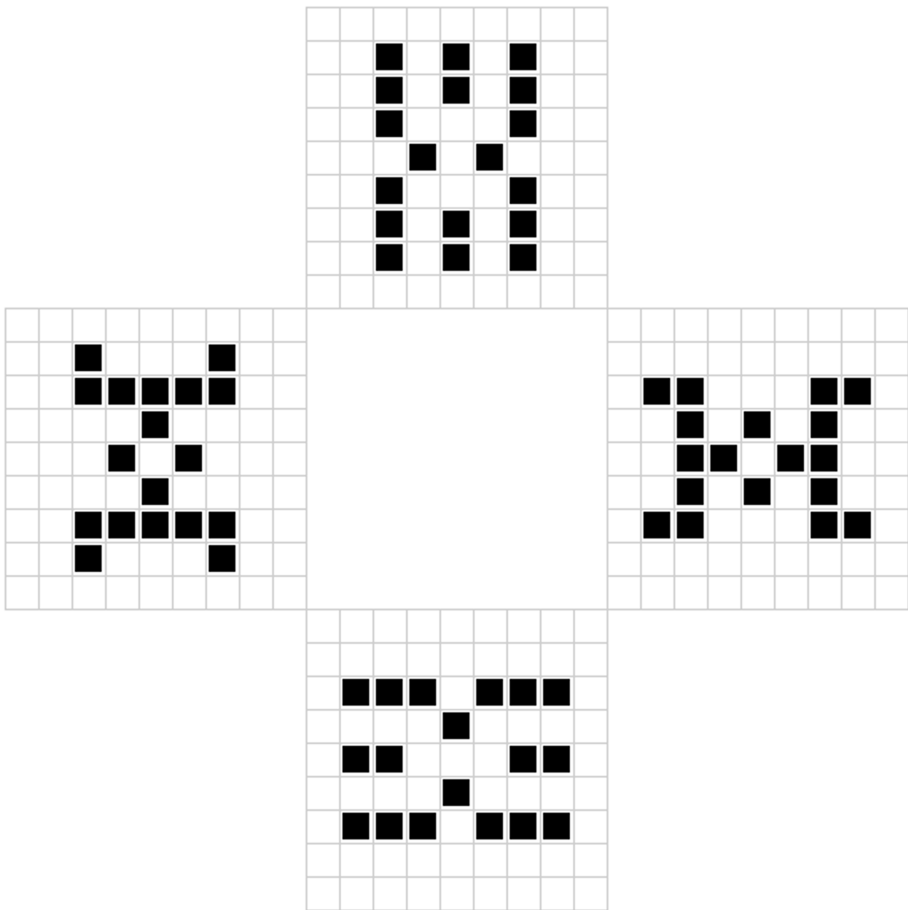
A Medley for Computational Complexity

With applications of Information Theory, Learning Theory,
and Ketan Mulmuley's Parametric Complexity Technique



Bruno Loff

A Medley for Computational Complexity

With applications of Information Theory, Learning Theory,
and Ketan Mulmuley's Parametric Complexity Technique

The cover is a period-4 oscilator for Conway's game of life, called "Monogram". It was discovered by Dean Hickerson.

A Medley for Computational Complexity

With applications of Information Theory, Learning Theory,
and Ketan Mulmuley's Parametric Complexity Technique

Academisch Proefschrift

ter verkrijging van de graad van doctor aan de
Universiteit van Amsterdam
op gezag van de Rector Magnificus
prof.dr. D.C. van den Boom
ten overstaan van een door het college voor
promoties ingestelde commissie, in het openbaar
te verdedigen in de Agnietenkapel
op dinsdag 21 januari 2014, te 10.00 uur

door

Bruno Serra Loff Barreto

geboren te Lissabon, Portugal

Promotor:	Prof. Dr. H. Buhrman
Overige leden:	Dr. P. van Emde Boas
	Prof. Dr. M. Koucký
	Dr. L. Torenvliet
	Prof. Dr. Yde Venema
	Prof. Dr. R. de Wolf

Faculteit der Natuurwetenschappen, Wiskunde en Informatica

to Pedro, Jetty, and Andrea

As seraph onto Poe, so you are to me:
An enchanted island in a stormy sea.

Acknowledgments

This thesis would not have existed without Harry. Of course, it is often the case that a PhD thesis will not come to light without an advisor, but that is not what I mean here. Naturally I acknowledge Harry's stimulus, his questions, his collaboration, and his friendship, which I greatly esteem. However I would like to mention something more fundamental.

Throughout the last five years I have been on the verge of giving up on this thesis at least three times. My depression made it very difficult for me to concentrate, and provided me with a constant influx of pain. Besides the resulting disillusionment with the world around me, I was struck by a profound disbelief in myself, and it felt to me that anyone in their good sense should have judged me inept. Yet Harry never did. And in those crucial moments when this work required an informed belief in my ability to complete it — I owe it entirely to Harry for providing. You're reading this, so I guess he was right!

I've learned the most and had the most fun when collaborating with others, and I am thankful to all those with whom I have exchanged ideas. Because this is particularly satisfying when there is a tangible outcome, my co-authors deserve special mention: thank you Eric Allender, Joshua Brody, Lance Fortnow, Luke Friedman, John Hitchcock, Michal Koucký, Florian Speelman, Leen Torenvliet and Nikolay Vereshchagin.

I am further indebted to the members of this thesis' committee (p. iv above), both for their participation, and for their careful reading and helpful comments.

None of this would have been possible without financial support. The Portuguese science foundation, FCT, has paid for my sustenance for the greater part of my stay here, and during this final year I was paid by CWI: I truly appreciate this. I am also grateful to Peter Bro Milterson and his mathematical computer science group in Aarhus for funding my visits there on two occasions.

Amsterdam Bruno Loff
December, 2013.

Abstract

This thesis contains four parts. Each part studies a topic within computational complexity by applying techniques from other fields in theoretical computer science.

In **Chapter 1** we will use Kolmogorov complexity to study probabilistic polynomial-time algorithms. Let R denote the set of Kolmogorov-random strings, which are those strings x whose Kolmogorov complexity $K(x)$ is at least as large as their length $|x|$. There are two main results. First, we show that any bounded-error probabilistic polynomial-time algorithm can be simulated by a deterministic polynomial-time algorithm that is allowed to make non-adaptive queries to R. Second, we show that for a time-bounded analogue of R (defined using time-bounded Kolmogorov complexity), it holds that any polynomial-time algorithm that makes non-adaptive queries to R can be simulated both in polynomial space and by circuits of polynomial size.

This indicates that we are near to an alternative characterization of probabilistic polynomial-time as being exactly deterministic polynomial-time with non-adaptive queries to R. Such characterizations ultimately aim at using techniques from Kolmogorov complexity and computability to study the relationship between different complexity classes. As can be expected, the proofs in this chapter make essential use of such techniques.

In **Chapter 2** we make an effort at extending Mahaney's theorem [74] to more general reductions, or — seen another way — strengthening the Karp-Lipton theorem [64] to a stronger collapse of the polynomial-time hierarchy. Mahaney's theorem states that if SAT is m-reducible to a sparse set, then $\mathrm{P} = \mathrm{NP}$, and the Karp-Lipton theorem (more precisely, the strengthened version of Cai [40]) says that if SAT is Turing-reducible to a sparse set, then $\mathrm{PH} \subseteq \mathrm{ZPP}^{\mathrm{NP}}$.

We prove that if a class of functions $\mathcal{C}$ has a polynomial-time learning algorithm in Angluin's bounded error learning model, then if SAT is m-reducible to $\mathcal{C}$, it follows that $\mathrm{PH} \subseteq \mathrm{P}^{\mathrm{NP}}$.

Then from the existence of such an algorithm for linear-threshold functions, we conclude that if SAT is m-reducible to a linear-threshold function, then $\mathrm{PH} \subseteq \mathrm{P}^{\mathrm{NP}}$. It will be seen that both disjunctive and majority truth-table (non-adaptive) reductions to sparse sets are a special case of m-reductions to linear-threshold functions, and hence our results hold also for these kinds of

reductions.

We also prove a somewhat stronger result of independent interest. For such a class of functions $\mathcal{C}$, it holds that if SAT m-reduces to $\mathcal{C}$, then we can answer any number of SAT-queries of length n by asking only n (larger) queries to SAT.

There are two main results in **Chapter 3**. First, we prove a more refined NP-hardness result for knapsack and related problems. We will construct a reduction from the satisfiability of fan-in-2 circuits of size S with k input bits to an instance of the subset-sum problem having bit-length $O(S+k)$. A corollary of this is a simple proof that there is no approximation algorithm for the knapsack problem which gives a better-than-inverse-polynomial approximation ratio, unless the exponential-time hypothesis of Impagliazzo and Paturi [55] fails to hold.

Secondly, we will use the technique we just developed, together with Ketan Mulmuley's parametric complexity technique, in order to prove an unconditional lower bound in Mulmuley's parallel semi-algebraic PRAM model [77]. We will show that, in that model, there is no algorithm for solving the knapsack problem in time $o(x^{1/4})$ using $2^{o(n^{1/4})}$ processors, even when the bit-length of the weights is restricted to n. The significance of this result follows from the fact that pretty much every known parallel algorithm can be implemented in this model.

In **Chapter 4**, we turn to communication complexity and information complexity [28]. We prove several theorems: (1) we show a "Reverse Newman's Theorem", stating that a private-coin q-round protocol that reveals I bits of information can be simulated by a public-coin q-round protocol that reveals $I+\tilde{O}(q)$ bits of information; (2) we show that public-coin protocols that reveal I bits of information can be simulated by protocols that communicate only $\tilde{O}(I)$ bits (but possibly use many more rounds); (3) we prove a constant-round two-way variant of the Slepian–Wolf theorem, and use it to show that q-round public-coin protocols that reveal I bits of information can be simulated by protocols that communicate only $O(I)+\tilde{O}(q)$ bits on average, and make use of $O(q)$ rounds, also on average; and (4) as a consequence of (1) and (3) we show a direct-sum theorem for bounded-round protocols, which states that, for any function f which needs C bits of (average) communication to be computed by randomized protocols in $O(q)$-average-rounds, computing k copies of the function using a q-round protocol requires $\Omega(kC)-\tilde{O}(q)$ bits of communication.

Samenvatting

Dit proefschrift is opgebouwd uit vier delen. In elk deel wordt een onderwerp binnen de computationele complexiteitstheorie bestudeerd door het toepassen van technieken uit andere gebieden van de theoretische informatica.

In **Hoofdstuk 1** zullen wij Kolmogorov-complexiteit gebruiken om probabilistische polynomiale-tijd algoritmen te bestuderen. Neem R als de verzameling van Kolmogorov-willekeurige binaire sequenties, dat zijn die sequenties x waarvan de Kolmogorov complexiteit $K(x)$ minstens zo groot is als hun lengte $|x|$. De twee belangrijkste resultaten in dit hoofdstuk zijn de volgende. In de eerste plaats laten we zien dat elk probabilistisch polynomiale-tijd algoritme met begrensde foutkans gesimuleerd kan worden door een deterministisch polynomiale-tijd algoritme dat niet-adaptieve vragen aan R kan stellen. Ten tweede laten we zien dat wanneer we een tijdsbegrensde variant van R gebruiken (gedefinieerd met behulp van tijdsbegrensde Kolmogorov-complexiteit), elk polynomiale-tijd algoritme dat niet-adaptieve vragen stelt aan R gesimuleerd kan worden door zowel Turing machines met polynomiale ruimte als door circuits van polynomiale grootte.

Deze resultaten geven aan dat we in de buurt komen van een alternatieve beschrijving van probabilistische polynomiale tijd, namelijk precies als deterministische polynomiale-tijd berekeningen met niet-adaptieve vragen naar R. Zulke beschrijvingen mikken uiteindelijk op het gebruik van technieken uit de Kolmogorov-complexiteit en berenbaarheidstheorie om de onderlinge verhoudingen van verschillende complexiteitsklassen te bestuderen. Zoals te verwachten maken de bewijzen in dit hoofdstuk sterk gebruik van zulke technieken.

In **Hoofdstuk 2** doen we een aanzet om Mahaney's stelling [74] uit te breiden naar algemenere reducties, of — anders bekeken — de ineenstorting van de polynomiale-tijd hierarchie in de Karp-Lipton stelling [64] te versterken. Mahaney's stelling zegt dat als SAT m-reduceerbaar is naar een ijle verzameling, dan volgt P = NP, en de Karp-Lipton stelling (of preciezer, de sterkere variant van Cai [40]) vertelt ons dat gegeven dat SAT Turing-reduceerbaar is naar een ijle verzameling, er dan geldt dat $\mathrm{PH} \subseteq \mathrm{ZPP}^{\mathrm{NP}}$.

We bewijzen dat wanneer een functie-klasse $\mathcal{C}$ een polynomiale-tijd leeralgoritme heeft in Angluin's *bounded error learning* model, dan krijgen we SAT is m-reduceerbaar naar $\mathcal{C}$ impliceert dat $\mathrm{PH} \subseteq \mathrm{P}^{\mathrm{NP}}$.

Vanuit het bestaan van zo een algoritme voor linear-threshold functies, con-

cluderen we dat als SAT m-reduceerbaar is naar een linear-threshold functie, dan volgt PH $\subseteq$ P^{NP}. Het zal te zien dat dan zowel disjunctieve als majority truth-table (niet-adaptieve) reducties naar ijle verzamelingen een specifieke instantie van m-reducties naar linear-threshold functies zijn, en vandaar gelden onze resultaten ook voor deze typen reducties.

Ook bewijzen we het volgende sterkere resultaat, wat van zelfstandig belang is. Voor een klasse functies $\mathcal{C}$ geldt het dat wanneer SAT m-reduceerbaar is naar $\mathcal{C}$, dan kunnen we elk aantal SAT-vragen van lengte n beantwoorden door slechts n (grotere) vragen aan SAT te stellen.

De twee voornaamste resultaten te vinden in **Hoofdstuk 3** zijn de volgende. Ten eerste bewijzen we een verfijnder NP-moeilijkheidsresultaat voor het knapsakprobleem en gerelateerde problemen. We construeren een reductie vanaf vervulbaarheid van fan-in-2 circuits van grootte S met k invoer-bits naar een instantie van het subset-sum probleem met bit-lengte $O(S+k)$. Als corollarium hiervan krijgen we een simpel bewijs dat er geen benaderings-algoritme voor het knapzakprobleem is dat een beter-dan-reciproke-polynome benaderingsratio geeft, tenzij de exponentiële-tijd hypothese van Impagliazzo en Paturi [55] niet waar is.

Ten tweede zullen we de zojuist ontwikkelde methode gebruiken, samen met Ketan Mulmuley's parametrische complexiteitstechniek, om een onvoorwaardelijke ondergrens te bewijzen in Mulmuley's parallelle semi-algebraïsche PRAM model [77]. We zullen laten zien dat, in het betreffende model, er geen algoritme bestaat wat het knapzakprobleem oplost in tijd $o(x^{1/4})$, gebruik makend van $2^{o(n^{1/4})}$ processoren, zelfs wanneer de bit-lengte van de gewichten begrensd is tot n. De significantie van dit resultaat komt voort uit het feit dat vrijwel elk bekend parallel algoritme geïmplementeerd kan worden in dit model.

In **Hoofdstuk 4** richten we ons op communicatie-complexiteit en informatie-complexiteit [28]. We bewijzen verscheidene stellingen: (1) we geven een "Reverse Newman's Theorem" (omgekeerde Newman's stelling), deze stelt dat een private-coin q-ronde protocol welke I bits aan informatie openbaart, gesimuleerd kan worden door een public-coin q-ronde protocol welke $I + \tilde{O}(q)$ bits aan informatie openbaart; (2) we laten zien dat public-coin protocollen welke I bits aan informatie tonen gesimuleerd kunnen worden door procollen die slechts $\tilde{O}(I)$ bits communiceren (in mogelijk meer rondes); (3) we bewijzen een constante-ronde two-way variant van de Slepian–Wolf stelling, en gebruiken deze om te laten zien dat q-ronde public-coin protocollen die I bits informatie openbaren, gesimuleerd kunnen worden door protocollen die slechts gemiddeld $O(I) + \tilde{O}(q)$ bits communiceren, en gemiddeld $O(q)$ rondes gebruiken; en (4) gebruik makend van (1) en (3) laten we een direct-sum stelling zien voor protocollen met een begrensd aantal rondes, welke zegt dat, voor elke functie f die gemiddeld C bits communicatie nodig heeft om berekend te worden door gerandomiseerde protocollen in gemiddeld $O(q)$ rondes, er $\Omega(kC) - \tilde{O}(q)$ bits aan communicatie nodig zijn om k kopieën van deze functie te berekenen met een protocol wat q rondes gebruikt.

Contents

There is no index in this thesis.
To do a keyword search, please use the electronic version, found at:
`https://www.dropbox.com/s/54gb8giopxo0y4r/thesis.loff.pdf`

Introduction for a general audience

Computational complexity, also known as complexity theory, is the study of how much of a given resource is necessary to solve a given mathematical problem. An example that everyone is likely to know is the following: suppose we wish to multiply two integers x and y. We are given the digits of x and y in decimal notation — suppose both x and y have n digits — and we wish to obtain the digits of the number $z = x \times y$. The typical primary-school multiplication method works as follows: we place the number x on top of y, aligning the digits to the right, and then we proceed by multiplying the rightmost digit y_1 of y with x, and writing it down, then multiplying the second rightmost digit y_2 of y with x, and writing it below the previous number while shifting all its digits one position to the left, and so on, for all n digits of y; this gives us n numbers $z^{(1)}, \ldots, z^{(n)}$, where each $z^{(i)} = y_i \times x$ is the i-th digit of y multiplied by x, a number which could have up to $n+1$ digits. To end the multiplication, we sum all of the $z^{(i)}$'s, and this gives us the digits of $z = x \times y$. When we apply this method using paper divided into squares, like in a child's arithmetic book, we get something that looks like this:

					x_n	$\ldots$	x_1	
				$\times$	y_n	$\ldots$	y_1	
			$z^{(1)}_{n+1}$	$z^{(1)}_n$	$\ldots$	$z^{(1)}_1$		
		$z^{(2)}_{n+1}$	$z^{(2)}_n$	$\ldots$	$z^{(2)}_1$			
	$\iddots$	$\iddots$	$\iddots$	$\iddots$				
$+$	$z^{(n)}_{n+1}$	$z^{(n)}_n$	$\ldots$	$z^{(n)}_1$				
	z_{2n}	z_{2n-1}	$\ldots$	$\ldots$	$\ldots$	z_1		

Now let us look at the following resource: how many squares are we using in order to multiply the two numbers? In our tally, let us ignore the squares that we absolutely must use, namely the squares that hold the digits of x, the digits of y, and those for the result $x \times y$.[1] How many additional squares do we need to use?

[1] We do this because we are interested in comparing different methods for multiplication, with respect to the use of squares... but any method for multiplication is forced to use at least those squares required to write the input and the output. Hence counting these squares would be superfluous.

For instance, the primary-school method which we have just seen uses $n+1$ squares for each $z^{(i)}$, and there are n such $z^{(i)}$. So, in total, the number of squares used by this method is $n \times (n+1) = n^2 + n$.

For the problem of multiplying two numbers, and for this particular resource, a complexity theorist is primarily interested in answering the following question: What is the smallest number of squares that we need to use, in order multiply two n-digit numbers x and y? The answer to this question depends on n, and is given by a function $f(n)$, which a complexity theorist would call *the space-complexity of multiplication*. We have just shown that $f(n) \leq n^2 + n$. But this is not the minimum! If we use a slightly more clever method, we can show that $f(n) \leq 2n + 4$ (can the reader figure out how? the method requires the use of an erasor, in order to re-use the squares). The best method we know achieves $f(n) \leq c \log n$ for some fixed constant c that does not depend on n,[2] and it can be shown that this is the best possible[3].

Other problems can be studied. For instance, how much space do we need in order to find the shortest path between two cities on a map having n cities in total? How much space do we need to compute the number of different paths that can be taken between two cities? Other resources can be studied as well, such as time, or communication needed to solve a given problem. Interaction between resources is also interesting: If we are allowed to use more space than the strict minimum, can we do with less time? If we are given many computers to do calculations in parallel, can we solve the problem using less time? Given a method to *recognize* a solution to a problem using little time, can we devise a procedure that *finds* such a solution in a comparable amount of time?[4] ... and so on.

This thesis is a collection of answers to a few questions in computational complexity. We use a varied palette of mathematical techniques in order to answer these questions. The general theme is that of solving a problem in computational complexity by applying tools from a different field in theoretical computer science, or at least from a different context within computational complexity.

Chapter 1. In this chapter, we use techniques from Kolmogorov Complexity and Computability in order to study the time complexity of randomized algorithms.

Randomized algorithms are algorithms that are allowed to toss a fair coin during their execution, and have their behavior depend on the outcome of these coin tosses. Such an algorithm is said to solve some problem if it

[2] See wikipedia [96] for the algorithm. The quantity $\log n$ is approximately the number of digits needed to write the number n in decimal notation. The number n itself is the number of digits needed to write x and y.

[3] This is proven by means of a pumping lemma, as in [70, Lemma 3].

[4] For instance, it is easy to quickly recognize if a filled sudoku puzzle on n^2-by-n^2 boards of n-by-n blocks is correct, just by going through each line, each row and each block making sure that there are no repeated numbers (this takes time roughly n^4). But is it possible to actually solve such a sudoku puzzle — find a solution to a partially filled sudoku — in time n^c for some constant c? Answering this question amounts to solving the famous P versus NP problem.

gives the correct answer with high (99%) probability. A normal (deterministic, non-randomized) algorithm can be thought of as a randomized algorithm that always gives the correct answer (with 100% probability), so, in principle, it could be the case that randomized algorithms can solve problems faster than deterministic algorithms, and whether this is the case or not is a longstanding open question in computational complexity.

Kolmogorov Complexity measures how complex a string of symbols is by how large a program to output said string needs to be. A string of symbols x is called *Kolmogorov-random* if there is no program that outputs x and has a length smaller than the length of x. Computability is the study of what can and cannot be solved by computers, regardless of the amount of resources used, and an example of an uncomputable problem is that of knowing whether a given string is Kolmogorov-random. We give some evidence showing that a problem can be solved by a randomized algorithm in polynomial-time if and only if it can be solved by a *deterministic* polynomial-time algorithm that has an unusual extra ability: the ability to know which strings have high Kolmogorov complexity, among a list of strings which it produces after seeing the input (for a precise and technical description of what this means see the technical overview, and the introductory section of this chapter).

Chapter 2. In this chapter we use computational learning theory to prove some results on circuit-size complexity.

Circuit-size complexity measures how big a circuit must be in order to solve a given problem.[5] It is easy to show that most functions are hard to compute by small circuits, and it is widely believed that certain natural problems are hard to solve by small circuits. However, thus far there is no known natural example of such a hard problem, and coming up with such an example — and proving that it cannot be computed by small circuits — is regarded as the holy grail of the field. Complexity theorists have tried to circumvent this difficult problem by showing conditional results, of the form "if problem X can be solved by small circuits, then such and such unlikely consequence must follow"; such results can be thought of as evidence (but not proof) that problem X cannot be solved by small circuits. The result presented in this chapter is a conditional result of this form.

The proof makes use of computational learning theory. This field studies when and how computer programs can be taught to classify data. Typically, the learning algorithm is fed a number examples (x, y) with $y = f(x)$, where f belongs to a restricted class $\mathcal{F}$ of functions, and has to learn which function f was used in generating them. If this can be done efficiently (for instance, if the number of counter-examples needed is small, and/or if the learning algorithm is a polynomial-time algorithm),

[5] By circuit we mean a boolean circuit, which is a mathematical model of the digital logic circuits used in modern computer hardware. For a precise description see `http://en.wikipedia.org/wiki/Boolean_circuit`. We are interested in the size of circuits because if the circuit is too big, then building it will be impossible, or too expensive.

then we say that $\mathcal{F}$ is "learnable". In this chapter we show that if $\mathcal{F}$ is "learnable," then certain natural problems are unlikely to be solvable by certain kinds of circuits that make use of functions from $\mathcal{F}$ (again, see the technical overview for precision).

Chapter 3. Here we use a mix of techniques to show that a problem — called the knapsack problem — cannot be solved by certain kinds of circuits, i.e., that it has high complexity for a certain model of computation.

The model of computation in question is Ketan Mulmuley's parallel PRAM without bit operations. In this model there are several processors executing instructions on a shared memory, which is somewhat similar to a modern GPU (a Graphics Processing Unit, used in modern graphics boards). In some ways, the model is much more powerful than a modern GPU, because it is a multiple-instruction multiple-data (MIMD) model where access to shared memory is instantaneous and conflict-free, and in other ways it is less powerful, because the model is not allowed to operate on the individual bits of the memory, and instead is only allowed to treat numbers stored in memory as self-contained units to which the processors apply given operations (such as addition and multiplication).

The knapsack problem is the problem of, when given a set of n (unbreakable) bars of gold of various sizes, having weights $w_1, \ldots, w_n$, determine the maximum amount of gold that we can carry in a bag which can hold at most W units of weight.

We show that this problem cannot be solved with less than $n^{1/4}$ time and $2^{n^{1/4}}$ processors, even when the given numbers have no more than n bits. Because there is no known parallel algorithm that makes crucial use of bit-access, we believe that our impossibility result also holds for general parallel algorithms.

Chapter 4. In this chapter we use a mix of Information Theory and combinatorial properties of random graphs in order to transform private-coin protocols into public coin protocols that reveal the same amount of information, and prove new direct-sum results in communication complexity.

Communication complexity concerns itself with the following scenario: two parties — customarily called Alice and Bob — are each given two strings of length n as input — respectively x and y — and wish to compute a joint function of this input, i.e., they wish to know $f(x, y)$, for some given f. The function f may depend on both inputs, and so Alice and Bob are required to communicate with each other, which they do according to a predefined set of rules called a *protocol*. We then define the *communication complexity* of f (for length n) as the minimum number of bits of communication required for any protocol that allows the two parties to compute f (on every input of length n). This scenario is ubiquitous, we can think of Alice and Bob as two computers communicating over a network, or as two pieces of hardware communicating over a memory bus, *etc.*

A natural question in this setting is the following *direct-sum question*: suppose that the communication complexity of f is C, i.e., that I need to communicate C bits in order to jointly compute f on all inputs of length n, then what is the communication complexity of computing k copies of f simultaneously? Is it necessarily at least kC, or can we do something smarter?

In a related subject, called "information complexity," we have the same scenario, Alice and Bob wanting to compute a function of their joint input, but now instead of measuring the amount of bits that they need to exchange, we measure the amount of information that those bits reveal about their input. It could happen, for instance, that Alice's part in the protocol requires her to send to Bob the outcome of many random coin tosses, and in this case Bob will receive many bits, but they are completely uncorrelated with Alice's input and hence reveal no information about it; this is an example of (part of) a protocol that has high communication but reveals no information. The *information complexity* of a function f is the minimum number of bits of information that the parties must necessarily reveal to each other about their inputs, when executing any protocol that computes f.

At the meeting of these two subjects, we arrive at the following *protocol compression question*: can we take a protocol that reveals little information about the player's inputs and convert it ("compress it") into a protocol that uses little communication?

When Alice and Bob are allowed to make private coin tosses, it sometimes happens that they are able to use the outcome of these tosses to somehow "obfuscate" their inputs and compute f without revealing a lot of information. However, if they are forced to share the outcome of their coin tosses, it is not clear how to avoid revealing more information. In the literature on the subject, the following question had been asked [58]: whether or not private coin tosses confer any advantage over public coin tosses in this setting.

We will show essentially that these last two questions are intimately related: modulo certain insignificant factors, protocol compression is possible if and only if any private-coin protocol can be simulated by a public-coin protocol that reveals no additional information. We then show that such a public-coin simulation is possible, in the case of bounded-round protocols, and as a consequence we prove a direct-sum property for the communication complexity of bounded-round protocols.

Chapter 1

Power from the set of random strings

The Kolmogorov complexity $K(x)$ of a string x is (roughly) the length of the smallest program that generates x. If x has length n and cannot be generated by a program of smaller length, we take this to mean that x has no discernible pattern, and call x a *Kolmogorov-random string* (or just *random*, for short). In this chapter we are interested in the set of Kolmogorov-random strings:

$$R_K = \{x \in \{0,1\}^* \mid K(x) \geq |x|\}.$$

It is easy to show that R_K is undecidable by *reductio ad absurdum*; in fact, if some program could decide whether strings are random, then the lexicographically-least random string of a given length n would itself be given by a small program. It is also possible to show that query access to R_K is computationally useful; whereas one intuitively expects that the distribution of random strings should itself be "random", and hence perhaps useless, Martin [75] has actually shown that oracle access to R_K allows us to decide any recursively enumerable set, and Kummer [66] showed that, in fact, disjunctive truth-table access to R_K is enough.

A series of recent papers have extended these results, by showing that R_K can be used as an oracle to increase computational power also for resource-bounded reductions. Allender et al. [5] have shown that PSPACE $\subseteq \mathrm{P}^{R_K}$ and Allender et al. [4] proved that NEXP $\subseteq \mathrm{NP}^{R_K}$. These positive results seem to imply that R_K actually has a deep and rich structure.

On the other hand, a series of negative results tells us that, for resource-bounded reductions, the extra power granted by R_K is more limited than in the computability setting. Hence, for instance, in [4] it was proven that if a decidable set A is polynomial-time disjunctive truth-table reducible to R_K, then A must be decidable in polynomial-time without the help of any oracle, and Allender et al. [8] have shown that if a decidable set A is truth-table reducible to R_K, then A must lie in PSPACE, and if a decidable set A is Turing reducible to R_K, then A must lie in EXPSPACE.[1]

[1] In all cases A must be reducible to R_K for any choice of universal Turing machine in the definition of R_K; see the preliminaries section in this chapter for the precise statements. Contrast with the results of Kummer [66], which show that disjunctive truth-table reductions to R_K with significantly larger time bounds would allow us to decide any recursively enumerable set.

In this chapter we extend this line of research in various ways for the case of truth-table reductions: we show that BPP is truth-table reducible to R_K (§1.4), by first proving that the prefixes of the characteristic string of R_K are hard, in a non-uniform complexity sense (§1.2). We then prove that this holds even for resource-bounded variants of Kolmogorov complexity (also in §1.4), and give some evidence that these inclusions cannot be significantly improved (in §1.5-1.7). The next section includes the full definitions, as well as a more detailed introduction to the contents of this chapter.

The results in this chapter are based on the two papers:

- Harry Buhrman, Lance Fortnow, Michal Koucký, and Bruno Loff. Derandomizing from random strings. In *Proceedings of the 25th CCC*, pages 58–63, 2010.
- Eric Allender, Harry Buhrman, Luke B. Friedman, and Bruno Loff. Reductions to the set of random strings: the resource-bounded case. In *Proceedings of the 37th MFCS*, pages 88–99, 2012.

1.1 Preliminaries and an extended introduction

In this section we present detailed definitions, which will then allow us to give a more complete and thorough overview of the results in this chapter.

We assume the reader is familiar with basic complexity theory [21] and Kolmogorov complexity [69]. We will use $|x|$ to denote the length of a string x. For an integer n and set $A \subseteq \{0,1\}^*$, $A^{=n} = A \cap \{0,1\}^n$, $A^{\leq n} = \cup_{k \leq n} A^{=k}$, and $A_{1:n}$ denotes the first n bits in the characteristic sequence of A. A *time bound* t will be taken to mean a strictly growing time-constructible function; for two time bounds t and t', the notation $t \geq t'$ refers to almost-everywhere point-wise inequality.

1.1.1 Kolmogorov complexity

Let M be a Turing machine. For any string $x \in \{0,1\}^*$, the *Kolmogorov complexity* of x relative to M is:

$$C_M(x) = \min\{\ |p| \mid p \in \{0,1\}^* \text{ and } M(p) = x\}.$$

For a time bound t we define the time-bounded Kolmogorov complexity relative to M:

$$C^t_M(x) = \min\{\ |p| \mid M(p) = x \text{ and } M(p) \text{ uses at most } t(|x|) \text{ steps}\}.$$

Note that unlike traditional computational complexity the time bound is a function of the length of the output of U.

If we restrict these definitions to Turing machines whose domain is a prefix-free set, called *prefix-free* Turing machines, then we obtain the *prefix-free* version of these notions, denoted $K_M(x)$ and $K^t_M(x)$.

We call a (prefix-free) Turing machine U *universal*, if there exists some constant c such that, for any other (prefix-free) Turing machine M and time

bounds t and t', where $t \in \omega((t')^c)$, there exists a constant c_M such that, for all strings x, $C^t_U(x) \leq C^{t'}_M(x) + c_M$ (resp. $K^t_U(x) \leq K^{t'}_M(x) + c_M$) and $C_U(x) \leq C_M(x) + c_M$ (resp. $K_U(x) \leq K_M(x) + c_M$).

In this chapter we will often make definitions and statements about Kolmogorov complexity where we omit the machine M relative to which $C_M(x)$ (or K_M) is defined. This is taken to mean that the definitions are sound and the statements true, whenever M is universal (for regular Turing machines in the case of C, and for prefix-free Turing machines in the case of K). These definitions are to be interpreted as being dependent on the choice of universal M, and statements should then be taken as true for an arbitrary universal (regular or prefix-free) Turing machine. We will make the choice of universal Turing machine explicit whenever this may lead to an ambiguity in the interpretation of the definition or statement.

A string x is said to be *C-random* (or *K-random*) if $C(x) \geq |x|$ (resp. $K(x) \geq |x|$). The set of C-random strings is denoted by

$$R_C = \{x \in \{0,1\}^* \mid C(x) \geq |x|\},$$

and the set of K-random strings R_K is defined in the same way, replacing C with K. Analogous notation is also used for time-bounded Kolmogorov complexity, replacing C with C^t and K with K^t.

1.1.2 Complexity classes and reductions

We will refer to computation with advice. We deviate slightly from the usual definition of computation with advice in the way how we express and measure the running time. For an *advice function* $\alpha : \{0\}^* \to \{0,1\}^*$, we say that $L \in \mathrm{P}/\alpha$ if there is a Turing machine M such that for every $x \in \{0,1\}^*$, $M(x, \alpha(0^{|x|}))$ runs in time polynomial in the length of x and $M(x, \alpha(0^{|x|}))$ accepts iff $x \in L$. We assume that M has random access to its input so the length of $\alpha(0^n)$ can grow faster than any polynomial in n. Similarly, we define EXP/α where we allow the machine M to run in exponential time in the length of x on the input $(x, \alpha(0^{|x|}))$. Furthermore, we are interested not only in Boolean languages (decision problems) but also in functions, so we naturally extend both definitions also to computation of functions with advice. Typically we are interested in the amount of advice that we need for inputs of length n so for $f : \mathbf{N} \to \mathbf{N}$, C/f is the union of all C/α for α satisfying $|\alpha(0^n)| \leq f(n)$.

Let L be a language and C be a language class. We say that $L \in$ i.o.$-C$ if there exists a language $L' \in C$ such that for infinitely many n, $L^{=n} = L'^{=n}$. For a Turing machine M, we say $L \in$ i.o.$-M/f$ if there is some advice function α with $|\alpha(0^n)| \leq f(n)$ such that for infinitely many n, $L^{=n} = \{x \in \{0,1\}^n \mid M(x, \alpha(0^{|x|}))$ accepts$\}$. The definitions are similar for functions instead of languages.

We will refer to probabilistic computation with advice. For $f : \mathbf{N} \to \mathbf{N}$, $BPP//f$ is the class of sets decidable by a bounded-error probabilistic Turing machine with an advice function α such that $|\alpha(0^n)| \leq f(n)$, where the machine is only required to have a bounded-error when given advice from α. More precisely, for every input x, $M(x, \alpha(|x|))$ must correctly decide x with

probability bounded away from 1/2, but, for $\beta \neq \alpha(|x|)$, $M(x, \beta)$ can have any probability distribution.

