Round Complexity Versus Randomness Complexity in Interactive Proofs

Maya Leshkowitz
Weizmann Institute of Science, Rehovot, Israel
mleshkowitz@gmail.com

Abstract

Consider an interactive proof system for some set S that has randomness complexity $r(n)$ for instances of length n, and arbitrary round complexity. We show a public-coin interactive proof system for S of round complexity $O(r(n)/\log n)$. Furthermore, the randomness complexity is preserved up to a constant factor, and the resulting interactive proof system has perfect completeness.

2012 ACM Subject Classification Theory of computation → Interactive proof systems

Keywords and phrases Interactive Proofs

Digital Object Identifier 10.4230/LIPIcs.APPROX-RANDOM.2018.49

Related Version A full version of the paper is available at `https://eccc.weizmann.ac.il/report/2017/055/`.

Acknowledgements I would like to thank my advisor, Prof. Oded Goldreich, for his encouragement to approach the problem studied in this paper, and for his guidance and support in the research and the writing process. I also thank Dr. Guy Rothblum for useful discussions.

1 Introduction

The notion of interactive proof systems, put forward by Goldwasser, Micali, and Rackoff [8], and the demonstration of their power by Lund, Fortnow, Karloff, Nisan [12] and Shamir [14] are among the most celebrated achievements of complexity theory.

Loosely speaking, interactive proof systems capture the most general way in which one party can efficiently verify claims made by another, more powerful, party. The definition of interactive proof systems generalizes the traditional notion of a proof system (indeed, an NP-proof system), by allowing both *interaction* and *randomness*.

It is well known that both interaction and randomness are inherent to the power of interactive proof systems, where here we mean the extra power above that of NP-proof systems. Interactive proofs with no randomness can be easily transformed into NP-proof systems, whereas randomized non-interactive proofs, captured by the class $\mathcal{MA}$ (defined by Babai [1]), are essentially the randomized version of $\mathcal{NP}$ (e.g., loosely speaking, if $\mathcal{BPP} = \mathcal{P}$, then $\mathcal{MA} = \mathcal{NP}$).

Both *interaction* and *randomness* are quantitative notions; that is, one talks of the "amount of interaction", which is commonly associated with the (total) number of messages exchanged (a.k.a number of *rounds*), and of the amount of randomness (a.k.a *randomness complexity*). While the previous paragraph refers to the qualitative question and asserts that both interaction and randomness are essential, a finer study of the quantitative question is called for; that is, it is natural to ask about the necessary amount of interaction and randomness in various interactive proof systems.

The study of round-complexity in interactive proof systems has some well known results: Babai and Moran showed that the round complexity of any public-coin interactive proof system (a.k.a Arthur-Merlin proofs) can be reduced by a constant factor [2], whereas the public-coin transformation of Goldwasser and Sipser [9] (which essentially preserves the number of rounds) extends this result to general interactive proof systems. It is also known that a stronger round reduction is quite unlikely, since it would place SAT in co-AM-time($2^{o(n)}$), whereas AM-time(T) may equal Ntime(poly(T)). (This is the case due to a combination of results reviewed below in the paragraph titled "conditional tightness".)

In contrast, we are only aware of one study that focuses on the randomness complexity of interactive proof systems [1]. Specifically, Bellare et al. [3] studied the randomness complexity of *error reduction* in the context of interactive proof systems. (We mention that the randomness complexity is also of interest in [5], which provides an alternative transformation of general interactive proof systems to public-coin ones.)

Regarding the completeness in interactive proofs, the fact that every interactive proof system can be transformed into one with perfect completeness was shown in [4].

1.1 Round complexity versus randomness complexity

A natural question, which to the best of our knowledge was not considered before, is what is the relation between the two foregoing complexity measures. We do suspect that there are cases when the randomness complexity cannot be made smaller than the round complexity without trivially increasing the number of rounds. This is because constant-round interactive proof systems seem more powerful than NP-proof systems (see, e.g., the Graph Non-Isomorphism proof of [6]), whereas a logarithmic amount of randomness is clearly useless. But *can the randomness complexity be smaller than the round complexity?*

The answer is definitely negative if we consider *public-coin* interactive proof systems. Recall that in these proof systems, in each round, the verifier sends the outcome of fresh coins that it has tossed at the beginning of the current round, and so by definition the number of coin tosses is at least as large as the number of rounds.[2] However, it is not clear what happens in case of general interactive proofs. (In particular, the transformation of [9] significantly increases the randomness complexity by a factor depending on the number of rounds.)

Recall that in a general interactive proof system, the verifier may toss all coins at the very beginning, but its message in each round may be a complex function of the outcome of these coins (and the messages it has received from the prover). In particular, the verifier's messages may have very little information contents (from the prover's point of view), and so we may have many more rounds than the number of coins tossed. Furthermore, it is not clear how to collapse rounds that yield verifier messages of low information contents. These are the issues we deal with when showing that also in general interactive proof systems, randomness complexity $r(n)$ yields round complexity $O(r(n)/\log n)$, and hence in interactive proof systems *the round complexity is smaller than the randomness complexity.*

▶ **Theorem 1** (Main Theorem). *Suppose that S has an interactive proof system of randomness complexity $r(n)$ for instances of length n. Then, S has a* public-coin *interactive proof system*

[1] We refer to unconditional results and not to the long line of research of randomness versus hardness trade-off that rely on uniform or non-uniform assumptions, see, e.g. [10, 13].

[2] This presumes that the definition requires the verifier to send a non-empty message in each round. But otherwise (i.e., if the definition allows empty messages), rounds in which the verifier sends nothing can be collapsed.

of round complexity $O(r(n)/\log n)$ and randomness complexity $O(r(n))$. Furthermore, the resulting interactive proof system has perfect completeness.

Note that, in addition to obtaining the public-coin feature, we obtain perfect completeness for free. That is, even if the original system does not have perfect completeness, the new one has this feature.

We note that it is easier to prove a weaker version of the main theorem, which does not obtain the public-coin and perfect completeness features. The proof of this weaker result, outlined in Section 1.2, and given in detail in the full vesion of this paper [11, Apdx C], illustrates one of the ideas that underlie the proof of Theorem 1.

