISP**?

   For $S = \Omega(n)$, we prove that $\mathbf{SPACE}[S] \subseteq \mathbf{ITIME}[\tilde{O}(S^2)]$, but only show that $\mathbf{NSPACE}[S] \subseteq \mathbf{ITIME}[\tilde{O}(S^3)]$. Does nondeterministic space require more verifier time than deterministic space?

3. Can we get double efficiency with a similar verifier time? Is it true that

$$\mathbf{TISP}[T, S] \subseteq \mathbf{ITIME}[\mathbf{polylog}(T)S, \mathbf{poly}(T)]?$$

   This would make the verification of polynomial time algorithms only take as long (up to polylogairthmic factors) as the space of those algorithms, with a proof that still only takes polynomial time. This would make directly using interactive proofs for delegating computation more practical.

References

[AS98] Sanjeev Arora and Shmuel Safra. "Probabilistic Checking of Proofs: A New Characterization of NP". In: *J. ACM* 45.1 (Jan. 1998), 70–122. ISSN: 0004-5411. DOI: 10.1145/273865.273901. URL: https://doi.org/10.1145/273865.273901.

[Aro+98] Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. "Proof Verification and the Hardness of Approximation Problems". In: *J. ACM* 45.3 (1998), 501–555. ISSN: 0004-5411. DOI: 10.1145/278298.278306. URL: https://doi.org/10.1145/278298.278306.

[BFL90] L. Babai, L. Fortnow, and C. Lund. "Nondeterministic exponential time has two-prover interactive protocols". In: *Proceedings [1990] 31st Annual Symposium on Foundations of Computer Science*. 1990, 16–25 vol.1. DOI: 10.1109/FSCS.1990.89520.

[CT22] Lijie Chen and Roei Tell. "Hardness vs Randomness, Revised: Uniform, Non-Black-Box, and Instance-Wise". In: *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*. 2022, pp. 125–136. DOI: 10.1109/FOCS52979.2021.00021.

[GKR15] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. "Delegating Computation: Interactive Proofs for Muggles". In: *J. ACM* 62.4 (Sept. 2015). ISSN: 0004-5411. DOI: 10.1145/2699436. URL: https://doi.org/10.1145/2699436.

[Gol+07] Shafi Goldwasser, Dan Gutfreund, Alexander Healy, Tali Kaufman, and Guy N. Rothblum. "Verifying and Decoding in Constant Depth". In: *Proceedings of the Thirty-Ninth Annual ACM Symposium on Theory of Computing*. STOC '07. San Diego, California, USA: Association for Computing Machinery, 2007, 440–449. ISBN: 9781595936318. DOI: 10.1145/1250790.1250855. URL: https://doi.org/10.1145/1250790.1250855.

[Gol18] Oded Goldreich. *On Doubly-Efficient Interactive Proof Systems*. 2018. URL: https://www.wisdom.weizmann.ac.il/~oded/de-ip.html.

[Lun+90] C. Lund, L. Fortnow, H. Karloff, and N. Nisan. "Algebraic methods for interactive proof systems". In: *Proceedings [1990] 31st Annual Symposium on Foundations of Computer Science*. 1990, 2–10 vol.1. DOI: 10.1109/FSCS.1990.89518.

[MC22] Dana Moshkovitz and Joshua Cook. *Tighter MA/1 Circuit Lower Bounds From Verifier Efficient PCPs for PSPACE*. 2022. URL: https://eccc.weizmann.ac.il/report/2022/014/.

[MW18] Cody Murray and Ryan Williams. "Circuit Lower Bounds for Nondeterministic Quasi-Polytime: An Easy Witness Lemma for NP and NQP". In: *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*. STOC 2018. Los Angeles, CA, USA: Association for Computing Machinery, 2018, 890–901. ISBN: 9781450355599. DOI: 10.1145/3188745.3188910. URL: https://doi.org/10.1145/3188745.3188910.

[Mil75] Gary L. Miller. "Riemann's Hypothesis and Tests for Primality". In: *Proceedings of the Seventh Annual ACM Symposium on Theory of Computing*. STOC '75. Albuquerque, New Mexico, USA: Association for Computing Machinery, 1975, 234–239. ISBN: 9781450374194. DOI: 10.1145/800116.803773. URL: https://doi.org/10.1145/800116.803773.

[Nis90] Noam Nisan. "Pseudorandom Generators for Space-Bounded Computations". In: *Proceedings of the Twenty-Second Annual ACM Symposium on Theory of Computing*. STOC '90. Baltimore, Maryland, USA: Association for Computing Machinery, 1990, 204–212. ISBN: 0897913612. DOI: 10.1145/100216.100242. URL: https://doi.org/10.1145/100216.100242.

[RRR16] Omer Reingold, Guy N. Rothblum, and Ron D. Rothblum. "Constant-Round Interactive Proofs for Delegating Computation". In: STOC '16. Cambridge, MA, USA: Association for Computing Machinery, 2016, 49–62. ISBN: 9781450341325. DOI: 10.1145/2897518.2897652. URL: https://doi.org/10.1145/2897518.2897652.

[Rab80] Michael O Rabin. "Probabilistic algorithm for testing primality". In: *Journal of Number Theory* 12.1 (1980), pp. 128–138. ISSN: 0022-314X. DOI: https://doi.org/10.1016/0022-314X(80)90084-0. URL: https://www.sciencedirect.com/science/article/pii/0022314X80900840.

[SZ99] Michael Saks and Shiyu Zhou. "$BP_HSPACE(S) \subseteq DSPACE(S^{3/2})$". In: *J. Comput. Syst. Sci.* 58.2 (Apr. 1999), 376–403. ISSN: 0022-0000. DOI: 10.1006/jcss.1998.1616. URL: https://doi.org/10.1006/jcss.1998.1616.

[San07] Rahul Santhanam. "Circuit Lower Bounds for Merlin-Arthur Classes". In: *Proceedings of the Thirty-Ninth Annual ACM Symposium on Theory of Computing*. STOC '07. San Diego, California, USA: Association for Computing Machinery, 2007, 275–283. ISBN: 9781595936318. DOI: 10.1145/1250790.1250832. URL: https://doi.org/10.1145/1250790.1250832.

[Sav70] Walter J. Savitch. "Relationships between Nondeterministic and Deterministic Tape Complexities". In: *J. Comput. Syst. Sci.* 4.2 (Apr. 1970), 177–192. ISSN: 0022-0000. DOI: 10.1016/S0022-0000(70)80006-X. URL: https://doi.org/10.1016/S0022-0000(70)80006-X.

[Sha92] Adi Shamir. "IP = PSPACE". In: *J. ACM* 39.4 (Oct. 1992), 869–877. ISSN: 0004-5411. DOI: 10.1145/146585.146609. URL: https://doi.org/10.1145/146585.146609.

ECCC ISSN 1433-8092

https://eccc.weizmann.ac.il