We say that a set A polynomial-time Turing reduces to a set B, if there is an oracle machine M that on input x runs in polynomial time and with oracle B decides whether $x \in A$. If M asks its questions *non-adaptively*, i.e., each oracle question does not depend on the answers to the previous oracle questions, we say that A polynomial-time truth-table reduces to B ($A \leq^{\mathrm{p}}_{\mathrm{tt}} B$). Moreover, we say that A polynomial-time truth-table disjunctively reduces to B, written $A \leq^{\mathrm{p}}_{\mathrm{dtt}} B$, if the truth-table reduction M outputs as its answer the disjunction of the oracle answers.

The notation P^A (and P^A_{tt}) will be used to refer to the class of sets that are polynomial-time Turing (resp. truth-table) reducible to A.

1.1.3 Complexity classes based on reductions to R_K

We would like to understand what it means for a set to be efficiently reducible to R_K; it is clear that for any universal machine U, very complex sets reduce efficiently to R_{K_U} (starting with R_{K_U} itself), but it is not clear that any of these sets reduce to R_K for an *arbitrary* universal machine. This suggests the following natural definition:

1.1.1. Definition. DTTR is the class of decidable sets A such that $A \leq^p_{tt} R_{K_U}$ holds for every universal Turing machine U.[2]

We will also work with a time-bounded analogue of DTTR:

1.1.2. Definition. TTRT is the class of languages L such that there exists a time bound t_0 (depending on L) such that for all universal prefix-free machines U and for all time bounds $t \geq t_0$, $L \leq^p_{tt} R_{K^t_U}$.

1.1.4 R_K versus R_C

Most inclusions that we prove are valid for both R_C and R_K, meaning that if a set A is reducible to R_K, then the same proof typically shows that it is reducible to R_C, and vice-versa. Analogues of DTTR and TTRT can be defined for the regular (non prefix-free) Kolmogorov complexity. However, we do know how to prove upper bounds for sets reducible to R_C, meaning that, as far as it is known, it could well be that every recursive set is polytime truth-table reducible to R_C. An outstanding open problem from [8] is to prove the same PSPACE upper bound in this setting. So we will generally avoid discussing R_C in this context, to avoid excessive repetition.

However, there is one exception: our hardness results of §1.2 are slightly better for R_C than they are for R_K, and hence our proof of Barzdin's result (in §1.3) only works for R_C. This is why we work with both regular and prefix-free Kolmogorov complexity.

[2] Recent results by Miller et al. [76] indicate that decidability of A is implied by the fact that $A \leq_{tt} R_{K_U}$ for every universal machine U, and hence we could well omit the requirement that A is decidable in the definition of DTTR. Because this is unpublished material, and we are yet to see the proof ourselves, we opted to explicitly state that A is decidable.

1.1.5 A detailed overview of our results

Now that all of the relevant definitions have been given, we can discuss our results in more detail, and also how they fit with previous work. In this chapter we continue to explore the question of which sets can be efficiently reduced to R_K. This question was explored in various earlier papers [5, 4]. It was shown, for instance, that polynomial-time adaptive (Turing) access to R_K enables one to do PSPACE-computations, i.e., PSPACE $\subseteq P^{R_K}$. One of the ingredients in the proof was to show how one may, on input 0^n and in polynomial time with *adaptive* access to R_K, generate a polynomially long Kolmogorov random string.

However, a polynomial-time algorithm with *non-adaptive* access to R_K, on input 0^n, can only produce random strings of length at most $O(\log n)$. Allender, Buhrman, and Koucký [4] begun a systematic study of weaker and non-adaptive access to R_C and R_K. They showed for example that

$$\mathrm{P} = \mathrm{REC} \cap \bigcap_U \{A \mid A \leq^{\mathrm{p}}_{\mathrm{dtt}} R_{K_U}\}.$$

This result and the fact that with non-adaptive access to R_K in general only logarithmically small random strings can be found, seem to suggest that adaptive access to R_K is needed to obtain some useful gain.

The first result in this chapter proves that this intuition is misguided: We show that the characteristic sequence formed by the strings of length $c \log n$, $R_K^{=c \log n}$, itself a string of length n^c, is hard in a non-uniform complexity sense. More precisely, we will show (in §1.2) that for every time bound t,

$$R_K \notin \text{i.o.-DTIME}(t)/2^{n - \log n}.$$

This will allow us to use (in §1.4) the hardness versus randomness framework of Impagliazzo and Wigderson [56] to construct a pseudo-random generator for BPP that uses R_K as an oracle, and conclude that BPP $\subseteq$ DTTR. The proof will also go through for time-bounded Kolmogorov complexity, for sufficiently high time bounds, and BPP $\subseteq$ TTRT will also follow.

This non-uniform hardness stands in stark contrast with the case of algorithms that are not time-bounded, for in that case only n bits of advice are necessary to decide R_K.[3]

We will also prove a tighter result for R_C in §1.2, namely that for every time bound t, there is a constant c such that $R_C \notin \text{i.o.-DTIME}(t)/2^{n-c}$. As a consequence we obtain (in §1.3) an alternative proof of a result due to Barzdin [25], showing the existence of a r.e. set A such that for all time bounds t, there exists c such that

$$K^{t(n)}(A_{1:n} \mid n) \geq n/c. \tag{1.1}$$

holds almost everywhere. We simply take for A the complement of R_C. We also show that the dependence of c on t is necessary to get (1.1) almost everywhere, but that there is a single c such that (1.1) holds infinitely often, for any t.

[3] The n bits tell us how many strings of length n are not random, and given this we can decide R_K simply by enumerating all non-random strings in a dovetail fashion. It can then be shown that such an algorithm, on this particular advice, has no computable time bound.

Next we try to establish whether BPP can be characterized as the class of sets that non-adaptively reduce to R_K (in polynomial time). Allender, Friedman, and Gasarch [8] have shown that DTTR $\subseteq$ PSPACE, from which we may conclude that DTTR is a legitimate complexity class lying somewhere between BPP and PSPACE. We will present some evidence suggesting that DTTR sits closer to BPP (and hence P) than it does to PSPACE.

First of all, notice that we can view the truth-table reduction to R_K as a computation with advice of $K^{t(n)}$-complexity $\Omega(n)$. We will show (in §1.5) that for sets in EXP and $t(n) \in 2^{n^{\Omega(1)}}$, polynomial-time computation with polynomial (or exponential) size advice of $K^{t(n)}$ complexity $n - O(\log n)$ (resp. $n - O(\log \log n)$) can be simulated by bounded-error probabilistic machine with almost linear-size advice. For paddable sets that are complete for NP, $\mathrm{P}^{\#\mathrm{P}}$, PSPACE, or EXP we may even dispense linear-size advice. Hence, advice of very high $K^{t(n)}$ complexity is no better than a truly random string. However, $n - O(\log n)$ is much higher than $\Omega(n)$, and hence this result merely suggests, but does not prove, that DTTR lies close to BPP.

Secondly, we will show (in §1.6-1.7) that:

$$\mathrm{TTRT} \subseteq \mathrm{P}/\mathit{poly} \cap \mathrm{PSPACE}/O(1).$$

The PSPACE/$O(1)$ inclusion follows from an adaptation of the results of [8] to the case of resource-bounded Kolmogorov complexity, and the P/*poly* inclusion follows from a technique which cannot be applied to the resource-unbounded Kolmogorov complexity (we will explain why in §1.6). Since there is no known natural complexity class within P/*poly* $\cap$ PSPACE/$O(1)$ other than BPP/$O(1)$, this result suggests that TTRT (and hence perhaps DTTR) lies closer to BPP than to PSPACE.[4]

1.1.6 Basic propositions for later use

Let us begin by proving that there are many random strings.

1.1.3. Proposition. *There exists some constant d, such that each of the sets $R_C^{=n}, R_{C^t}^{=n}, R_K^{=n}$ and $R_{K^t}^{=n}$ has at least $2^n/d$ strings, for any choice of t and for n sufficiently large.*

Proof. We prove this for the unbounded-time variants. The respective time-bounded cases follow, because if a string requires a (prefix-free) program of size ℓ to be printed, in particular it requires a (prefix-free) program of size ℓ to be printed in bounded time t.

Let d' be the size of a self-delimiting program which takes as input a number s of length $n - d' - 1$, enumerates the first s strings of length n that are output by programs of length less than n, and outputs the first string of lenght n not in that enumeration. The number s cannot equal the number N of non-random strings of length n, otherwise the program would output a random string with a total description length $n - 1$. Hence $N \geq 2^{n-d'-1}$, and we can set $d = 2^{d'+1}$.

[4] It is possible to define TTRT in such a way that the $O(1)$ advice can be removed, but this requires discussing a subtle technical point about the notion of efficient universal Turing machine, which we will avoid doing in this thesis. See [6, §5] for the result.

The proof for $R_K^{=n}$ is simpler and grants us $d = 2$. Suppose that more than half of the strings of length n were non-random, for every n in some infinite set S. Then for each of these lengths we would have 2^{n-1} strings of complexity at most $n - 1$. We could then derive

$$\sum_{n\in S}\sum_{z\in\{0,1\}^n} 2^{-K(z)} \geq \sum_{n\in S} 2^{n-1}\cdot 2^{-n+1} = \infty,$$

which would contradict the Kraft inequality [69, Theorem 1.11.1]. ■

1.1.4. Proposition. *For any machine M and $t'(|x|) > 2^{|x|}t(|x|)$, the query $x \in R_{K_M^t}$? can be computed in time t'.*

Proof. Simulate the machine M on every string of length less than $|x|$ for $t(|x|)$ steps. Because there are fewer than $2^{|x|}$ such strings, the bound follows. ■

1.1.5. Proposition. *Let $L \leq_{tt}^p R_{K_U^t}$ for some time-bound t. Then there exists a constant k such that the language L can be computed in $t_L(n) = 2^{n^k}t(n^k)$ time.*

Proof. Let M be a machine that computes L by running the polynomial-time truth-table reduction from L to $R_{K_U^t}$, and computing by brute-force the answer to any queries from the reduction. Using Proposition 1.1.4, we have that for large enough k, M runs in at most $t_L(n) = 2^{n^k}t(n^k)$ time, so L is decidable within this time-bound. ■

It is the ability to compute R_{K^t} for short strings that makes the time-bounded case different from the ordinary case. This will be seen in proofs throughout the paper.

1.2 High circuit complexity of R_C and R_K

In this section we prove that the characteristic sequences of R_C and R_K have high circuit complexity almost everywhere. We will first prove the following lemma.

1.2.1. Lemma. *For every total Turing machine M there is a constant c_M such that R_C is not in i.o.-$M/2^{n-c_M}$.*

By *total* Turing machine we mean a Turing machine that halts on every input. There is a (non-total) Turing machine M such that R_C is in $M/n + 1$ where the advice is the number of strings in $R_C^{=n}$. With this advice, M can find all the non-random strings of length n, but will not halt if the advice underestimates the number of random strings.

Proof of Lemma 1.2.1. Suppose we have a total Turing machine M, and that, for some advice α of length k, $x \in R_C^{=n} \iff (x, \alpha) \in L(M)$ for infinitely many lengths n.

For strings β of length at most k, let $R_\beta = \{x \in \Sigma^n \mid (x, \beta) \in L(M)\}$. By Proposition 1.1.3, for some integer $d > 0$, $|R_\alpha| \geq 2^n/d$. So we know that if $|R_\beta| < 2^n/d$ then $\beta \neq \alpha$. We call β *good* if $|R_\beta| \geq 2^n/d$.

Fix a good β and choose $x_1, \ldots, x_m$ at random. The probability that all the x_i are not in R_β is at most $(1 - 1/d)^m < 2^{-m/d}$. So for $m = d(k+1)$, we have $2^{-m/d} = 2^{-k-1}$, and thus we can find a sequence $x_1, \ldots, x_m$ such that every good β will have an $x_i \in R_\beta$ for some i. This also means $x_i \in R_\alpha$ for some i, so one of the x_i's is random.

Now choose $c_M = \ell + 2$, where $\ell = |P|$ is the length of a self-delimiting encoding of a program P which:

1. given as input an index i padded to length $n - \ell - 1$,
2. will find the lexicographically least sequence of n-bit strings $x_1, \ldots, x_m$, such that for any good advice β of length at most 2^{n-c_M}, there is some x_j for which $M(x_j, \beta) = 1$; and, when given an $i \leq m$,
3. outputs x_i.

The values referenced in P are allowed to depend on ℓ by the use of Kleene's second recursion theorem.[5] Now we must have $k \geq 2^{n-c_M}$ as otherwise the sequence of x_i's found by the program would have $m \leq d(k+1) \leq 2^{n-\ell-1}$ elements, and then the random x_i would have a description of length $\ell + (n - \ell - 1) = n - 1$. ■

A similar theorem can be proven for R_K. Here it should be noted that the index i given to the program P should then itself be encoded in a prefix-free fashion, and this will cause the prefix-free complexity of the x_i in the sequence to go up, and our bound becomes a little worse. We could obtain, for instance, the following theorem:

1.2.2. Lemma. *For every total Turing machine M it holds that R_K is not in i.o.-$M/2^{n-\log n-2\log\log n}$.*

In order to get our statement about time-bounded advice classes we instantiate Lemma 1.2.1 with universal machines U_t that run for t steps, use the first part of their advice, in prefix-free form, as a code for a machine to simulate and have the second part of the advice for U_t as the advice for the simulated machine. The following is a direct consequence of Lemmas 1.2.1 and 1.2.2.

1.2.3. Lemma. *(a) For every time bound t and universal advice machine U_t there is a constant c_t such that R_C is not in i.o.-$U_t/2^{n-c_t}$.*

(b) For every time bound t and prefix-free universal machine U_t, R_K is not in i.o.-$U_t/2^{n-\log n-2\log\log n}$.

We are now ready to prove the main theorem from this section.

[5] Kleene's second recursion theorem states that for any computable function $f(e, x)$, where e and x are binary strings, there exists a program p such that $p(x) = f(p, x)$. If $f(e, x)$ can be computed in time $t(|e|, |x|)$ by a program of length ℓ, then p will have length $\ell + O(1)$, and will run in time $O(|p| + t(|p|, |x|) \log t(|p|, |x|))$.

1.2.4. THEOREM. *(a) For every time bound t there is a constant d_t such that $R_C \notin$ i.o.-DTIME$(t)/2^{n-d_t}$.*

(b) For every time bound t it holds that $R_K \notin$ i.o.-DTIME$(t)/2^{n-2\log n}$.

Proof. To prove (a), suppose that the theorem is false, i.e., that is there is a time bound t such that for every d there is a machine M_d that runs in time t such that $R_C \in$ i.o.-$M_d/2^{n-d}$. Set $t' = t\log^2 t$ and let $c_{t'}$ be the constant that comes out of Lemma 1.2.3 when instantiated with time bound t'. Set $d = c_{t'}+1$ and let the code of machine M_d from the (false) assumption have size e. So we have that $R_C \in$ i.o.-$M_d/2^{n-d}$. This in turn implies that $R_C \in$ i.o.-$U_{t'}/(2^{n-d}+e+2\log e)$, which implies that $R_C \in$ i.o.-$U_{t'}/2^{n-c_{t'}}$ a contradiction with Lemma 1.2.3. The last step is true because the universal machine running for at most $t' = t\log^2 t$ steps, can simulate M_d, which runs in time t. Part (b) is proven in the same fashion. ■

We can also obtain a variant of Theorem 1.2.4 for the time-bounded prefix-free Kolmogorov complexity (we will not need the R_{C^t} variant, so we do not include it here).

1.2.5. LEMMA. *There exists a constant c such that for any time bounds t and t', if $t = \omega(2^{2^{n+c}}t'(n)\log t'(n))$ then for every prefix-free Turing machine M running in time t' it holds that R_{K^t} is not in i.o.-$M/2^{n-\log n-2\log\log n}$.*

Proof. An inspection of the proof of Lemma 1.2.1 reveals that the running time of the search procedure for $x_1,\ldots,x_m$ can be bounded by $2^{2^{n+c}}t'(n)$ for some universal constant c independent of M. We now use the same argument of that proof, replacing the use of Kleene's second recursion theorem and Kolmogorov complexity with their time-bounded variants. ■

In the same way as before we obtain the following corollary.

1.2.6. COROLLARY. *There exists a constant c such that for any time bounds t and t', if $t \in \omega(2^{2^{n+c}}t'(n)\log^3 t'(n))$ then R_{K^t} is not in i.o.-DTIME$(t')/2^{n-2\log n}$.*

For example, if we instantiate $t = 2^{2^{2n}}$ and $t' = 2^{n^{\log n}}$ in the above corollary we get that R_{K^t} is not in i.o.-EXP$/2^{n-2\log n}$.

1.3 A new proof of a theorem by Barzdin

Barzdin [25] showed a construction of a computably enumerable set whose characteristic string has high resource-bounded Kolgomorov complexity almost everywhere. As an immediate corollary of Theorem 1.2.4 we get an alternative, more natural candidate for such a set, namely $\overline{R_C}$ (the complement of R_C).

1.3.1. COROLLARY. *For every time bound t there is a constant c such that*

$$C^t(\overline{R_C}(1:n) \mid n) \geq n/c.$$

Barzdin [25] also showed that this lower bound is optimal. That is, the dependence of c on the time bound t is needed for the characteristic sequence of every r.e. set. Hence the dependence on t is also necessary in our Theorem 1.2.4. Indeed it can be shown that:

1.3.2. Theorem. *(a) For every c, there is a time bound t for which R_C is in i.o.-$\mathrm{DTIME}(t)/2^{n-c}$.*

(b) There exists some c such that, for every time bound t, $R_C \notin \mathrm{DTIME}(t)/2^{n-c}$.

Theorem 1.3.2 gives us an interesting contrast.

1.3.3. Corollary. *The following hold:*

(a) For every c, $R_C \in$ i.o.-$\mathrm{REC}/2^{n-c}$.

(b) For some c, $R_C \notin \mathrm{REC}/2^{n-c}$. The constant c depends only on the universal machine defining R_C.

Proof of Theorem 1.3.2. (a) Let H denote the binary entropy function, and choose $0 < \alpha < 1/2$, such that $H(\alpha) < 2^{-c}$. Let $\sigma = \liminf_{n\to\infty} |\overline{R_C^{=n}}|/2^n$, and set $\tilde{\sigma}$ to the dyadic rational[6] written with the first $1 + \log(1/\alpha)$ bits of σ. Note $\tilde{\sigma}$ is independent of n and can be hardwired into our machine.

Then, on input 0^n, we can (1) enumerate $\tilde{\sigma}2^n$ many non-random strings of length n, and (2) assume that some given advice tells us, among the strings which we did not enumerate, which are in R_C. Since for sufficiently large n, we can always find a fraction of $\tilde{\sigma}$ non-random strings of length n, this algorithm halts almost everywhere. And because, for infinitely many n, $\left|\tilde{\sigma}2^n - |\overline{R_C^{=n}}|\right| < \alpha 2^n$, after (1) we are left with at most $\alpha 2^n$ non-random strings still to be enumerated. Since $\log \sum_{k\le\alpha 2^n} \binom{2^n}{k} \le H(\alpha)2^n \le 2^{n-c}$ [cf. 61, p.283], then 2^{n-c} many bits of advice will be able to point out exactly these strings.
(b) Fix d such that $|R_C^{=n}| > n/d$ for all n.

Let $\{M_i\}_{i\in\mathbb{N}}$ computably enumerate all Turing machines. Choose $c = \ell + \log d + 2$, where ℓ is the length of a self-delimiting encoding of the program which:

1. given as input an index i padded to length $n - \ell - 1$,
2. will work with $M = M_n$, and, as in the proof of Lemma 1.2.1, assuming M is total,
3. will find the lexicographically least sequence of n-bit strings $x_1, \ldots, x_m$, such that for any good advice β of length at most 2^{n-c}, there is some x_j for which $M(x_j, \beta) = 1$; and, when given an $i \le m$,
4. output x_i.

Then notice that if M_n is total, regardless of which good advice β, of length 2^{n-c}, it is given, there will always be some n-bit string x_j which is not random (it is described using $n - 1$ bits), and for which $M_n(x_j, \beta) = 1$. So M_n does not decide R_C with 2^{n-c} bits of advice. ■

[6] A dyadic rational is a rational number of the form $\frac{a}{2^b}$.

1.4 BPP $\leq^p_{tt} R_K$

In this section we show that we can derandomize any bounded-error algorithm if we have non-adaptive access to R_K. It was shown in [56] that if a function is sufficiently hard to approximate by large enough circuits, we could use this function to build a useful pseudo-random generator. We start with the following definition:

1.4.1. Definition. A boolean function $f : \{0,1\}^n \to \{0,1\}$ is (ε, S)-hard (or, alternatively, ε-hard for circuits of size S) if any circuit of size S can correctly compute f on at most an $\frac{1}{2} + \epsilon$ fraction of its inputs.

Then it will follow from Theorem 1.2.4 that R_K is hard to approximate.

1.4.2. Lemma. *For any n, the truth table $R_K^{=n}$ is $\frac{1}{n^4}$-hard for circuits of size $2^{\frac{1}{4}n}$.*

Proof. We show that any function $f : \{0,1\}^n \to \{0,1\}$ that is not $(\frac{1}{n^4}, 2^{\frac{1}{4}n})$-hard is in i.o.-P/$2^{n-2\log n}$.

Under this assumption, there exist infinitely often circuits C_n of size $2^{\frac{1}{4}n}$ which compute f on all but a $\frac{1}{n^4}$ fraction of the inputs. Then, given as advice C_n and the list of x's for which $C_n(x) \neq f(x)$, we may compute f exactly for all inputs of size n. A description for C_n can be made to have length, say, $2^{\frac{1}{2}n}$, and the list has length $n\frac{2^n}{n^4} = 2^{n-3\log n}$. This makes for a total of no more than $2^{n-2\log n}$ bits of advice. ■

1.4.3. Corollary. *For some $\epsilon > 0$, there is a $(2^{\epsilon n}, 2^{\epsilon n})$-hard function $f \leq^p_{tt} R_K$.*

Proof. This follows from various hardness amplification results. First, given non-adaptive access to the characteristic function of $R_K^{=cn}$ (for some large enough constant c), which is $(\frac{1}{n^4}, 2^{\frac{1}{4}n})$-hard, [54] can produce a function $f' : \{0,1\}^{c'n} \to \{0,1\}$ (for some large enough $c' < c$) which is $\Omega(1)$-hard for circuits of size $\frac{2^{\frac{1}{4}n}}{n^{O(1)}}$. Second, given non-adaptive access to f', [56] construct a direct product generator which may be used to obtain a $(2^{\epsilon n}, 2^{\epsilon n})$-hard function $f : \{0,1\}^n \to \{0,1\}$. ■

1.4.4. Corollary. BPP $\leq^p_{tt} R_K$.

Proof. [84]'s pseudo-random generator can be used to derandomise BPP, when given a $(2^{\epsilon n}, 2^{\epsilon n})$-hard function f. The generator is a function $G_f : \{0,1\}^s \to \{0,1\}^n$ which takes a seed of length $s = O(\log n)$, and queries f on inputs $x_1, \ldots, x_n$. Each x_i has length $O(\log n)$ and is constructed by selecting specific bits from the seed (using a combinatorial construction called a *design*). So in order to compute G_f, it is only necessary to have the values of f for every input of length up to $O(\log n)$, and these values are $poly(\log n)$-time truth-table reducible to R_K. ■

By using Corollary 1.2.6 instead of Theorem 1.2.4, the hardness of approximation results above will also hold for R_{K^t}, for suitably large t. The following variant can then be proven:

1.4.5. Corollary. *For any time bound* $t = \Omega(2^{2^{2n}})$, $\mathrm{BPP} \leq^p_{tt} R_{K^t}$.

It is interesting to compare this statement with the result of Buhrman and Mayordomo [32] that $\mathrm{EXP} \not\subseteq P^{R_{K^t}}$ for $t = 2^{n^2}$. Because of this result, it will be rather difficult to prove the above corollary for $t = 2^{n^2}$, for then it would follow EXP $\neq$ BPP, solving a longstanding open question.

We can conclude, for the classes we defined earlier:

1.4.6. Corollary. $\mathrm{BPP} \subseteq \mathrm{DTTR} \cap \mathrm{TTRT}$.

1.5 High-complexity advice can only be used as randomness

Our goal now would be to show that using R_K as a source of randomness is the only way to make use of it. Ideally we would like to show that any recursive set that is truth-table reducible to R_K must be in BPP. We fall short of such a goal. However we can show the following claim.

1.5.1. Theorem. *Let* $\alpha : \{0\}^* \to \{0,1\}^*$ *be a length-preserving function and* $c > 0$ *be a constant, such that* $\alpha(0^n) \notin$ *i.o.*-$\mathrm{EXP}/n - c \log n$. *Then for every* $A \in \mathrm{EXP}$ *if* $A \in \mathrm{P}/\alpha(0^{n^d})$ *for some* $d > 0$ *then* $A \in \mathrm{BPP}//O(n \log n)$.

This theorem says that very Kolmogorov-random advice of polynomial size can be replaced by almost linear-size advice and true randomness. It can be proven using diagonalization that such advice functions exist, and these can be used to derandomize BPP as we did above.

We fall short of proving a converse of Corollary 1.4.4 in two respects. First, the advice is supposed to model the initial segment of the characteristic sequence of R_K which the truth-table can access. However, by providing only polynomial size advice we restrict the hypothetical truth-table reduction to query strings of only logarithmic length. Second, the randomness that we require from the initial segment is much stronger than what one can prove and what is in fact true for the initial segment of the characteristic sequence of R_K. We know how to deal with the first issue in one of two ways: we can make an even stronger assumption on the advice string, as is done by Theorem 1.5.2, or we can restrict ourselves to resource-bounded Kolmogorov complexity, for which it will be shown (in §1.6) that querying R_{K^t} for strings of up to polynomial length is enough. We do not know how to deal with the second issue.

Proof. Let M be a polynomial-time Turing machine and $A \in \mathrm{EXP}$ be a set such that $A(x) = M(x, \alpha(0^{|x|^d}))$. We claim that for all n large enough there is a non-negligible fraction of advice strings r of size n^d that could be used in place of $\alpha(0^{n^d})$. More precisely:

$$\Pr_{r \in \{0,1\}^{n^d}}[\forall x, x \in A \iff M(x,r) = 1] > \frac{1}{n^{cd}}.$$

To prove the claim consider the set $G = \{r \in \{0,1\}^{n^d} \mid \forall x \in \{0,1\}^n, x \in A \iff M(x,r) = 1\}$. Clearly, $G \in$ EXP and $\alpha(0^{n^d}) \in G$. If $|G^{=n^d}| \leq 2^{n^d}/n^{cd}$ then $\alpha(0^{n^d})$ can be computed in exponential time from its index in the set $G^{=n^d}$ of length $n^d - cd \log n$. Since $\alpha(0^{n^d}) \notin$ i.o.-EXP$/n^d - cd \log n$ this cannot happen infinitely often.

Now we present an algorithm which on input x, and using only $O(n \log n)$ bits of advice (in fact $O(\log n)$ entries from the truth table of A), will with high probability produce a string in $r \in G$, and output $A(x)$:

1. Given an input x of length n, and an advice string $x_1, A(x_1), ..., x_k, A(x_k)$,
2. sample at most $2n^{cd}$ strings of length n^d until the first string r is found such that $M(x_i, r) = A(x_i)$ for all $i \in \{1, \ldots, k\}$.
3. If such r is found then output $M(x,r)$, otherwise output 0.

For all n large enough the probability that the second step does not find r compatible with the advice is upper-bounded by the probability that we do not sample any string from G which is at most $(1 - \frac{1}{n^{cd}})^{2n^{cd}} < e^{-2} < 1/6$.

It suffices to show that we can find an advice sequence such that, for at least 4/5-fraction of the r's compatible with the advice, $M(x,r) = A(x)$. For given n, we will find the advice by pruning iteratively the set of bad random strings $B = \{0,1\}^{n^d} \setminus G$. Let $i = 0, 1, \ldots, \log_{5/4} 4n^{cd}$. Set $B_0 = B$. If there is a string $x \in \{0,1\}^n$ such that for at least 1/5 of $r \in B_i$, $M(x,r) \neq A(x)$, then set $x_{i+1} = x$ and $B_{i+1} = B_i \cap \{r \in \{0,1\}^{n^d} \mid M(x_{i+1}, r) = A(x_{i+1})\}$. If there is no such string x then stop and the x_i's obtained so far will form our advice. Notice, if we stop for some $i < \log_{5/4} 4n^{cd}$ then for all $x \in \{0,1\}^n$, $\Pr_{r \in B_i}[M(x,r) \neq A(x)] < 1/5$. Hence, for any given input, the r found by the algorithm to be compatible with the advice will give the correct answer with probability at least 4/5. On the other hand, if we stop building the advice at $k = \log_{5/4} 4n^{cd}$ then $|B_k| \leq 2^{n^d} \cdot (4/5)^{\log_{5/4} 4n^{cd}} \leq |G^{=n^d}|/4$. Hence, any string r found by the algorithm to be compatible with the advice $x_1, A(x_1), ..., x_k, A(x_k)$ will come from G with probability at least 4/5. ■

The following theorem can be established by a similar argument. It again relies on the fact that a polynomially large fraction of all advice strings of length 2^{n^d} must work well as an advice. By a pruning procedure similar to the proof of Theorem 1.5.1 we can avoid bad advice. In the BPP algorithm one does not have to explicitly guess the whole advice but only the part relevant to the pruning advice and to the current input.

1.5.2. THEOREM. *Let $\alpha : \{0\}^* \to \{0,1\}^*$ be a length preserving function and $c > 0$ be a constant. If $\alpha(0^n) \notin$ i.o.-EXP$/n - c \log \log n$ then for every $A \in$ EXP if $A \in$ P$/\alpha(0^{2^{n^d}})$ for some $d > 0$ then $A \in$ BPP$//O(n \log n)$.*

We show next that if the set A has some suitable properties we can dispense with the linear advice all together and replace it with only random bits. Thus for example if SAT $\in$ P$/\alpha(0^n)$ for some computationally hard advice $\alpha(0^n)$ then SAT $\in$ BPP.

1.5.3. THEOREM. *Let $\alpha : \{0\}^* \to \{0,1\}^*$ be a length preserving function and $c > 0$ be a constant such that $\alpha(0^n) \notin$ i.o.-EXP$/n - c \log n$. Let A be paddable and polynomial-time many-one-complete for a class $\mathcal{C} \in \{\mathrm{NP}, \mathrm{P}^{\#\mathrm{P}}, \mathrm{PSPACE}, \mathrm{EXP}\}$. If $A \in \mathrm{P}/\alpha(0^{n^d})$ for some $d > 0$ then $A \in$ BPP (and hence $\mathcal{C} \subseteq$ BPP).*

To prove the theorem we will need the notion of instance checkers. We use the definition of Trevisan and Vadhan [94].

1.5.4. DEFINITION. An *instance checker* C for a boolean function f is a polynomial-time probabilistic oracle machine whose output is in $\{0, 1, \mathrm{fail}\}$ such that

- for all inputs x, $\Pr[C^f(x) = f(x)] = 1$, and
- for all inputs x, and all oracles f', $\Pr[C^{f'}(x) \notin \{f(x), \mathrm{fail}\}] \leq 1/4$.

It is immediate that by linearly many repetitions and taking the majority answer one can reduce the error of an instance checker to 2^{-n}. Note also the following theorem [attributed to 19, 72, 91]:

1.5.5. THEOREM. *Any problem that is complete for* EXP, PSPACE *or* $\mathrm{P}^{\#\mathrm{P}}$ *has an instance checker. Moreover, there are* EXP*-complete problems,* PSPACE*-complete problems, and* $\mathrm{P}^{\#\mathrm{P}}$*-complete problems for which the instance checker C only makes oracle queries of length exactly $\ell(n)$ on inputs of length n for some polynomial $\ell(n)$.*

However, it is not known whether NP has instance checkers.

Proof of Theorem 1.5.3. To prove the claim for $\mathrm{P}^{\#\mathrm{P}}$-, PSPACE- and EXP-complete problems we use the instance checkers. We use the same notation as in the proof of Theorem 1.5.1, i.e., M is a Turing machine such that $A(x) = M(x, \alpha(0^{|x|^d}))$ and the set of good advice is G. We know from the previous proof that $|G^{=n^d}| \geq 2^{n^d}/n^{cd}$ because $\alpha(0^n) \notin$ i.o.-EXP$/n - c \log n$.

Let C be the instance checker for A which on input of length n asks oracle queries of length only $\ell(n)$ and makes error on a wrong oracle at most 2^{-n}. The following algorithm is a bounded-error polynomial-time algorithm for A:

1. On input x of length n, repeat at most $2n^{cd}$ times
   (a) Pick a random string r of length $(\ell(n))^d$.
   (b) Run the instance checker C on input x and answer each of his oracle queries y by $M(y, r)$.
   (c) If C outputs "fail" continue with another iteration otherwise output the output of C.
2. Output 0.

Clearly, if we sample $r \in G$ then the instance checker will provide a correct answer and we stop. The algorithm can produce a wrong answer either if the instance checker always fails (so we never sample $r \in G$ during the iterations) or if the instance checker gives a wrong answer. Probability of not sampling

a good r is at most $(1 - \frac{1}{n^{cd}})^{2n^{cd}} < e^{-2} < 1/6$. The probability of getting a wrong answer from the instance checker in any of the iterations is at most $2n^{cd}/2^n$. Thus the algorithm provides the correct answer with probability at least 2/3.

To prove the claim for NP-complete languages we show it for the canonical example of SAT. The following algorithm solves SAT correctly with probability at least 5/6:

1. On input ϕ of length n, repeat at most $2n^{cd}$ times
   (a) Pick a random string r of length n^d.
   (b) If $M(\phi, r) = 1$ then use the self-reducibility of SAT to find a presumably satisfying assignment a of ϕ while asking queries ψ of size n and answering them according to $M(\psi, r)$. If the assignment a indeed satisfies ϕ then output 1 otherwise continue with another iteration.
2. Output 0.

Clearly, if ϕ is satisfiable we will answer 1 with probability at least 5/6. If ϕ is not satisfiable we will always answer 0. ■

1.6 TTRT $\subseteq$ P/*poly*

In this section we will show that TTRT $\subseteq$ P/*poly*, and explain why the proof technique we use will not be helpful to settle the DTTR case.

At a first glance, it seems reasonable to guess that a polynomial-time reduction would have difficulty telling the difference between an oracle for R_K and an oracle for R_{K^t}, for large enough t. Indeed, most theorems about Kolmogorov complexity can be proven for its resource-bounded variants, e.g. symmetry of information [67], Muchnik's theorem [80, 79], etc. At first one might think that t would not need to be very large so that R_{K^t} looks sufficiently like R_K to fool a polynomial-time reduction into behaving the same way with both oracles. However, this intuition is wrong, and here is an example for adaptive polynomial-time reductions, which follows from results in [34, 32]:

1.6.1. Observation. *There is a polynomial-time algorithm which, given oracle access to R_K and input 1^n, outputs a K-random string of length n. However, for any time-bound t such that*

$$t(n+1) \gg 2^n t(n),$$

there is no polynomial-time algorithm which, given oracle access to R_{K^t} and input 1^n, outputs a K^t-random string of length n.

For the algorithm, see [34]; roughly, we start with a small random string and then use [34, Theorem 15] (described later) to get a successively larger random string. But in the time-bounded case in [32] it is shown that on input 1^n, no polynomial-time machine M can query (or output) any K^t-random string of

length n: in fact, $M(1^n)$ is the same for both oracles R_{K^t} and $R' = R_{K^t}^{\leq n-1}$. This is proven as follows: since R' can be computed in time $t(n)$ (by Proposition 1.1.4), then any query of length $\geq n$ made by $M^{R'}(1^n)$ is described by a pointer of length $O(\log n)$ in time $t(n)$, and hence is not in R_{K^t}.

1.6.1 Small circuits for sets reducible to R_{K^t}

We now prove that TTRT is a subset of P/*poly*. Actually, we will prove that this holds even for *Turing* reductions to R_{K_U}.