Conditional tightness

The round-complexity obtained by Theorem 1 is the best one may hope for at this time, since a result asserting round complexity $o(r(n)/\log n)$ for any set that has an interactive proof system of randomness complexity $r(n)$ would yield an unexpected result that conflicts with common beliefs and seems currently out of reach. Specifically, it would place $\mathtt{SAT}$ in co-AM-time$(2^{o(n)})$, which does contradict common beliefs. The full reasoning is as follows:

1. A variant of the celebrated interactive proof system for $\overline{\mathtt{SAT}}$ yields an interactive proof system of randomness complexity $O(n)$ for unsatisfiable CNFs with n variables. (This interactive proof consists of $n/\log n$ rounds such that in each round we strip a single variable in the sum-check that sums over $n/\log n$ variables with values in $[n]$, while using a finite field of size poly(n). See [7, Sec 8]).[3]
2. On the other hand, any set having an m-round interactive proof system is in AM-time$(n^{O(m)})$, see [7, Apdx B]. Hence, if unsatisfiable CNFs have an interactive proof of round complexity $o(n/\log n)$, then such instances can be refuted in AM-time$(2^{o(n)})$, whereas AM-time(T) may equal Ntime(poly(T)).

A different perspective

Theorem 1 may be viewed as an alternative transformation of general interactive proof systems into public-coin ones. Recall that the transformation of Goldwasser and Sipser [9] preserves the round-complexity of the original system (up to an additive constant), but increases the randomness complexity (i.e., raising it to a constant power). The same holds in the variant of that transformation presented in [5]. In contrast, Theorem 1 preserves the randomness complexity of the original system (up to a constant factor), but does not necessarily preserve the round-complexity. Taking this perspective, the fact that the round-complexity is bounded in terms of the randomness complexity is a consequence of the fact that the resulting scheme is of the public-coin type.

[3] This is done by packing a sequence of $\log n$ bits of the boolean variables into a symbol of $H = [n] \subseteq F$ where F is some field. For $i \in \ell$, where $\ell = \log n$, denote by $f_i(x) : F \to F$ the polynomial of degree $n-1$ that maps each element in H to the value of its ith boolean variable. Now, take the standard arithmetization of the CNF and replace each occurrence of the variable indexed $j \cdot \ell + i$ by the polynomial $f_i(x_j)$, where x_j is the F variable that represents the jth block of boolean variables. The resulting polynomial is a n/ℓ variable polynomial of total degree $O(m \cdot n)$, where m is the number of clauses. Finally, the number of satisfying assignments is given by the sum over all $(y_1, ..., y_{n/\ell}) \in H^{n/\ell}$ of the polynomial derived above. Furthermore, we do not execute the sum-check protocol over an exponentially large finite field but rather over a finite field of prime cardinality $p = \text{poly}(n)$, where p is selected by the verifier at random among such primes.

1.2 On the proof of Theorem 1

We start by giving an overview of a proof of a weaker result, in which we show how to transform any interactive proof, of randomness complexity $r(n)$, to a *private-coin* interactive proof for the same set that uses $O(r(n)/\log n)$ rounds, while maintaining the randomness complexity of $r(n)$. This proof gives a flavor of the proof of the main theorem, but is significantly simpler.

1.2.1 Private-coin emulation protocol

The idea of the emulation protocol is that, in every iteration, we would like the prover to send possible continuations of the current transcript (describing execution segments of possibly different number of rounds) that reveal much information about the verifier's random coins. Hence, the prover sends partial transcripts of *maximal* length such that each account for a large fraction of the residual probability mass[4]. The verifier then checks if one of these transcripts is consistent with the strategy determined by the values of its random coins, which were tossed upfront. If so, the verifier picks the maximal transcript consistent with its strategy and the verifier and prover proceed their interaction from that point. Otherwise, the verifier sends its next message (based on the aforementioned coins) without using the continuations suggested by the prover. We stress that the only source of the verifier's randomness is its private coins tossed upfront, which are used to determine the continuation of the transcript in each subsequent iteration.

We wish to elaborate on how the prover determines the continuations of the transcripts. Fixing an iteration, we denote the current transcript by γ and its residual probability mass by $p(\gamma)$. Each transcript the prover sends on this iteration is a possible continuation of γ of *maximal* length that is a "heavy continuation". By a heavy continuation γ', we mean that γ' has probability mass greater than $p(\gamma)/n$, when subtracting from it the probability mass of the continuations of γ that were either sent by the prover in previous iterations, or determined in this one.

This conditioning allows the prover to send several continuations of the transcript that are also continuations of each other. Consider for example the case that the prover sends $\gamma\alpha_1\beta_1\alpha_2\beta_2$ and $\gamma\alpha_1\beta_1$. In this case if the verifier chooses $\gamma\alpha_1\beta_1$ it means that in the next iteration the continuation of this transcript cannot begin with $\alpha_2\beta_2$.

The benefit of this method of determining transcript-continuations is that we guarantee that in the *next* iteration the probability mass of the new transcript is *lower* than $p(\gamma)/n$. The reasoning is as follows. If the verifier chooses one of the transcripts suggested by the prover, then on the *next* iteration the residual probability mass of each of its continuations is lower than $p(\gamma)/n$, otherwise this continuation should have been suggested by the prover on the previous iteration. If the verifier did not choose any of the transcripts, and instead continued the transcript with its own message $\widetilde{\alpha_1}$, then it follows that the residual probability mass of the transcript $\gamma\widetilde{\alpha_1}$ (under the conditioning of the appropriate events) is also lower than $p(\gamma)/n$, otherwise the continuation $\gamma\widetilde{\alpha_1}$ should have been suggested by the prover.

It follows that after $O(r(n)/\log n)$ rounds of interaction the probability of the transcript generated is at most $(1/n)^{r(n)/\log n} = 1/2^{r(n)}$, which means that there is a unique value of the coin tosses consistent with the transcript. Hence, a complete transcript is generated and

[4] Note that in the eyes of an observer, a verifier that samples its random coins at the beginning of the interaction and proceeds accordingly, is equivalent to a verifier that on each round samples a message with probability proportional to its residual probability mass.

the verifier can reject or accept at this point. It is easy to show that if the prover follows the prescribed emulation, then the verifier accepts with the same probability as in the original interactive proof system, and hence completeness is maintained.

Note that the above emulation per se does not suffice. It is essential to include validation checks that guarantee that the transcripts provided by the prover are consistent with some prover strategy for the original protocol. This means that if the prover provides two transcripts that share a prefix, this common prefix must end with a prover's message. This implies that the prover answers in the same way to the same verifier messages, which means that the prover's strategy is consistent with some prover of the original emulation, and so the soundness of the original proof system is maintained.

1.2.2 Public-coin emulation protocol

The simplified private-coin emulation protocol captures one of the key ideas of our public-coin emulation protocol. The difficulty that we face when seeking a public-coin emulation is that we cannot rely on hidden coins tossed upfront by the verifier. Thus, when presented with a list of heavy continuations, it is unclear how the verifier should select one at random, since the selection probability should be determined by the residual probability masses that are unknown to it. Our solution is to have the prover provide these probabilities, but this raises the need to verify these claimed values. (Needless to say, the verifier rejects upfront if the sum of these probabilities does not match the claimed probability of the transcript as determined before the current round.) In the last iteration, a complete transcript is sampled, containing the verifier's private coins, hence the validity of the transcript and the claim can be checked.

The foregoing description raises a few issues. Firstly, the prover should find a way to communicate all the transcripts to the verifier, and not only the ones with high residual probability mass as before. Second, it is not clear what happens when the prover provides wrong values for the residual probabilities. As for the second issue, note that maliciously raising the probability of a transcript does contribute towards having the sum of probabilities meet the prior claim, but it makes the probability that this transcript is selected higher, and so puts the prover in greater problem in the next round. Indeed, a careful analysis shows that actually the prover gains nothing by such behaviour, since when the transcript is complete, false claims about its residual probability are easily detected.

Turning back to the first issue, we note that the issue is that there may be too many short transcripts that each account for a small fraction of the residual probability mass. To deal with this case, we pack many transcripts into a single auxiliary message, which means that we use a succinct representation of a sequence that contains many of the transcripts but not all of them (since otherwise we would have made no progress at the current round). The succinct representation should support the verification that the corresponding sequences are disjoint. Now, each such "pack" of transcripts will be assigned the corresponding probability mass, and be treated as if it were an actual transcript.

Needless to say, the foregoing is but a very rough sketch of the structure of the derived proof system. The actual proof system uses a carefully designed verification procedure that ensures that its executions can be mapped to executions in the original proof system.

We note that while the above description of the public-coin emulation refers to the probability that various transcripts appear in the original proof system (when the prover uses an optimal strategy), our actual construction refers only to accepting transcripts (i.e., transcripts that lead the original verifier to accept). Consequently, we obtain a proof system of perfect completeness, even if the original proof system had two-sided error probability.

1.3 Organization

Towards proving the main theorem we shall show how to emulate an existing interactive proof system with a public coin emulation protocol that has $O(r(n)/\log n)$ rounds. We begin by introducing the notion of "protocol trees" in Section 3, which we use to describe the interaction of the verifier and prover of the original interactive proof system. In Section 4, we shall show how to transform the protocol tree into an "emulation tree", that contains the continuations of the transcripts that the prover sends on each iteration along with their probability masses. Using this emulation tree, we then turn to describing the public-coin emulation protocol for the new prover and verifier in Section 5. The analysis of the emulation, which is partitioned into completeness and soundness, is given in the full version of the paper [11, Sec 6]. Also given in the full version [11, Apdx C] is a private-coin emulation protocol for emulating the interactive proof system and its analysis, which is less involved than the public-coin emulation.

2 Preliminaries

Let us start by formally defining interactive proof systems, where the completeness and soundness bounds are parameters.

▶ **Definition 2** (Interactive Proof Systems)**.** Let $c, s : \mathbb{N} \to [0,1]$ such that $c(|x|) \geq s(|x|) + \frac{1}{poly(|x|)}$. An interactive proof system for a set S is a two party game, between a verifier executing a probabilistic polynomial time strategy, denoted V, and a prover executing a *(computationally unbounded)* strategy, denoted P, satisfying the following two conditions:

- Completeness with bound c: For every $x \in S$, the verifier V accepts after interacting with the prover P on common input x with probability at least $c(|x|)$.
- Soundness with bound s: For every $x \notin S$ and every prover strategy $\widetilde{P}$, the verifier V accepts after interacting with $\widetilde{P}$ on common input x with probability at most $s(|x|)$.

When c and s are not specified, we mean $c \equiv 2/3$ and $s \equiv 1/3$. We denote by $\mathcal{IP}$ the class of sets having interactive proof systems. When $c \equiv 1$, we say that the system has perfect completeness.

3 The protocol tree of the original proof system

Fixing an interactive proof and an instance x of length n, we describe the possible prover-verifier interactions of the system on common input x using a tree whose height corresponds to the number of rounds of interaction. For some $\ell = \ell(n)$, we assume without loss of generality that in each round the verifier sends a message $\alpha \in \{0,1\}^\ell$, and the prover's responds with a message $\beta \in \{0,1\}^\ell$. We can also assume, without loss of generality, that the prover's strategy is deterministic and fixed. Each node v in level j represents a possible prover-verifier transcript for the first j rounds of the interaction. The branching of the tree represents the possible ways to extend the transcript to the next round. The number of ways to extend the transcript depends only on the verifier's message, since we fixed the prover's strategy. Hence, each node has *at most* $d \coloneqq 2^\ell$ children, corresponding to the 2^ℓ possible verifier messages for the next round. The prover's response to each such message is included in the **description** of the corresponding node.

The description of a node u on level j contains the partial **transcript** $\gamma(u) = \alpha_1\beta_1, \ldots, \alpha_j\beta_j$ of the interaction up to the j'th round. The root (at level zero) has an empty transcript, whereas a leaf of the tree represents a complete prover-verifier interaction.

We can assume, without loss of generality, that the verifier sends its private coins on the last round, and hence every leaf is associated with a sequence of coin tosses which either leads the verifier to accept or to reject. Hence, we can represent the possible interactions generated by the interactive proof system using a tree of height m which has $2^{r(n)}$ leaves, where m is the number of rounds and $r(n)$ is the number of coin tosses. Using a constant number of parallel repetitions, we can assume that the interactive proof system has completeness parameter $\frac{9}{10}$ and soundness parameter $\frac{1}{10}$. Note that this blows up the randomness complexity only by a constant factor (as compared to our interactive proof for the standard $\frac{1}{3}$, $\frac{2}{3}$ parameters). Therefore, if x is a yes-instance then at least $\frac{9}{10} \cdot 2^{r(n)}$ of its leaves represent accepting runs, and if x is a no-instance then at most $\frac{1}{10} \cdot 2^{r(n)}$ of its leaves represent accepting runs.