1.6.2. Theorem. *Suppose $A \in \mathrm{DTIME}(t_1)$ and $M : A \leq_T^p R_{K^t}$, for some time-bounds t, t_1 with*[7]

$$t(n+1) \geq 2^n t(n) + 2^{2^n} t_1(2^n).$$

*Then $A \in$ P/*poly*; in fact, if M runs in time n^c, and $R' = R_{K^t}^{\leq \lceil (c+1)\log n \rceil}$, then $\forall x \in \{0,1\}^n \; M^{R'}(x) = A(x)$.*

Proof. Let $\ell(n) = \lceil (c+1)\log n \rceil$, $R'(n) = R_{K^t}^{\leq \ell(n)}$, and suppose that $M^{R'(n)}(x) \neq A(x)$ for some x of length n. Then we may find the first such x in time $2^{\ell(n)} t(\ell(n)) + 2^{n+1}(t_1(n) + O(n^c))$ (cf. Proposition 1.1.4), and each query made by $M^{R'(n)}(x)$ can be output by a program of length $c \log n + O(1)$, running in the same time bound. But since $A(x) \neq M^{R'(n)}(x)$, it must be that, with $R'(n)$ as oracle, M makes some query of size $m \geq \ell(n)+1$ which is random for t-bounded Kolmogorov complexity (because both small and nonrandom queries are answered correctly when using R' instead of R_{K^t}). Let q be the first such query. We then have both that q is supposed to be random, and that q can be output by a program of length $< \ell(n)$ in time $2^{\ell(n)} t(\ell(n)) + 2^{n+1}(t_1(n) + O(n^c)) \ll 2^{\ell(n)} t(\ell(n)) + 2^{2^{\ell(n)}} t_1(2^{\ell(n)}) \leq t(\ell(n)+1) \leq t(m)$, which is a contradiction. ■

1.6.3. Corollary. TTRT $\subseteq$ P/*poly*.

Proof. Let $L \in$ TTRT. By the definition of TTRT, $L \leq_{tt}^p R_{K^{t_0}}$ for some t_0. Using Proposition 1.1.5, we then have that L is decidable in time $t_L(n) = 2^{n^k} t_0(n^k)$ for some constant k. Choose a time-bound t such that $t(n+1) \geq 2^n t(n) + 2^{2^n} t_L(2^n)$. By the definition of TTRT, since $t > t_0$, we have that $L \leq_{tt}^p R_{K_{U_0}^t}$, from which by Theorem 1.6.2 it follows that $L \in$ P/*poly*. ■

PSPACE $\leq_T^p R_K$ [5], but Theorem 1.6.2 implies that PSPACE $\not\leq_T^p R_{K^t}$ for sufficiently-large t, unless PSPACE $\subseteq$ P/*poly*. This highlights the difference between the time-bounded and ordinary Kolmogorov complexity, and how this comes to the surface when working with reductions to the corresponding sets of random strings. We wish to emphasize at this point that the proof of the inclusion PSPACE $\leq_T^p R_K$ relies on the ability of a P^{R_K} computation to construct a large element of R_K, whereas the P/*poly* upper bound in the time-bounded case relies on the *inability* to use the oracle to find such a string, in the time-bounded setting.

[7] For example, if $A \in$ EXP, then t can be doubly-exponential. If A is elementary-time computable, then t can be an exponential tower.

1.6.2 A reduction distinguishing R_K from R_{K^t}, and an incorrect conjecture

Theorem 1.6.2 shows that a polynomial-time truth-table reduction to R_{K^t} for sufficiently-large t will work just as well if only the logarithmically-short queries are answered correctly, and all of the other queries are simply answered "no".

The authors of [7] conjectured that a similar situation would hold if the oracle were R_K instead of R_{K^t}. More precisely, they proposed a proof-theoretic approach towards proving that DTTR is in P/*poly*: Let PA_0 denote Peano Arithmetic, and for $k > 0$ let PA_k denote PA_{k-1} augmented with the axiom "PA_{k-1} is consistent". Allender et al. [7] have shown that, for any polynomial-time truth-table reduction M reducing a decidable set A to R_K, one can construct a true statement of the form $\forall n \forall j \forall k \Psi(n, j, k)$ (which is provable in a theory such as Zermelo-Frankel), with the property that if for each fixed (**n**,**j**,**k**) there is some k' such that $\mathrm{PA}_{k'}$ proves $\psi(\mathbf{n},\mathbf{j},\mathbf{k})$, then DTTR $\subseteq$ P/*poly*. Furthermore, if these statements were provable in the given extensions of PA, it would follow that, for each input length n, there is a finite subset $R' \subseteq R_K$ consisting of strings having length at most $O(\log n)$, such that $M^{R'}(x) = A(x)$ for all strings x of length n.

Thus the authors of [7] implicitly conjectured that, for any polynomial-time truth-table reduction of a decidable set to R_K, and for any n, there would be some setting of the short queries so that the reduction would still work on inputs of length n, when all of the long queries are answered "no". While we have just seen that this is precisely the case for the time-bounded situation, the next theorem shows that this does not hold for R_K, even if "short" is interpreted as meaning "of length $< n$". (It follows that infinitely many of the statements $\psi(\mathbf{n},\mathbf{j},\mathbf{k})$ of [7] are independent of every $\mathrm{PA}_{k'}$.)

1.6.4. THEOREM. *There is a truth-table reduction $M : \{0,1\}^* \leq^p_{tt} R_K$, such that, for all large enough n:*

$$\forall R' \subseteq \{0,1\}^{\leq n-1} \exists x \in \{0,1\}^n \; M^{R'}(x) \neq 1.$$

Proof. Theorem 15 of [34] presents a polynomial-time procedure which, given a string z of even length $n-2$, will output a list of constantly-many strings $z_1, \ldots, z_c$ of length n, such that at least one of them will be K-random if z is. We use this to define our reduction M as follows: on input $x = 00\ldots0z$ of length n having even $|z|$, we query each of z, $z_1, \ldots, z_c$, and every string of length at most $\log n$. If there are no strings of length at most $\log n$ in the oracle, we reject. Else, if z is in the oracle but none of the z_i are, we reject. On all other cases we accept.

By [34, Theorem 15], and since R_K has strings at every length, it is clear that $M^{R'}$ accepts every string if $R' = R_K$, and rejects every string if $R' = \varnothing$. However, for any non-empty set $R' \subseteq \{0,1\}^{\leq n-1}$, let $\ell \leq n-1$ be the highest even length for which $R'^{=\ell} \neq \varnothing$, and pick $z \in R'^{=\ell}$. Then we will have $z \in R'^{=\ell}$ but every $z_i \notin R^{=\ell+2}$, hence $M^{R'}(00\ldots0z)$ rejects. ∎

In fact, if we let $R' = R_{K^t}^{\leq n-1}$, for even n, then for the first $x = 00z$ such that $M^{R'}(x) = 0$, we will have $z \in R' \subseteq R_{K^t}$, but each z_i can be given by a

small pointer in time $O(2^{n-1}t(n-1))$ (again we use Proposition 1.1.4), and hence $z_i \notin R_{K^t}$ for suitably fast-growing t. Thus $M^{R_{K^t}}(x) = 0 \neq M^{R_K}(x)$, and we conclude:

1.6.5. Observation. *If $t(n+1) \gg 2^n t(n)$, then the non-adaptive reduction M above behaves differently on the oracles R_K and R_{K^t}.*

1.7 TTRT $\subseteq$ PSPACE/$O(1)$

Our single goal for this section is proving the following:

1.7.1. Theorem. *For any computable unbounded function $f(n) = \omega(1)$,*

$$\text{TTRT} \subseteq \text{PSPACE}/f(n).$$

The proof of this theorem is patterned closely on related arguments in [8], although a number of complications arise in the time-bounded case. Although we aim to make the presentation here self-contained, [8] is a good primer and a source of additional intuition for the proof. Also, one can refer to the conference version of this chapter [6] for a presentation that is not self-contained but emphasizes the differences between the proof in the time-bounded case and the unbounded case. Before proving the theorem we present several supporting propositions.

1.7.2. Proposition. *For any time bound t and universal prefix-free machine U,*

$$\sum_{x \in \{0,1\}} 2^{-K^t_U(x)} \leq 1.$$

Proof. From the Kraft Inequality [see 69, Theorem 1.11.1], $\sum_{x \in \{0,1\}^*} 2^{-K_U(x)} \leq 1$ for any prefix-free machine U. For any time bound t and string x, $K^t_U(x) \geq K_U(x)$, so adding a time bound can only decrease the sum on the left side of this inequality. ■

1.7.3. Proposition (Analogue to Coding Theorem). *Let f be a function such that*

1. *$\sum_{x \in \{0,1\}^*} 2^{-f(x)} \leq 1$, and*
2. *there is a machine M computing $f(x)$ in time $t(|x|)$.*

Then if $t'(|x|) > 2^{2|x|}t(|x|)$ there exists some prefix-free machine M' such that $K^{t'}_{M'}(x) = f(x) + 2$.

Proof. The proof is similar to the proof of Proposition 5 from [8]. Let

$$E = \langle x_0, f(x_0)\rangle, \langle x_1, f(x_1)\rangle, \ldots$$

be an enumeration of the function f ordered lexicographically by the strings x_i.

We identify the set of infinite sequences $S = \{0,1\}^\infty$ with the half-open real interval $[0,1)$; that is, each real number r between 0 and 1 will be associated with the sequence(s) corresponding to the infinite binary expansion of r. We will associate each element $\langle x_i, f(x_i)\rangle$ from the enumeration E with a subinterval $I_i \subseteq S$ as follows:

$I_0 = [0, 2^{-f(x_0)})$, and for $i \geq 1$, $I_i = [\sum_{k<i} 2^{-f(x_k)}, \sum_{k\leq i} 2^{-f(x_k)})$. That is, I_i is the half-open interval of length $2^{-f(x_i)}$ that occurs immediately after the interval corresponding to the element $\langle x_{i-1}, f(x_{i-1})\rangle$ that appeared just prior to $\langle x_i, f(x_i)\rangle$ in the enumeration E.

Since $\sum_{i\geq 0} 2^{-f(x_i)} \leq 1$, each $I_i \subseteq S$.

Any *finite* string z also corresponds to a subinterval $\Gamma_z \subseteq S$ consisting of all infinite sequences that begin with z; Γ_z has length $2^{-|z|}$. Given any element $\langle x_i, f(x_i)\rangle$, there must exist a lexicographically first string z_i of length $f(x_i)+2$ such that $\Gamma_{z_i} \subseteq I_i$. Observe that, since the intervals I_i are disjoint, no string z_i is a prefix of any other.

Let M' be the following machine. On input z, M' runs M to compute the enumeration E until it finds an element $\langle x_i, f(x_i)\rangle$ that certifies that $z = z_i$. If it finds such an element then M' outputs x_i.

Suppose that M' outputs x_i on input z, and let $\langle x_i, f(x_i)\rangle$ be the element of E corresponding to x_i. Before outputting x_i, M' must compute $|\langle x_j, f(x_j)\rangle|$ for every string x_j such that $x_j < x_i$ (under the lexicographical ordering). There are at most $2^{|x_i|+1}$ strings x_j such that $x_j < x_i$, so overall this will take less than $2^{2|x_i|}t(|x_i|)$ time.

M' will be a prefix-free machine, and we have that $K^{t'}_{M'}(x) = f(x) + 2$. ■

1.7.4. Proposition (Analogue to Proposition 6 from [8]). *Let U be a universal prefix-free Turing machine and let M be any prefix-free Turing machine. Suppose that t, t', and t'' are time bounds and f, g are two time-constructible increasing functions, such that f is upper bounded by a linear function, and $t''(|x|) \geq \max\{f(t(|x|)), g(t'(|x|))\}$.*

Then there is a universal prefix-free machine U' such that

$$K^{t''}_{U'}(x) = \min(K^t_U(x), K^{t'}_M(x)) + 1.$$

Proof. On input $0y$, U' runs U on input y. If U would output string x on y after s steps, then U' outputs string x after $f(s)$ steps. Similarly, on input $1y$, U' runs M on input y. If M would output string x on y after s steps, then U' outputs string x after $g(s)$ steps.

Note that because U is an efficient universal prefix-free machine, U' will be an efficient universal prefix-free machine as well. ■

1.7.5. Proposition (Analogue of Proposition 7 from [8]). *Given any universal prefix-free machine U, time bound t, and constant $c \geq 0$, there is a universal prefix-free machine U' such that $K^t_{U'}(x) = K^t_U(x) + c$.*

Proof. On input $0^c x$, M' runs M on input x, and does not halt on other inputs. ■

Proof of Theorem 1.7.1. Fix f, and suppose for contradiction that $L \in \mathrm{TTRT}$ but $L \notin \mathrm{PSPACE}/f(n)$. Let t_0 be the time bound given in the definition of TTRT, and assume without loss of generality that $t_0(n)$ is greater than the time required to compute $f(n)$, and let U_0 be some arbitrary universal prefix-free machine. By the definition of TTRT, $L \leq^p_{tt} R_{K^{t_0}_{U_0}}$. Therefore, by Proposition 1.1.5, L is decidable in time $t_L(n) = 2^{n^k} t_0(n^k)$ for some constant k.

Let $t^*(n)$ be an extremely fast-growing time-constructible function, so that for any constant d, we have $t^*(\log(f(n))) > 2^{n^d} t_L(n)$ for all large n. To get our contradiction, we will show that there exists a universal prefix-free machine U such that $L \not\leq^p_{tt} R_{K^{t^{*3}}_U}$. Note that because $t^* > t_0$, this is a contradiction to the fact that $L \in \mathrm{TTRT}$.

For any function $f : \{0,1\}^* \to \mathbf{N}$, define $R_f = \{x : f(x) \geq |x|\}$. We will construct a function $F : \{0,1\}^* \to \mathbf{N}$ and use it to form a function $H : \{0,1\}^* \to \mathbf{N}$ such that:

1. F is a total function and $F(x)$ is computable in time $t^{*2}(|x|)$ by a machine M;
2. $H(x) = \min(K^{t^*}_{U_0}(x) + 5, F(x) + 3)$;
3. $\sum_{x \in \{0,1\}^*} 2^{-H(x)} \leq 1/8$;
4. $L \not\leq^p_{tt} R_H$.

As we will see below, these properties are enough to prove the theorem. ■

1.7.6. Claim (Analogue of Claim 1 from [8]). *Given the above properties $H = K^{t^{*3}}_U$ for some universal prefix-free machine U.*

By Property 4 this ensures that $L \not\leq^p_{tt} R_{K^{t^{*3}}_U}$ for some universal prefix-free machine U, and hence $L \notin \mathrm{TTRT}$, thus establishing Theorem 1.7.1.

Proof of Claim 1.7.6. By Property 3 we have that $\sum_{x \in \{0,1\}^*} 2^{-(F(x)+3)} \leq 1/8$. Hence it must hold that $\sum_{x \in \{0,1\}^*} 2^{-F(x)} \leq 1$. Using this along with Property 1, we then have by Proposition 1.7.3 that $K^{t^{*3}}_{M'} = F + 2$ for some prefix-free machine M'. By Proposition 1.7.5 we have that $K^{t^*}_{U'} = K^{t^*}_{U_0} + 4$ for some efficient universal prefix machine U'. Therefore, by Proposition 1.7.4, with $f(n) = n, g(n) = n^3$, we find that $H(x) = \min(K^{t^*}_{U_0}(x) + 5, F(x) + 3) = \min(K^{t^*}_{U'}(x), K^{t^{*3}}_{M'}) + 1$ is $K^{t^{*3}}_U$ for some efficient universal prefix machine U. ■

All we now need to show is that, for our given language L, we can always construct functions H and F with the four desired properties.

Let $\gamma_1, \gamma_2, \ldots$ be a list of all possible polynomial-time truth-table reductions from L to R_H. This is formed in the usual way: we take a list of all Turing machines and put a clock of $n^i + i$ on the ith one and we will interpret the output on a string x as an encoding of a Boolean circuit on atoms of the form "$z \in R_H$". (i.e. these atoms form the input gates of the circuit, and their truth

values determine the output of the circuit.) We will refer to the string z as a *query*.

As in [8], to ensure that $L \not\leq^p_{tt} R_H$ (Property 4), we need to satisfy an infinite list of requirements of the form

$R_e : \gamma_e$ is not a polynomial-time truth-table reduction of L to R_H.

As part of our construction we will set up and play a number of games, which will enable us to satisfy each of these requirements R_e in turn. Our moves in the game will define the function F (and thus indirectly H). Originally we have that $F(z) = 2|z| + 3$ for all strings z. Potentially during one of these games, we will play a move forcing a string z to be in the complement of R_H. To do this we will set $F(z) = |z| - 4$. Therefore, a machine M can compute $F(z)$ by running our construction, looking for the first time during the construction that $F(z)$ is set to $|z|-4$, and outputting $|z|-4$. If a certain amount of time elapses (to be determined later) during the construction without $F(z)$ ever being set to $|z| - 4$, then the machine M outputs the default value $2|z| + 3$.

1.7.1 Description of the games

Let us first describe abstractly the games that will be played during the construction; afterwards we will explain how we use these games to satisfy each requirement R_e (note that these games are defined differently than those in [8]).

For a given requirement R_e, a game $\mathcal{G}_{e,x}$ will be played as followed for some string x:

First we calculate the circuit $\gamma_{e,x}$, which is the output of the reduction γ_e on input x. Let F^* be the function F as it is at this point of the construction when the game $\mathcal{G}_{e,x}$ is about to be played. For any atom "$z_i \in R_H$" that is an input of this circuit such that $|z_i| \leq \log(f(|x|)) - 1$, we calculate $r_i = \min(K^{t^*}_{U_0}(z_i) + 5, F^*(z_i) + 3)$. If $r_i < |z_i|$ we substitute FALSE in for the atom, and simplify the circuit accordingly, otherwise we substitute TRUE in for the query, and simplify the circuit accordingly. (We will refer to this as the "pregame preprocessing phase".)

The remaining queries z_i are then ordered by increasing length. There are two players, the F player (whose moves will be played by us during the construction), and the K player (whose moves will be determined by $K^{t^*}_{U_0}$). As in [8], in each game the F player will either be playing on the YES side (trying to make the final value of the circuit equal TRUE), or the NO side (trying to make the final value of the circuit equal FALSE).

Let S_1 be the set of queries from $\gamma_{e,x}$ of smallest length, let S_2 be the set of queries that have the second smallest length, etc. So we can think of the queries being partitioned into an ordered set $\mathcal{S} = (S_1, S_2, \ldots, S_r)$ for some r.

The scoring for the game is similar to that in [8]; originally each player has a score of 0 and a player loses if his score exceeds some threshold ϵ. When playing a game $\mathcal{G}_{e,x}$, we set $\epsilon = 2^{-e-3}$.

Originally we have that the truth value of all the atoms in the game are TRUE. In round one of the game, the K player makes some (potentially empty) subset Z_1 of the queries from S_1 nonrandom; i.e. for each $z \in Z_1$ he sets the

atom "$z \in R_H$" to the value FALSE. For any $Z_1 \subseteq S_1$ that he chooses to make nonrandom, $\sum_{z \in Z_1}(2^{-(|z|-6)} - 2^{-(2|z|+3)})$ is added to his score. As in [8], a player can only legally make a move if doing so will not cause his score to exceed ϵ.

After the K player makes his move in round 1, the F player responds, by making some subset Y_1 of the queries from $S_1 - Z_1$ nonrandom. After the F player moves, $\sum_{z \in Y_1} 2^{-(|z|-4)} - 2^{-(2|z|+3)}$ is added to his score.

This is the end of round one. Then we continue on to round two, played in the same way. The K player goes first and makes some subset of the queries from S_2 nonrandom (which makes his score go up accordingly), and then the F player responds by making some subset of the remaining queries from S_2 nonrandom. Note that if a query from S_i is not made nonrandom by either the K player or the F player in round i, it cannot be made nonrandom by either player for the remainder of the game.

After r rounds are finished the game is done and we see who wins, by evaluating the circuit $\gamma_{e,x}$ using the answers to the queries that have been established by the play of the game. If the circuit evaluates to TRUE (FALSE) and the F player is playing as the YES (resp. NO) player, then the F player wins, otherwise the K player wins.

Note that the game is asymmetric between the F player and the K player; the F player has an advantage due to the fact that he plays second in each round and can make an identical move for fewer points than the K player. Because the game is asymmetric, it is possible that F can have a winning strategy playing on *both* the YES and NO sides. Thus we define a set $val(\mathcal{G}_{e,x'}) \subseteq \{0,1\}$ as follows: $0 \in val(\mathcal{G}_{e,x'})$ if the F player has a winning strategy playing on the NO side in $\mathcal{G}_{e,x'}$, and $1 \in val(\mathcal{G}_{e,x'})$ if the F player has a winning strategy playing on the YES side in $\mathcal{G}_{e,x'}$.

1.7.2 Description of the construction

Now we describe the construction. In contrast to the situation in [8], we do not need to worry about playing different games simultaneously or dealing with requirements in an unpredictable order; we will first satisfy R_1, then R_2, etc. To satisfy R_e we will set up a game $\mathcal{G}_{e,x}$ for an appropriate string x of our choice, and then play out the game in its entirety as the F player. We will choose x so that we can win the game $\mathcal{G}_{e,x}$, and will arrange that by winning the game we ensure that R_e is satisfied. It is possible that the K player will "cheat" on game $\mathcal{G}_{e,x}$, if our interpretation of the function $K^{t^*}_{U_0}$, which will determine the moves of the K player, does not translate into legal moves in the game. In this case we quit the game $\mathcal{G}_{e,x}$ and we play $\mathcal{G}_{e,x'}$ for some new x'. However, we will show that the K player cannot cheat infinitely often on games for a particular e, so eventually R_e will be satisfied.

Originally we define the function F so that $F(z) = 2|z| + 3$ for all strings z. Suppose s time steps have elapsed during the construction up to this point, and we are getting ready to construct a new game in order to satisfy requirement R_e. (Either because we just finished satisfying requirement R_{e-1}, or because K cheated on some game $\mathcal{G}_{e,x}$, so we have to start a new game $\mathcal{G}_{e,x'}$). Starting

with the string $0^{t^{*4}(s)}$ (i.e. the string of $t^{*4}(s)$ zeros), we search strings in lexicographical order until we find an x' such that $(1 - L(x')) \in val(\mathcal{G}_{e,x'})$. (Here, L denotes the characteristic function of the set L.)

Once we find such a string x' (which we will prove we always can), then we play out the game $\mathcal{G}_{e,x'}$ with the F player (us) playing on the YES side if $L(x') = 0$ and the NO side if $L(x') = 1$. To determine the K player's move in the ith round, we let $Z_i \subseteq S_i$ be those queries $z \in S_i$ for which $K^{t^*}_{U_0}(z) \leq |z|-6$. Our moves are determined by our winning strategy; whenever we play a move that makes a query z nonrandom, we update the function F so that $F(z) = |z| - 4$. Note that whenever either of the player plays a move involving a query z in one of the games (which we have called "making z nonrandom"), he *does* make the query z nonrandom in the sense that $R_H(z)$ is fixed to the value 0 for good.

To finish showing that Property 4 will be satisfied, it suffices to prove the following three claims.

1.7.7. Claim. *If during the construction we win a game $\mathcal{G}_{e,x}$, then R_e will be satisfied and will stay satisfied for the remainder of the construction.*

Proof. Suppose that we win a game $\mathcal{G}_{e,x}$. Let $H^* = \min(K^{t^*}_{U_0}+5, F^*+3)$, where F^* is the function F immediately after the game $\mathcal{G}_{e,x}$ is completed. Our having won the game implies that when evaluating the circuit $\gamma_{e,x}$, while substituting the truth value of "$z \in R_{H^*}$" for any query of the form "$z \in R_H$", we have that $\gamma_{e,x} \neq L(x)$, which means that the reduction γ_e does not output the correct value on input x and thus R_e is satisfied. For any game $\mathcal{G}_{e',x'}$ that is played later in the construction, by design x' is always chosen large enough so that any query that is not fixed during the pre-game preprocessing has not appeared in any game that was played previously, so $\mathcal{G}_{e',x'}$ will not conflict with $\mathcal{G}_{e,x}$ and R_e will remain satisfied for the remainder of the construction. ■

1.7.8. Claim. *For any given requirement R_e, the K player will only cheat on games $R_{e,x}$ for a finite number of strings x.*

Proof. If the K player cheats on a game $R_{e,x}$, it means that he makes moves that causes his score to exceed $\epsilon = 2^{-e-3}$. By the definition of how K's moves are determined, this implies that $\sum_{z \in Z_{e,x}} 2^{-(K^{t^*}_{U_0}(z)-6)} \geq \epsilon$, so $\sum_{z \in Z_{e,x}} 2^{-K^{t^*}_{U_0}(z)} \geq \epsilon/64$, where $Z_{e,x}$ is defined to be the set of all the queries that appear in the game $\mathcal{G}_{e,x}$ that are not fixed during the preprocessing stage. However, for any two games $G_{e,x}$ and $G_{e,x'}$ the sets $Z_{e,x}$ and $Z_{e,x'}$ are disjoint, so if K cheated on an infinite number of games associated with the requirement R_e, then this would imply that $\sum_{z \in \{0,1\}^*} 2^{-K^{t^*}_{U_0}(z)} \geq \epsilon/64 + \epsilon/64 + \cdots$. But this divergence would violate Lemma 1.7.2. ■

1.7.9. Claim. *During the construction, for any requirement R_e, we can always find a witness x with the needed properties to construct $\mathcal{G}_{e,x}$.*

Proof. Suppose for some requirement R_e, our lexicographical search goes on forever without finding an x such that $(1 - L(x')) \in val(\mathcal{G}_{e,x'})$. Then we will find that $L \in \text{PSPACE}/f(n)$, which is a contradiction.

Here is the PSPACE algorithm to decide L. Hardcode all the answers for the initial sequence of strings up to the point where we got stuck in the construction. Let F^* be the function F up to that point in the construction. On a general input x, construct $\gamma_{e,x}$. The advice function $f(n)$ will give the truth-table of $\min(K^t_{U_0}(z) + 5, F^*(z) + 3)$ for all queries z such that $|z| \leq \log(f(|x|)) - 1$. For any query z of $\gamma_{e,x}$ such that $|z| \leq \log(f(|x|)) - 1$, fix the answer to the query according to the advice.

If the F player had a winning strategy for both the YES and NO player on game $\mathcal{G}_{e,x}$, then we would not have gotten stuck on R_e. Also the F player must have a winning strategy for either the YES or the NO player, since he always has an advantage over the K player when playing the game. Therefore, because we got stuck, it must be that the F player has a winning strategy for the YES player if and only if $L(x) = 1$. Once the small queries have been fixed, finding the side (YES or NO) for which the F player has a winning strategy on $\mathcal{G}_{e,x}$, and hence whether $L(x) = 1$ or $L(x) = 0$, can be done in PSPACE.

To prove this, we will show that the predicate "The F player has a winning strategy as the YES player on $\mathcal{G}_{e,x}$" can be computed in alternating polynomial time, which by [43] is equal to PSPACE. To compute this predicate, we must determine if *for every* move of the K player in round 1, there *exists* a move for the F player in round 1, such that *for every* move of the K player in round 2, there *exists* a move for the F player in round 2... such that when the game is finished the circuit $\gamma_{e,x}$ evaluates to TRUE. We can represent any state of the game (i.e. which of the polynomial number of queries have been fixed to be nonrandom so far, the score of the players, the current round, and whose turn it is) by a number of bits bounded by a polynomial in $|x|$. Also, given a move by one of the players, it is easy to determine in polynomial time whether the move is legal and to compute the new score of the player after the move. (It suffices to add up a polynomial number of rationals of the form $a/2^b$ where $b = n^{O(1)}$). Also, because there are only a polynomial number of queries in the circuit $\gamma_{e,x}$, the total number of moves in the game is bounded by a polynomial. Finally, evaluating the circuit at the end of the game can be done in polynomial time. Thus the predicate in question can be computed in alternating polynomial time, which completes the proof. ■

The following claim shows that Property 1 is satisfied.

1.7.10. Claim. *$F(z)$ is computable in time $t^{*2}(|z|)$.*

Proof. The function F is determined by the moves we play in games during the construction. In order to prove the claim, we must show that if during the construction we as the F player make a move that involves setting a string z to be nonrandom, then fewer than $t^{*2}(|z|)$ time steps have elapsed during the construction up to that point. The machine M that computes F will on input z run the construction for $t^{*2}(|z|)$ steps. If, at some point before this during the construction, we as the F player make z nonrandom, then M outputs $|z| - 4$. Otherwise M outputs $2|z| + 3$.

Suppose during the construction that we as the F player make a move that sets a query z to be nonrandom during a game $\mathcal{G}_{e,x}$. Note that $|z| \geq \log(f(|x|))$, otherwise z would have been fixed during the preprocessing stage of the game.

There are at most $2^{|x|+1}$ strings x' that we could have considered during our lexicographic search to find a game for which we had a winning strategy before finally finding x. Let s be the number of time steps that have elapsed during the construction before this search began.

Let us first bound the amount of time it takes to reject each of these strings x'. To compute the circuit $\gamma_{e,x'}$ takes at most $|x'|^k$ time for some constant k. For each query y such that $|y| \leq \log(f(|x'|)) - 1$ we compute $\min(K^{t^*}_{U_0}(y) + 5, F^*(y) + 3)$. To calculate $F^*(y)$ it suffices to rerun the construction up to this point and check whether a move had been previously made on the string y. To do this takes s time steps, and by construction we have that $t^*(|z|) \geq t^*(\log f(|x|)) > |x| \geq |x'| \geq t^{*4}(s)$, so $s < |z|$. By Proposition 1.1.4, to compute $K^{t^*}_{U_0}(y)$ takes at most $2^{|y|}t^*(|y|) \leq 2^{|z|}t^*(|z|)$ time steps. Therefore, since there can be at most $|x'|^k$ such queries, altogether computing $\min(K^{t^*}_{U_0}(y)+5, F^*(y)+3)$ for all these y will take fewer than $|x'|^k 2^{|z|}t^*(|z|)$ time steps.

Then we must compute $L(x')$, and check whether $(1 - L(x')) \in val(\mathcal{G}_{e,x'})$. Computing $L(x')$ takes $t_L(|x'|)$ time. By Claim 1.7.9, once the small queries have been fixed appropriately, computing $val(\mathcal{G}_{e,x'})$ can be done in PSPACE, so it takes at most $2^{|x'|^d}$ time for some constant d.

Compiling all this information, and using the fact that for each of these x' we have that $|x'| \leq |x|$, we get that the total number of time steps needed to reject all of these x' is less than $2^{|x|^{d'}} 2^{|z|} t_L(|x|)t^*(|z|)$ for some constant d'.

During the actual game $\mathcal{G}_{e,x}$, before z is made nonrandom the construction might have to compute $K^{t^*}_{U_0}(y) + 5$ for all queries of $\gamma_{e,x}$ for which $|y| \leq |z|$. By Proposition 1.1.4 this takes at most $|x|^k 2^{|z|} t^*(|z|)$ time.

Therefore, overall, for some constant d'' the total amount of time steps elapsed before z is made nonrandom in the construction is at most

$$T = 2^{|x|^{d''}} 2^{|z|} t_L(|x|)t^*(|z|) + s < t^{*2}(|z|).$$

Here the inequality follows from the fact that $t^*(\log(f(|x|))) > 2^{|x|^d} t_L(|x|)$ for any constant d, and that $|z| \geq \log(f(|x|))$.

■

Finally, to finish the proof of the theorem we need to show that Property 3 is satisfied.

1.7.11. Claim. $\sum_{x \in \{0,1\}^*} 2^{-H(x)} \leq \frac{1}{8}$.

Proof. To begin, notice that

$$\begin{aligned} \sum_{x \in \{0,1\}^*} 2^{-H(x)} &= \sum_{x \in \{0,1\}^*} 2^{-\min(K^{t^*}_{U_0}(x)+5, F(x)+3)} \\ &\leq \sum_{x \in \{0,1\}^*} 2^{-(K^{t^*}_{U_0}(x)+5)} + \sum_{x \in \{0,1\}^*} 2^{-(F(x)+3)}. \end{aligned}$$

By Proposition 1.7.2, $\sum_{x\in\{0,1\}^*} 2^{-K^{t^*}_{U_0}(x)} \leq 1$, so $\sum_{x\in\{0,1\}^*} 2^{-(K^{t^*}_{U_0}(x)+5)} \leq 1/32$. We also have that $\sum_{x\in\{0,1\}^*} 2^{-(F(x)+3)} = (1/8)\sum_{x\in\{0,1\}^*} 2^{-F(x)}$. Therefore, it is enough that $\sum_{x\in\{0,1\}^*} 2^{-F(x)} \leq 1/2$, as this would imply that

$$\sum_{x\in\{0,1\}^*} 2^{-H(x)} \leq \frac{1}{32} + \frac{1}{8}\times\frac{1}{2} \leq \frac{1}{8}.$$

Let Z_F be the set of all those queries that we (the F player) make nonrandom during the construction by playing a move in one of the games. We have that

$$\begin{aligned}\sum_{x\in\{0,1\}^*} 2^{-F(x)} &= \sum_{x\in Z_F} 2^{-(|x|-4)} + \sum_{x\notin Z_F} 2^{-(2|x|+3)} \\ &= \sum_{x\in\{0,1\}^*} 2^{-(2|x|+3)} + \sum_{x\in Z_F} (2^{-(|x|-4)} - 2^{-(2|x|+3)}) \\ &\leq \frac{1}{8} + \sum_{x\in Z_F} (2^{-(|x|-4)} - 2^{(2|x|+3)}).\end{aligned}$$

Thus it now suffices to show that $tot_F = \sum_{x\in Z_F}(2^{-(|x|-4)} - 2^{(2|x|+3)}) \leq 1/4$. Notice that tot_F is exactly the total number of points that the F player accrues in all games throughout the lifetime of the construction. First let us consider those games on which the K player cheats. We know that in all these games, the F player accrues fewer points than the K player, and in particular accrues fewer points during these games than tot_K, the total number of points the K player accrues in all games throughout the lifetime of the construction. Let Z_K be the set of all those queries that the K player makes nonrandom during the construction by playing a move in one of the games. We have that

$$\begin{aligned}tot_K = \sum_{z\in Z_K} 2^{-(|z|-6)} - 2^{-(2|z|+3)} &\leq \sum_{z\in Z_K} 2^{-(K^{t^*}_{U_0}(z)+5)} \\ &\leq \sum_{z\in\{0,1\}^*} 2^{-(K^{t^*}_{U_0}(z)+5)} \leq \frac{1}{32},\end{aligned}$$

where the first inequality uses that for all $z \in Z_K$, $K^{t^*}_{U_0}(z) \leq |z| - 6$, and the last inequality again comes from Proposition 1.7.2.

Now consider games on which K does not cheat – for each R_e there will be exactly one of these. On each of these games the F player can accrue at most $\epsilon = 2^{-e-3}$ points. Thus the total number of points the F player accrues on all games that K does not cheat on is at most $\sum_{e=1}^{\infty} 2^{-e-3} = 1/8$.

Therefore $tot_F \leq 1/32 + 1/8 \leq 1/4$.

■

Chapter 2

Reductions to sparse sets

In this chapter we study the consequences of NP having non-uniform polynomial size circuits of various types. We continue the work of Agrawal and Arvind [2] who study the consequences that would follow if SAT were many-one reducible to functions computable by non-uniform circuits consisting of a single weighted threshold gate (written $\text{SAT} \leq_m^p \text{LT}_1$).

They claim that as a consequence P = NP follows, but unfortunately their proof was incorrect. We present a counter-example to their Splitting Lemma [2, §4, p. 203], due to Amir Shpilka.

We take up this question again and use results from computational learning theory to show that if $\text{SAT} \leq_m^p \text{LT}_1$ then $\text{PH} = \text{P}^{\text{NP}}$.

We furthermore show that if SAT disjunctive truth-table (or majority truth-table) reduces to a sparse set then $\text{SAT} \leq_m^p \text{LT}_1$ and hence a collapse of PH to P^{NP} also follows. Lastly we show several interesting consequences of $\text{SAT} \leq_{dtt}^p$ SPARSE.

The results in this chapter are based on the paper:

- Harry Buhrman, Lance Fortnow, John Hitchcock, and Bruno Loff. Learning reductions to sparse sets. In *Proceedings of the 38th MFCS*, pages 243–253, 2013.

2.1 Introduction

In this chapter we study consequences of NP having non-uniform polynomial size circuits of various types. This question is intimately related to the existence of sparse hard sets for NP under different types of reductions, and has played a central role in complexity theory starting with the work of Berman, Hartmanis, Karp, Lipton and Mahaney [27, 64, 74].