The description of a node also contains its **weight**, denoted $w(u)$. The weight of the node is the number of coin sequences that are consistent with the node and lead the verifier to accept at the end of the interaction. That is,

▶ **Definition 3** (Weight of a leaf)**.** Let u be a leaf with transcript $\gamma(u)$ which corresponds to the full transcript of the interaction of P and V on input x, when V uses coins ρ; that is,

$$\gamma(u) = (\alpha_1, \beta_1, \ldots, \alpha_m, \beta_m, (\rho, \sigma)) \tag{1}$$

where $\sigma = V(x, \rho, \beta_1, \ldots, \beta_m) \in \{0, 1\}$ is V's final verdict and for every $i = 1, \ldots, m$ it holds that $\alpha_i = V(x, \rho, \beta_1, \ldots, \beta_{i-1})$ and $\beta_i = P(x, \alpha_1, \ldots, \alpha_i)$. We define the weight $w(u)$ of u to be V's final verdict σ.

▶ **Definition 4** (Weight of a node)**.** The weight $w(u)$ of a node u in the protocol tree is the sum of the weights of the leaves that are descendants of u.

Note that $w(u)$ is proportional the probability that $\gamma(u)$ is generated and the verifier accepts at the end of the interaction.

4 The emulation tree

4.1 Overview

So far we explained how to represent the possible executions of a m-round interactive proof system on some instance x, where the protocol utilizes $r(n)$ coins. This resulted in a protocol tree of height m with $2^{r(n)}$ leaves. Our goal is to transform this protocol tree to an emulation tree that defines a prover strategy for a $O(r(n)/\log n)$-round public-coin emulation protocol. This transformation is done using the Build_Tree procedure. First we describe a very restricted case where the protocol tree is already suitable for our proposed public-coin emulation and the Build_Tree procedure is not required. Next, we explain how the transformation works in a restricted case when the degree of the protocol tree is bounded by poly(n), and finally in the case of a general protocol tree.

The protocol tree is of height $O(r(n)/\log n)$ and degree poly(n)

In order to convince the verifier that x is a yes-instance the prover makes an initial claim that the weight of the root of the protocol tree is at least $c \cdot 2^{r(n)}$ (where c is the completeness bound). The emulation is initiated at the root of the protocol tree and on each round of the emulation the prover assists the verifier at progressing one step down the protocol tree. (This assistance is required because the verifier does not have access to the protocol tree.) Each round consists of the prover providing the verifier with the descriptions of the children of the current node u that was sampled on the previous round, where these descriptions

contain the weights of the various children. The verifier performs validations to check that according to the descriptions these are legal children of u, and that their claimed weights sum up to $w(u)$. Then, the verifier samples a child with probability that is approximately proportional to its weight, up to a multiplicative factor of $1 + \frac{1}{n}$, using $O(\log n)$ public coins. On the last round, a leaf is sampled, whose description contains the complete prover-verifier interaction along with the coins tossed by the verifier. The new verifier accepts if and only if the transcript sampled is consistent with the original verifier's strategy according to the coin tosses revealed, and leads the original verifier to accept.

To see why this is indeed an interactive proof system for the original language, note that an honest prover can always convince the verifier of the correctness of a true claim using this emulation. Hence the interactive proof system we described has perfect completeness. On the other hand, for no-instances, a prover that wants to make the verifier accept must make an initial claim that the weight of the empty transcript is much larger than its real weight. Namely, the real weight of the empty transcript is at most $s \cdot 2^{r(n)}$, whereas the prover claims that the weight is at least $c \cdot 2^{r(n)}$, where $c > s$ are the completeness and soundness bounds of the original interactive proof system. Thus, there is a multiplicative gap of $\frac{s}{c}$ between the real weight and the one claimed. We can show that, in expectation, this gap is maintained throughout the emulation, up to a factor of $(1 + \frac{1}{n})^n$ that comes from the approximation factor. Therefore, the probability that a leaf that corresponds to an accepting run is sampled on the last round (and hence the verifier accepts) is at most $\frac{s}{c}(1 + \frac{1}{n})^n$, which is smaller than $\frac{1}{3}$ for a suitable choice of s and c.

The degree of the protocol tree is bounded by poly(n)

In this case the height of the protocol tree may be asymptotically larger than $\frac{r(n)}{\log n}$. We create a new tree of height $O(r(n)/\log n)$ to guide the prover's strategy, which we use in a way similar to how we used the protocol tree in the previous paragraph. We call this tree the **emulation tree**. The nodes in the emulation tree are nodes from the protocol tree, however the children of a node u in the emulation tree may be non-immediate descendants of u in the protocol tree.

We start with a protocol tree T rooted at r whose weight is $w(r)$, and on each step we modify this tree towards creating an emulation tree. We define a **heavy descendant** of r to be a node in T whose weight is at least $\frac{w(r)}{n}$, and the weight of each of its children is smaller than $\frac{w(r)}{n}$. Note that there are at most n such nodes.

We modify T so that the children of r in the emulation tree are its original children as well as the heavy descendants that we lift upwards to make them new children of r. This modification is performed using the Build_Tree(r) procedure, which when invoked on a node r identifies the nodes that will be children of r in the new tree, sets them as children of r, and then initiates recursive invocations on the (original and new) children of r, creating the new emulation tree rooted at r. Details follow.

Let T_r denote the temporary tree after the stage that we identify and set the children of r. We start by identifying the heavy descendants of r (they can also be children of r in T), which will become **heavy children** of r in T_r. We then proceed to the non-heavy children of r in T, which will also be children of r in T_r. After we identified the children of r in the new emulation tree, we call the Build_Tree procedure on each child of r in T_r, which creates an emulation tree rooted at that node.

Observe that in the final emulation tree, for each node u, the weight of each grandchild of u is at most $\frac{w(u)}{n}$. This is because if v is an heavy child of u in the emulation tree, then the weight of the descendants of v in T_u is at most $\frac{w(u)}{n}$. Since the children of v in the emulation tree are descendants of v in T_u the claim holds. Otherwise, v is non-heavy child of u, so its

weight is smaller than $\frac{w(u)}{n}$ and hence the weight of the children of v is also smaller than $\frac{w(u)}{n}$.

It follows that the height of the final emulation tree is $O(r(n)/\log n)$. The number of children of each node is at most $\text{poly}(n)$, because we add at most n heavy children to the original children of each node. Hence the emulation tree has properties similar to the protocol tree in the previous paragraph, so it is suitable for our public-coin emulation described in the previous subsection.

The general case

In general, the degree of the protocol tree is unbounded, and hence it may be exponential in n. Lifting the heavy children as we did in the case of unbounded height guarantees that the height of the new emulation tree is $O(r(n)/\log n)$, but its degree may be super-polynomial in n (due to the original children). Hence, we will not be able to perform the emulation. Thus, we also need to make sure the degree of the new tree is $\text{poly}(n)$.

In order to reduce the degree when it is too large, we group the non-heavy children of r under new children, which we call **interval children**. This is done in addition to handling the heavy children of r as before. In general, children of r in T may become non-immediate descendants in T_r, and non-immediate descendants of r may become immediate children of r in T_r (due to lifting).

Determining the children of r in T_r is done in two steps, as part of the Build_Tree(r) procedure. The first step is identifying the heavy descendants of r and lifting them to be children of r, creating heavy children in T_r. Next, we unite the non-heavy children of r into groups. We unite the children by lexicographic order of their transcript field, such that the weight of each group is larger than $\frac{w(r)}{n}$ and at most $\frac{2w(r)}{n}$ (except for, possibly, the last group which is only required to have weight smaller than $\frac{2w(r)}{n}$). We create a new **interval child** v for each such group, where the children of v are the nodes in the group.

After this step, the children of r in the final emulation tree are exactly the children of r in T_r. The number of children r has is at most n, since the weight of each child of r (except for possibly the last interval child) is at least $\frac{w(r)}{n}$. Next, the procedure is called recursively on the children of r in T_r in order to create the final emulation tree.

The description of a node u in the emulation tree is composed of the **transcript** field $\gamma(u) = \alpha_1\beta_1 \ldots \alpha_i\beta_i$ and a **weight** field $w(u)$ as in the original protocol tree, with an additional **range** field $R(u)$. The range of a node represents the possible range of its children's transcripts. After determining the heavy children of u, and before grouping the non-heavy children under interval children, the non-heavy children of u are all children of u in the original tree. Hence, the transcripts of the children of each interval child are the same up to the last verifier's message on which they differ, which corresponds to the branching of the protocol tree for the next round. Thus, we can label the range $R(u) = [s, e]$ where $s < e \in \{0,1\}^\ell$ according to the range of the last verifier message in the transcript field of the children of u. Heavy children have full range $\left[0^\ell, 1^\ell\right]$, whereas the range of interval children is a subinterval of $\left[0^\ell, 1^\ell\right]$ such that this subinterval corresponds to the transcripts of the descendants that are grouped under this node.

We show in the analysis that the height of the final emulation tree is $O(r(n)/\log n)$. The degree of nodes in the final emulation tree is at most n, hence it is suitable for public-coin emulation like in the previous subsection.

(The running time of this algorithm is at least the size of the protocol tree, which is exponential in $r(n)$, and thus it may be exponential in $|x|$. However, the prover is the one that runs this algorithm and the prover is computationally unbounded. Therefore the running time is not an issue.)

4.2 The Build Tree procedure

Denote the designated prover and verifier of the original interactive proof system by P_0 and V_0 respectively, and the protocol tree of P_0 and V_0 for a yes-instance x by T_{P_0,V_0}. The Build_Tree procedure is a recursive procedure that reads and updates a global tree T, which is initially set to equal the protocol tree T_{P_0,V_0} until obtaining the final emulation tree, denoted by E_{P_0,V_0}. When invoked on a node u in T, the procedure determines the children of u, updates the global tree and invokes the procedure recursively on the children of u.
We denote by $T(u)$ the subtree of T rooted at u.

Initialization

The tree T is initialized to be the original protocol tree, where each node has a description that contains the weight and transcript like in the original tree, and an additional range field which is initially left empty. We set the range of the root, denoted r, to be full range $R(r) = [0^\ell, 1^\ell]$. If the weight of r is zero we terminate the process. Otherwise, we invoke the Build_Tree procedure on r.

The main procedure: Build_Tree

If u is a leaf, the procedure returns without updating the global tree T. Otherwise, the Build_Tree procedure invokes two sub-procedures, Build_Heavy(u) and Build_Interval(u), in order to identify and update the children of u in T. Finally, the Build_Tree procedure is invoked recursively on all the children of u in T.

Build_Heavy

The Build_Heavy procedure identifies the **heavy descendants** of u in $T(u)$, which are descendants of large weight that have no children of large weight, and modifies the tree by lifting them to become **heavy children** of u.

▶ **Definition 5** (Heavy descendants)**.** We call v a heavy descendant of u if v is a descendant of u in T and the following conditions hold:

1. $w(v) \geq \frac{w(u)}{n}$
2. Either v is a leaf, or for each child z of v it holds that $w(z) < \frac{w(u)}{n}$.

For each heavy descendant, v, of u we perform the following process:

1. Update v's description: Set the range field of v to be full range $R(v) = [0^\ell, 1^\ell]$.
2. Modify the protocol tree if v is not already a child of u:
   a. Subtract $w(v)$ from the weight of the ancestors of v in $T(u)$, except for u whose weight stays the same.
   b. Move v (along with the subtree rooted at v) to be directly under u.

See Figure 1.
After we finish identifying and moving the heavy children of u we perform a clean up stage where we erase all the nodes in T with weight zero.