Karp and Lipton showed that if NP is Turing reducible to a sparse set then the polynomial time hierarchy collapses to its second level. This was later improved to a collapse of $\text{PH} = \text{ZPP}^{\text{NP}}$ [65, 31], and finally $\text{PH} = \text{S}_2^p$ [40].[1]

[1] S_2^p is the second Level of the symmetric hierarchy, a class lying somewhere between P^{NP} and ZPP^{NP}; see the complexity zoo [1] for more details.

Improvement of this result to a deeper collapse is a challenging open question whose positive solution would imply new unconditional circuit lower bounds.[2]

Mahaney [74] showed that if SAT reduces many-one to a sparse set then in fact P = NP. This implication was subsequently improved by Ogiwara and Watanabe [85] to bounded truth-table reductions, and later work extended this result to other weak reductions [15, 14, 13, 87, 11, 16, 18]. Notoriously open is to show a similar result for disjunctive truth-table reductions. The best known consequence of this is a collapse of PH to P^{NP} [17].

Agarwal and Arvind [2] studied the consequences of SAT many-one reducing to LT_1, the class of languages accepted by non-uniform circuits consisting of a single weighted linear-threshold gate. They claimed that SAT $\leq_m^p$ LT_1 implies P = NP — unfortunately, the proof in that paper was flawed, as it relied essentially on their incorrect *Splitting Lemma* (p. 203).[3]

We will give a simple counter-example, due to Amir Shpilka, showing that the Splitting Lemma fails. Then we take a fresh look at the SAT $\leq_m^p$ LT_1 setting and connect it with results in learning theory. We use an efficient deterministic algorithm from Maass and Turán [73] for learning half spaces, to obtain a collapse of the polynomial-time hierarchy to P^{NP} from the assumption that SAT $\leq_m^p$ LT_1. The main ingredient in the learning algorithm is the use of linear programming, which also featured prominently in the work of Agrawal and Arvind.

The use of learning theory in this area of complexity theory is not new and was used before by [65, 31, 49, 51], however the use of *deterministic* learning algorithms in relationship with the polynomial time hierarchy is new.

Next we examine the consequences of SAT $\leq_{dtt}^p$ SPARSE and make a link with the geometric approach above. Using the leftset technique from [85] it is easy to show for conjunctive truth-table reductions that if SAT $\leq_{ctt}^p$ SPARSE then P = NP. Frustratingly, for disjunctive truth table reductions the best known consequence is PH = P^{NP}, a result due to Arvind et al.[17], who use a complicated argument. We use error-correcting codes to show that SAT $\leq_{dtt}^p$ SPARSE implies that SAT $\leq_m^p$ LT_1, which with our previous result gives a new and more modular proof of the collapse to P^{NP}. Our new approach enables us to obtain the same collapse for majority reductions.

We finish with a handful of new consequences of SAT $\leq_{dtt}^p$ SPARSE and SAT $\leq_{maj}^p$ SPARSE. Interestingly, if we could improve our results to show that PH = $P_{\|}^{NP}$ follows from the assumption that SAT $\leq_{dtt}^p$ SPARSE, this would allow us to obtain the full collapse to P = NP (under the same assumption).

---

[2]Suppose, for instance, that NP $\subseteq$ P/*poly* implies that PH $\subseteq$ P^{NP}. Then either NP does not have polynomial-size circuits or Σ_2^p is in P^{NP}, and then it follows from a result of Kannan [63] that P^{NP} does not have circuits of fixed-polynomial size, and from Impagliazzo and Wigderson [56] that BPP $\subseteq$ P^{NP}.

[3]The mistake in this *Splitting Lemma* was not seen by any of the paper's referees, but instead was accidentally discovered years later. For an anecdotal account of the episode, please consult `http://blog.computationalcomplexity.org/2009/10/thanks-for-fuzzy-memories.html`.

2.2 Preliminaries

We assume that the reader is familiar with computational complexity, as expounded, for instance, in [21, 10]. Hence we will use the usual notation for complexity classes: P, NP, Σ_i^p, Π_i^p, Δ_i^p, ZPP, *etc.*

The least common notation we use, is: $\mathrm{P}_{\|}^{\mathrm{NP}}$ and $\mathrm{FP}_{\|}^{\mathrm{NP}}$, which are the classes of sets and functions, respectively, that are polynomial-time computable with non-adaptive queries to an NP oracle; $\mathrm{P}^{\mathrm{NP}[q]}$, and $\mathrm{FP}^{\mathrm{NP}[q]}$, the classes of sets and functions that are polynomial-time computable by asking no more than $q(n)$ (possibly adaptive) queries to an NP oracle.

A linear threshold function $L : \{0,1\}^m \to \{0,1\}$ is defined by a vector of m real numbers $w \in \mathbb{R}^m$, called weights, a threshold $\theta \in \mathbb{R}$, and the equation

$$L(z) = \begin{cases} 1 & \text{if } z \cdot w > \theta, \text{ and} \\ 0 & \text{if } z \cdot w \leq \theta. \end{cases}$$

Here $z \cdot w$ denotes the inner product $\sum_{i=1}^{m} z_i w_i$.

We let $\mathrm{LT}_1(m)$ denote the class of linear-threshold functions with m-bit binary inputs. We may freely assume, for functions in $\mathrm{LT}_1(m)$, that the weights and thresholds are integers of bit-length $m \log m$ [78, Thm. 16].

The following lemma is well-known [see 21]:

2.2.1. Lemma.

$$A \in \mathrm{P}/\mathit{poly} \iff A \in \mathrm{P}^{\mathrm{SPARSE}},$$

This result implies that a polynomial-time oracle reduction to a sparse set can be seen as a polynomial-time circuit. In this chapter we will study reductions to sparse sets for weaker notions of reductions. The weaker the reduction, the weaker the access to non-uniformity.

In this paper we are concerned with three such reductions:

2.2.2. Definition. **(*dtt* reductions)** A set A disjunctive truth-table reduces to a set S, written $A \leq_{dtt}^{p} S$, if there exists a polytime computable function Q, outputting a set of queries, such that

$$x \in A \iff Q(x) \cap S \neq \varnothing.$$

(majority reductions) A set A majority truth-table reduces to a set S, written $A \leq_{maj}^{p} S$, if there exists a function Q, as above, such that

$$x \in A \iff |Q(x) \cap S| > \frac{|Q(x)|}{2}$$

(LT_1 reductions) A set A reduces to linear-threshold functions, written $A \leq_{m}^{p} \mathrm{LT}_1$, if there exists a polytime computable function f, and a family $\{L_n\}_{n\in\mathbb{N}}$ of linear threshold functions, such that[4]

$$x \in A^{=n} \iff L_n(f(x)) = 1.$$

[4]Notice that the length of $f(x)$ must be a function of the length of x.

2.3 A counter example to the Splitting Lemma

Here is a construction due to Amir Shpilka. Suppose we he have a set of m row vectors $\{d_i\}_{i=1}^m$ in $\mathbb{Q}^n$, together with two sets of vectors $\{q_i\}_{i=1}^m$ and $\{q_{i,j}\}_{i,j=1}^m$ in $\mathbb{Q}^n$, such that

1. For each i, $d_i \cdot q_i > 0$ and $d_\ell \cdot q_i < 0$ if $\ell \neq i$; and
2. For each i, j, $d_i \cdot q_{i,j} > 0$, and $d_j \cdot q_{i,j} > 0$, but $d_\ell \cdot q_{i,j} < 0$ if $\ell \notin \{i, j\}$.

In the nomenclature of [2], the d_i's *split* the set of q_i and $q_{i,j}$'s. Then the Splitting Lemma claimed that $m \leq n + 1$. Now we present a counter-example showing that actually m can be infinite whenever $n \geq 5$.

For natural numbers i, j, define q_i and $q_{i,j}$ as the first n coefficients of the polynomials:

$$p_i(x) = 1 - 2(x - i)^2 = \sum_{k=0}^{n-1} q_i(k)x^k$$

and

$$p_{i,j}(x) = 1 - 2(x - i)^2(x - j)^2 = \sum_{k=0}^{n-1} q_{i,j}(k)x^k.$$

Then for $d_i = (i^0, i^1, \ldots, i^{n-1})$, whenever $n \geq 5$ (the number of non-zero coefficients of $p_{i,j}$), it will hold that $d_\ell \cdot q_i = p_i(\ell)$ and $d_\ell \cdot q_{i,j} = p_{i,j}(\ell)$. Now, for the infinity of points $\{d_i\}_{i\geq 1}$, $\{q_i\}_{i\geq 1}$, and $\{q_{i,j}\}_{i,j\geq 1}$, it is clear by inspection that the splitting property holds.

2.4 If $\text{SAT} \leq_m^p \text{LT}_1$ ...

Attempting to derive P = NP from $\text{SAT} \leq_m^p \text{LT}_1$ should prove difficult, since by the results in the next section this would imply the same collapse for *dtt* and majority reductions to sparse sets, and this problem has been open for a long time. $A \in \text{P}/poly$ follows from $A \leq_m^p \text{LT}_1$, and so from $\text{SAT} \leq_m^p \text{LT}_1$ and [40] we get $\text{PH} = \text{S}_2^p$. This collapse can be improved in the following way:

2.4.1. THEOREM. *If* $\text{SAT} \leq_m^p \text{LT}_1$, *then* $\text{PH} = \text{P}^{\text{NP}}$.

We take a similar approach as [31]: the existence of a suitable learning algorithm will, under the assumption that $\text{SAT} \leq_m^p \text{LT}_1$, collapse the polynomial-time hierarchy. The difference is that we have a *deterministic* learning algorithm for linear threshold functions, but only (zero-error) probabilistic algorithms with access to an NP oracle are known that can learn general circuits.

Our learning model is the online learning model of Angluin [9] for learning with counter-examples. In our case, the learner wishes to identify an unknown linear threshold function, say $L \in \text{LT}_1(m)$. At each learning step, the algorithm proposes some hypothesis $H \in \text{LT}_1(m)$. If $H \neq L$, then the algorithm is given a counter-example x such that $H(x) \neq L(x)$. The algorithm is not allowed to make any assumptions on the choice of counter-example, which could very

well be adversarially chosen. Based on the previous counter-examples and hypotheses, the algorithm suggests a new hypothesis which is correct on the inputs seen so far, and the process is repeated until $H = L$. The learning complexity of such an algorithm is the maximum number of such steps that it will need in order to learn any function in $\mathrm{LT}_1(m)$.

2.4.2. THEOREM ([73]). *There is a deterministic polynomial-time algorithm for learning* $\mathrm{LT}_1(m)$ *functions in* $O(m^3 \log m)$ *steps.*

As a corollary, we will be able to prove Theorem 2.4.1, and the forthcoming Theorem 2.4.3. It should be noted that both of these theorems hold for polynomial-time many-one reductions to any class of functions which, like LT_1, have a polynomial-time algorithm for learning with counter-examples.

Proof of of Theorem 2.4.1. Suppose SAT $\leq_m^p$ LT_1, and let L_n be a family of linear threshold functions, and f a polytime reduction, such that

$$\psi \in \mathrm{SAT}^{=n} \iff L_n(f(\psi)) = 1. \tag{2.1}$$

For a given formula of length n, we use the algorithm of Theorem 2.4.2 in order to uncover a linear threshold function H with the same property (2.1) as L_n, in polynomial time with the help of an NP oracle.

Let $m = |f(\psi)|$ on inputs ψ of length n. We proceed as follows: we start with an initial hypothesis H for L_n, given by the learning algorithm for $\mathrm{LT}_1(m)$. Then at each step in the learning process we ask the NP oracle if there exists some formula ψ of length n such that:

1. ψ has no variables and evaluates to true, but $H(f(\psi)) = 0$, or
2. ψ has no variables and evaluates to false, but $H(f(\psi)) = 1$, or
3. $H(f(\psi)) = 1$ but both $H(f(\psi_0)) = 0$ and $H(f(\psi_1)) = 0$, or
4. $H(f(\psi)) = 0$, but $H(f(\psi_0)) = 1$ or $H(f(\psi_1)) = 1$.

Above, ψ_0 and ψ_1 are obtained by replacing the first variable of ψ respectively with 0 or 1. Essentially, we are asking whether the set $\mathrm{SAT}(H) = \{\psi \mid H(f(\psi)) = 1\}$ violates the self-reducibility of SAT. If this is not the case, then necessarily $\mathrm{SAT}(H) = \mathrm{SAT}^{=n}$ — we have found a hypothesis H that is good enough to decide $\mathrm{SAT}^{=n}$, so we are done.

But if the self-reducibility is violated, then for at least one $\phi \in \{\psi, \psi_0, \psi_1\}$, we must have $H(f(\phi)) \neq L_n(f(\phi))$, and so $f(\phi)$ gives us a counter-example to update the hypothesis H. We use prefix-search to obtain such a formula ϕ, and from equation (2.1) this will provide us with a counter-example, i.e., $H(f(\phi)) \neq L_n(f(\phi))$.

After $O(m^3 \log m) = \mathrm{poly}(n)$ many iterations, we will either have learned L_n, or otherwise obtained an hypothesis H suitable for the purpose of querying $\mathrm{SAT}^{=n}$.

It now holds that if the NP oracle is given the bits representing H, it can use these bits to decide $\mathrm{SAT}^{=n}$, and it can then simulate an $\mathrm{NP}^{\mathrm{NP}} = \Sigma_2^p$ oracle. So Σ_2^p, and consequently all of PH, collapses to P^{NP}. □ ■

In order to solve $\text{SAT}^{=n}$, the algorithm above potentially asks $\Omega(nm^3 \log m)$ adaptive queries to SAT. We can be a bit more clever, and actually reduce this number to n. This will give us the following:

2.4.3. THEOREM. *If* $\text{SAT} \leq_m^p \text{LT}_1$, *then* $\text{NP}^{\text{SAT}^{=n}} \subseteq \text{P}^{\text{SAT}[n]} \cap \text{NP/}lin$.

Proof. The idea is to use the self-reducibility of SAT once again, in order to learn $\text{SAT}^{=n}$ first for formulas with no variables (formulas over constants, which evaluate to true or false), then for formulas with 1 variable, then 2 variables, and so on. Let $\text{SAT}_k^{=n}$ be the set of satisfiable formulas having exactly k variables. Starting with the initial hypothesis H, we set out to learn $\text{SAT}_0^{=n}$. What is the largest number of mistakes that we can make, i.e., how many times might we need to change our hypothesis H until we have properly learned $\text{SAT}_0^{=n}$?

Using a SAT oracle, we can ask: *is there a sequence* $\psi_1, \ldots, \psi_\ell$ *of* ℓ *formulas, having* 0 *variables, such that* ψ_{i+1} *is always a counter-example to the hypothesis constructed by our learning algorithm after seeing* $\psi_1, \ldots, \psi_i$*?*[5]

We know that such a sequence will have at most $\text{poly}(n)$ formulas, and so using binary search and with $O(\log n)$ such queries, we can find the length of the largest sequence of counter-examples which can be given to our learning algorithm before it necessarily learns $\text{SAT}_0^{=n}$. Let this length be ℓ_0.

Then because ℓ_0 is maximal, at this point we know that if the learning algorithm is given *any* sequence of ℓ_0 counter-examples having no variables, the constructed hypothesis H will be correct on $\text{SAT}_0^{=n}$, in the sense that $\psi \in \text{SAT}_0^{=n} \iff H(f(\psi)) = 1$.

Now that we know ℓ_0, we set out to learn $\text{SAT}_1^{=n}$. Using SAT as an oracle, we may ask: *Is there a sequence of* ℓ_0 *counter-examples with* 0 *variables, followed by* ℓ *counter-examples with* 1 *variable?* Thus we may obtain ℓ_1, the length of the largest sequence of counter-examples with 1 var, that can be given to the learning algorithm *after it has already learned every possible formula with* 0 *variables.*

In general we know $\ell_0, \ldots, \ell_{k-1}$, and we set out to learn $\text{SAT}_k^{=n}$. Using SAT as an oracle, we ask: *Is there a sequence of* ℓ_0 *counter-examples with* 0*-variables, followed by* ℓ_1 *counter-examples with* 1*-variable, ..., followed by* ℓ_{k-1} *counter-examples with* $k-1$ *variables, followed by* ℓ *counter-examples with* k *variables?*

The key observation is that in order for the SAT oracle to be able to tell whether a formula ψ with k variables is a counter-example to hypothesis H, i.e., whether $H(f(\psi)) \neq \text{SAT}(\psi)$, it will need to know whether ψ is or is not satisfiable. In order to know this, the SAT oracle uses H itself, which at this point is known to be correct for formulas with $k-1$ variables, and thus $\psi \in \text{SAT} \iff H(f(\psi_0)) = 1$ or $H(f(\psi_1)) = 1$.

[5] Formalizing the question as an NP-set gives us:

$$A = \{\langle 0^n, 0^\ell \rangle \mid \exists \bar{\psi}, \bar{H} \forall i \; H_i = \text{Learner}(\psi_1, \ldots, \psi_i) \wedge H_{i-1}(f(\psi_i)) \neq \text{SAT}(\psi_i)\},$$

where $\bar{\psi}$ is a sequence of ℓ formulas with 0 variables, $\bar{H}$ is a sequence of ℓ threshold functions, and $i \in \{1, \ldots, \ell\}$. Notice that $H_{i-1}(f(\psi_i)) \neq \text{SAT}(\psi_i)$ is decidable in polynomial time because the formulas ψ_i have no variables.

In the end we have $n+1$ numbers $\ell_0, \ldots, \ell_n$, and we know that if the learning algorithm is given *any* sequence of ℓ_0 counter-examples having no variables, followed by ℓ_1 counter-examples having 1 variable, ..., followed by ℓ_n counter-examples having n variables, then the constructed hypothesis H will be correct on all of $\mathrm{SAT}^{=n}$. Furthermore, such a sequence must exist by construction.

These numbers take up at most $O(n \log n)$ bits, and each bit is the outcome of one (much larger, adaptive) query to SAT. Having access to $\ell_0, \ldots, \ell_n$, an NP machine can guess a proper sequence of counter-examples, and it will thus obtain an hypothesis H which it can use to answer any query to $\mathrm{SAT}^{=n}$. Thus $\mathrm{NP}^{\mathrm{SAT}^{=n}} \subseteq \mathrm{P}^{\mathrm{SAT}[n \log n]}$, and $\mathrm{NP}^{\mathrm{SAT}^{=n}} \subseteq \mathrm{NP}/n \log n$.

In order to improve $n \log n$ to n bits, or even $\frac{n}{c \log n}$ bits, the proof is similar, but instead of learning how to decide $\mathrm{SAT}^{=n}$ for one extra variable at a time, we learn $O(\log n)$ extra variables at a time — this requires us to unfold the self-reduction tree $O(\log n)$-deep. ■

Under the assumption that SAT has polynomial-size circuits, we may decide, in coNP, whether a given string $\alpha(n)$ encodes a circuit correct for $\mathrm{SAT}^{=n}$. However, there will possibly be many strings with this property — the following theorem gives us a way to single out, in coNP, a *unique* advice string $\alpha(n)$ suitable to decide $\mathrm{SAT}^{=n}$.

2.4.4. THEOREM. *If* $\mathrm{NP} \subseteq \mathrm{P}/poly$, *and* $\mathrm{PH} \subseteq \mathrm{P}^{\mathrm{NP}}$, *then* $\mathrm{PH} \subseteq \mathrm{P}/\alpha$ *for some polynomial advice function* $0^n \mapsto \alpha(n)$ *whose graph* $G_\alpha = \{\langle 0^n, \alpha(n)\rangle | n \in \mathbb{N}\} \in$ coNP.

Proof. Let A be Δ_2-complete.[6] Then there is a polytime machine $\mathcal{M}$ that decides $A^{=n}$ with polynomially-long advice $\gamma(n)$, where $\gamma(n)$ codes a circuit solving $\mathrm{SAT}^{=m}$, for some $m = \mathrm{poly}(n)$. The machine $\mathcal{M}$ uses $\gamma(n)$ to answer the queries needed in the Δ_2 computation of A. Furthermore, the function $0^n \mapsto \tilde{\alpha}(n)$, given by

$$\begin{aligned} \tilde{\alpha}(n) \quad & \text{is the lexicographically smallest string} \\ & \text{such that } x \in A^{=n} \iff \mathcal{M}(x)/\tilde{\alpha}(n) = 1, \end{aligned}$$

is in PH and thus in $\mathrm{FP}^{\mathrm{SAT}}$. Then let $\mathcal{N}$ be a polytime machine computing $\tilde{\alpha}$ with a SAT oracle, and let's say it makes k queries to compute $\tilde{\alpha}(n)$. Let $S \in$ coNP be the set of strings $\langle 0^n, \tilde{\alpha}, a_1, \ldots, a_k, y_1, \ldots, y_k\rangle$ such that

1. $\mathcal{N}^{\bar{a}}(0^n) = \tilde{\alpha}$ (i.e., when $a_1, \ldots, a_k$ are given as answers to the queries of $\mathcal{N}$),
2. if $a_i = 1$ then y_i is the lexicographically smallest satisfying assignment of the i-th formula queried by $\mathcal{N}^{\bar{a}}$, and
3. if $a_i = 0$ then $y_i = \lambda$ (the empty string) and the i-th formula queried by $\mathcal{N}^{\bar{a}}$ is not satisfiable.

[6] For instance, A is the set of pairs $\langle \psi, z\rangle$ such that z prefixes the lexicographically-least satisfying assignment to the boolean formula ψ.

Notice that for a given n, the string $\langle 0^n, \tilde{\alpha}, a_1, \ldots, a_k, y_1, \ldots, y_k\rangle \in S$ is uniquely defined, so S is the graph of $\alpha(n) = \langle \tilde{\alpha}, a_1, \ldots, a_k, y_1, \ldots, y_k\rangle$. When given $\alpha(n)$, an algorithm for A can simply check if $\mathcal{M}(x)/\tilde{\alpha} = 1$. ■

2.4.5. Corollary. *If* SAT $\leq_m^p$ LT_1, *then* PH $\subseteq$ P/α *for some polynomial advice function* $0^n \mapsto \alpha(n)$ *whose graph* $G_\alpha \in$ coNP.

2.5 LT_1 versus *dtt* and *maj* reductions

In this section we show that LT_1 reductions can simulate *dtt* and majority reductions to sparse sets. Thus, effectively, the collapses we have proven for LT_1 reductions imply similar collapses for *dtt* and majority reductions.

2.5.1. Theorem. *If* $A \leq_{dtt}^p$ SPARSE *or* $A \leq_{maj}^p$ SPARSE, *then* $A \leq_m^p \mathrm{LT}_1$.

Proof. We will use a Reed-Solomon code to construct the LT_1 reduction. Suppose $A \leq_{dtt}^p S \in$ SPARSE, and assume w.l.o.g. that the *dtt* reduction is given by a polytime computable function Q, such that

$$x \in A^{=n} \iff S^{=m} \cap Q(x) \neq \varnothing, \tag{2.2}$$

$$|S^{=m}| = m, \text{ and}$$

$$|Q(x)| = d.$$

That is, for every input x of length n, $Q(x) = \{y_1, \ldots, y_d\}$ always queries the same number of $d = d(n)$ strings of the same length $m = m(n)$, and that there will be exactly m strings in $S^{=m}$. Such an assumption can always be made by tweaking the reduction and changing S accordingly.

We will be working over the field $\mathbb{F}_{2^\ell}$, for $\ell \geq \lceil \log dm^2 \rceil$. For any given binary string s of length m, we define the polynomial $p_s(z) = \sum_{i=1}^m s_i z^{i-1}$. Now let $C(s)$ be the encoding of s as a $2^\ell \times 2^\ell$-long binary string: this string is the concatenation of $p_s(a)$, as a goes through all the 2^ℓ elements of $\mathbb{F}_{2^\ell}$; each $p_s(a)$ is in turn encoded by a binary string of length 2^ℓ, having a 1 at position $p_s(a)$ (for some fixed enumeration of $\mathbb{F}_{2^\ell}$), and 0s elsewhere.

Note that $|C(s)| = O(d^2m^4) = \mathrm{poly}(n)$. Then vitally note that by encoding strings this way, the number of bit positions where $C(s)$ and $C(y)$ are both 1, given by the inner product $C(s) \cdot C(y)$,[7] is exactly the number of elements $a \in \mathbb{F}_{2^\ell}$ where $p_s(a) = p_y(a)$. So for any two words $s, y \in \{0,1\}^m$, using the fact that $p_s - p_y$ is either identically zero, or has at most $m-1$ roots,

$$\begin{cases} C(y) \cdot C(s) \leq m-1 & \text{if } y \neq s, \text{ and} \\ C(y) \cdot C(s) \geq dm^2 & \text{if } y = s. \end{cases}$$

[7] Note that the binary strings $C(s)$ and $C(y)$ are seen as 0-1 vectors, and that the inner product is a natural number $\sum_{j=1}^{2^\ell \times 2^\ell} C(s)_j C(y)_j$.

Define $g(x) = \bigvee_{i=1}^{d} C(y_i)$, where $Q(x) = \{y_1, \ldots, y_d\}$, and by $\bigvee$ we mean bitwise-OR. Then

$$\begin{cases} g(x) \cdot C(s) \leq \sum_{i=1}^{d} C(y_i) \cdot C(s) \leq d(m-1) & \text{if } s \notin Q(x), \text{ and} \\ g(x) \cdot C(s) \geq dm^2 & \text{if } s \in Q(x). \end{cases}$$

Finally, let $w_n = \oplus_{s \in S^{=m}} C(s)$, and $f(x) = (g(x))^{\oplus m}$, where by $\oplus$ we mean the direct sum of vectors / concatenation of strings. Then $f(x) \cdot w_n = \sum_{s \in S^{=m}} g(x) \cdot C(s)$, and we come to

$$\begin{cases} f(x) \cdot w_n \leq md(m-1) & \text{if } S^{=m} \cap Q(x) = \varnothing, \text{ and} \\ f(x) \cdot w_n \geq dm^2 & \text{if } S^{=m} \cap Q(x) \neq \varnothing. \end{cases} \tag{2.3}$$

So $x \in A \iff f(x) \cdot w_n > dm(m-1)$, showing that $A \leq^p_m \mathrm{LT}_1$.

The transformation for *maj* reductions is similar. We begin with a *dtt* reduction function Q, which is like before, except that now Equation (2.2) is replaced with

$$x \in A^{=n} \iff |S^{=m} \cap Q(x)| > \frac{d}{2}.$$

Then both the LT_1 reduction function f, and the set of weights w_n are constructed exactly in the same way, but over a slightly larger field. Working through the proof, if 2^ℓ is the size of our chosen field, and $K = |S^{=m} \cap Q(x)|$, then Equation (2.3) becomes:

$$2^\ell K \leq f(x) \cdot w_n \leq 2^\ell K + d(m-1)(m-K).$$

Now choose $\ell \geq \lceil \log 4dm^2 \rceil$ as the size of our field. Using the defining property of the *maj* reduction, a small computation will show us that

$$x \in A^{=n} \iff K > \frac{d}{2} \iff f(x) \cdot w_n > 2^\ell \left(\frac{d}{2} + \frac{1}{4} \right)$$

— this defines our LT_1 reduction. ■

2.6 If SAT $\leq^p_{dtt}$ SPARSE ...

Disjunctive truth-table reductions to sparse sets are powerful enough to simulate bounded truth-table reductions to sparse sets [3]. But the collapses that are known, under the assumption that SAT $\leq^p_{dtt}$ SPARSE, are not as strong as those for btt reductions. We can summarize what was known about SAT $\leq^p_{dtt}$ SPARSE, in the following two theorems:

2.6.1. CONSEQUENCE ([41, 12]). ... *then* $\mathrm{FP}^{\mathrm{NP}}_{\|} = \mathrm{FP}^{\mathrm{NP[log]}}$, $\mathrm{UP} \subseteq \mathrm{P}$, *and* $\mathrm{NP} = \mathrm{RP}$.

2.6.2. CONSEQUENCE ([17]). ... *then* $\mathrm{PH} = \mathrm{P}^{\mathrm{NP}} = \mathrm{P}^{\mathrm{RP}} = \mathrm{BPP}$.

To these consequences, we append our own observations, which follow from the results in the previous sections.

2.6.3. Consequence. *... then* $\mathrm{NP}^{\mathrm{SAT}^{=n}} \subseteq \mathrm{P}^{\mathrm{SAT}[n]}$, $\mathrm{NP}^{\mathrm{SAT}^{=n}} \subseteq \mathrm{NP}/\mathit{lin}$.

2.6.4. Consequence. *... then* $\mathrm{PH} \subseteq \mathrm{P}/\alpha$ *for some function* $0^n \mapsto \alpha(n)$ *whose graph* $G_\alpha \in \mathrm{coNP}$.

Finally, we note that we are not far away from obtaining the final consequence $\mathrm{P} = \mathrm{NP}$.

2.6.5. Consequence. *... then* $\mathrm{EXP} \not\subseteq \mathrm{NP}/\log$.

2.6.6. Consequence. *... then* $\mathrm{E}^{\mathrm{NP}} \not\subseteq \mathrm{SIZE}(2^{\varepsilon n})$ *for some* $\varepsilon > 0$.

2.6.7. Consequence. *... then the following statements are all equivalent:*

1. $\mathrm{P} = \mathrm{NP}$.
2. $\mathrm{P}^{\mathrm{NP}} = \mathrm{P}_{\|}^{\mathrm{NP}}$.
3. $\mathrm{coNP} \cap \mathrm{SPARSE} \subseteq \mathrm{NP}$.
4. $\mathrm{E}^{\mathrm{NP}} = \mathrm{E}_{\|}^{\mathrm{NP}}$.

Proof of of Consequence 2.6.5. [47] show that

$$\mathrm{EXP} \subseteq \mathrm{P}_{\|}^{\mathrm{NP}} \iff \mathrm{EXP} \subseteq \mathrm{NP}/\log.$$

But if we had $\mathrm{EXP} \subseteq \mathrm{P}_{\|}^{\mathrm{NP}}$, then we could compute the lexicographically least satisfying assignment of a given formula in $\mathrm{FP}_{\|}^{\mathrm{NP}}$, and thus in $\mathrm{FP}^{\mathrm{NP}[\log]}$, by Consequence 2.6.1. But then we could also do it in FP alone, simply by trying every possible answer to the queries made by the $\mathrm{FP}^{\mathrm{NP}[\log]}$ computation. But then $\mathrm{P} = \mathrm{NP}$, and the necessary conclusion $\mathrm{EXP} \subseteq \mathrm{PH} \subseteq \mathrm{P}$ would contradict the time-hierarchy theorem. ■

Proof of of Consequence 2.6.6. By counting there is a function $f : \{0,1\}^{\log n} \to \{0,1\} \notin \mathrm{SIZE}(n^\varepsilon)$ which can be found in P^{Σ_2} [cf. 63], and thus, by Consequence 2.6.2, in P^{NP}. Translating this upwards we get a set in E^{NP} with no circuits of size $2^{\varepsilon n}$. ■

Proof of of Consequence 2.6.7. All of the items (2, 3, 4) follow trivially from $\mathrm{P} = \mathrm{NP}$. We now prove each converse in turn.

(2⇒1) As in the proof of Consequence 2.6.5, $\mathrm{P} = \mathrm{NP}$ follows if we are able to compute the least satisfying assignment of a given formula in $\mathrm{FP}_{\|}^{\mathrm{NP}}$. This is trivially the case when $\mathrm{P}^{\mathrm{NP}} = \mathrm{P}_{\|}^{\mathrm{NP}}$.

(3⇒1) If $\mathrm{SPARSE} \cap \mathrm{coNP} \subseteq \mathrm{NP}$, then, from Consequence 2.6.4, we get $\mathrm{PH} \subseteq \mathrm{NP}^{G_\alpha} \subseteq \mathrm{NP}^{\mathrm{NP} \cap \mathrm{SPARSE}}$: the non-deterministic machine just guesses the advice α and checks it using the oracle. But $\mathrm{NP}^{\mathrm{NP} \cap \mathrm{SPARSE}} \subseteq \mathrm{NP}$ [cf. 62], and thus the least satisfying assignment of a given formula can be obtained in $\mathrm{FP}_{\|}^{\mathrm{NP}}$.

(4⇒1) To see the third equivalence, notice that $\mathrm{E}^{\mathrm{NP}} = \mathrm{E}_{\|}^{\mathrm{NP}}$, which together with Consequence 2.6.6 implies we can derandomize BPP in $\mathrm{P}_{\|}^{\mathrm{NP}}$ [cf. 84, 56,

the proof is analogous to that in §1.4]. If we further take into account that PH $\subseteq$ BPP, by Consequence 2.6.2, then it follows that PH $\subseteq \mathrm{P}^{\mathrm{NP}}_{\|}$, and the least satisfying assignment can be found in $\mathrm{FP}^{\mathrm{NP}}_{\|}$. ■

Chapter 3
Hardness results for Knapsack problems

In this chapter we show various hardness results for knapsack and related problems; in particular we will show that unless the Exponential-Time Hypothesis is false, subset-sum cannot be approximated any better than with an FPTAS. We also provide new unconditional lower bounds for approximating knapsack in Ketan Mulmuley's parallel PRAM model. Furthermore, we give a simple new algorithm for approximating knapsack and subset-sum, that can be adapted to work for small space, or in small parallel time.

The results in this chapter are based on the paper:

- Harry Buhrman, Bruno Loff, and Leen Torenvliet. New hardness results for knapsack problems. Submitted.

3.1 Introduction

The Knapsack problem is a natural example of an NP-complete optimization problem which nevertheless has a fully-polynomial time approximation scheme (FPTAS). However, there is no *a priori* reason to think that FPTAS would be the best one could hope for. In fact, in order to prove NP-hardness of knapsack we require the problem to be solved exactly, so in principle there could exist polynomial-time approximation algorithms for knapsack with an approximation ratio strictly closer to 1 than inverse-polynomial.

We begin this work by giving evidence that inverse-polynomial is as good an approximation as we are likely to get. For example, we obtain the following:

3.1.1. Proposition. *If there is a polynomial-time algorithm to approximate knapsack with inverse-super-polynomial error ratio, then we can decide the satisfiability of n-variable* NC *circuits of size $\omega(n)$ in time $2^{o(n)}$.*

In a way, this is a refinement of the NP-hardness of the knapsack problem. The reduction actually offers a robust trade-off, in that algorithms with a successively better approximation ratio can be used to simulate successively larger non-deterministic NC-computations.

A reduction from 3Sat instead of NC circuit satisfiability was independently discovered by Cygan et al. [45]. Their proof is more contrived, as it is

the result of several intermediate reductions, and their result is slightly weaker, though some of the reduction gadgets they use have a similar flavor to our own. Our proof, however, is simple, short and direct.

From the proposition above, and the sparsification lemma of [57], we get, in particular, a $2^{o(n)}$-time algorithm for 3SAT from the assumption that such a good approximation can be obtained in polynomial time. This would contradicting the Exponential–Time Hypothesis [55]. This is the first hardness-of-approximation result of its kind, since hardness of approximation has been shown for every other approximation ratio (cf. Section 3.4). This also follows as a consequence of [45], although in that paper the result was not clearly formulated.

The techniques in these proofs will allow us to prove a new unconditional lower bound for solving knapsack in parallel, using Mulmuley's parametric complexity technique [77]. Our reduction techniques can be used to show that the knapsack problem has high parametric complexity, even when the bit-lengths of the inputs are small. It will follow that it is impossible to approximate knapsack within a factor of $1+\varepsilon$ in time $o((\log \frac{1}{\varepsilon})^{\frac{1}{4}})$, and using $2^{o((\log \frac{1}{\varepsilon})^{\frac{1}{4}})}$ processors, in Mulmuley's parallel PRAM model without bit operations.

To complement these hardness results, we provide a simple, and to our knowledge new, algorithm for approximating knapsack and subset-sum in parallel, obtained through a mix of two standard results: the meet-in-the-middle algorithm for knapsack [53], and the PSPACE completeness of alternating time [93]. We are able to approximate subset-sum up to error ratio $1+\varepsilon$ in space $\log \frac{1}{\varepsilon} \cdot \log n$. A linear arithmetic PRAM [see 77] can compute it in time $\log n$ using $(\frac{1}{\varepsilon})^{\log n}$ processors. Alternatively, it may be implemented by an $O(\log n)$-depth AC circuit of size $(\frac{1}{\varepsilon})^{\log n}$.