Build_Interval(u)

This procedure groups the non-heavy children of u under **interval children**. Denote the range field of u by $R(u) = [s(u), e(u)]$. (Note that $s(u) = 0^\ell$ and $e(u) = 1^\ell$ unless u is an interval node, in which case its range is partial.) We partition the range of u into a sequence

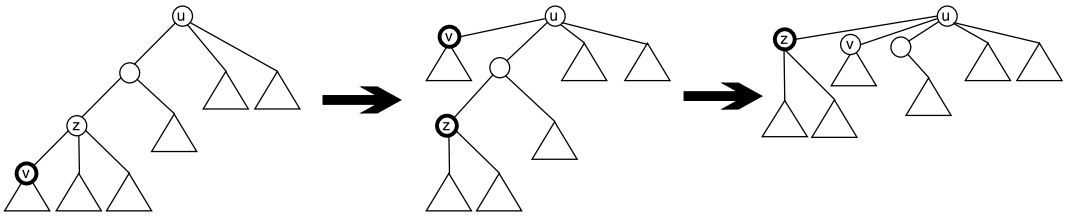

Figure 1 Build_Heavy: In the first step, v is identified as a heavy descendant of u and moved to be a heavy child of u. In the second step, z is identified and moved to be a heavy child of u. The triangles represent subtrees of the original tree.

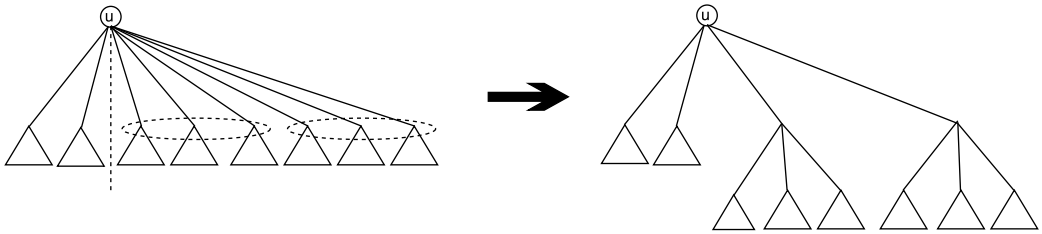

Figure 2 Build_Interval: The left diagram represents the tree before the Build_Interval procedure. The nodes to the left of the dashed line are heavy children of u. The group of nodes inside each dashed circle are united under an interval node. The tree on the right is the result of applying the Build_Interval procedure.

of consecutive intervals, each one representing the range of a new child of u. As long as we have not partitioned all of the range $[s(u), e(u)]$ we perform the following procedure.

1. Determine s', the starting point of the interval child's range: Initially, for the first interval child of u we set $s' = s(u)$. For the next interval children, if the end of the range of the previously created interval child is $\tilde{e}$, then we set $s' = \tilde{e} + 1$.
2. Determine e', the ending point of the interval child's range: For each $e \in \{0,1\}^\ell$, denote by $non\_heavy(s', e)$ the set of children of u in $T(u)$ whose weights are smaller than $\frac{w(u)}{n}$ and their last verifier message α (in the transcript field) is in the range $[s', e]$. Note that when $[s', e] \neq [s(u), e(u)]$ the set $non\_heavy(s', e)$ can be a proper subset of set of non-heavy children of u. We define the weight of the set $non\_heavy(s', e)$, which we denote by $W(s', e)$, as the sum of the weights of nodes in $non\_heavy(s', e)$.
   We set e', to be the minimal $e \in \{0,1\}^\ell$ that satisfies $W(s', e) \geq \frac{w(u)}{n}$. If no such e exists and $W(s', e(u)) > 0$, we set $e' = e(u)$. If $W(s', e(u)) = 0$ there is no need to create another interval child so we return to the Build_Tree procedure. (This guarantees that the weight of an interval child is at least 1).
3. Create a new node v: We set the transcript of v to be like the transcript of u, $\gamma(v) = \gamma(u)$, its range to be $R(v) = [s', e']$ and its weight to be $w(v) = W(s', e')$.
4. Place v in the tree: disconnect u from the nodes in $non\_heavy(s', e')$. Set u as a parent of v and let v be the parent of all nodes that are in $non\_heavy(s', e')$.

See Figure 2. Note that the weight of an interval child of u is at most $\frac{2w(u)}{n}$ and at least $\frac{w(u)}{n}$, except possibly for the last interval child, whose weight is at least 1.

4.3 Properties of the emulation tree

Recall that in the original protocol tree, v was a child of u if and only if $\gamma(v) = \gamma(u)\alpha\beta$ where $\alpha, \beta \in \{0,1\}^\ell$ denote the next verifier message and the prover's response to it. However, in the new emulation tree, E_{P_0,V_0}, this is not the case. Namely, if v is a child of u in E_{P_0,V_0},

then it could be that v is an heavy child of u and hence $\gamma(v) = \gamma(u)\alpha_1\beta_1, \ldots, \alpha_k\beta_k$ for some $\alpha_1, \beta_1, \ldots, \alpha_k, \beta_k \in \{0,1\}^\ell$, or v is an interval child hence $\gamma(v) = \gamma(u)$ and $R(v) \subseteq R(u)$. Nevertheless, the following properties of the final emulation tree E_{P_0,V_0} hold. The proofs can be found in the full version of the paper [11].

▶ **Claim 6** (Node degree). *Each node u in the final emulation tree E_{P_0,V_0} has at most n children.*

▶ **Claim 7** (Weight reduction). *For every node u in the final emulation tree E_{P_0,V_0} the weight of each grandchild of u in E_{P_0,V_0} is at most $\frac{2w(u)}{n}$.*

▶ **Corollary 8** (Corollary to Claim 7). *The height of the final emulation tree E_{P_0,V_0} is $O(r(n)/\log n)$.*

▶ **Claim 9** (Leaves). *The leaves of E_{P_0,V_0} are exactly the leaves of protocol tree T_{P_0,V_0} whose weights are 1.*

5 Public-coin emulation

Next, we describe the strategy of the designated prover P and verifier V in the new ("emulation") protocol. The strategy of the designated prover P for a yes-instance x uses the emulation tree E_{P_0,V_0} of x constructed in the previous section. The prover assists the verifier V in progressing down the emulation tree. On each iteration, the prover provides the descriptions of the children $v_1, \ldots, v_d$ of the current node u, which was sampled in the previous iteration. The verifier preforms validations on the list supplied by the prover (to be detailed below), and then samples one of the children for the next iteration according to its weight. The verifier does not have access to the emulation tree, and its validations consist of structural requirements on the part of the emulation tree seen so far. On the last round the verifier checks that the full transcript, along with the sequence of coin tosses, leads the original verifier V_0 to accept.

One of the structural validations that the verifier makes is that the nodes provided by the prover may be children of u in the emulation tree. For nodes in the original protocol tree, v is a child of u if and only if the transcript of v extends the transcript of u by one pair of messages, and thus v is a descendant of u if and only if the transcript of u is a proper prefix of the transcript of v. For nodes in the emulation tree the situation is more complex. If v is an interval child of u, then the transcript of v equals the transcript of u, and the range of v is a partial range of the range of u. If v is a heavy child of u, then the transcript of u is a proper prefix of the transcript of v. Furthermore, if $\gamma(u) = \alpha_1^u\beta_1^u, \ldots, \alpha_i^u\beta_i^u$, then α_{i+1}^v (the $i+1$ verifier's message in the transcript of v) should be in the range of u.

With these two cases in mind, we define the conditions required of the descriptions of two nodes u and v in order for v to be a descendant (not necessarily a child) of u in the emulation tree. We say that v is a **transcript descendant** of u if these required conditions hold.

▶ **Definition 10** (Transcript Descendant). Denote by u and v nodes in the emulation tree with transcripts $\gamma(u) = \alpha_1^u\beta_1^u, \ldots, \alpha_i^u\beta_i^u$ and $\gamma(v) = \alpha_1^v\beta_1^v \ldots \alpha_j^v\beta_j^v$ and with range field $R(u)$ and $R(v)$, respectively. We say that v is a transcript descendant of u if one of the following conditions hold:

(i) $\gamma(v) = \gamma(u)$ and $R(v) \subseteq R(u)$

(ii) $\gamma(u)$ is a proper prefix of $\gamma(v)$ and $\alpha_{i+1}^v \in R(u)$

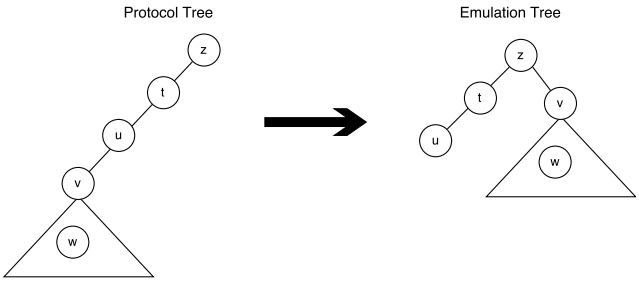

Figure 3 Truncation during Build_Tree. The node v is identified as a heavy descendant of z, so it is moved (along with the subtree that is rooted at it) to be a heavy child of z.

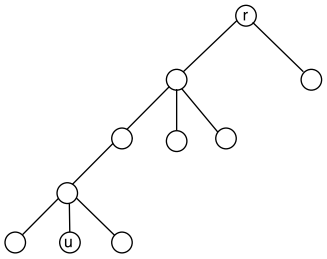

Figure 4 The nodes in the list of seen nodes, S, in the iteration that the input node is u.

Note that it is possible that v is a descendant of u in the emulation tree and the description of u is equal to the description of v. For example, this can happen when v is the only interval child of u.

It is easy to show that for every node u in E_{P_0,V_0}, every descendant of u is a transcript descendant of u. The proof is similar to the proof in the completeness part of the analysis [11, Sec 6].

We state the following claim regarding the transitivity property of transcript descendancy, which we use in the analysis of the emulation.

▶ **Claim 11** (Transitivity). *For nodes u, v and z in the emulation tree such that z is a transcript descendant of v and v is a transcript descendant of u, z is a transcript descendant of u.*

The proof of the claim is given in [11, Apdx A]. It follows by case analysis of the different transcript descendancy types between u, v and z.

The condition of v being a transcript descendant of u is not sufficient to guarantee that v may be a descendant of u in the emulation tree E_{P_0,V_0}. For example, suppose that v was a child of u in T_{P_0,V_0}, and is a heavy descendant of a node z that is an ancestor of u in E_{P_0,V_0}. Then, v becomes a heavy child of z in E_{P_0,V_0}, which means that the subtree rooted at v was truncated and moved up to be a direct descendant of z. Therefore, in order to check if v may be a legal descendant of u in the emulation tree, the verifier needs to check that v does not belong to a part of the tree that was truncated and moved to a different part of the emulation tree (such as nodes v and w in Figure 3). For this reason the verifier keeps a list S of nodes that were seen during the emulation, and updates the list at every iteration with the new nodes seen. Note that the nodes in S, which the verifier sees up to some iteration, are a subtree of the emulation tree that is composed of a path from the root of the tree to the current input node, augmented with the children of the nodes in the path. See Figure 4.

Another structural validation requires the transcripts that the verifier sees during the execution to be consistent with a deterministic prover strategy.

▶ **Definition 12** (Prover consistent)**.** We say that two transcripts $\gamma(u)$ and $\gamma(v)$ are **prover consistent** if the maximal prefix they agree on is either empty or ends with a prover's message. That is, the prover should respond in the same way on the same prefix of the transcripts (i.e., for every j smaller than the length of the shorter transcript, if $(\alpha_1^u\beta_1^u,\ldots,\alpha_j^u)=(\alpha_1^v\beta_1^v,\ldots,\alpha_j^v)$ then $\beta_j^u=\beta_j^v$).

This condition will allow for extracting a prover's strategy for the original protocol from the transcripts in E_{P_0,V_0}, and then to claim that since the original prover cannot fool the verifier with high probability, the new prover cannot either.

Initially, for the first iteration, the transcript of the root r is the empty transcript and the range is full range, $[0^\ell,1^\ell]$. The prover provides the weight of the root r and the verifier checks that the claimed weight is at least $\frac{9}{10}\cdot 2^{r(n)}$. The verifier adds the description of r to the list of seen nodes S. The rest of the first iteration, as well as subsequent iterations, proceed as follows.