After some preliminaries in Section 3.2, we study in Section 3.3 both old and new approximation algorithms for knapsack and related problems. In Section 3.4 we show how to reduce satisfiability problems to subset-sum with small bit-weights, and then proceed with the lower-bounds for Mulmuley's model in Section 3.5.

3.2 Preliminaries

For a given natural number n, $[n]$ denotes the set $\{1,\ldots,n\}$, and $\mathbb{S}_n$ denotes the group of permutations of n letters. For $\sigma_1,\sigma_2 \in \mathbb{S}_n$, the permutation $\sigma_1\sigma_2$ is defined by $(\sigma_1\sigma_2)(i) = \sigma_2(\sigma_1(i))$ for all $i \in [n]$. For a given predicate P, we will use the notation $[P?]$ to denote 0 if P is false, and 1 if P is true.

3.2.1 Circuit classes and bounded non-determinism

A *width-5 permutation branching program* P over k input bits $y_1,\ldots,y_k$ is a set (of so-called *instructions*) $\{(j,z_j,\alpha_j,\beta_j)\}_{j=1}^{S}$, with $z_j \in [k]$, $\alpha_j,\beta_j \in \mathbb{S}_5$. For a given input $\bar{y} \in \{0,1\}^k$, the evaluation of the program $P(\bar{y}) \in \mathbb{S}_5$ is equal to $\gamma_1\gamma_2\ldots\gamma_S$, where γ_j equals α_j if $y_{z_j}=0$ and equals β_j if $y_{z_j}=1$. We usually

write $P(\bar{y}) = 1$ when $P(\bar{y}) = e$ (the identity) and write $P(\bar{y}) = 0$ to mean $P(\bar{y}) \neq e$.

We will let NC-SAT(S, k) denote the set of circuits C composed of k boolean input gates and S fan-in-2 NAND gates, such that $C(\bar{y}) = 1$ for some choice of $\bar{y} \in \{0,1\}^k$. We use $\exists\mathrm{NC}_1(k)$ to denote the class of sets accepted by log-depth fan-in-2 uniformly generated circuits of polynomial size which make use of k non-deterministic bits — i.e., a set A is in $\exists\mathrm{NC}_1(k)$ if there is a uniform family of polysize boolean formulae F_n such that $x \in A$ iff $F_{|x|}(x, y) = 1$ for some $y \in \{0,1\}^k$.[1] Finally, we let NC_1-SAT(S, k) denote the set of satisfiable size-S formulae with k boolean variables. Notice that NC_1-SAT(S, k) denotes a set and $\exists\mathrm{NC}_1(k)$ denotes a class of sets for which NC_1-SAT$(\mathrm{poly}(n), k)$ is a complete problem.

3.2.2 Mulmuley's parallel model of computation

Mulmuley's model is a semi-algebraic model of parallel computation. The inputs are usually thought of as having a binary part, giving the combinatorial structure (for instance, the adjacency matrix of a graph), and an integer part, giving numerical data about this structure (such as the weights of the edges of said graph). The model treats these two types of data differently with respect to pointer jumping, which makes its full description more involved than it needs to be for our purpose. In fact, the knapsack problem has no combinatorial structure, and so we can look at a simpler form of Mulmuley's model, and leave the full details to [77].

A *(numerical) problem* in this setting is a family A_n of subsets of $\mathbb{Z}^n$. The model of computation is the *arithmetic PRAM*, a device composed of a certain number of registers and processors. At the beginning of the computation, a tuple $\bar{a} = (a_1, \ldots, a_n)$ is given as input by placing each a_i in register i. Each processor is given a numbered sequence of instructions, each of which is one of the following:

- $w = u \circ v$, where w is a register and u and v are either registers or constants, and $\circ$ is one of $+, -, \times$;
- If register i is greater than zero, then go to instruction ℓ, or else go to instruction ℓ'.

At every time-step, each processor executes its current instruction simultaneously; we assume that concurrent reads and writes are OK, and are handled, for instance, by ordering the processors according to some priority. All instructions have unit cost, which actually means that the model can handle very large numbers. The machine eventually halts, for instance by letting processor 1 execute a special instruction, and we say that $\bar{a}$ was accepted if register 1 holds a zero (meaning $\bar{a} \in A$), and was rejected otherwise.

It should be noted that division is not part of the basic instructions. As a result of this, it can be proven that the model cannot know the least significant bit of an n-bit register in $o(\sqrt{n})$ steps and using $2^{o(\sqrt{n})}$ processors; because of

[1] We may assume that such formula have logarithmic-depth [38, 39].

this, the model is sometimes called *PRAM without bit operations*. This is an algebraic model, in that the contents of the registers at any given time-step are essentially polynomials in the numerical parameters given as the input.[2]

Now we may define *Ketan's class* $\mathrm{KC}(P,T)$ as the class of problems A which can be decided by such a device using P processors and T time. An algebraic algorithm will have P and T depend only on n, whereas a semi-algebraic algorithm will have them depend on the bit-length of the integers a_i. This model is quite powerful, and in fact it is capable of implementing every parallel algorithm that the we know of. Nevertheless, Mulmuley [77] shows that the decision version of maximum flow is not in $\mathrm{KC}(2^{\frac{\sqrt{n}}{a}}, \frac{\sqrt{n}}{a})$, for some constant a, even when the bit-lengths of the flow capacities are at most $O(n^2)$. The result extends to any numerical problem, such as the traveling salesman problem, to which max-flow reduces by a parallel algebraic reduction. But subset-sum is not known to be such a problem.[3] In the paper [90], it was shown that subset-sum is not in $\mathrm{KC}(2^{\frac{\sqrt{n}}{a}}, \frac{\sqrt{n}}{a})$, but using a different technique than Mulmuley's, which gives no limit on the bit-length of the inputs among which a hard instance will be found (we could say that the lower bound was proven for algebraic algorithms, and not for semi-algebraic algorithms). In Section 3.5 we prove that subset-sum is not in $\mathrm{KC}(2^{\frac{1}{a}n^{1/4}}, \frac{1}{a}n^{1/4})$, even for bit-length of the inputs bounded by $O(n^2)$.

3.2.3 Knapsack and generalizations

As part of our study we will include a variant of the knapsack problem which we call the *symmetric knapsack problem*, and which is obtained by adding additional constraints to the knapsack instance, in the form of equations over the symmetric group. It will be seen that the symmetric knapsack problem is significantly harder to solve than the knapsack problem. Our reason to study this problem here is two-fold. First of all, not many hardness results are known for problems in non-commutative algebra [cf. 81], and symmetric subset-sum is a natural problem in that setting (see Observation 3.2.4 below). Secondly, our results show that the addition of even the simplest combinatorial structure to the knapsack problem already makes it significantly harder.

In the definition below, it helps to think of $\{1,\ldots,n\}$ as a set of items, the $v(i)$ as values, $w(i)$ as weights, and $\sigma(i)$ as patterns of these items, and that our task is to fit a set of items of maximum total value into a knapsack that

[2] More precisely, the contents of a register at any given time-step is a polynomial in these parameters, but precisely which polynomial depends on the outcomes of previous conditionals (greater-than-zero instructions).

[3] While the max-flow problem, like any problem in P, reduces to the knapsack problem, it does not seem to be possible to have such a reduction in a way that preserves the bit-length of the numerical parameters (the flow capacities). So the lower bound for max-flow does not imply a lower bound for the knapsack problem.

Intuitively, there is good reason to believe that such a reduction does not exist. The subset-sum problem is a "purely numerical" problem, in the sense that it has no combinatorial structure whatsoever; hence the combinatorial structure of a max-flow problem (the graph itself) will need to be somehow encoded in the numbers of the subset-sum problem we would wish to reduce to; but this graph can be much larger than the bit-lengths of the edge weights.

can carry W weight, with the added restriction that the items we pick must orderly fit together according to a certain pattern Σ.

3.2.1. Definition. The *0–1 symmetric knapsack problem*, which we denote with $\textsc{SymK}(n, a, b)$ for given integer parameters $n, a, b \geq 0$, is defined as the following optimization problem: We are given $v : [n] \to [2^a], w : [n] \to [2^b]$, $\sigma : [n] \to \mathbb{S}_5$, as well as $W \in [2^b], \Sigma \in \mathbb{S}_5$, and then, going over all subsets $I \subseteq [n]$, we wish to

$$\begin{aligned} \text{maximize} \quad & \textstyle\sum_{i\in I} v(i) \\ \text{s.t.} \quad & \textstyle\sum_{i\in I} w(i) \leq W \\ & \textstyle\bigodot_{i\in I} \sigma(i) = \Sigma \end{aligned}$$

The notation $\odot_{i\in I}\sigma(i)$ means the product of the elements $\sigma(i)$ for $i \in I$, *in ascending order of* i, and for completion we let $\odot_{i\in\varnothing}\sigma(i)$ denote e, the identity permutation. Note that the order is important since $\mathbb{S}_5$ is not commutative.

Throughout, we will assume that every $w(i) \leq W$, since of course larger items do not fit into the knapsack, and can be ignored.

The symmetric subset-sum problem, $\textsc{SymSS}(n, b)$, is the problem of deciding, when given an instance $X = (v, w, \sigma, W, \Sigma)$ of $\textsc{SymK}(n, b, b)$ with $v = w$, if there exists a feasible solution matching the bound W:

3.2.2. Definition. The symmetric subset-sum problem $\textsc{SymSS}(n, b)$ is defined as the following decision problem: We are given $w : [n] \to [2^b]$, $\sigma : [n] \to \mathbb{S}_5$, $W \in [2^b]$, and $\Sigma \in \mathbb{S}_5$, and we wish to decide if there exists an $I \subseteq [n]$ such that $\sum_{i\in I} w(i) = W$ and $\odot_{i\in I}\sigma(i) = \Sigma$.

Again it will be convenient to think of $[n]$ as a set of items, $w(i)$ as weights, and $\sigma(i)$ as patterns. Intuitively, our goal is now to find a choice of items matching a specific weight with a specific pattern. Note that subset-sum is defined as a decision problem whereas knapsack is defined as an optimization problem. If the we restrict σ_i to be the identity of $\mathbb{S}_5$, we have the original knapsack and subset-sum problems:

3.2.3. Definition. The 0–1 knapsack problem $\mathrm{K}(n, a, b)$ is $\textsc{SymK}(n, a, b)$ restricted to the case when σ_i, Σ are all e. The 0–1 subset-sum problem $\mathrm{SS}(n, b)$ is defined in the same way.

3.2.4. Observation. *The symmetric subset-sum problem, as defined here, is a natural problem in the setting of non-commutative algebra, because it is a special case of the subset-sum problem over general groups (studied in [81]). Indeed, to an instance* (w, σ, W, Σ) *of* $\textsc{SymSS}(n, b)$ *corresponds an instance of the subset-sum problem over the symmetric group, as follows. Let* $M = p_1 \dots p_m$ *be the smallest product of the first* m *primes such that* $M > n2^b$*, and let* $\phi(w, \sigma) = (w \bmod p_1, \dots, w \bmod p_m, \sigma)$ *be the canonical (additive) group isomorphism from* $\mathbb{Z}_M \times \mathbb{S}_5$ *to* $\mathbb{Z}_{p_1} \times \cdots \times \mathbb{Z}_{p_m} \times \mathbb{S}_5$*, itself a subgroup of* $\mathbb{S}_B$*, with* $B = p_1 + \ldots + p_m + 5$.

Then there exists an $I \subseteq [n]$ *such that* $\sum_{i\in I} w(i) = W$ *and* $\odot_{i\in I}\sigma(i) = \Sigma$*, if and only if there exists an* $I \subseteq [n]$ *such that* $\odot_{i\in I}\phi(w(i), \sigma(i)) = \phi(W, \Sigma)$*, and this latter problem is an instance of subset-sum over* $\mathbb{S}_B$.

From $m \leq b \log n$ *and [20, §2.7], we get* $B \leq (b \log n)^2 \ln(b \log n) + 5$.

3.3 Approximation algorithms: Old and new

Given an instance $X = (v, w, \sigma, W, \Sigma)$ of symmetric knapsack, it will have a unique optimum value m^*, corresponding to one (among possibly many) optimal solution I^*.

The goal of an approximation algorithm is to estimate the (unknown) value of m^*. The algorithm is said to achieve approximation ratio α if it always outputs a value m such that $m^* \leq \alpha m$.

3.3.1 Equivalence of approximation and exact solution for small weights

A classical observation in the study of knapsack problems is that, modulo a multiplicative factor of n, solving an n-item knapsack approximately to ratio $1+\varepsilon$ is equivalent to solving instances of knapsack having integer values bounded by $\frac{1}{\varepsilon}$. This holds the exact same way for the symmetric variant.

3.3.1. THEOREM. *Let $\varepsilon > 0$; then each problem in the following list reduces to the one below:*

1. *Solving* $\textsc{SymK}(n, a, b)$ *with approximation ratio* $1 + \varepsilon$*;*
2. *Solving* $\textsc{SymK}(n, \log n + \log(1 + \frac{1}{\varepsilon}), b)$ *exactly;*
3. *Solving* $\textsc{SymK}(n, a, b)$ *with approximation ratio* $1 + \frac{\varepsilon}{n^2}$*.*

This theorem will allow us to prove both upper and lower bounds for approximation algorithms by ignoring the approximation ratio altogether, and working with instances having small values $v(i)$. The proof is standard [95], and we write it here for completeness.

Proof. (1 reduces to 2) Let $X = (v, w, \sigma, W, \Sigma)$ be an instance of $\textsc{SymK}(n, a, b)$, and let $V = \max_i v(i)$. Set $t = \log(\frac{\varepsilon}{1+\varepsilon} \cdot \frac{V}{n})$, and make $v'(i) = \lfloor v(i)/2^t \rfloor$. Define X' to be X with v replaced by v'. Our approximate solution I will be an optimal solution of X'. Note that X' is an instance of $\textsc{SymK}(n, a', b)$ with $a' = \log n + \log(1+\frac{1}{\varepsilon})$, and that furthermore since w, σ, W, Σ remain unchanged, then any I feasible for X' will be feasible for X also (and vice-versa).

To conclude, we prove an upper bound on the approximation ratio. So let $m = \sum_{i \in I} v(i)$ be the value of I (i.e., m' is the X-value of the X'-optimal solution), and m^* the value of an optimal solution I^* of X. Since X' is X with the t least significant bits truncated, it is easy to see that $m^* - m \leq n2^t$. Since, furthermore, $V \leq m^*$, we conclude that

$$\frac{m^* - m}{m} \leq \frac{n2^t}{V}$$

which implies, by simple algebraic manipulation and our choice of t, that

$$m^* \leq \left(\frac{V}{V - n2^t}\right) m \leq (1 + \varepsilon)m.$$

Item 2 reduces to 3 via the following more general claim:

3.3.2. Claim. *Solving* SymK(n, a, b) *exactly reduces to solving* SymK(n, a, b) *with approximation ratio* $1 + \frac{1}{n2^a}$.

This is almost trivial to see. Note that the optimum of any instance of SymK(n, a, b) is bounded by $m^* \leq n2^a$. Hence any value m such that $m^* \leq (1 + \frac{1}{n2^a})m$ will have $m^* - m \leq m\left(1 - \frac{1}{1+\frac{1}{n2^a}}\right) < 1$, and so $m^* = m$ (as they are both integers). ■

3.3.3. Observation. *When the inputs are given in binary, the above reductions can be computed by an* NC^0 *circuit. When the inputs are given as numerical parameters to Mulmuley's PRAM model, the reductions can be computed in time* $O(\log n + \log(1 + \frac{1}{\varepsilon}))$ *using* $O(n/\varepsilon)$ *processors.*

3.3.2 Dynamic programming algorithm and a FPTAS

We now show that the original dynamic programming algorithm for classical 0–1 knapsack [26], together with its derived FPTAS, will also work for the symmetric knapsack problem. Again the proof is standard.

3.3.4. Theorem. *There is a dynamic programming algorithm for finding the optimum of* SymK(n, a, b), *in time* $O(n^2 2^a B)$, *where* B *is the time required to sum and compare* $(b + \log n)$*-bit numbers.*

Proof. Let $X = (v, w, \sigma, W, \Sigma)$ be an instance of SymK(n, a, b), and let $V = \max_i v(i)$. Throughout the algorithm, $I = I(k, \tilde{v}, \tilde{\sigma})$, for $k \in [n], \tilde{v} \in [nV], \tilde{\sigma} \in \mathbb{M}$, when defined, will denote a solution which:

(1) is a subset of $[k]$, meaning $i \notin I$ for $i > k$;

(2) has value $\sum_{i \in I} v(i) = \tilde{v}$,

(3) pattern $\odot_{i \in I} \sigma(i) = \tilde{\sigma}$, and

(4) is weight-optimal, in the sense that $\sum_{i \in I} w(i)$ is minimum among the weights of all solutions obeying (1-3).

Now I is computed iteratively: for $k = 1$, we set $I(1, 0, e) = \varnothing$ and $I(1, v(1), \sigma(1)) = \{1\}$, and let $I(1, \tilde{v}, \tilde{\sigma})$ remain undefined for all other pairs of $\tilde{v}$ and $\tilde{\sigma}$ — i.e., with only one item we can either put it in I with value $v(1)$ and pattern $\sigma(1)$, or not put it in I, which results in value 0 and pattern e.

Suppose we have defined I up to $k - 1$. Then let $I' = I(k - 1, \tilde{v}, \tilde{\sigma})$, and $I'' = I(k - 1, \tilde{v} - v(k), \tilde{\sigma}\sigma_k^{-1}) \cup \{k\}$. Then we define $I(k, \tilde{v}, \tilde{\sigma})$ to be equal to the I among I' and I'' that has minimum weight.[4] Again the same reasoning applies (we can either put k into I or not), and in this way we prove that the requirement of minimum weight is inductively maintained.

If we keep I and W as arrays in a random access memory, then each iteration can be computed in time $O(nVB) = O(n2^a B)$.

[4] And undefined if I' and I'' are both undefined.

Finally, the optimum solution for X is the highest value $\tilde{v}$ for which $I = I(n, \tilde{v}, \Sigma)$ is defined, and has weight no greater than W. ■

As in the original 0–1 knapsack problem, this algorithm can be used to obtain an FPTAS.

3.3.5. Corollary. *Solutions of* $\text{SymK}(n, a, b)$ *achieving approximation ratio* $1 + \varepsilon$ *can be obtained in time* $O((1 + \frac{1}{\varepsilon})n^3 B)$.

3.3.3 A new, simple algorithm for small space and small parallel time

We present a very simple algorithm, to our knowledge new, that solves subset-sum in $O(W^{\log n})$ time using $O(\log W \cdot \log n)$ space. Notice that if W is at least $2^{\Omega(n/\log n)}$, then we can instead run the trivial $O(n)$-space algorithm that tries every possible setting of the items, for an improved 2^n time bound. So this algorithm is only interesting for smaller values of W; but because approximation algorithms are equivalent to algorithms for small weights (Theorem 3.3.1) this makes our algorithm particularly suitable for approximation.

Our algorithm can be made to work for both the subset-sum and knapsack problems. It is a mix of ideas from two classical papers from the 70s: the *meet-in-the-middle* algorithm [53], and the PSPACE hardness of alternating time [93]. It can appropriately be called *recursive meet-in-the-middle algorithm.*

3.3.6. Theorem (Recursive meet-in-the-middle). *There is an algorithm for* $\text{SS}(n, b)$ *that works in time* $O(W^{\log n})$ *and space* $O(\log W \cdot \log n)$.

3.3.7. Observation. *It is interesting to compare the time and space parameters of our algorithm with those of [71]. In that paper, the authors present a clever algorithm for subset-sum based on the use of an adequate Fourier transform, which runs in time* $\tilde{O}(n^2 W \log W)$ *and space* $\tilde{O}(n^2)$. *Our algorithm is slower, but when applying it to solve subset-sum approximately, our algorithm uses less space, because usually* $W \ll 2^{n/\log n}$ *in this scenario.*

We will present the algorithm for the subset-sum problem first, and then explain how it generalizes to the symmetric version and the knapsack problem.

Proof of Theorem 3.3.6. We show a recursive procedure for the following problem: we are given an instance $X = (w, W) \in \text{SS}(n, b)$, and a value $m \in [W]$; we will output whether there exists a subset of items $I \subseteq [n]$ such that $\sum_{i \in I} w(i) = m$. The procedure works as follows:

(1) (Base case) If $n = 1$, we return *true* iff $w(1) = m$, else we return *false.*

(2) We go through each possible value $m' \leq m$;

(3) then we consider the partial problems X_1 having the first half of the items, and X_2, having the second half;

(4) we recursively call our procedure on inputs X_1, m' and $X_2, m - m'$: if no solution was found for either of the sub-problems, we move on to the next choice of m';

(5) if solutions were found, return *true*.

(6) If we have gone through every value, we return *false*.

To see that we return *true* in case a solution exists, note that if I is a solution of X having value m, then $I_1 = I \cap \{1, \ldots, \lfloor \frac{n}{2} \rfloor\}$ and $I_2 = I \setminus I_1$ will have some values, respectively m_1 and m_2. Since I has value m, it must be that $m_2 = m - m_1$; hence the algorithm will find some solution for X_1 and X_2 when calling (4) with $m' = m_1$ (here we assume inductively that solutions will be found for smaller n).

Finally, an algorithm for $\mathrm{SS}(n, b)$ will call this procedure with $m = W$. To bound the time and space used, we notice that the recursion has depth $\log n$, and each call in the stack uses $O(\log W)$ space to store m and m'. ■

By analyzing the algorithm above, we can see that it is given by an alternation of ORs and ANDs. Let $\mathrm{SS}(X, m)$ mean that there exists a solution for X of value m. Then it is clear that

$$\mathrm{SS}(X, m) \iff \bigvee_{m' \in [m]} \left[\mathrm{SS}(X_1, m') \wedge \mathrm{SS}(X_2, m - m')\right].$$

Expanding this out in a circuit gives us $O(\log n)$ layers. The bottom layer will simply contain equality tests. Hence we conclude that:

3.3.8. Corollary. *A parallel version of the algorithm can be implemented with* AC *circuits of depth* $O(\log n)$ *and size* $O(W^{\log n} b)$*, or in an arithmetic PRAM with* $O(W^{\log n})$ *processors running for* $O(\log n)$ *time.*

3.3.9. Observation. *For the knapsack problem* $\mathrm{K}[n, a, b]$*, where* w *and* v *may be distinct, each recursive call will look for solutions whose weight is* at most m*, rather than exactly* m*, and maintain in memory the maximum value among the admissible solutions found so far (rather than simply whether there exists a solution or not); i.e., the recursion is now:*

$$\mathrm{K}(X, m) = \max\{\mathrm{K}(X_1, m') + \mathrm{K}(X_2, m - m') | m' \in [m]\},$$

where $\mathrm{K}(X, m)$ *is the maximum value attainable for the instance* X *with weight up to* m*. The basic case is* $v(1)$ *if* $w(1) \leq m$*, and* 0 *otherwise. This is computable serially in* $O(W^{\log n})$ *time and* $(\log W + a) \log n$ *space, or by means of an* AC *circuit of depth* $O(\log n)$ *and size* $O(W^{\log n}(a^2 W^2 + b))$*.*[5]

3.4 Hardness of approximation

In the last few decades, theoretical computer science has classified most known NP-hard optimization problems according to how close to the optimum solution a polynomial time algorithm is likely to get. For instance, if P ≠ NP, then it holds that:

[5]To see this, notice that the maximum of W-many a-bit numbers can be computed by AC circuits of size $O(W^2 a^2)$, and the basic case can be computed by a circuit of size $O(b)$.

- For some constants $c > c'$, SetCover can be approximated to error ratio $c \log n$, but not $c' \log n$ [88].

- For some constants $c > c'$, Max2Sat can be approximated to error ratio c [68], but not c' [50].

- The problem of finding an optimal multi-processor scheduling with speed factors[6] has a PTAS, but no FPTAS [44, 52].

It is now natural to ask: how about problems that *do* have a FPTAS, can we prove that this kind of approximability is optimal, under a hardness assumption of some kind? We now show that the answer is yes, and that in this context the (much stronger) exponential-time hypothesis is the adequate choice.

3.4.1 Hardness of approximation for knapsack

Our strategy is the usual: we reduce a hard problem to the task of approximating subset-sum with ratio better than $1 + \frac{1}{\text{poly}}$.

3.4.1. Theorem. NC-Sat$(S, k) \leq_m^{\mathrm{AC}_0}$ SS$(O(S+k), O(S+k))$.

Proof. We will show that there is a uniform AC_0 circuit which, given as input a size-S circuit $C(\bar{y})$ over k variables $\bar{y} = y_1, \ldots, y_k$, will output an instance $X = (w, W)$ of

$$\mathrm{SS}(2k + 3S, 2k + 4S + 1),$$

that admits a subset-sum of weight W if and only if $C(\bar{y}) = 1$ for some choice of $\bar{y}$.

Suppose C is given as a sequence $(G_1, \ldots, G_S)$ of binary NAND gates, where each gate G_j is defined by the two gates feeding into it (these could be input gates or gates $G_{j'}$ for $j' < j$). Then X will have two items $Y_i^{(0)}$ and $Y_i^{(1)}$ for each $i \in [k]$, and three items $G_j^{(00)}, G_j^{(01)}, G_j^{(10)}$ for each NAND gate $j \in [S]$. We will construct our knapsack instance in such a way that any optimal solution must choose exactly one of the $Y_i^{(0)}, Y_i^{(1)}$ items for each $i \in [k]$, and exactly one of the G_j^{ab} items for each $j \in [S]$. For any optimal solution, a choice of the Y items will force a choice of gate items corresponding to an evaluation of the circuit.

It will help comprehension if we present the weights of the items in our instance both algebraically and in table form. For $i \in [k]$, $j \in [S]$, let $c(i, j)$ be 2 if input i is the first input of gate j, 1 if it is the second input, and 0 otherwise. Similarly with gates, for $j, j' \in [S]$, let $d(j, j')$ be 2 if gate j is the first input of gate j', 1 if it is the second input, and 0 otherwise. We will set

[6] This is the problem of scheduling tasks to processors of different speeds, while attempting to minimize execution time. Hence each task t in a set of tasks T has a length $l(t)$, and each processor p in a set of processors P has a speed factor $s(p)$. We wish to find an assignment of the tasks $f : T \to P$ minimizing completion time: $\max_{p \in P} \sum_{t : f(t) = p} l(t) s(p)$.

the weights to:

$$\begin{aligned}
w(Y_i^{(0)}) &= 2^{2(k-i)\,+\,4S+1} \\
w(Y_i^{(1)}) &= 2^{2(k-i)\,+\,4S+1} + \sum_{j=1}^{S} c(i,j)2^{4(S-j)+1} \\
w(G_j^{(00)}) &= (2+1) \times 2^{4(S-j)+1} + \sum_{j'=j+1}^{S} d(j,j')2^{4(S-j')+1} + [j = S?] \\
w(G_j^{(01)}) &= (2+0) \times 2^{4(S-j)+1} + \sum_{j'=j+1}^{S} d(j,j')2^{4(S-j')+1} + [j = S?] \\
w(G_j^{(10)}) &= (0+1) \times 2^{4(S-j)+1} + \sum_{j'=j+1}^{S} d(j,j')2^{4(S-j')+1} + [j = S?]
\end{aligned}$$

(Recall that $[j = S?]$ denotes 1 when $j = S$ and 0 otherwise.) We set the maximum weight to:

$$W = \sum_{i=1}^{k} 2^{2(k-i)\,+\,4S+1} + \sum_{j=1}^{S} 3 \times 2^{4(S-j)+1} + 1$$

The DLOGTIME uniformity of the reduction follows from the simplicity of the above expressions, which make the circuit computing the bits of the instance very easy to describe. Some global lookup is required to compute the functions c and d, which makes the reduction AC_0, rather than simply NC_0.[7]

Let us see what these numbers look like when written in binary. Since each number is less than $2^{2k+4S+1}$, we will write them in a table with $k + S + 1$ blocks. The first k blocks have two bits each, the middle blocks have four bits each, and the last block has one bit.

w	1 … i … k	G_1	…	G_j	…	$G_{j'}$	…	G_S	*out*
$Y_i^{(0)}$	00 … 01 … 00	0000	…	0000	…	0000	…	0000	0
$Y_i^{(1)}$	00 … 01 … 00	$c(i,1)$	…	$c(i,j)$	…	$c(i,j')$	…	$c(i,S)$	0
$G_j^{(00)}$	00 … 00 … 00	0000	**0**	0011	…	$d(j,j')$	…	$d(j,S)$	$[j = S?]$
$G_j^{(01)}$	00 … 00 … 00	0000	**0**	0010	…	$d(j,j')$	…	$d(j,S)$	$[j = S?]$
$G_j^{(10)}$	00 … 00 … 00	0000	**0**	0001	…	$d(j,j')$	…	$d(j,S)$	$[j = S?]$
	…								
W	01 … 01 … 01	0011	…	0011	…	0011	…	0011	1

It is vital to notice that in each column G_j there will be exactly five non-zero entries: two correspond to the inputs, having binary form 10 and 01, and three correspond to the gate G_j, and have binary form 11, 10 and 01. These are chosen so that for any setting $ab \in \{0,1\}^2$ of the items corresponding to the two inputs, then if we wish to achieve the maximum weight W, we will be forced to pick the gate item $G_j^{(ab)}$. Formally, we wish to prove the following claim, from which the theorem follows immediately.

[7] The encoding of the circuit could be changed to make this lookup easier.

3.4.2. Claim. *The solutions I of X having weight exactly W, are in 1–1 correspondence with the assignments of $\bar{y}$ causing $C(\bar{y}) = 1$.*

Given a choice of $\bar{y}$, we can define the solution $I(\bar{y})$ inductively as follows:

- $\{Y_i^{(y_i)}\} \subseteq I$;
- For each gate G_j, we let
  - $a_j = 1$ if input i is the first input to gate j and $Y_i^{(1)} \in I$, or if gate ℓ is the first input to gate j and $G_\ell^{(a_\ell b_\ell)} \in I$; and $a_j = 0$ otherwise;
  - similarly for b_j and the second input to gate j.
- Then $G_j^{(a_j b_j)} \in I$ if a_j and b_j are not both 1.

This solution will have weight exactly W, if $C(\bar{y}) = 1$, and weight $W - 1$, if $C(\bar{y}) = 0$. This can be seen easily by inspection of the table above: for instance, if $G_j = \text{NAND}(y_i, y_{i'})$, $y_i = 1$ and $y_{i'} = 0$, then $Y_i^{(1)}$ contributes (binary) weight 10 to column G_j, $Y_i^{(0)}$ contributes weight 0, and $G_j^{(10)}$ contributes weight 01, for a total of 11, which is exactly W for that column. The final column will be 1 iff the output gate G_S evaluates to 1.

To complete the proof of the claim, and hence of the theorem, it suffices to show that any solution achieving weight W must be of the form $I(\bar{y})$ for some unique choice of $\bar{y}$. Let I be any solution with total weight W. Again by inspection of the table above, we will find that, for each i, exactly one of $Y_i^{(0)}$ or $Y_i^{(1)}$ must be in I. This follows by letting i be the hypothetical first coordinate where this fails to hold: if both $Y_i^{(0)}$ and $Y_i^{(1)}$ are in I, then the total weight will surpass W, and if both $Y_i^{(0)}$ and $Y_i^{(1)}$ are missed, the total weight must be less than W because of the low weight of the items which are yet to be fixed.[8] So let $Y_i^{(y_i)}$, for $i \in [k]$, give us the set of chosen Y items. Then again in an inductive fashion we establish that for each gate G_j we must pick at most one item among $G_j^{(ab)}$, exactly as defined in $I(\bar{y})$: again it holds that this is the unique choice which fits the weight W exactly right. ■

This allows us to obtain a hardness-of-approximation result for the knapsack problem:

3.4.3. Corollary. *For every $f(n) = \omega(1)$, there exists $g(n) = \omega(n)$, such that if subset-sum can be approximated to ratio $1 + n^{-f(n)}$ in polynomial time, then* NC-Sat$(g(n), g(n)) \subseteq$ DTIME$(2^{o(n)})$ *and the Exponential–Time Hypothesis is false.*

Proof. If subset-sum can be approximated thus, then by Claim 3.3.2 (more precisely, by the analogous claim for subset-sum), we can solve $\text{SS}(n, h(n) \log n)$ in time n^c, for some non-decreasing super-constant $h(n)$. By an appropriate

[8] Here it is worth pointing out: this is the reason we need to "pad" each block of the weights with additional bits. This bears some resemblance to the superincreasing sequences used in the Merkle-Hellman knapsack cryptosystem.

substitution (say $n \leftarrow 2^{n/\sqrt{h(n)}}$), this implies we can solve $\mathrm{SS}(2^n, \omega(n))$ in time $2^{o(n)}$. By the previous theorem we can now conclude that NC-SAT$(\omega(n), \omega(n))$ can be solved in time $2^{o(n)}$.

In particular this implies that the satisfiability of 3CNF formulas with $\omega(n)$-many clauses can be decided in time $2^{o(n)}$ But the sparsification lemma of [57] will then imply that the satisfiability of 3CNF formulas of any size can *also* be decided in time $2^{o(n)}$, i.e., that the Exponential–Time Hypothesis is false.

More precisely, the sparsification lemma states that there is a function f such that, for any $\varepsilon > 0$, any 3CNF formula is equivalent to the OR of $O(2^{\varepsilon n})$ 3CNF formulas in which the number of clauses is $f(\varepsilon)n$, and these formulas can be produced in time $\mathrm{poly}(n)2^{\varepsilon n}$. Hence we let $\varepsilon = \varepsilon(n) = o(1)$ decrease slowly enough, causing $f(\varepsilon(n)) = \omega(1)$ to be sufficiently small, so that we can solve 3CNF formulas with $f(\varepsilon(n))n$ clauses in time $2^{o(n)}$. Then producing and evaluating the big OR of 3CNFs would take time $\mathrm{poly}(n)2^{2\varepsilon n + o(n)} = 2^{o(n)}$ in total. ■

Notice that this result is optimal, since for any constant c it *is* possible to approximate knapsack to ratio $1 + \frac{1}{n^c}$ in polynomial time.

3.4.2 Hardness of approximation for symmetric knapsack

It is immediate to see that SYMSS(n, n) is NP-hard, since it includes the original subset-sum problem as a special case. However, the addition of a small amount of combinatorial structure to the problem will allow for a hardness result where the bit-length of the weights only depends logarithmically on the formula size.

3.4.4. THEOREM. NC_1-SAT$(S, k) \leq_m^{\mathrm{AC}_0}$ SYMSS$(O(S^2 + k), O(k \log S))$.

Proof. We will show that there is a uniform AC_0 circuit which, given as input a width-5 permutation branching program $P(\bar{y})$ over k variables $\bar{y} = y_1, \ldots, y_k$ having size S, will output an instance $X = (w, \sigma, W, \Sigma)$ of

$$\mathrm{SYMSS}(2(k+S), k(2+\ell(S))), \qquad (\ell(S) = \lceil \log S + 1 \rceil)$$

that admits a subset-sum of weight W if and only if $P(\bar{y}) = e$ for some choice of $\bar{y}$. The result then follows by Barrington's theorem [24], which states that any NC^0 circuit can be evaluated by such a width-5 branching program.

So let $P = \{(j, z_j, \alpha_j, \beta_j)\}_{j=1}^{S}$, with $z_j \in [k]$, $\alpha_j, \beta_j \in \mathbb{S}_5$. We will always use i to index the y's, and j to index the instructions of P.

In our symmetric subset-sum problem, we will have two items for each possible y, denoted by $Y_i^{(1)}$ and $Y_i^{(0)}$; these items represent the possible assignments of the respective variable. We will also have two items for each instruction, denoted by A_j and B_j, which represent the choice of either α_j or β_j.