▶ **Construction 13** (the ith iteration)**.** On input a non-leaf node u and list S.[5]

1. The prover provides the descriptions of the children $v_1,\ldots,v_d$ of u:

$$(\gamma(v_1),R(v_1),w'(v_1)),\ldots,(\gamma(v_d),R(v_d),w'(v_d))$$

2. The verifier preforms the following validations and rejects if any of them fails:

a. The verifier checks that all nodes are different (according to their descriptions) that is, for each distinct $i,j\in[d]$, if $\gamma(v_i)=\gamma(v_j)$, then $R(v_i)\neq R(v_j)$.

b. The verifier checks that the weights of the children of u sum up to $w'(u)$; that is,

$$w'(u)=\sum_{j=1}^{d}w'(v_j) \tag{2}$$

c. For each $j\in[d]$, the verifier checks that v_j is a transcript descendant of u.

d. For each interval child v_j, the verifier checks that $\gamma(v_j)=\gamma(u)$; that is, if $R(v_j)\neq[0^\ell,1^\ell]$, then $\gamma(v_j)=\gamma(u)$ must hold.

e. For each $j\in[d]$ the verifier checks that v_j is not in a part of the emulation tree that was truncated. (See discussion following Definition 10.) Specifically, for $v\in S(u)$ then if v is a transcript descendant of u, then v_j should not be a transcript descendant of v. For illustration consider Figure 3, where $u,t,z,v\in S$. In that case, the verifier checks that v_j is not a transcript descendant of v (where v is a transcript descendant of u since it was a descendant of u in the original protocol tree).
(Note that it can be that v_j is a transcript descendant of some node in S that is not a transcript descendant of u, and this is not considered a violation. For example, all the nodes are transcript descendants of the root r, which is in S.)

f. The verifier checks that the ranges of all the interval children are disjoint; that is, for every two interval children v_j and v_k, the verifier checks that $R(v_j)\cap R(v_k)=\emptyset$.

g. For each $j\in[d]$ the verifier checks that $\gamma(v_j)$ is **prover consistent** (see Definition 12) with respect to the other transcripts of nodes in S and with regarding to the transcripts of the other children $\gamma(v_k)$, where $k\neq j$.

[5] Because of Step 4, the input node u will always be a non-leaf node.

3. The verifier chooses a child according to the probability distribution J that assigns each $j \in [d]$ probability approximately proportional to $w'(v_j)$ using $O(\log n)$ coin tosses. That is, J is such that

$$\mathbf{Pr}[J = j] \leq \frac{w'(v_j)}{\sum_{i=1}^{d} w'(v_i)} \cdot \left(1 + \frac{1}{n}\right) \tag{5.1}$$

   We can only afford to use $O(\log n)$ public-coins per round, and hence we compromise on sampling each child with probability proportional to $w'(v_j)$, and instead sample with approximate probability. See explanation for approximate sampling in [11, Apdx B].
4. The verifier adds all the children of u to the list S; that is $S \leftarrow S \cup \{v_1, \ldots, v_d\}$. Unless $\gamma(v_j)$ is the complete transcript (which includes the outcome of the coin tosses), the next iteration will start with node v_j and the set S. Otherwise, we proceed to the final checks.

By our conventions, the last message the verifier sends, denoted α_m, contains the outcomes $\rho \in \{0,1\}^{r(n)}$ of the $r(n)$ coins tossed. Thus, if the last node chosen is v, then ρ can be easily extracted from $\gamma(v) = \alpha_1\beta_1, \ldots, \alpha_m\beta_m$. After the last iteration the verifier performs final checks and accepts if all of them hold:

(i) Check that ρ is accepting for $\gamma(v)$ and consistent with it: It checks that $V_0(x, \rho, \beta_1, \ldots, \beta_m) = 1$, and that for every $i = 1, \ldots, m$ it holds that $\alpha_i = V_0(x, \rho, \beta_1, \ldots, \beta_{i-1})$. Note that the verifier needs ρ in order to verify these conditions, so this check can only be done after the last iteration. Also note that if these checks pass then $w(v) = 1$ (rather than $w(v) = 0$).

(ii) Check that $w'(v) = 1$; in other words the prover's last claim should be that the weight of the last node chosen is 1 (and not more than 1).

Clearly, the number of rounds of the emulation is $O(r(n)/\log n)$ because the height of the emulation tree is $O(r(n)/\log n)$, and the prover and verifier proceed one step down the tree on each round. Since the verifier uses $O(\log n)$ public coins on each round, the randomness complexity of the emulation is $O(r(n))$.

References

1 Laszlo Babai. Trading group theory for randomness. In *ACM Symposium on the Theory of Computing*, pages 421–429, 1985.

2 Laszlo Babai and Shlomo Moran. Arthur-merlin games: A randomized proof system and a hierarchy of complexity classes. *Journal of Computer and System Science*, 36:254–276, 1988.

3 Mihir Bellare, Oded Goldreich, and Shafi Goldwasser. Randomness in interactive proofs. *Computational Complexity*, 3:319–354, 1993.

4 Martin Fürer, Oded Goldreich, Yishay Mansour, Michael Sipser, and Stathis Zachos. On completeness and soundness in interactive proof systems. *Advances in Computing Research*, 5:429–442, 1989.

5 Oded Goldreich and Maya Leshkowitz. On emulating interactive proofs with public coins. *Electronic Colloquium on Computational Complexity (ECCC)*, 23:66, 2016.

6 Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems. *Journal of the ACM*, 38(3):691–729, 1991. Preliminary version in *27th FOCS*, 1986.

7 Oded Goldreich, Salil P. Vadhan, and Avi Wigderson. On interactive proofs with a laconic prover. *Computational Complexity*, 11(1-2):1–53, 2002.

8 Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18(1):186–208, 1989. Preliminary version in *17th STOC*, 1985. Earlier versions date to 1982.

9 Shafi Goldwasser and Michael Sipser. Private coins versus public coins in interactive proof systems. *Advances in Computing Research*, 5:73–90, 1989. Extended abstract in *18th STOC*, 1986.

10 Adam R. Klivans and Dieter van Melkebeek. Graph nonisomorphism has subexponential size proofs unless the polynomial-time hierarchy collapses. *SIAM Journal on Computing*, 31(5):1501–1526, 2002.

11 Maya Leshkowitz. Round complexity versus randomness complexity in interactive proofs. *Electronic Colloquium on Computational Complexity (ECCC)*, 24:55, 2017. Full version of the paper.

12 Carsten Lund, Lance Fortnow, Howard J. Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. *Journal of the ACM*, 39(4):859–868, 1992. Extended abstract in *31st FOCS*, 1990.

13 Ronen Shaltiel and Christopher Umans. Low-end uniform hardness versus randomness tradeoffs for AM. *SIAM Journal on Computing*, 39(3):1006–1037, 2009.

14 Adi Shamir. IP = PSPACE. *Journal of the ACM*, 39(4):869–877, 1992. Preliminary version in *31st FOCS*, 1990.