We will construct our knapsack instance in such a way that the optimal solution will choose exactly one of the $Y_i^{(0)}, Y_i^{(1)}$ items for each i, and this choice will force the correct choice of either A_j or B_j, whenever y_i is the variable consulted in the j-th instruction of P (i.e., whenever $z_j = i$). If we choose to

take item $Y_i^{(0)}$, for instance, then, in order to achieve weight W, we will be forced to always pick every A_j whenever $z_j = i$, and won't be able to pick any of the B_j.

Let $N(i) \leq S$ denote the number of times that y_i is consulted by P, i.e., the number of j such that $z_j = i$, and make $\ell = \lceil \log S + 1 \rceil$, so that each $N(i) < 2^\ell - 1$.

As before, we present the weights of the items in our instance both algebraically and in table form.

We will set the weights to:

$$\begin{aligned} w(Y_i^{(0)}) &= 2^{2(k-i)\,+\,2k\ell} + N(i)2^{2(k-i)\ell} \\ w(Y_i^{(1)}) &= 2^{2(k-i)\,+\,2k\ell} + N(i)2^{\ell+2(k-i)\ell} \\ \text{if } z_j = i, \text{ then } w(A_j) &= 2^{\ell+2(k-i)\ell} \\ \text{and } w(B_j) &= 2^{2(k-i)\ell} \end{aligned}$$

We set the maximum weight to:

$$W = \sum_{i=1}^{k} 2^{2(k-i)\,+\,2k\ell} + N(i)\left(2^{2(k-i)\ell} + 2^{\ell+2(k-i)\ell}\right).$$

The DLOGTIME uniformity of the reduction follows from the simplicity of the above expressions, which make the circuit computing the bits of the instance very easy to describe. Sum is required to compute $N(i)$, which makes the reduction AC_0, rather than simply NC_0.

Let us again see what these numbers look like in table form. Since each number is less than $2^{2k\,+\,2k\ell}$, we will write them in a table with $3k$ blocks. The first k blocks have two bits each, and the last $2k$ blocks have ℓ bits each:

w	1	…	i	…	k	$A^{(1)}$	$B^{(1)}$	…	$A^{(i)}$	$B^{(i)}$	…	$A^{(k)}$	$B^{(k)}$
$Y_i^{(0)}$	00	…	01	…	00	0	0	…	0	$N(i)$	…	0	0
$Y_i^{(1)}$	00	…	01	…	00	0	0	…	$N(i)$	0	…	0	0
$A_j\|_{z_j=i}$	00	…	00	…	00	0	0	…	1	0	…	0	0
$B_j\|_{z_j=i}$	00	…	00	…	00	0	0	…	0	1	…	0	0
						…							
W	01	…	01	…	01	$N(1)$	$N(1)$	…	$N(i)$	$N(i)$	…	$N(k)$	$N(k)$

Finally, we set $\sigma(Y_i^{(0)}) = \sigma(Y_i^{(1)}) = e$, $\sigma(A_j) = \alpha_j$ and $\sigma(B_j) = \beta_j$. Furthermore, in our sorting of the items (which is relevant for the outcome of $\odot_i \sigma(i)$), we ensure that the items A_j, B_j appear in growing order of j.

Here, our result follows from the following claim by setting $\Sigma = e$:

3.4.5. Claim. *The solutions I of X having weight exactly W and pattern Σ, are in 1–1 correspondence with the assignments of $\bar{y}$ causing $P(\bar{y}) = \Sigma$.*

To see this, notice that given a choice of $\bar{y}$, we define the solution $I(\bar{y}) = \{Y_i^{(y_i)}\} \cup \{A_j | z_j = i \wedge y_i = 0\} \cup \{B_j | z_j = i \wedge y_i = 1\}$, and this solution will have

weight exactly W, and pattern $P(\bar{y})$. This can be seen easily by inspection of the table above: for instance, if $y_i = 0$, the choice of $Y_i^{(0)}$ contributes weight 1 to the i-th column of the i-th block, and weight $N(i)$ to the column $B^{(i)}$; then the choice of $N(i)$-many A_j variables will contribute with a total weight of $N(i)$ to the column $A^{(i)}$. Now the pattern of $I(\bar{y})$ will simply be $P(\bar{y})$, by construction.

To complete the proof of the claim, and hence of the theorem, it suffices to show that any solution achieving weight W must be of the form $I(\bar{y})$ for some unique choice of $\bar{y}$. Let I be any solution with total weight W. Again by inspection of the table above, we will find that, for each i, exactly one of $Y_i^{(0)}$ or $Y_i^{(1)}$ must be in I. This follows by letting i be the hypothetical first coordinate where this fails to hold: if both $Y_i^{(0)}$ and $Y_i^{(1)}$ are missed, the total weight will be less than W, and if both $Y_i^{(0)}$ and $Y_i^{(1)}$ are in I, then the total weight will surpass W. So let $Y_i^{(y_i)}$, for $i \in [k]$, give us the set of chosen Y items. In the same way as before, it is easy to see that, for each i, every A_j with $z_j = i$ must be picked, if $y_i = 0$, or otherwise every corresponding B_j must be picked. Hence $I = I(\bar{y})$, as intended. ■

Let us instantiate k, in order to see what kind of parameters show up:

3.4.6. Corollary. *Symmetric knapsack cannot be approximated to ratio* $1 + 2^{-(\log n)^3}$ *in polynomial time, unless* $\exists \mathrm{NC}_1((\log n)^2) \subseteq \mathrm{P}$.

3.5 Lower bounds for parallel time

We will now prove that the knapsack problem cannot be efficiently parallelized in Mulmuley's model. The hardness of knapsack has already been proven in [90] in the fully-algebraic setting, and now we present a proof for the optimization version of subset-sum in the semi-algebraic setting. We do this using Mulmuley's parametric complexity technique. We will stick to the least general definition that applies to the subset-sum case.

The optimization version of subset-sum is simply the knapsack problem restricted to the case where the values of the items equal their weights; i.e., we are given $(\bar{w}, W)$ and we wish to compute:

$$S(\bar{w}, W) = \max\{\bar{a} \cdot \bar{w} | \bar{a} \in \{0,1\}^n \text{ and } \bar{a} \cdot \bar{w} \leq W\}$$

We will now refer to the problem of computing S as simply *the knapsack problem*. A *linear parametrization* for the knapsack problem is a function $P_n : \mathbb{R} \to \mathbb{R}^{n+1}$ mapping some interval $[A, B]$ into instances $(\bar{w}, W)$, where W is fixed and each w_i is given by a linear function of a single parameter λ. P_n is said to have bit-length $\beta = \beta(n)$ if all the coefficients of these linear functions are integers of bit-length β. For each $\lambda \in [A, B]$, let $m(\lambda) = S(P_n(\lambda))$.[9] Then it is easy to see that $m(\lambda)$ is a piecewise-linear and convex function of λ. The

[9]Here we extend the knapsack problem to the reals. This can easily be done due to it being a homogeneous optimization problem, i.e. $S(\alpha\bar{w}, \alpha W) = \alpha S(\bar{w}, W)$ for any constant α. We could avoid this entirely, except that it makes the definitions easier to visualize.

complexity $\rho(n)$ of P_n is then the number of different slopes of $m(\lambda)$ as λ goes from A to B.

The *parametric complexity* of the knapsack problem for bit-length $\beta(n)$, $\phi(n, \beta(n))$, is equal to the maximum complexity $\rho(n)$, over all parametrizations P_n of bit-length $\beta(n)$. Now let us define the following decision version of the knapsack problem: $A_n = \{(\bar{w}, W, z) | S(\bar{w}, W) \leq z\}$. Then the following is proven in [77]:

3.5.1. THEOREM (THEOREM 3.3 OF [77]). *There exist large enough constants a, b such that A_n cannot be solved in* $\mathrm{KC}(2^{\sqrt{\log \phi(n,\beta(n))}/b}, \sqrt{\log \phi(n, \beta(n))}/b)$*; this is so even if we restrict every numeric parameter in the input to be an integer with bit-length at most $a\beta(n)$.*

Below we will prove exponential lower bounds on $\phi(O(n^2), O(n^2))$, which imply that A_n is not in $\mathrm{KC}(2^{n^{\frac{1}{4}}/b}, n^{\frac{1}{4}}/b)$, even for instances where each weight has at most *an* bits. This is a stronger setting than the results in [90], which prove that subset-sum is not in $\mathrm{KC}(2^{\sqrt{n}/b}, \sqrt{n}/b)$, but without any restriction on the bit-length at which the hard instances will be found. Unfortunately we were unable to show a parallel algebraic reduction from A_n to subset-sum, and hence our results hold only for the knapsack problem, rather than subset-sum.

3.5.1 High parametric complexity for knapsack

3.5.2. THEOREM. *$\phi(O(n^2), O(n^2)) \geq 2^n$. Hence A_n is not in* $\mathrm{KC}(2^{n^{\frac{1}{4}}/b}, n^{\frac{1}{4}}/b)$*, even for bit-lengths restricted to $O(n)$.*

The proofs require a careful blending of the techniques of Theorem 3.4.1 and of the paper [77].

Proof. We will construct a linear parametrization $P(\lambda) : [0, 2^k] \to \mathrm{SS}(O(k + K), O(k + K))$ such that the weight of each item in $P(\lambda)$ is a linear function of $\lambda \in [0, 2^k]$, and such that the graph of the optimum value $m(\lambda)$ of $P(\lambda)$ will be a piecewise linear convex graph with 2^k-many different slopes. The hardness then follows from Theorem 3.5.1.

Let $\bar{y} = y_1, \ldots, y_{k+K}$ denote binary vectors of length $k+K = k+\frac{k(k-1)}{2}$. For $1 \leq i_1 < i_2 \leq k$, we let (i_1, i_2) stand for some bijection with $k+1, \ldots, k+K$, and we make use of the notation $y_{(i_1,i_2)}$ for convenience.

Let $C(\bar{y})$ be the formula:

$$\bigwedge_{1 \leq i_1 < i_2 \leq k} \left(y_{(i_1,i_2)} = y_{i_1} \wedge \neg y_{i_2} \right), \tag{3.1}$$

and let $X = X_C$ be the corresponding subset-sum instance given by Theorem 3.4.1. Since C can be implemented by an NC circuit with $10K$ NAND gates, then X will have $2k + 30K$ items and the weights have bit-lengths at most $2k + 40K + 1$.

Our $P(\lambda)$ will have the same items as X, namely for each variable y_i there are two items designated $Y_j^{(0)}$ and $Y_j^{(1)}$,[10] and we will use the generic designation C_j, for $j = 1, \ldots, 30K$, to denote the remaining items[11]. However, the weights of $P(\lambda)$ will be different: although the most significant bits of $P(\lambda)$ will be set in the same way as in X (and thus do not depend on λ), the least significant bits will be set so that as λ goes from 0 to 2^k, the optimum value of $P(\lambda)$ will have 2^k-many different slopes.

Again we will present $P(\lambda)$ both numerically and in table form. Let w_X and W_X denote the weights of X. We will set the weights of $P(\lambda)$ as follows:

$$\begin{aligned} w(Y_i^{(0)}) &= 2^{k(k+5)} w_X(Y_i^{(0)}) + f_i^{(0)}(\lambda) \\ w(Y_i^{(1)}) &= 2^{k(k+5)} w_X(Y_i^{(1)}) + f_i^{(1)}(\lambda) \\ w(C_j) &= 2^{k(k+5)} w_X(C_j) \\ W &= 2^{k(k+5)} W_X + 2^{k(k+5)} - 1 \end{aligned}$$

We define f as follows:

$$\begin{aligned} \text{for } 1 \le i \le k,\ f_i^{(0)}(\lambda) &= 2^{(k-i)(k+5)}(2^{k-i+1} - \lambda) \\ f_i^{(1)}(\lambda) &= 2^{(k-i)(k+5)}(2^{k-i}) \\ \text{for } 1 \le i_1 < i_2 \le k,\ f_{(i_1,i_2)}^{(0)}(\lambda) &= 0 \\ f_{(i_1,i_2)}^{(1)}(\lambda) &= 2^{(k-i_2)(k+5)} 2^{k-i_1} \end{aligned}$$

Notice that the weights w are a left shift of w_X by $k(k+5)$ bits, except for the items $Y_j^{(0)}$ and $Y_j^{(1)}$, who additionally get a least-significant value parametrized by λ. See also that the values of f, written in binary, can be partitioned into k blocks of $k+5$ bits each.

Let us illustrate this with the following table:

	$(2k + 40K + 1$ bits)	1	...	i	...	k
$Y_i^{(0)}$	Thm. 3.4.1	0	...	$2^{k-i+1} - \lambda$	...	0
$Y_i^{(1)}$	Thm. 3.4.1	0	...	2^{k-i}	...	0
$Y_{(i_1,i)}^{(0)}\|_{i_1<i}$	Thm. 3.4.1	0	...	0	...	0
$Y_{(i_1,i)}^{(1)}\|_{i_1<i}$	Thm. 3.4.1	0	...	2^{k-i_1}	...	0
W	Thm. 3.4.1	$2^{k+5} - 1$	...	$2^{k+5} - 1$	...	$2^{k+5} - 1$

The similarity with Theorem 3.4.1 ensures that we must pick exactly one of $Y_i^{(0)}$ and $Y_i^{(1)}$, and that $C(\bar{y}) = 1$ for the vector $\bar{y}$ corresponding to this choice, i.e.:

[10] We will also use the notation $Y_{(i_1,i_2)}^{(0)}, Y_{(i_1,i_2)}^{(1)}$ to denote the items corresponding to the variable $y_{(i_1,i_2)}$.

[11] By this we mean that each C_j, for $j = 1, \ldots, 30K$, is one of the three items $G_j'^{(00)}, G_j'^{(01)}, G_j'^{(10)}$ for some j', that appear in the proof of Theorem 3.4.1. We group them together because we will treat them in the exact same way.

3.5.3. Claim. *Each optimal solution of $P(\lambda)$ corresponds to a unique choice of $\bar{y}$ obeying equation (3.1).*

The proof of this claim is exactly as before. The correspondence is no longer bijective, since there might be values of $\bar{y}$ obeying 3.1 which have sub-optimal value due to f. However, any solution which is optimal up to the first $2k + 40K + 1$ bits must already satisfy the formula C, and in this case these bits must be set to W_X as in the proof of Theorem 3.4.1. We take this one step further, by noticing that (3.1) fully determines $\bar{y}$ from the first k bits $y_1, \ldots, y_k$.

So let $\hat{y}$ denote $y_1, \ldots, y_k$, and $I(\hat{y})$ be the solution containing the items $Y_i^{(y_i)}$, $Y_{(i_1,i_2)}^{(y_1 \wedge \neg y_2)}$, and the corresponding C_j's that simulate the computation of the circuit. Notice that $I(\hat{y})$ maximizes the first $2k + 10K + 1$ bits of the value to be exactly W_X: so we define $g_\lambda : \{0,1\}^k \to [2^{k(k+5)} - 1]$, to be the optimum of $I(\hat{y})$ minus $2^{k(k+5)} W_X$. Now the following claim holds:

3.5.4. Claim. *$\hat{y} \mapsto I(\hat{y})$ is a bijective correspondence between $\{0,1\}^k$ and solutions of $P(\lambda)$ having value at least $2^{k(k+5)} W_X$. Furthermore, the optimum solutions of $P(\lambda)$ are those $I(\hat{y})$ which maximize $g_\lambda(\hat{y})$.*

From this we may define $m^*(\lambda) = \max_{\hat{y} \in \{0,1\}^k} g_\lambda(\hat{y})$, and this will be the value of the optimum solution of $P(\lambda)$, minus $2^{k(k+5)} W_X$. The instance $P(\lambda)$ was constructed so that the optimum is given exactly by λ:

3.5.5. Claim. *For a given $\lambda \in [0, 2^k)$ of the form $\lambda = \lfloor \lambda \rfloor + \Delta, \Delta \in (\frac{1}{4}, \frac{3}{4})$, the maximum value of $g_\lambda(\hat{y})$ is attained when $\hat{y}$ is the binary expansion of $\lfloor \lambda \rfloor$.*

We now prove this claim. Fix some choice $\hat{y} = y_1, \ldots, y_k$. Then, by summing over the columns of the above table, and by equation (3.1), one sees that the following formula holds:

$$g_\lambda(\hat{y}) = \sum_{i=1}^{k} \left(y_i 2^{k-i} + (1 - y_i) \left(2^{k-i+1} - \lambda + \sum_{i_1 < i} y_{i_1} 2^{k-i_1} \right) \right) 2^{(k-i)(k+5)} \tag{3.2}$$

Take the binary expansion of $\lfloor \lambda \rfloor = \lambda_1 2^{k-1} + \ldots + \lambda_k 2^0$. We prove that the maximum of g_λ is achieved when $\hat{y} = \lambda_1 \ldots \lambda_k$. For any $\hat{y} \neq \lambda_1 \ldots \lambda_k$, let d be the first bit for which $\lambda_d \neq y_d$. Then we let $\tilde{y} = \lambda_1 \ldots \lambda_d y_{d+1} \ldots y_k$, and show that $g_\lambda(\tilde{y}) \geq g_\lambda(\hat{y})$.

Let $g_\lambda^{(i)}(\hat{y}) 2^{(k-i)(k+5)}$ be the i-th term in the summand of equation (3.2), and $G_i = g_\lambda^{(i)}(\tilde{y}) - g_\lambda^{(i)}(\hat{y})$. Then $G_i = 0$ for $i < d$, and so

$$g_\lambda(\tilde{y}) - g_\lambda(\hat{y}) = G_d 2^{(k-d)(k+5)} + \sum_{i > d} G_i 2^{(k-i)(k+5)}.$$

We may lower bound G_d as follows:

$$\begin{aligned} G_d &= (\lambda_d - y_d)2^{k-d} + (y_d - \lambda_d)\left(2^{k-d+1} - \lambda + \sum_{i_1<d} \lambda_{i_1} 2^{k-i_1}\right) \\ &= (\lambda_d - y_d)\left(\lambda_d 2^{k-d} - 2^{k-d} + \Delta + \sum_{i>d} \lambda_i 2^{k-i}\right) \\ &\geq \min(\Delta, 1-\Delta) \geq \frac{1}{4} \end{aligned}$$

Since every $|G_i|$ is upper bounded by 2^{k+3}, it now follows that

$$g_\lambda(\tilde{y}) - g_\lambda(\hat{y}) \geq G_d 2^{(k-d)(k+5)} - \sum_{i>d} |G_i| 2^{(k-i)(k+5)} > 0,$$

and the claim is proven.

Finally, we establish that the parametric complexity of knapsack is $\Omega(2^k)$:

3.5.6. Claim. *The slope of $m^*(\lambda)$ is unique for each interval*

$$(\lfloor\lambda\rfloor + \frac{1}{4}, \lfloor\lambda\rfloor + \frac{3}{4}) \subset [0, 2^k].$$

Looking at equation 3.2, from the previous claim it follows that $m^*(\lambda) = g_\lambda(\lfloor\lambda\rfloor)$, which simplifies to the linear form:

$$m^*(\lambda) = g_\lambda(\lambda_1, \ldots, \lambda_k) = \sum_{i:\lambda_i=1} 2^{(k-i)(k+5)}\lambda + A_{\lfloor\lambda\rfloor}$$

for some constant $A_{\lfloor\lambda\rfloor}$. And so each binary expansion of $\lfloor\lambda\rfloor$ gives rise to a unique slope. ■

We now state the consequences for parallel algorithms attempting to approximate the knapsack problem.

3.5.7. Corollary. *Knapsack instances with n items cannot be approximated with error ratio $1+\varepsilon$ in time $(\log \frac{1}{\varepsilon})^{\frac{1}{4}}/a$, and using $2^{(\log \frac{1}{\varepsilon})^{\frac{1}{4}}/a}$ processors, for some positive constant a, in Mulmuley's parallel PRAM model without bit operations.*

Proof. By the same proof as in Claim 3.3.2 we can show that optimally solving a knapsack instance (having $v = w$) with n items and bit-length n reduces to solving the same knapsack instance with error ratio $1 + \frac{1}{n2^n}$. So setting $\frac{1}{\varepsilon} = n2^n$, by Theorem 3.5.2 this cannot be done in $\mathrm{KC}(2^{n^{\frac{1}{4}}/b}, n^{\frac{1}{4}}/b)$ for some b, and hence neither in $\mathrm{KC}(2^{(\log \frac{1}{\varepsilon})^{\frac{1}{4}}/a}, (\log \frac{1}{\varepsilon})^{\frac{1}{4}}/a)$ for some larger a. ■

From our construction we also derive a new lower-bound for solving knapsack exactly, which matches the lower bound Mulmuley [77] proved for max-flow.

3.5.8. Corollary. *Knapsack instances with n items and bit-length $O(n^2)$ cannot be solved in time $\sqrt{n}/a$, using $2^{\sqrt{n}/a}$ processors, for some positive constant a, in Mulmuley's parallel PRAM model without bit operations.*

3.6 Open Problems

We consider the following two open problems worthy of further research. First, we have proven a lower bound for the optimization version of the subset-sum problem (the problem of computing $S(\bar{w}, W)$ exactly), even in the case of small input bit-lengths. Can we prove a lower bound for the decision version of subset-sum? Surprisingly, there is a reduction from the optimization version to the decision version [cf. 82], but this reduction is not algebraic, or even semi-algebraic, and so the question remains open.

Second, we have proven that we can approximate knapsack up to error $1 + \varepsilon$ in $\log n$ time with $(\frac{1}{\varepsilon})^{\log n}$ processors, and that we cannot do it in time $(\log \frac{1}{\varepsilon})^{\frac{1}{4}}/a$, and using $2^{(\log \frac{1}{\varepsilon})^{\frac{1}{4}}/a}$ processors. Can this gap be closed?

Chapter 4

Towards a reverse Newman's theorem in information complexity

Newman's theorem states that we can take any public-coin communication protocol and convert it into one that uses only private randomness with but a little increase in communication complexity. We consider a reversed scenario in the context of information complexity: can we take a protocol that uses private randomness and convert it into one that only uses public randomness while preserving the information revealed to each player?

We prove that the answer is yes, at least for protocols that use a bounded number of rounds. As an application, we prove new direct sum theorems through the compression of interactive communication in the bounded-round setting. To obtain this application, we prove a new one-shot variant of the Slepian-Wolf coding theorem, interesting in its own right.

Furthermore, we show that if a Reverse Newman's Theorem can be proven in full generality, then full compression of interactive communication and fully-general direct-sum theorems will result.

The results in this chapter are based on the paper:

- Joshua Brody, Harry Buhrman, Michal Koucký, Bruno Loff, Florian Speelman, and Nikolay Vereshchagin. Towards a reverse Newman's theorem in interactive information complexity. In *Proceedings of the 23rd CCC*, pages 24–33, 2013.

4.1 Introduction

Information cost was introduced by a series of papers [42, 22, 59, 23, 29] as a complexity measure for two-player communication protocols. Internal information cost measures the amount of information that each player learns about the input of the other player while executing a given protocol. In the usual setting of communication complexity we have two players, Alice and Bob, each having an input x and y, respectively. Their goal is to determine the value $f(x, y)$ for some predetermined function f. They achieve the goal by communicating to each other some amount of information about their inputs according to some *protocol*.

The usual measure considered in this setting is the *number of bits* exchanged by Alice and Bob, whereas the internal information cost measures the amount of *information* transferred between the players during the communication. Clearly, the amount of information is upper bounded by the number of bits exchanged but not vice versa. There might be a lengthy protocol (say even of exponential size) that reveals very little information about the players' inputs.

In recent years, a substantial research effort was devoted to proving the converse relationship between the information cost and the length of protocols, i.e., to proving that a protocol which reveals only I bits of information can be simulated by a different protocol which communicates only (roughly) I bits. Such results are known as *compression theorems*. [23] prove that a protocol that communicates C bits and has internal information cost I can be replaced by another protocol that communicates $O(\sqrt{I \cdot C})$ bits. For the case when the inputs of Alice and Bob are sampled from independent distributions they also obtain a protocol that communicates $O(I \cdot \log C)$ bits. These conversions do not preserve the number of rounds. In a follow-up paper, [29] consider a bounded round setting and give a technique that converts the original q-round protocol into a protocol with $O(q \cdot \log I)$ rounds that communicates $O(I + q \log \frac{q}{\varepsilon})$ bits with additional error ε.

All known compression theorems are in the randomized setting. We distinguish two types of randomness — *public* and *private*. Public random bits are seen by both communicating players, and both players can take actions based on these bits. Private random bits are seen only by one of the parties, either Alice or Bob. We use *public-coin* (*private-coin*) to denote protocols that use only public (private) randomness. If a protocol uses both public and private randomness, we call it a *mixed-coin* protocol.

Simulating a private-coin protocol using public randomness is straightforward: Alice views a part of the public random bits as her private random bits, Bob does the same using some other portion of the public bits, and they communicate according to the original private-coin protocol. This new protocol communicates the same number of bits as the original protocol and computes the same function. In the other direction, an efficient simulation of a public-coin protocol using private randomness is provided by Newman's Theorem [83]. Sending over Alice's private random bits to make them public could in general be costly as they may need e.g., polynomially many public random bits, but Newman showed that it suffices for Alice to transfer only $O(\log n + \log \frac{1}{\delta})$ random bits to be able to simulate the original public-coin protocol, up to an additional error of δ.

In the setting of information cost the situation is quite the opposite. Simulating public randomness by private randomness is straightforward: one of the players sends a part of his private random bits to the other player and then they run the original protocol using these bits as the public randomness. Since the random bits contain no information about either input, this simulation reveals no additional information about the inputs; thus the information cost of the protocol stays the same. This is despite the fact that the new protocol may communicate many more bits than the original one.

However, the conversion of a private-randomness protocol into a public-randomness protocol seems significantly harder. For instance, consider a protocol in which in the first round Alice sends to Bob her input x bit-wise XOR-ed with her private randomness. Such a message does not reveal any information to Bob about Alice's input — as from Bob's perspective he observes a random string — but were Alice to reveal her private randomness to Bob, he would learn her complete input x. This illustrates the difficulty in converting private randomness into public.

We will generally call "Reverse Newman's Theorem" (R.N.T.) a result that makes randomness public in an interactive protocol without revealing more information. This chapter is devoted to attacking the following:

> **R.N.T. Question.** *Can we take a private-coin protocol with information cost I and convert it into a public-coin protocol with the same behavior and information cost $\tilde{O}(I)$?*

Interestingly, the known compression theorems [23, 29, 60] give compressed protocols that use only public randomness, and hence as a by-product they give a conversion of private-randomness protocols into public-randomness equivalents. However, the parameters of this conversion are far from the desired ones.[1] In Section 4.4 we show that the R.N.T. question represents the core difficulty in proving full compression theorems; namely, we will prove that any public-coin protocol that reveals I bits of information can already be compressed to a protocol that uses $\tilde{O}(I)$ bits of communication, and hence a fully general R.N.T. would result in fully general compression results, together with the direct-sum results that would follow as a consequence. This was discovered independently by Denis Pankratov, who in his MSc thesis [86] extended the analysis of the [23] compression schemes to show that they achieve full compression in the case when only public randomness is used. Our compression scheme is similar but slightly different: we discovered it originally while studying the compression problem in a Kolmogorov complexity setting (as in [35]), and our proof for the Shannon setting arises from the proper "translation" of this proof; we include it for completeness and because we think it makes for a more elementary proof.

4.1.1 Main results

Our main contribution is a Reverse Newman's Theorem in the bounded-round scenario. We will show that any q-round private-coin protocol can be converted to an $O(q)$-round public-coin protocol that reveals only additional $\tilde{O}(q)$ bits of information (Theorem 4.3.1). Our techniques are new and interesting. Our main technical tool is a conversion of one-round private-randomness protocols into one-round public-randomness protocols. This conversion proceeds in two main steps. After *discretizing* the protocol so that the private randomness is sampled uniformly from some finite domain, we convert the protocol into what we call a 1-1 protocol, which is a protocol having the property that for each input and each message there is at most one choice of private random bits that

[1] We discuss the differences in more detail in Section 4.5.

will lead the players to send that message. We show that such a conversion can be done without revealing too much extra information. In the second step we take any 1-1 protocol and convert it into a public-coin protocol while leaking only a small additional amount of information about the input. This part relies on constructing special bipartite graphs that contain a large matching between the right partition and any large subset of left vertices.

Furthermore, we will prove two compression results for public-randomness protocols: a round-preserving compression scheme to be used in the bounded-round case, and a general (not round-preserving) compression scheme which can be used with a fully general R.N.T. Either of these protocols achieves much better parameters than those currently available for general protocols (that make use of private randomness as well as public). The round-preserving compression scheme is essentially a constant-round average-case one-shot version of the Slepian-Wolf coding theorem [92], and is interesting in its own right.

As a result of our R.N.T. and our round-preserving compression scheme, we will get a new compression result for general (mixed-coin) bounded-round protocols. Whereas previous results for the bounded-round scenario [29] gave compression schemes with communication complexity similar to our own result, their protocols were not round-preserving. We prove that a q-round protocol that reveals I bits of information can be compressed to an $O(q)$-round protocol that communicates $O(I+1) + q\log(\frac{qn}{\delta})$ bits, with additional error δ. As a consequence we will also improve the bounded-round direct-sum theorem of [29].

Organization of the chapter. In Section 4.3 we discuss our Reverse Newman's Theorem. In Section 4.4 we will prove our compression results. Section 4.5 will give applications to direct-sum theorems. Finally, Section 4.6 is dedicated to showing alternatives to the constructions we have presented, as well as bounds that prevent further improvement to our techniques.

4.2 Preliminaries

We use capital letters to denote random variables, calligraphic letters to denote sets, and lower-case letters to denote elements in the corresponding sets. So typically A is a random variable distributed over the set $\mathcal{A}$, and a is an element of $\mathcal{A}$. We will also use capital and lower-case letters to denote integers numbering or indexing certain sequences. We use $\Delta(A, A')$ to denote the *statistical distance* between random variables A and A':

$$\Delta(A, A') = \frac{1}{2}\sum_{a\in\mathcal{A}} |\Pr[A=a] - \Pr[A'=a]|.$$

4.2.1 Information theory

For a given probability random variable A distributed over the support $\mathcal{A}$, its entropy is

$$H(A) = \sum_{a\in\mathcal{A}} p_a \log\frac{1}{p_a},$$

where $p_a = \Pr[A = a]$. Given a second random variable B that has a joint distribution with A, the conditional entropy $H(A|B)$ equals

$$\mathbb{E}_{b \in B}[H(A|B = b)].$$

In this chapter, and when clear from the context, we denote a conditional distribution $A|B = b$ more succinctly by $A|b$.

4.2.1. Fact. If A has n possible outcomes then

$$H(A) \leq \log n.$$

4.2.2. Fact.

$$H(A|B) \leq H(A) \leq H(A, B), \quad H(A|B, C) \leq H(A|C) \leq H(A, B|C).$$

4.2.3. Fact.

$$H(A, B) = H(A) + H(B|A), \quad H(A, B|C) = H(A|C) + H(B|A, C).$$

We let $I(A : B)$ denote the Shannon mutual information between A and B, and $I(A : B|C)$ denote the Shannon mutual information between A and B, conditional on C:

$$\begin{aligned} I(A : B) &= H(A) - H(A|B) = H(B) - H(B|A), \\ I(A : B|C) &= H(A|C) - H(A|B, C) = H(B|C) - H(B|A, C). \end{aligned}$$

Notice that $I(A : B|C)$ may be larger than $I(A : B)$, for instance when C is the bitwise XOR of independent A and B.

4.2.4. Fact. The following equality is called *chain rule*:

$$I(A_1, \ldots, A_k : B|C) = I(A_1 : B|C) + \sum_{i=2}^{k} I(A_i : B|C, A_1, \ldots, A_{i-1})$$

Here $A_1, \ldots, A_k$ stands for a random variable in the set of k-tuples and A_i stands for its ith projection.

4.2.5. Fact. A and B are independent conditional on C (which means that whatever outcome c of C we fix, A and B become independent conditional on the event $C = c$) if and only if $I(A : B|C) = 0$.

4.2.6. Fact. If A and B are independent conditional on D then

$$I(A : C|B, D) = I(A : BC|D) \leq I(A : C|D).$$

4.2.7. Fact. If A and C are independent conditional on the pair B, D then

$$I(A : B, C|D) = I(A : B|D).$$

4.2.8. FACT. For any two random variables A, B over the same universe $\mathcal{U}$, it holds that

$$H(A) - H(B) \leq \log(|\mathcal{U}|)\Delta(A, B) + 1,$$

Proof. For each u in $\mathcal{U}$, let $c_u = \min\{\Pr[A = u], \Pr[B = u]\}$, $a_u = |Pr[A = u] - c_u|$ and $b_u = |\Pr[B = u] - c_u|$. Then $\delta := \Delta(A, B) = \sum_u a_u = \sum_u b_u$, and $1 - \delta = \sum_u c_u$.

So let μ_c, μ_a, μ_b be distributions, with $\mu_c(u) = c_u/(1-\delta)$, $\mu_a(u) = a_u/\delta$, and $\mu_b(u) = b_u/\delta$. Then we can think of A as being generated by tossing a coin A' with bias $\Pr[A' = 1] = \delta$, and if $A' = 1$, then we sample according to μ_a, and if $A' = 0$, we sample according to μ_c. Similary we think of B as being generatedby the toss of a coin B' with the same bias, then sampling according to μ_b if $B' = 1$, and according to μ_c otherwise.

It now follows that:

$$H(A) \leq H(A, A') = H(A') + H(A|A') = H_2(\delta) + (1-\delta)H(\mu_c) + \delta H(\mu_a),$$

where H_2 is the binary entropy function. On the other hand,

$$H(B) \geq H(B|B') \geq (1-\delta)H(\mu_c).$$

This gives us the claimed bound, since $H(\mu_a) \leq \log|\mathcal{U}|$ and $H_2(\delta) \leq 1$. ■

4.2.2 Two-player protocols

We will be dealing with protocols that have both public and private randomness; this is not very common, so we will give the full definitions, which are essentially those of [23, 29]. We will be working exclusively in the distributional setting, meaning that our inputs will be drawn from some distribution, and we will be interested in the average case communication complexity, round complexity, *etc.* From here onwards, we will assume that the input is given to two players, Alice and Bob, by way of two random variables X, Y sampled from a possibly correlated distribution μ over the support $\mathcal{X} \times \mathcal{Y}$.

A *private-coin protocol* π with output set $\mathcal{Z}$ is defined as a rooted tree, called the *protocol tree*, in the following way:

1. Each non-leaf node is owned by either Alice or Bob.
2. If v is a non-leaf node belonging to Alice, then:
   (a) The children of v are owned by Bob; each child is labeled with a binary string, and the set $\mathcal{C}(v)$ of labels of v's children is prefix-free.
   (b) Associated with v is a set $\mathcal{R}_v$, and a function $M_v : \mathcal{X} \times \mathcal{R}_v \to \mathcal{C}(v)$.
3. The situation is analogous for Bob's nodes.
4. With each leaf we associate an *output value* in $\mathcal{Z}$.

On input x, y the protocol is executed as follows:

1. Set v to be the root of the protocol tree.
2. If v is a leaf, the protocol ends and outputs the value associated with v.

3. If v is owned by Alice, she picks a string $r_{A,v}$ uniformly at random from $\mathcal{R}_v$ and sends the label of $M_v(x, r_{A,v})$ to Bob, they both set $v := M_v(x, r_{A,v})$, and return to step 2.
4. If v is owned by Bob, he picks a string $r_{B,v}$ uniformly at random from $\mathcal{R}_v$ and sends the label of $M_v(x, r_{B,v})$ to Alice, they both set $v := M_v(x, r_{B,v})$, and return to step 2.

A general, or *mixed-coin*, protocol is given by a distribution over private-coin protocols. The players run such a protocol by using shared randomness to pick an index r (independently of X and Y) and then executing the corresponding private-coin protocol π_r. A protocol is called *public-coin* if every $\mathcal{R}_v$ has size 1, i.e., no private randomness is used.

We let $\pi(x, y, r, r_A, r_B)$ denote the messages exchanged during the execution of π, for given inputs x, y, and random choices r, r_A and r_B, and $\textsc{Out}_\pi(x, y, r, r_A, r_B)$ be the output of π for said execution. The random variable R is the public randomness, R_A is Alice's private randomness, and R_B is Bob's private randomness; we use Π to denote the random variable $\pi(X, Y, R, R_A, R_B)$.

4.2.9. Definition. The *worst-case communication complexity* of a protocol π, $\mathsf{CC}(\pi)$, is the maximum number of bits that can be transmitted in a run of π on any given input and choice of random strings. The *average communication complexity* of a protocol π, with respect to the input distribution μ, denoted $\mathsf{ACC}_\mu(\pi)$, is the average number of bits that are transmitted in an execution of π, for inputs drawn from μ. The *worst-case number of rounds* of π, $\mathsf{RC}(\pi)$, is the maximum depth reached in the protocol tree by a run of π on any given input. The *average number of rounds* of π, w.r.t. μ, denoted $\mathsf{ARC}_\mu(\pi)$, is the average depth reached in the protocol tree by an execution of π on input distribution μ.

4.2.10. Definition. The *(internal) information cost* of protocol π with respect to μ is:

$$\mathsf{IC}_\mu(\pi) = I(Y : \Pi, R, R_A | X) + I(X : \Pi, R, R_B | Y)$$

Here the term $I(Y : \Pi, R, R_A|X)$ stands for the amount of information Alice learns about Bob's input after the execution of the protocol (and the meaning of the second term is similar). This term can be re-written in several different ways:

$$I(Y : \Pi, R, R_A|X) = I(Y : \Pi|X, R, R_A) = I(Y : \Pi, R|X, R_A),$$
$$I(Y : \Pi, R, R_A|X) = I(Y : \Pi, R|X) = I(Y : \Pi|X, R).$$

Here the first equality holds, as Bob's input Y is independent from randomness R, R_A conditional on X, which is obvious (see Fact 4.2.6 from the preliminaries). The second equality holds, since Y is independent from randomness R conditional on X, R_A, which is also obvious.

The third equality holds, as Y is independent from R_A conditional on Π, X, R (Fact 4.2.7). This independence follows from the rectangle property of protocols: for every fixed Π, X, R the set of all pairs $((Y, R_B), R_A)$ producing

the transcript Π is a rectangle and thus the pair (Y, R_B) (and hence Y) is independent from R_A conditional on Π, X, R. The fourth equality is proven similarly to the first and the second ones.

The expressions $I(Y : \Pi, R|X)$ and $I(Y : \Pi|X, R)$ for the information revealed to Alice are the most convenient ones and we will use them throughout this chapter. Similar transformations can be applied to the second term in Definition 4.2.10.

4.2.11. Definition. A protocol π is said to compute function $f : \mathcal{X} \times \mathcal{Y} \to \mathcal{Z}$ with error probability ε over distribution μ if

$$\Pr_{\mu,R,R_A,R_B}[\textsc{Out}_\pi(x, y, r, r_A, r_B) = f(x, y)] \geq 1 - \varepsilon \ .$$

Many of our technical results require that the protocol uses a limited amount of randomness at each step. This motivates the following definition.

4.2.12. Definition. A protocol π is an *ℓ-discrete protocol*[2] if $|\mathcal{R}_v| = 2^\ell$ at every node of the protocol tree.

When a protocol is ℓ-discrete, we say that it uses ℓ bits of randomness for each message; when ℓ is clear from context, we omit it. While the standard communication model allows players to use an infinite amount of randomness at each step, this is almost never an issue, since one may always "round the message probabilities" to a finite precision. This intuition is captured in the following observation.

4.2.13. Observation. *Suppose π is a private-coin protocol. Then, there exists an ℓ-discrete protocol π' with $\ell = O(\log(|\mathcal{X}|) + \log(|\mathcal{Y}|) + \mathsf{CC}(\pi))$ such that (i) $\mathsf{CC}(\pi') \leq \mathsf{CC}(\pi)$, (ii) $\mathsf{RC}(\pi') \leq \mathsf{RC}(\pi)$, and (iii) for all x, y we have*

$$\Delta\left(\Pi'(x, y, R_A, R_B), \Pi(x, y, R_A, R_B)\right) \leq 2^{-\Omega(\ell)}.$$

Furthermore, for any input distribution μ, the error of π' is at most the error of π plus $2^{-\ell}$. Equally small differences hold between $\mathsf{ACC}_\mu(\pi')$, $\mathsf{ARC}_\mu(\pi')$, and their π equivalents, and $\mathsf{IC}_\mu(\pi')$ is within an additive constant of $\mathsf{IC}_\mu(\pi)$.

Proof. Let π be given by its protocol tree; for each node v, let its corresponding function be $M_v : \mathcal{X} \times \mathcal{R} \to \mathcal{C}(v)$ (if it is Alice's node) or $M_v : \mathcal{Y} \times \mathcal{R}_v \to \mathcal{C}(v)$.

We let π' be given by the same protocol tree but where the functions M_v are restricted to a finite set $\mathcal{R}'_v$ of size $\leq k = 2^{10\ell}$, with $\ell = \log|\mathcal{X}||\mathcal{Y}| + \mathsf{CC}(\pi)$. Hence by construction π' has the same worst-case communication and number of rounds as π.

Let R_v be a random variable uniformly distributed over $\mathcal{R}_v$ and R'_v be a random variable uniformly distributed over $\mathcal{R}'_v$.

[2] In a discrete protocol, we restrict only the amount of private randomness in this definition. It is perhaps natural to also restrict the public randomness, but we will not need to.

4.2.14. Claim. *For any node v of Alice's there is a choice of $\mathcal{R}'_v$ of size $\leq 2^{10\ell}$ such that*

$$|\Pr[M_v(x, R_v) = m] - \Pr[M_v(x, R'_v) = m]| \leq 2^{-4\ell}$$

for every x and m. The obvious analogue holds for Bob's nodes.

We prove that $\mathcal{R}'_v$ exists by the probabilistic method. Let $\tilde{\mathcal{R}} = \{r_1, \ldots, r_k\}$ be a random variable which is a multiset obtained by picking k elements uniformly from $\mathcal{R}_v$, and define R'_v as the random variable which picks an element $r_i \in \tilde{\mathcal{R}}$ uniformly at random (counting multiplicities). Let P_m denote the random variable that is

$$P_m = \Pr[M_v(x, R'_v) = m] = \frac{\sum_{i=1}^{k} [M_v(x, r_i) = m]}{k}.$$

By linearity of expectation we find that:

$$\mathbb{E}[P_m] = \frac{\sum_{i=1}^{k} \mathbb{E}[M_v(x, r_i) = m]}{k} = \Pr[M_v(x, R_v) = m].$$

And hence by Hoeffding's inequality we conclude that:

$$\Pr[|P_m - \Pr[M_v(x, R_v) = m]| > 2^{-4\ell}] \leq 2\exp\left(-2k2^{-8\ell}\right) \ll 2^{-\ell}.$$

Hence by a union bound there must exist a choice for $\tilde{\mathcal{R}}$ such that

$$|P_m - \Pr[M_v(x, R_v) = m]| \leq 2^{-4\ell}$$

holds for every x and m; this choice is $\mathcal{R}'_v$.

Now fix x, y; from the claim it follows that for any transcript t,

$$|\Pr[\pi(x, y) = t] - \Pr[\pi'(x, y) = t]| \leq 2^{-3\ell},$$

which in turn implies that

$$\Delta\left(\Pi(x, y, r_A, r_B), \Pi'(x, y, \mathcal{R}'^{(a)}, \mathcal{R}'^{(b)})\right) \leq 2^{-2\ell}.$$

This results in a difference of $\leq 2^{-\ell}$ in success probability, average communication complexity, and average number of rounds, for any given input distribution. The technique we used is very similar to Newman's proof of his theorem, and we could have bounded the ammount of private randomness to something exponentially smaller, while achieving similar bounds.

However, to prove that there is a small difference in information cost, we need ℓ to be as large as $\log|\mathcal{X}||\mathcal{Y}| + \mathsf{CC}(\pi)$. Begin by noting that:

$$I(\Pi : X|Y) = H(\pi(X, Y, R)|Y) - H(\pi(X, Y, R)|X, Y),$$

and then use Fact 4.2.8 to conclude that

1. $|H(\pi(X, Y, R)|Y = y) - H(\pi'(X, Y, R')|Y = y)| = O(1)$ for all y, and

2. $|H(\pi(X,Y,R)|X=x,Y=y) - H(\pi'(X,Y,R')|X=x,Y=y)| = O(1)$ for any x, y, and hence

3. $|I(\Pi : X|Y) - I(\Pi' : X|Y)| = O(1)$,

By a symmetric reasoning for Bob, we find that $|\mathsf{IC}_\mu(\pi) - \mathsf{IC}_\mu(\pi)| = O(1)$. ■

Hence, while working exclusively with discretized protocols, our theorems will also hold for non-discretized protocols, except with an additional exponentially small error term. We consider this error negligible, and hence avoid discussing it beyond this point; the reader should bear in mind, though, that when we say that we are able to simulate a discretized protocol exactly, this will imply that we can simulate any protocol with $2^{-\Omega(\ell)}$ error.

We are particularly interested in the case of one-way protocols. In a one-way protocol, Alice sends a single message to Bob, who must determine the output. A one-way protocol π is thus given by a function $M_\pi : \mathcal{X} \times \mathcal{R} \mapsto \mathcal{M}$; on input x Alice randomly generates r and sends $M_\pi(x, r)$. Note that if π is private-coin, then $\mathsf{IC}_\mu(\pi) = I(X : M(X, R_A)|Y)$, and similarly, if π is public-coin, then $\mathsf{IC}_\mu(\pi) = I(X : R, M(X, R)|Y)$.

Finally, we close this section with a further restriction on protocols, which we call 1–1. Proving an R.N.T. result for 1–1 protocols will be a useful intermediate step in the general R.N.T. proof.

4.2.15. Definition. A one-way protocol π is a 1–1 protocol if $M_\pi(x, \cdot)$ is 1–1 for all x.

4.3 Towards a Reverse Newman's Theorem

Our main result is the following:

4.3.1. Theorem (Reverse Newman's Theorem, bounded-round version). *Let π be an arbitrary, ℓ-discrete, mixed-coin, q-round protocol, and let $C = \mathsf{CC}(\pi)$, $n = \max\{\log|\mathcal{X}|, \log|\mathcal{Y}|\}$. Suppose that π's public randomness R is chosen from the uniform distribution over the set $\mathcal{R}$, and π's private randomness R_A and R_B is chosen from uniform distributions over the sets $\mathcal{R}_A$ and $\mathcal{R}_B$, respectively.*

Then there exists a public-coin, q-round protocol $\tilde{\pi}$, whose public randomness R' is drawn uniformly from $\mathcal{R} \times \mathcal{R}_A \times \mathcal{R}_B$, and that has the exact same transcript distribution, i.e., for any input pair x, y and any message transcript t,

$$\Pr[\pi(x, y, R, R_A, R_B) = t] = \Pr[\tilde{\pi}(x, y, R') = t],$$

and for any distribution μ giving the input (X, Y),

$$\mathsf{IC}_\mu(\tilde{\pi}) \leq \mathsf{IC}_\mu(\pi) + O\left(q \log\left(2n\ell\right)\right). \tag{4.1}$$

We conjecture, furthermore, that a fully general R.N.T. holds:

4.3.2. Conjecture. *Theorem 4.3.1 holds with (4.1) replaced by*

$$\mathsf{IC}_\mu(\tilde{\pi}) \leq \tilde{O}(\mathsf{IC}_\mu(\pi)),$$

where $\tilde{O}(\cdot)$ suppresses terms and factors logarithmic in $\mathsf{IC}_\mu(\pi)$ and $\mathsf{CC}(\pi)$.

In Sections 4.4 and 4.5, we show that R.N.T.s imply fully general compression of interactive communication, and hence the resulting direct-sum theorems in information complexity. This results in new compression and direct-sum theorems for the bounded-round case. We believe that attacking Conjecture 4.3.2, perhaps with an improvement of our techniques, is a sound and new approach to proving these theorems.

Before proving Theorem 4.3.1 let us first remark that it suffices to show it only for protocols π without public randomness (with an absolute constant in the O-notation). To see this, fix any outcome r of the random variable R, and look at the protocol π conditioned on $R = r$. This is a protocol without public randomness, let us denote it by π_r. Using the expression

$$I(X : \Pi|Y, R) + I(Y : \Pi|X, R)$$

for information cost of π, we see that it equals the average information cost of the protocol π_r. Therefore, assuming that we are able to convert π_r into a public-coin protocol $\tilde{\pi}_r$, as in Theorem 4.3.1, we can let the protocol $\tilde{\pi}$ pick a random r and then run $\tilde{\pi}_r$. As the information cost of the resulting protocol $\tilde{\pi}$ again equals the average information cost of $\tilde{\pi}_r$, the inequality (4.1) follows from similar inequalities for π_r and $\tilde{\pi}_r$. For this reason, the theorems below will be proven for private-coin — rather than mixed-coin — protocols.

The $O(q \log(2n\ell))$-term of (4.1) suggests that we have some loss of information on each round. Indeed, Theorem 4.3.1 will be derived from its one-way version.

4.3.1 Reverse Newman's Theorem for one-way protocols

4.3.3. Theorem (R.N.T. for one-way protocols). *For any one-way private-coin ℓ-discrete protocol π there exists a one-way public-coin ℓ-discrete protocol π' such that π and π' generate the same message distributions, and for any input distribution $(X, Y) \sim \mu$, we have*

$$\mathsf{IC}_\mu(\pi') \leq \mathsf{IC}_\mu(\pi) + O(\log(2n\ell)),$$

where $n = \log |\mathcal{X}|$.

Proof. We first sketch the proof. The public randomness R' used by the new protocol π' will be the very same randomness R used by π. So we seem to have very little room for changing π, but actually there is one change that we are allowed to make. Let $M_\pi : \mathcal{X} \times \mathcal{R} \mapsto \mathcal{M}$ be the function Alice uses to generate her message. It will be helpful to think of M_π as a table, with rows corresponding to possible inputs x, columns corresponding to possible choices of the private random string r, and the (x, r) entry being the message

$M_\pi(x,r)$. Noticing that r is picked uniformly, Alice might instead send message $M_\pi(x,\phi_x(r))$, where ϕ_x is some permutation of $\mathcal{R}$. In other words, she may permute each row in the table using a permutation ϕ_x for the row x. The permutation ϕ_x will "scramble" the formerly-private now-public randomness R into some new string $\tilde{r} = \phi_x(r)$ about which Bob hopefully knows nothing. This "scrambling" keeps the message distribution exactly as it was, changing only which R results in which message. We will see that this can be done in such a way that, in spite of knowing r, Bob has no hope of knowing $\tilde{r} = \phi_x(r)$, unless he already knows x to begin with.

To understand what permutation ϕ_x we need, we first note the following. Let $M' = M_{\pi'}(X,R)$ denote the message that the protocol π' we have to design sends for input X and public randomness R. Then the information cost of π' is

$$I(M', R : X|Y).$$

The information cost of the original protocol π is

$$I(M : X|Y) = I(M' : X|Y),$$

where the equality holds as the distributions of the triples (M,X,Y) and (M',X,Y) are identical (regardless of the chosen permutations ϕ_x). Thus the difference between the information costs of π' and π equals

$$I(M', R : X|Y) - I(M' : X|Y) = I(R : X|M', Y),$$

which is at most $H(R|M',Y)$. If we permute each row of the table in such a way that every message m appears in at most $d = (n \cdot \ell)^{O(1)}$ columns, then given m we can specify the column (the random-choice R) used to pick m with $O(\log n\ell)$ bits, and hence

$$H(R|M', Y) = O(\log n\ell).$$

Unfortunately, it may happen that there are no such permutations. For instance, this is the case when a row has the same message m in every column.

We will show that if all messages in a row are distinct, then we can "almost" achieve the goal: one can permute each row in such a way that with probability at least $1 - 1/n^2$ the message $M' = M_{\pi'}(X,R)$ appears in at most $d = (n \cdot \ell)^{O(1)}$ columns. Thus we first prove Theorem 4.3.3 for the special case of 1–1 protocols, i.e. for protocols where each row has distinct messages.

The proof of Theorem 4.3.3 for 1–1 protocols. We first will construct a special bipartite graph G, which we call a *matching graph*. Its left nodes will be all possible messages m and its right nodes will be all random strings r. Our strategy will be to find a way of permuting each row of our table so that for every row x and most columns r (in row x) the message $M_{\pi'}(x,r)$ in the cell (x,r) of the table is connected by an edge to r in the graph G.

4.3.4. Definition. An (m,ℓ,d,δ)-matching graph is a bipartite graph $G = (\mathcal{M} \cup \mathcal{R}, \mathcal{E})$ such that $|\mathcal{M}| = 2^m$, $|\mathcal{R}| = 2^\ell$, $\deg(u) = d$ for each $u \in \mathcal{M}$, and such that for all $\mathcal{M}' \subseteq \mathcal{M}$ with $|\mathcal{M}'| = 2^\ell$, $G_{\mathcal{M}' \cup \mathcal{R}}$ has a matching of size at least $2^\ell(1-\delta)$.

To gain some intuition about what is happening, suppose we had the following *fictional object*: an $(m, \ell, n, 0)$-matching graph — i.e., we have a degree-n graph with the property that any left-set of size $|\mathcal{R}|$ will have a perfect matching with $\mathcal{R}$ that uses only edges in the graph. Now let $\mathcal{M}_x = M_\pi(x, \mathcal{R})$ be the set of messages that π can send on input x; then in the new protocol π', $M_{\pi'}(x, r)$ is the message that is matched with r in the perfect matching between $\mathcal{M}_x$ and $\mathcal{R}$ (see Figure 4.3.1). It should be clear that π' gives each message exactly the same probability mass.

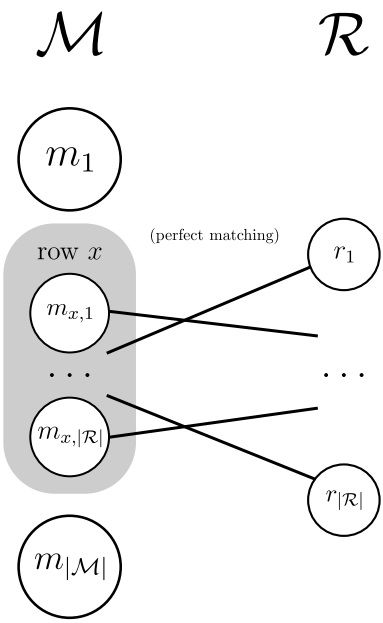


Figure 4.1: An ideal 'matching graph'.

To see that, in this new protocol π', R reveals little information about X when M' is known, notice that if we know the message $m' = M_{\pi'}(x, r)$, then in order to specify r we only need to say which edge in the graph must be followed; this is specified with $\log n$ bits because our graph has degree n. Hence $I(X : R|M) \leq H(R|M) \leq \log n$.

In truth, matching graphs with such good parameters do not exist. But we *can* have good-enough approximations, and we can show that this is enough for our purposes. These graphs are obtained through the Probabilistic Method.

4.3.5. Lemma. *For all integers $\ell \leq m$ and positive δ there is an (m, ℓ, d, δ)-matching graph with $d = O(m/\delta)$.*

In Section 4.6.1 we will show that the lemma holds also $d = O((m - \ell)/\delta^2) + \ln(1/\delta)/\delta$ (Lemma 4.6.1). That bound has better dependence on m, ℓ (especially when $m - \ell \ll m$). However, it has worse dependence on δ. In Section 4.6.2 we show a lower bound of $d = \Omega((m - \ell)/\delta)$, which almost matches our upper bounds.

Proof. Hall's theorem [48] states that if in a bipartite graph *every* left subset of cardinality $i \leq L$ has at least i neighbors then *every* left subset of cardinality $i \leq L$ has a matching in the graph.

Thus it suffices to construct a bipartite graph having this property for $L = (1-\delta)2^\ell$. By the union bound, a random graph[3] of degree d fails to have this property with probability at most

$$\sum_{i=1}^{L} 2^{mi} 2^{\ell i} (i/2^\ell)^{di}.$$

Here 2^{mi} is an upper bound for the number $\binom{2^m}{i}$ of i-element left subsets $\mathcal{M}'$, $2^{\ell i}$ is an upper bound for the number of $i-1$-element right subsets $\mathcal{R}'$, and $(i/2^\ell)^{di}$ is an upper bound for the probability that all neighbors of $\mathcal{M}'$ fall into $\mathcal{R}'$. For $L \leq (1-\delta)2^\ell$ this sum is upper bounded by a geometric series

$$\sum_{i=1}^{L} \left(2^m 2^\ell (1-\delta)^d\right)^i.$$

Thus we are done, if the base of this series $2^m 2^\ell (1-\delta)^d$ is less than $1/2$, say, which happens for sufficiently large $d = O(m/\delta)$. ∎

Now the proof of Theorem 4.3.3 for 1–1 protocols proceeds as follows. Let $n = \log|\mathcal{X}|$ and $\ell = \log|\mathcal{R}|$. Assume without loss of generality that $\mathcal{M} = M(\mathcal{X}, \mathcal{R})$; then $|\mathcal{M}| \leq 2^{n+\ell}$. Now let G be an $(n+\ell, \ell, d, \delta)$-matching graph having $\mathcal{M}$ as a subset of its left set and $\mathcal{R}$ as its right set, for $\delta = \frac{1}{n^2}$. For these parameters, we are assured by Lemma 4.3.5 that such a matching graph exists having left-degree $d = O((n+\ell)n^2)$.

We construct the new protocol π' as follows. For each $x \in \mathcal{X}$ let $\mathcal{M}_x = M(x, \mathcal{R})$ be the set of messages that might be sent on input x. Noticing that $|\mathcal{M}_x| = 2^\ell$, consider a partial G-matching between $\mathcal{M}_x$ and $\mathcal{R}$ pairing all but a δ-fraction of $\mathcal{M}_x$; then define a bijection $M'_x : \mathcal{R} \to \mathcal{M}_x$ by setting $M'_x(r) = m$ if (m, r) is an edge in the matching, and pairing the unmatched m and r's arbitrarily (possibly using edges not in G). Finally, set $M'(x, r) = M'_x(r)$.

Since $M'(x, r) = M'_x(r)$ for some bijection M'_x between $\mathcal{R}$ and $\mathcal{M}_x$, it is clear that M and M' generate the same transcript distribution for any input x.

Now we prove that M' does not reveal much more information than M. We have seen that the difference between the information costs of π' and π is at most $H(R|M', Y)$. Thus it suffices to show that $H(R|M', Y)$ is at most the logarithm of the left degree of the matching graph plus a constant. As $H(R|M', Y)$ is the average of $H(R|M', Y = y)$ over all choices of y, it suffices to show that

$$H(R|M', Y = y) \leq \log d + 3$$

for every y. While proving this inequality, we will drop the condition $Y = y$ to simplify notation.

[3] For each left vertex, we pick each of the d neighbours independently and uniformly from the right-set.

Let us introduce a new random variable K, which is a function of X, R, M' and takes the value 1 if (M', R) is an edge of the matching graph and is equal to 0 otherwise. Recall that for every x the pair $(M'(x, R), R)$ is an edge of the matching graph with probability at least $1 - 1/n^2$. Therefore, $K = 0$ with probability at most $1/n^2$. Call a message m *bad* if the probability that $K = 0$ conditional to $M' = m$ (that is, the fraction of rows x, among all rows containing m, such that m was not matched within the graph in the row x) is more than $1/n$. Then M' is bad with probability less than $1/n$, otherwise $K = 0$ would happen with probability greater than $1/n^2$.

The conditional entropy $H(R|M')$ is the average of

$$H(R|M' = m)$$

for m chosen according to the distribution of M'. Notice that $H(R|M' = m)$ is at most the log-cardinality of $\mathcal{X}$, because in 1–1 protocols R is a function of the pair (M', X). Thus $H(R|M' = m) \le n$ for all m, and hence the total contribution of all bad m's in $H(R|M')$ is at most 1. Thus it suffices to show that for all good m,

$$H(R|M' = m) \le \log d + 2.$$

To this end notice that

$$H(R|M' = m) \le H(K|M' = m) + H(R|K, M' = m) \le 1 + H(R|K, M' = m).$$

Thus it is enough to prove that $H(R|K, M' = m) \le \log d + 1$ for all good m. Again, $H(R|K, M' = m)$ can be represented as the weighted sum of two terms,

$$H(R|K = 1, M' = m) \text{ and } H(R|K = 0, M' = m).$$

The former term is at most $\log d$, because when $K = 1$ and $M' = m$ we can specify R by the number of the edge (m, R) in the matching graph. The latter term is at most n, but its weight is at most $1/n$, since m is good. This completes the proof of Theorem 4.3.3 for 1-1 protocols.

The proof of Theorem 4.3.3 in general case. The general case follows naturally from the 1–1-case and the following lemma, which makes a protocol 1–1 by adding a small amount of communication.

4.3.6. Lemma (A 1–1 conversion which reveals little information). *Given a one-round ℓ-discrete private-coin protocol π, there is a one-round 1–1 ℓ-discrete private-coin protocol π' whose message is of the form*[4]

$$M_{\pi'}(x, r) = (M_\pi(x, r), J(x, r)),$$

for some function J, and such that

$$\mathsf{IC}_\mu(\pi') \le \mathsf{IC}_\mu(\pi) + \log \ell + 1.$$

[4] On any input x and any choice of randomness r, $M_{\pi'}(x, r)$ is obtained by taking $M_\pi(x, r)$ and adding some additional communication $J(x, r)$.

Proof. We think of $M(\cdot,\cdot)$ as a table, where the inputs $x \in \mathcal{X}$ are the rows and the random choices $r \in \mathcal{R}$ are the columns, and fix some ordering $r_1 < r_2 < \dots$ of $\mathcal{R}$. The second part $J(x,r)$ of $M_{\pi'}$ will be the ordinal number of the message $M(x,r)$ inside the row x i.e.,

$$J(x,r) = |\{r' \le r | M(x,r') = M(x,r)\}|.$$

This ensures that $M_{\pi'}$ is 1–1.

The difference between the information costs of π' and π is

$$I(M,J:X|Y) - I(M:X|Y) = I(J:X|Y,M).$$

Thus, it suffices to show that for every particular y,m we have[5]

$$I(J:X|Y=y,M=m) \le \log \ell + 1. \tag{4.2}$$

Fix any y and m, and drop the conditions $Y=y, M=m$ to simplify the notation. By definition, $I(J:X) = H(J) - H(J|X)$. For any fixed x the random variable J has the uniform distribution over the set $\{1,2,\dots,W_x\}$, where W_x stands for the number of occurrences of the message m in row x of the table.

Let us partition the x's into ℓ classes so that if x is in the ith class then $2^{i-1} \le W_x < 2^i$. Let $Z = Z_{y,m}$ be the class to which X belongs. The entropy of Z is at most $\log \ell$ and hence we have

$$I(J:X) \le I(J:X|Z) + H(Z) \le I(J:X|Z) + \log \ell.$$

Thus it suffices to show that for every i we have

$$I(J:X|Z=i) \le 1.$$

Notice that

$$H(J|Z=i) \le i,$$

as for all x in the ith class we have $W_x \le 2^i$. On the other hand,

$$H(J|X,Z=i) \ge i-1,$$

as for every x in the ith class we have $W_x \ge 2^{i-1}$ and the distribution of J conditional to $X=x, Y=y, M=m, Z=i$ is uniform. Thus

$$I(J:X|Z=i) = H(J|Z=i) - H(J|X,Z=i) \le i - (i-1) = 1.$$

■

Now we are able to finish the proof of Theorem 4.3.3 in the general case. Suppose π is a given one-way private-coin ℓ-discrete protocol. Let π_2 be the 1–1 protocol guaranteed by Lemma 4.3.6, and let π_3 be the protocol constructed from π_2 in the proof of Theorem 4.3.3 for the 1–1 case. Note that π_3's message

[5] In Section 4.6.3 we will prove a corresponding lower bound, implying that this upper-bound is tight up to a constant term.

is of the form $M_{\pi_3}(X,R) = (M_\pi(X,R), J(X,R))$, since it is equidistributed with M_{π_2}. Furthermore, we have

$$\mathsf{IC}_\mu(\pi_3) \le \mathsf{IC}_\mu(\pi) + O(\log 2n\ell).$$

Now, create a protocol π_4, which is identical to π_3, except that Alice omits $J(X,R)$. Since for each x the message $M_{\pi_4}(x,r)$ sent by π_4 equals $M(x,\phi_x(r))$ for some permutation ϕ_x of $\mathcal{R}$, it is clear that M and M' generate the same transcript distribution for any input x. And

$$\mathsf{IC}_\mu(\pi_4) \le \mathsf{IC}_\mu(\pi_3) \le \mathsf{IC}_\mu(\pi) + O(\log 2n\ell)\,.$$

This completes the proof of Theorem 4.3.3. ■

4.3.2 R.N.T. for many-round protocols

Let us derive Theorem 4.3.1 from Theorem 4.3.3.

Proof of Theorem 4.3.1. Let c be the constant hidden in the O-notation in Theorem 4.3.3 so that every one-round private-coin ℓ-discrete protocol π with $|\mathcal{X}|, |\mathcal{Y}| \le 2^n$ can be converted into a one-round public-coin protocol π' generating the same distribution on transcripts with

$$\mathsf{IC}(\pi') \le \mathsf{IC}(\pi) + c \log 2n\ell.$$

We are given a q-round private-coin protocol ρ and will simulate it by a public-coin protocol ρ' with

$$\mathsf{IC}(\rho') \le \mathsf{IC}(\rho) + 2qc \log 2n\ell.$$

The transformation of ρ into ρ' is as one can expect: in each node v of the protocol tree ρ we use a permutation of messages that depends on the input of the player communicating in that node. More specifically, let $m_{<j}$ denote the concatenation of messages sent by ρ' up to round j. In the jth round of ρ' we apply the protocol $\rho'_{m_{<j}}$, which is obtained by the transformation of Theorem 4.3.3 from the 1-round sub-protocol $\rho_{m_{<j}}$ of ρ rooted from the node $m_{<j}$ of the protocol tree of ρ. This change does not affect the probability distribution over messages sent in each node and hence the resulting protocol ρ' generates exactly the same distribution on transcripts. The protocol ρ' uses the same randomness as ρ; however, unlike ρ it uses public and not private randomness.

We have to relate now the information cost of ρ' to that of ρ. To this end we split the information cost of ρ' into the sum of information costs of each round of ρ'. Specifically, by the Chain rule (Fact 4.2.4) the amount of information revealed by ρ' to Bob (say) equals

$$I(X : M_1, R_1, \ldots, M_q, R_q | Y) = \sum_j I(X : M_j, R_j | Y, M_{<j}, R_{<j}).$$

where R_j denotes the randomness used in the jth round of ρ' and $M_j = \rho'_{M_{<j}}(X, R_j)$ denotes the message sent in the jth round of ρ'.

From $I(R_{<j} : M_j, R_j | Y, M_{<j}) = 0$,[6] we conclude from Theorem 4.3.3 — using Facts 4.2.5 and 4.2.6 from the preliminaries — that

$$I(X : M_j, R_j | Y, M_{<j}, R_{<j}) \leq I(X : M_j, R_j | Y, M_{<j}) \leq I(X : M_j | Y, M_{<j}) + c \log 2n\ell,$$

where $I(X : M_j | Y, M_{<j})$ in the right-hand side is the information cost of the jth round of the original protocol ρ. Summing up this inequality over all $j = 1, \ldots, q$ and applying the Chain rule to ρ we see that

$$I(X : M_1, R_1, \ldots, M_q, R_q | Y) \leq I(X : M_1, \ldots, M_q | Y) + qc \log 2n\ell.$$

The similar inequality for the amount of information revealed by ρ and ρ' to Alice is proved analogously. ■

4.4 Compression for public-coin protocols

We present in this section two results of the following general form: we will take a public-coin protocol π that reveals little information, and "compress" it into a protocol ρ that uses little *communication* to perform the same task with about the same error probability. It turns out that the results in this setting are simpler and give stronger compression than in the case where Alice and Bob have private randomness (such as in [23, 29]). We present two bounds, one that is dependent on the number of rounds of π, but which is also round-efficient, in the sense that ρ will not use many more rounds than π; and one that is independent of the number of rounds of π, but where the compression is not as good when the number of rounds of π is small. We begin with the latter.

4.4.1. Theorem. *Suppose there exists a public-coin protocol π to compute $f : \{0,1\}^n \times \{0,1\}^n \to \mathcal{Z}$ over the distribution μ with error probability δ', and let $C = \mathsf{CC}(\pi)$, $I = \mathsf{IC}_\mu(\pi)$. Then for any positive δ there is a public-coin protocol ρ computing f over μ with error $\delta' + \delta$, and with $\mathsf{ACC}_\mu(\rho) = O(I \cdot \log(2Cn/\delta))$.*

Proof. Our compression scheme is similar, but not identical, to that of [23]—the absence of private randomness allows for a more elementary proof.

It suffices to prove the theorem only for deterministic protocols—the case for public-coin protocols can be proved as follows. By fixing any outcome r of randomness R of a public-coin protocol π, we obtain a protocol π_r without public randomness and can apply Theorem 4.4.1 to π_r. The average communication length of the resulting deterministic protocol ρ_r is at most $O(I(\pi_r) \cdot \log(2Cn/\delta))$. Thus the average communication of the public-coin protocol ρ that chooses a random r and runs ρ_r will be at most $O(I \cdot \log(2Cn/\delta))$.

[6] The reader is reminded that we defined protocols so that the message in each round depends only on public randomness, the previous messages, and on a source of private randomness that is independent from the private randomness used in previous rounds. It is easy to see that such an assumption can be made.

Thus we have to show that any deterministic protocol π can be simulated with communication roughly:

$$I(Y : \Pi|X) + I(X : \Pi|Y) = H(\Pi|X) + H(\Pi|Y)$$

(the equality follows because $H(\Pi|X, Y) = 0$, since the transcript Π is a function of X and Y). As we do not relate in this theorem the round complexity of ρ to that of π, we may assume that in the protocol π every message is just a bit (and the turn to communicate does not necessarily alternate). In other words, the protocol tree has binary branching.

Given her input x, Alice knows the distribution of $\Pi|x$, and she can hence compute the conditional probability $\Pr[\pi(X, Y) = t|X = x]$ for each leaf t of the protocol tree. We will use the notation $w_a(t|x)$ for this conditional probability. Likewise Bob computes $w_b(t|y) = \Pr[\pi(X, Y) = t|Y = y]$. Now it must hold that $\pi(x, y)$ is the unique leaf such that both $w_a(t|x), w_b(t|y)$ are positive. Alice and Bob then proceed in stages to find that leaf: at a given stage they have agreed that a certain *partial transcript*, which is a node in the protocol tree of π, is a prefix of $\pi(x, y)$. Then each of them chooses a *candidate transcript*, which is a leaf extending their partial transcript (the candidate transcripts of Alice and Bob may be different). Then they find the largest common prefix (lcp) of their two candidate transcripts, i.e., find the first bit at which their candidate transcripts disagree. Now, because one of the players actually knows what that bit should be (that bit depends either on x or on y), the player who got it wrong can change her/his bit to its correct value, and this will give the new partial transcripts they agree upon. They proceed this way until they both know $\pi(x, y)$.

It will be seen that the candidate leaf can be chosen in such a way that the total probability mass under the nodes they have agreed upon halves at every correction, and this will be enough to show that Alice will only need to correct her candidate transcript $H(\Pi|X)$ times (and Bob $H(\Pi|Y)$ times) on average. Efficient protocols for finding the lcp of two strings will then give us the required bounds.

We first construct an interactive protocol that makes use of a special device, which we call *lcp box*. This is a conceptual interactive device with the following behavior: Alice takes a string u and puts it in the lcp box, Bob takes a string v and puts it in the lcp box, then a button is pressed, and Alice and Bob both learn the largest common prefix of u and v. Using an lcp box will allow us to ignore error events until the very end of the proof, avoiding an annoying technicality that offers no additional insight.

4.4.2. Lemma. *For any given probability distribution μ over input pairs and for every deterministic protocol π with information cost I (w.r.t. μ) and worst case communication C there is a deterministic protocol $\tilde{\rho}$ with zero communication computing the same function with the same error probability (w.r.t. μ) as π, and using an lcp box for C-bitstrings at most I times on average (w.r.t. μ).*

Proof. On inputs x and y, in the new protocol $\tilde{\rho}$ Alice and Bob compute weights $w_a(t|x), w_b(t|y)$ of every leaf of the protocol tree of π, as explained

above. Furthermore, for every binary string s let $w_a(s|x)$ denote the sum of weights $w_a(t|x)$ over all leaves t under s. Define $w_b(s|y)$ in a similar way.

The protocol $\tilde{\rho}$ runs in stages: before each stage i Alice and Bob have agreed on a binary string $s = s_{i-1}$, which is a prefix of $\pi(x, y)$. Initially $s = s_0$ is empty.

On stage i Alice defines the candidate transcript t_a as follows: she appends 0 to $s = s_{i-1}$ if $w_a(s0|x) > w_a(s1|x)$ and she appends 1 to s otherwise. Let s' denote the resulting string. Again, she appends 0 to s' if $w_a(s'0|x) > w_a(s'1|x)$ and she appends 1 to s' otherwise. She proceeds in this way until she gets a leaf of the tree (by construction its weight is positive). Bob defines his candidate transcript t_b in a similar way. Then they put t_a and t_b in the lcp box and they learn the largest common prefix s' of t_a and t_b. By construction both $w_a(s'|x)$ and $w_b(s'|y)$ are positive and hence s' is a prefix of $\pi(x, y)$. Recall that no leaf of the protocol tree is a prefix of another leaf. Therefore either $s' = t_a = t_b$, in which case they stop the protocol, as they both know $\pi(x, y)$. Or s' is a proper prefix of both t_a and t_b. If the node s' of the protocol tree belongs to Alice, then Bob's next bit is incorrect, and otherwise Alice's next bit is incorrect. They both add the correct bit to s' and let s_i be the resulting string.

Each time Alice's bit is incorrect $w_a(s|x)$ decreases by a factor of $1/2$, and similarly each time Bob's bit is incorrect $w_b(s|y)$ decreases by a factor of $1/2$. At the start we have $w_a(s|x) = w_b(s|y) = 1$ and at the end we have $w_a(s|x) = w_a(\pi(x, y)|x)$ and $w_b(s|y) = w_b(\pi(x, y)|y)$. Hence they use the lcp box at most

$$\log_2 1/w_a(\pi(x, y)|x) + \log_2 1/w_b(\pi(x, y)|y)$$

times. By definition of the conditional entropy the average of $\log_2 1/w_a(\pi(X, Y)|X)$ is equal to $H(\Pi|X)$ and the average of $\log_2 1/w_b(\pi(X, Y)|Y)$ equals $H(\Pi|Y)$. Thus Alice and Bob use the lcp box at most I times on average. ■

Now we have to transform the protocol of Lemma 4.4.2 to a randomized public-coin protocol computing f that does not use an lcp box, with additional error δ. The use of an lcp box can be simulated with an error-prone implementation:

4.4.3. Lemma ([46]). *For every positive ε and every natural C there is a randomized public-coin protocol such that on input two C-bit strings x, y, it outputs the largest common prefix of x, y with probability at least $1 - \varepsilon$; its worst-case communication complexity is $O(\log(C/\varepsilon))$.*

The lemma is proven by hashing (as in the randomized protocol for equality) and binary search. From this lemma we obtain the following corollary.

4.4.4. Lemma. *Let $\tilde{\rho}$ be a protocol that computes $f : \{0,1\}^n \times \{0,1\}^n \to \mathcal{Z}$, while using an lcp box $\ell \leq 2n$ times on average for strings of length at most C. Then $\tilde{\rho}$ can be simulated with error δ by a protocol ρ that does not use an lcp box, and communicates $O(\ell \log(\frac{2Cn}{\delta}))$ bits more on average.*

Proof. The protocol ρ simulates $\tilde{\rho}$ by replacing each use of the lcp box with the protocol given by Lemma 4.4.3 with some error parameter ε (to be specified later). The simulation continues while the total communication is less than n.

Once it becomes n, we stop the simulation and Alice just sends her input to Bob.

Notice that the additional error probability introduced by the failure of the protocol of Lemma 4.4.3 is at most $\varepsilon\ell$: for each input pair (x, y) the error probability is at most $\varepsilon i(x, y)$, where $i(x, y)$ stands for the number of times we invoke lcp box for that particular pair, and the average of $\varepsilon i(x, y)$ over (x, y) equals $\varepsilon\ell$. Thus if we take $\varepsilon \leq \delta/\ell$, the error probability introduced by failures of the lcp box is it most δ.

Each call to the lcp box costs $O(\log(C/\varepsilon)$. Thus the communication of ρ is at most

$$O(\ell \log(C/\varepsilon)) + (\ell\varepsilon)(2n)$$

more on average than that of $\tilde{\rho}$. Here the first term is an upper bound for the average communication over all triples (x, y, randomness for the lcp box) such that no lcp failure occurs and the second term accounts for the average communication over all remaining triples.

Let $\varepsilon = \delta/2n$ (which is less than δ/ℓ, as we assume that $\ell \leq 2n$) so that the average communication is at most $O(\ell \log(\frac{2Cn}{\delta}) + \ell\delta) = O(\ell \log(\frac{2Cn}{\delta}))$. ■

We are now able to finish the proof of the theorem. Notice that the information cost of the initial protocol is at most $2n$. Hence we can apply Lemma 4.4.4 for $\ell = I$ to the protocol of Lemma 4.4.2. The average communication of the resulting protocol ρ is at most $O(I \cdot \log(2Cn/\delta))$. ■

The proof of Theorem 4.4.1 offers no guarantee on the number of rounds of the compressed protocol ρ. It is possible to compress a public-coin protocol on a round-by-round basis while preserving, up to a multiplicative constant, the total number of rounds used.

4.4.5. Theorem. *Suppose there exists a public-coin protocol π to compute $f : \{0,1\}^n \times \{0,1\}^n \to \mathcal{Z}$ over input distribution μ with error probability δ', and let $I = \mathsf{IC}_\mu(\pi)$ and $q = \mathsf{RC}(\pi)$. Then there exists a public-coin protocol ρ that computes f over μ with error $\delta' + \delta$, and with $\mathsf{ACC}_\mu(\rho) = O(I+1) + q \log(nq/\delta)$ and $\mathsf{ARC}_\mu(\rho) = O(q)$.*

Proof. Again it suffices to prove the theorem for deterministic protocols π. The idea of the proof is to show the result one round at a time. In round i, Alice, say, must send a certain message m_i to Bob. From Bob's point of view, this message is drawn according to the random variable $M_i = M_i(\tilde{X}, y, m_1, \ldots, m_{i-1})$ where $\tilde{X}$ is Alice's input conditioned on Bob's input being y and on the messages $m_1, \ldots, m_{i-1}$ that were previously exchanged. We will show that there is a sub-protocol σ_i that can simulate round i with small error by using constantly-many rounds and with

$$O(H(M_i|y, m_1, \ldots, m_{i-1})) = O(I(X : M_i|y, m_1, \ldots, m_{i-1}))$$

bits of communication *on average*. Then putting these sub-protocols together, and truncating the resulting protocol whenever the communication is excessive, we obtain the protocol ρ which simulates π.

The procedure to compress each round is achieved through an interactive variant of the Slepian-Wolf theorem ([92, 89, 35]). We could not apply the known theorems directly, however, since they were made to work in different settings.

In a similar fashion to the proof of Theorem 4.4.1, we will make use of a special interactive device, which we call a *transmission μ-box*, where μ is a probability distribution over input pairs (X, Y). Its behavior is as follows: one player takes a string x and puts it in the transmission box, the other player takes a string y and puts it in the box, a button is pressed, and then the second player knows x. The usage of a transmission μ-box is charged in such a way that the average cost when the input pair (X, Y) is drawn at random with respect to μ is $O(H(X|Y) + 1)$ bits of communication and $O(1)$ rounds.

4.4.6. Lemma. *Let π be any deterministic q-round protocol, and let μ be the distribution of the inputs (X, Y). Then there exists a deterministic protocol $\tilde{\rho}$ that makes use of the transmission box (each time for a different distribution) to achieve the following properties.*

1. *The average communication of $\tilde{\rho}$ is $\mathsf{ACC}_\mu(\tilde{\rho}) = O(\mathsf{IC}_\mu(\pi) + q)$;*
2. *The average number of rounds of $\tilde{\rho}$ is $\mathsf{ARC}_\mu(\tilde{\rho}) = O(q)$;*
3. *$\tilde{\rho}$ uses a transmission box q times; and*
4. *After $\tilde{\rho}$ is run on the inputs x, y, both players know $\pi(x, y)$.*

Proof. Let $\pi_{<j}(x, y)$ denote the sequence of messages sent by π in the first $j-1$ rounds for inputs x, y. The protocol $\tilde{\rho}$ simulates π on a round-per-round basis.

Assume that in the new protocol $j-1$ rounds were played. Let $m_{<j}$ denote the sequence of $j-1$ messages sent earlier and let x, y stand for inputs. Assume further that in jth round of π Alice has to communicate. Her message is a function M of the sequence $m_{<j}$ and her input x. Let ν denote the probability distribution on pairs (m, y) where

$$\nu(m, y) = \Pr[M(X, m_{<j}) = m,\ Y = y | \pi_{<j}(X, Y) = m_{<j}].$$

In round j of protocol $\tilde{\rho}$, Alice puts the string $M(x, m_{<j})$ into the transmission ν-box and Bob puts there his input y and they press the button. If it is Bob's turn to communicate, then they reverse their positions.

Items 2, 3 and 4 from the statement of the Lemma follow from construction of $\tilde{\rho}$ and from the description of the transmission box. It remains to bound the average communication length of $\tilde{\rho}$. Again by assumption on the transmission box, the average communication in round j is at most $O(I_j + 1)$ where

$$I_j = H(M(X, \pi_{<j}(X, Y)) | Y, \pi_{<j}(X, Y)),$$

if it is Alice's turn to communicate and

$$I_j = H(M(Y, \pi_{<j}(X, Y)) | X, \pi_{<j}(X, Y)),$$

otherwise. From the chain rule (Fact 4.2.4) it follows that the sum of I_j over all j of the first type is equal to $I(\Pi : X|Y)$, while the sum of I_j over all j of the second type is equal to $I(\Pi : Y|X)$. ■

To proceed we need a protocol simulating the transmission box.

4.4.7. Lemma (Constant-round average-case one-shot Slepian–Wolf). *Let μ be the distribution of the inputs (X, Y). For every positive ε there is a public-coin communication protocol with the following properties:*

1. *For all x, y, after execution of the protocol Bob learns x with probability at least $1 - \varepsilon$.*
2. *When (X, Y) are drawn according to μ, the protocol communicates an*

$$O(H(X|Y) + 1) + \log(1/\varepsilon)$$

average number of bits in $O(1)$ average number of rounds.

Contrast this to the classical Slepian–Wolf theorem, where Alice and Bob are given a stream of i.i.d. pairs $(X_1, Y_1), \ldots, (X_n, Y_n)$, and Alice gets to transmit $X_1, \ldots, X_n$ by using only one-way communication, and with an *amortized* communication of $H(X|Y)$.

Proof. Let y be Bob's given input. For a given x in the support of X, let $p(x) = \Pr[X = x|Y = y]$, and for a given subset $\mathcal{X}$ of the same support, let $p(\mathcal{X}) = \Pr[X \in \mathcal{X}|Y = y]$. Then Bob begins by arranging the x's in the support of X by decreasing order of the probability $p(x)$. He then defines the two sets

$$\mathcal{X}_1 = \{x_1, \ldots, x_{i(1)}\}, \qquad \mathcal{Z}_1 = \mathcal{X}_1,$$

where $i(1)$ is the minimal index which makes $p(\mathcal{X}_1) \geq 1/2$. Inductively, while $Z_k \neq X$, he then defines:

$$\mathcal{X}_{k+1} = \{x_{i(k)+1}, \cdots, x_{i(k+1)}\}, \qquad \mathcal{Z}_{k+1} = \mathcal{Z}_k \cup \mathcal{X}_{k+1},$$

where $i(k+1) > i(k)$ is the minimal index which makes $p(\mathcal{X}_{k+1}) \geq \frac{1-p(\mathcal{Z}_k)}{2}$. In other words, $\mathcal{X}_{k+1}$ is the smallest set which takes the remaining highest-probability x's so that they total at least half of the remaining probability mass.

Because at least one new x_i is added at every step, this inductive procedure gives Bob a finite number of sets $\mathcal{Z}_1, \ldots, \mathcal{Z}_K = X$. Then the protocol consists of applying the protocol of the following lemma, which will be proved later.

4.4.8. Lemma. *For every natural number m and every positive ε there exists a randomized public-coin protocol with the following behavior. Suppose that Bob is given a family of finite sets $\mathcal{Z}_1 \subseteq \cdots \subseteq \mathcal{Z}_K \subset \{0,1\}^m$ and Alice is given a string $z \in \mathcal{Z}_K$. Then the protocol transmits z to Bob, except with a failure probability of at most ε. For k the smallest index for which $z \in \mathcal{Z}_k$, the run of this protocol uses at most $2k + 1$ rounds and $2\log|\mathcal{Z}_k| + \log\frac{1}{\varepsilon} + 4k$ bits of communication.*

Now let us bound the average number of rounds and communication complexity. First notice that $p(\mathcal{X}_k) \leq 2^{1-k}$, and hence, taking the average over Alice's inputs, we find that

$$\sum_{k=1}^{K} p(\mathcal{X}_k) 4k = O(1)$$

must upper-bound the average number of rounds, as well as the contribution of the $4k$ term to the average communication. To upper-bound the contribution of the $2\log|\mathcal{Z}_k|$ term, we first settle that:

(i) $p(\mathcal{X}_k) \leq 2p(\mathcal{X}_{k+1}) + 2p(x_{i(k)})$, which can be seen by summing two inequalities that follow from the minimality of $i(k)$ in the definition of $\mathcal{X}_k$:

$$p(\mathcal{X}_k) - p(x_{i(k)}) \leq \frac{1 - p(\mathcal{Z}_{k-1})}{2}, \qquad \frac{1 - p(\mathcal{Z}_k)}{2} \leq p(\mathcal{X}_{k+1}),$$

after which we get

$$\frac{p(\mathcal{X}_k)}{2} - p(x_{i(k)}) \leq p(\mathcal{X}_{k+1}).$$

(ii) $|\mathcal{Z}_k| \leq \frac{1}{p(x)}$ for any $x \in \mathcal{X}_{k+1} \cup \{x_{i(k)}\}$, which follows since every $x' \in \mathcal{Z}_k$ has a higher-or-equal probability than the x's in $\mathcal{X}_{k+1} \cup \{x_{i(k)}\}$, but the sum of all the $p(x')$ still adds up to less than 1.

Now we are ready to bound the remaining term in the average communication:

$$\sum_{k=1}^{K} p(\mathcal{X}_k)\log|\mathcal{Z}_k| \leq 2\sum_{k=1}^{K-1} p(\mathcal{X}_{k+1})\log|\mathcal{Z}_k| + p(\mathcal{X}_K)\log|\mathcal{Z}_K| + 2\sum_{k=1}^{K} p(x_{i(k)})\log|\mathcal{Z}_k|$$

$$\leq 5\sum_{x} p(x)\log\frac{1}{p(x)} = O(H(X|Y=y));$$

above, the first inequality follows from (i), and the second from (ii). ■

Proof of Lemma 4.4.8. The protocol is divided into stages and works as follows. On the first stage, Bob begins by sending the number $\ell_1 = \log|\mathcal{Z}_1|$ in unary to Alice, and Alice responds by picking $L_1 = \ell_1 + \log\frac{1}{\varepsilon} + 1$ random linear functions $f_1^{(1)}, \ldots, f_{L_1}^{(1)} : \mathbb{Z}_2^n \to \mathbb{Z}_2$ using public randomness, and sending Bob the hash values $f_1^{(1)}(z), \ldots, f_{L_1}^{(1)}(z)$. Bob then looks for a string $z' \in \mathcal{Z}_1$ that has the same hash values he just received; if there is such a string, then Bob says so, and the protocol is finished with Bob assuming that $z' = z$.

Otherwise, the protocol continues. At stage k, Bob computes the number $\ell_k = \log|\mathcal{Z}_k|$, and sends the number $\ell_k - \ell_{k-1}$ in unary to Alice; Alice responds by picking $L_k = \ell_k - \ell_{k-1} + 1$ random linear functions $f_1^{(k)}, \ldots, f_{L_k}^{(k)}$, whose evaluation on z she sends over to Bob. Bob then looks for a string $z' \in \mathcal{Z}_k$ that has the same hash values for all the hash functions which were picked in

this and previous stages; if there is such a string, then Bob says so, and the protocol is finished with Bob assuming that $z' = z$. If the protocol has not halted in K rounds, Alice just sends her input to Bob.

An error will occur whenever a $z' \neq z$ is found that has the same fingerprint as z. The probability that this happens at stage k for a specific $z' \in \mathcal{Z}_k$ is 2^{-L}, where $L = \ell_k + k + \log \frac{1}{\varepsilon}$ is the total number of hash functions picked up to this stage. By a union bound, the probability that such a z' exists is at most $|\mathcal{Z}_k| 2^{-\ell_k} \frac{\varepsilon}{2^k} \leq \frac{\varepsilon}{2^k}$. Again by a union bound, summing over all stages k we get a total error probability of ε.

To bound the communication for $z \in \mathcal{Z}_k$, notice that sending all $\ell_1 \ldots, \ell_k$ costs Bob at most $\log |\mathcal{Z}_k| + k$ bits of total communication[7], that the total number of hash values sent by Alice is at most $\log |\mathcal{Z}_k| + 2k + \log \frac{1}{\varepsilon}$, and that Bob's reply (saying whether the protocol should continue) costs him k bits. ■

From Lemma 4.4.7 we get an analogue of Lemma 4.4.4.

4.4.9. LEMMA. *Let $\tilde{\rho}$ be a protocol to compute $f : \{0,1\}^n \times \{0,1\}^n \to \mathcal{Z}$ that uses transmission boxes q times. Then, for any positive δ, $\tilde{\rho}$ can be simulated with error δ by a protocol ρ that does not use transmission boxes, and communicates $q \log(\frac{qn}{\delta}) + 1$ bits more than $\tilde{\rho}$.*

Proof. The protocol ρ simulates $\tilde{\rho}$ by replacing each use of a transmission box with the protocol given by Lemma 4.4.7 with some error parameter ε (to be specified later). The simulation continues while the total communication is less than n. Once it becomes n, we stop the simulation and Alice just sends her input to Bob.

The additional error probability introduced by the failure of the protocol of Lemma 4.4.7 is at most $q\varepsilon$. Assuming that $\varepsilon \leq \delta/q$, the error probability introduced by a transmission box failure is it most δ.

Each call of a transmission box costs $\log(1/\varepsilon)$ bits of communication more than we have charged the protocol $\tilde{\rho}$. Thus the communication of ρ is at most

$$q \log(1/\varepsilon) + (q\varepsilon)(2n)$$

longer than that of $\tilde{\rho}$. Let $\varepsilon = \delta/qn$ so that the communication of ρ be at most

$$q \log(qn/\delta) + \delta/2 \leq q \log(qn/\delta) + 1$$

more than that of $\tilde{\rho}$. ■

We are able now to finish the proof of the theorem. Applying Lemma 4.4.9 to the protocol of Lemma 4.4.6 we get the desired protocol. ■

4.5 Applications

From the combination of Theorems 4.3.1 and 4.4.5, and Observation 4.2.13, we can obtain a new compression result for general protocols.

[7] We have added 1 bit per message because, sending ℓ_i ones to Alice, Bob should append a zero to them — recall that the messages must form a prefix-free set.

4.5.1. Corollary. *Suppose there exists a mixed-coin, q-round protocol π to compute f over the input distribution μ with error probability ε, and let $C = \mathsf{CC}(\pi)$, $I = \mathsf{IC}_\mu(\pi)$, $n = \log|\mathcal{X}| + \log|\mathcal{Y}|$. Then there exists a public-coin, $O(q)$-average-round protocol ρ that computes f over μ with error $\varepsilon + \delta$, and with*

$$\mathsf{CC}(\rho) \leq O\left(I + q\log\left(\frac{qnC}{\delta}\right)\right). \tag{4.3}$$

As we will see in the following sub-section, this will result in a new direct sum theorem for bounded-round protocols. In general, given that we have already proven Theorem 4.4.1, and given that this approach shows promise in the bounded-round case, it becomes worthwhile to investigate whether we can prove Conjecture 4.3.2 with similar techniques.

4.5.1 Direct-sum theorems for the bounded-round case

The following theorem was proven in [23]:

4.5.2. Theorem. **([23], Theorem 12.)** *Suppose that there is a q-round protocol π^k that computes k copies of f with communication complexity C and error ε, over the k-fold distribution μ^k. Then there exists a q-round mixed-coin protocol π that computes a single copy of f with communication complexity C and the same error probability ε, but with information cost $\mathsf{IC}_\mu(\pi) \leq \frac{2C}{k}$ for any input distribution μ.*

As a consequence of this theorem, and of Corollary 4.5.1, we will be able to prove a direct sum theorem. The proof is a simple application of Theorem 4.5.2, and Corollary 4.5.1.

4.5.3. Theorem (Direct sum theorem for the bounded-round case). *There is some constant d such that, for any input distribution μ and any $0 < \varepsilon < \delta < 1$, if f requires, on average, at least*

$$C + q\log\left(\frac{qnC}{\delta - \varepsilon}\right)$$

bits of communication, to be computed over μ with error δ in dq (average) rounds, then $f^{\otimes k}$ requires at least kC bits of communication, in the worst case, to be computed over $\mu^{\otimes k}$ with error ε in q rounds.

4.5.2 Comparison with previous results

We may compare Corollary 4.5.1 with the results of [29]. In that paper, the nC factor is missing inside the log of equation (4.3), but the number of rounds of the compressed protocol is $O(q\log I)$ instead of $O(q)$. A similar difference appears in the resulting direct-sum theorems.

We remark that the compression of Jain et al. [60] is also achieved with a round-by-round proof. Our direct-sum theorem is incomparable with their more ambitious direct-product result. It is no surprise, then, that the communication complexity of their compression scheme is $O(\frac{qI}{\delta})$, i.e., it incurs a factor

of q, whereas we pay only an additive term of $\tilde{O}(q)$. However, their direct-product result also preserves the number of rounds in the protocol, whereas in our result the number of rounds is only preserved within a constant factor.

4.6 Alternative constructions and matching lower bounds

4.6.1 A different upper bound on the degree of matching graphs

4.6.1. LEMMA. *For every integer $\ell \leq m$ and real $\delta > 0$ there is an (m, ℓ, d, δ)-matching graph with $d = (2 + (m - \ell)\ln 2)/\delta^2 + \ln(1/\delta)/\delta$.*

Proof. We show the existence of such a graph using a probabilistic argument. Let A and B be any sets of $M = 2^m$ left and $L = 2^\ell$ right nodes, respectively. Construct a random graph G by choosing d random neighbors independently for each $u \in A$. Different neighbors of the same node u are also chosen independently, thus they might coincide. For any $A' \subseteq A$ of size L, let $E_{A'}$ be the event that $G_{A' \cup B}$ does *not* have a matching of size $L(1-\delta)$, and let $\text{BAD} := \bigvee_{A'} E_{A'}$. Note that the lemma holds if $\Pr[\text{BAD}] < 1$.

Next, we bound $\Pr[E_{A'}]$. Let $A' = \{u_1, \ldots, u_L\}$ be any set of L left nodes. Let $\mathcal{N}(u)$ denote the neighborhood of a vertex u. Consider the following procedure for generating a matching for $G_{A' \cup B}$:

FIND-MATCHING
1 Matching $\leftarrow \emptyset$
2 $V \leftarrow \emptyset$
3 **for** $i \leftarrow 1$ **to** L
4 **if** $\mathcal{N}(u_i) \not\subseteq V$
5 pick arbitrary $v_i \in \mathcal{N}(u_i) \setminus V$
6 Matching $\leftarrow$ Matching $\cup \{(u_i, v_i)\}$
7 $V \leftarrow V \cup \{v_i\}$
8 **return** Matching

Define the indicator variables $X_1, \ldots, X_L$ as follows: $X_i = 1$ if the condition in the 4th line of Find-Matching is true and 0 otherwise. From the definition of these variables it follows that for all i and all $b = (b_1, \ldots, b_i) \in \{0,1\}^i$ the conditional probability of $X_{i+1} = 0$ given $X_1 = b_1, \ldots, X_i = b_i$ is equal to

$$(|b|/L)^d,$$

where $|b|$ stands for Hamming weight of vector b, i.e. the number of 1s in $b = (b_1, \ldots, b_i)$. Consider also similar random variables $Y_1, \ldots, Y_L$ where the distribution of $Y_1, \ldots, Y_L$ is defined by the formula

$$\Pr[Y_{i+1} = 0 | Y_1 = b_1, \ldots, Y_i = b_i] = \begin{cases} (|b|/L)^d, & \text{if } |b| < (1-\delta)L, \\ 1, & \text{if } |b| \geq (1-\delta)L. \end{cases}$$

In terms of $X_1, \ldots, X_L$ the event $E_{A'}$ happens if and only if $X_1 + \cdots + X_L < (1-\delta)L$. For every string b of Hamming weight less than $(1-\delta)L$ the probabilities $\Pr[X = b]$ and $\Pr[Y = b]$ coincide. Thus it suffices to upper bound the probability $\Pr[Y_1 + \cdots + Y_L < (1-\delta)L]$. To this end consider independent random variables $Z_1, \ldots, Z_L \in \{0,1\}$, where the probability of $Z_i = 1$ is $(1-\delta)^d$.

4.6.2. Claim. $\Pr[|Y| < (1-\delta)L] \leq \Pr[|Z| < (1-\delta)L]$.

Proof. We prove this using the coupling method. We claim that there is a joint distribution of Y and Z such that the marginal distributions are as defined above, and with probability 1 it holds that $Z_i \leq Y_i$ for all i. This joint distribution is defined by the following process: we pick L independent reals $r_1, \ldots, r_L \in [0;1]$ and let

$$Z_i = \begin{cases} 0, & \text{if } r_i < (1-\delta)^d; \\ 1, & \text{otherwise.} \end{cases}$$

$$Y_i = \begin{cases} 0, & \text{if } r_i < \left(\frac{Y_1+\cdots+Y_{i-1}}{L}\right)^d \text{ and } \frac{Y_1+\cdots+Y_{i-1}}{L} < 1-\delta; \\ 1, & \text{otherwise.} \end{cases}$$

We claim that the inequality $Z_i \leq Y_i$ (holding with probability 1) implies that for every downward closed set $E \subset \{0,1\}$ it holds $\Pr[Y \in E] \leq \Pr[Z \in E]$ (we call a set E downward closed if $b \in E$ and $b' \leq b$, component-wise, implies $b' \in E$). Indeed,

$$\Pr[Y \in E] \leq \Pr[Y \in E, Z \in E] \leq \Pr[Z \in E],$$

where the first inequality holds, since E is downward closed and thus $Y \in E$ implies $Z \in E$. The set of Boolean vectors $b \in \{0,1\}^L$ of Hamming weight less than $(1-\delta)L$ is downward closed hence the statement. ■

By this lemma it suffices to upper bound the probability

$$\Pr[Z_1 + \cdots + Z_L < (1-\delta)L],$$

which can be obtained by Chernoff bound.

Let $S := \sum(1 - Z_i)$, and let $\mu := \mathbb{E}[S]$, $p = (1-\delta)^d$. Note that $\mu = pL$. Also, let $\psi := \delta/p - 1$. Using the multiplicative version of the Chernoff bound,

so long as $\psi > 0$, we have

$$\begin{aligned}
\Pr[S > \delta L] &= \Pr[S > pL \cdot (\delta/p)] \\
&= \Pr[S > \mu(1+\psi)] \\
&< \left(\frac{e^{\psi}}{(1+\psi)^{(1+\psi)}}\right)^{\mu} \\
&= \exp\left(\mu\left(\frac{\delta}{p} - 1 - \frac{\delta}{p}\ln(\frac{\delta}{p})\right)\right) \\
&< \exp\left(\mu\left(\frac{\delta}{p} - \frac{\delta}{p}\ln(\frac{\delta}{p})\right)\right) \\
&= \exp\left(pL\frac{\delta}{p}\left(1 - \ln\delta + \ln p\right)\right) \\
&= \exp\left(\delta L + \delta L\ln(1/\delta) + \delta L\ln p\right) \\
&= \exp\left(\delta L\left(1 + \ln(1/\delta) + \ln p\right)\right) .
\end{aligned}$$

Thus for every set A' of L left nodes we have $\Pr[E_{A'}] < e^{\delta L(1+\ln(1/\delta)+\ln p)}$. There are $\binom{M}{L}$ subsets of A of size L. By Stirling's Formula, we have

$$\binom{M}{L} \le \frac{(M)^L}{L!} \le \left(\frac{Me}{L}\right)^L = \exp(L(1+\ln M/L)) .$$

By union bound we have

$$\begin{aligned}
\Pr[BAD] &\le \exp\left(M(1+\ln M/L)\right) \cdot \exp\left(\delta M(1+\ln(1/\delta)+\ln p)\right) \\
&= \exp\left(M + M\ln M/L + \delta M + \delta M\ln(1/\delta) + \delta M\ln p\right) \\
&< \exp\left(M + M\ln M/L + \delta M + \delta M\ln(1/\delta) - d\delta^2 M\right) \\
&< 1 ,
\end{aligned}$$

where the final inequality uses $d = (2 + \ln M/L)/\delta^2 + \ln(1/\delta)/\delta$, which also ensures that $\psi > 0$ whenever δ is sufficiently small. ■

4.6.2 A lower bound on the degree of matching graphs

4.6.3. Lemma. *An (m, ℓ, d, δ)-matching graph must have*

$$d = \Omega\left(\min\left(\frac{m-\ell}{\delta}, \delta 2^{\ell}\right)\right) .$$

Proof. We will prove that in such a bipartite graph there must exist a left-set A of size $2^m(1-4\delta)^d$ whose neighbours are contained in a right-set B of size $(1-2\delta)2^{\ell}$. If the graph is a matching graph with said parameters, it must then follow that $|A| \le 2^{\ell}$, hence $d \ge (m-\ell)/\log(1-4\delta) = \Omega((m-\ell)/\delta)$.

We show this through the probabilistic method. Let us pick a random right-set B of size $(1-2\delta)2^{\ell}$. For a given left-node a, the probability that all its neighbours fall into B is at least

$$\binom{2^{\ell}-d}{(1-2\delta)2^{\ell}-d} \Big/ \binom{2^{\ell}}{(1-2\delta)2^{\ell}} \ge (1-2\delta)^d \left(1 - \frac{2d}{2^{\ell}}\right)^d .$$

Under the assumption that $d \leq \delta 2^\ell$, the left-hand side is at least $(1-4\delta)^d$.

It must then hold that for such random B, the expected number of left-nodes that map into B is $2^m(1-4\delta)$. Hence, for some choice of B, there will exist a left-set A of the same size whose neighbours are all in B. ■

4.6.3 A lower bound for equation (4.2) of the proof of Lemma 4.3.6

4.6.4. LEMMA. *There is an ℓ-discrete private-coin one-way protocol π, and a message m sent by π, such that for J defined as in Lemma 4.3.6, it holds that*

$$I(J : X | M_\pi = m) = \Omega(\log \ell).$$

Proof. Suppose Alice is given an input X uniformly distributed over $\{x_1, \ldots, x_N\}$, and private randomness uniformly distributed over $\{r_1, \ldots, r_N\}$, so that $\ell = \log N$. Let π be a one-way protocol given by

$$M_\pi(x_j, r_k) = \begin{cases} 0 & \text{if } k \leq \left\lfloor \frac{N}{j+1} \right\rfloor, \\ 1 & \text{otherwise.} \end{cases}$$

Then conditioned on $M_\pi = 0$, we will have $J(x_j, r_k) = k$. Let $M = \sum_{i=1}^N \lfloor \frac{N}{i+1} \rfloor$ be the size of $M_\pi^{-1}(0)$. Finally, let m denote the event $M_\pi = 0$. Then

$$\begin{aligned} I(X : J|m) &= H(X|m) - H(X|m, J) \\ &= \sum_{j=1}^N \frac{1}{M} \cdot \left\lfloor \frac{N}{j+1} \right\rfloor \log \frac{M}{\left\lfloor \frac{N}{j+1} \right\rfloor} - \sum_{k=1}^N \frac{1}{M} \cdot \left\lfloor \frac{N}{k+1} \right\rfloor \log \left\lfloor \frac{N}{k+1} \right\rfloor \\ &= \log M - \frac{2}{M} \sum_{i=1}^N \left\lfloor \frac{N}{i+1} \right\rfloor \log \left\lfloor \frac{N}{i+1} \right\rfloor, \end{aligned}$$

which is $\geq U$ iff:

$$2 \sum_{i=1}^N \left\lfloor \frac{N}{i+1} \right\rfloor \log \left\lfloor \frac{N}{i+1} \right\rfloor \leq M(\log M - U) \tag{4.4}$$

Let us denote the left-hand side with A and the right-hand side with B. Because $\frac{N}{x}$ is monotonically decreasing for $x \geq 1$, then:

$$A \leq \frac{2}{\ln 2} \int_1^{N+1} \frac{N}{x} \ln \frac{N}{x} dx.$$

The relevant primitive is $\int \frac{N}{x} \ln \frac{N}{x} dx = -\frac{1}{2} N (\ln \frac{N}{x})^2$ and hence

$$\begin{aligned} A &\leq \frac{2}{\ln 2} \left(-\frac{1}{2} N \left(\ln \frac{N}{N+1} \right)^2 + \frac{1}{2} N (\ln N)^2 \right) \\ &= \frac{2}{\ln 2} \left(N \ln N \ln(N+1) - \frac{1}{2} N (\ln(N+1))^2 \right). \end{aligned}$$

We denote this last quantity by A'. Good bounds for M are:[8]

$$N \ln N - 3N \leq M = \sum_{i=1}^{N} \left\lfloor \frac{N}{i+1} \right\rfloor \leq N \ln N + N$$

Let $B' := N \ln N - 3N$, so that $B \geq B'(\log B' - U)$. Then we will show that for an appropriate choice of U,

$$A' \leq B'(\log B' - U)$$

and hence $A \leq B$ and also $I(X : J|m) \geq U$. Equivalently,

$$A' - B' \log B' + B'U \leq 0 \tag{4.5}$$

For convenience, let $\alpha = \frac{\ln(N+1)}{\ln N}$ (which goes to 1 as N goes to ∞). Then $A' = \frac{1}{\ln 2} N (\ln N)^2 (2\alpha - \alpha^2)$ and $B' \log B' = \frac{1}{\ln 2} N (\ln N)^2 + \frac{1}{\ln 2} N \ln N \ln \ln N + O(N \ln N)$. Now the proof follows from the following:

4.6.5. Claim. $N(\ln N)^2(2\alpha - \alpha^2 - 1) \to -\frac{1}{N}$ *as* $N \to \infty$.

Because under this claim, the dominant negative term in (4.5) is $\frac{1}{\ln 2} N \ln N \ln \ln N$, and thus all we need to do is set U to be $c \ln \ln N$ for some $c < \frac{1}{\ln 2}$, that this ensures (4.5) is negative. For such a choice of U, it will hold that

$$I(X : J|m) \geq U = c \ln \ln N = \Omega(\log \ell).$$

Unfortunately, l'Hopital's rule does not seem to help us, as the terms become too complicated. Instead we estimate how fast $(2\alpha - \alpha^2 - 1)$ approaches 0 as N goes to infinity. For this, let $\beta = \frac{\ln(\frac{1}{x}+1)}{\ln \frac{1}{x}}$ and let us estimate β as x approaches 0. For x close to, but different from, 0, we have:

$$\beta = 1 - \frac{1}{\ln x} \ln(x+1) = 1 - \frac{x}{\ln x} + \frac{x^2}{2 \ln x} \pm O\left(\frac{x^3}{\ln x}\right)$$

(the last equality is by the Taylor expansion of $\ln(x+1)$ around 0). We also have

$$\beta^2 = \left(1 - \frac{x}{\ln x} + \frac{x^2}{2 \ln x} - O\left(\frac{x^3}{\ln x}\right)\right)^2 = \beta - \frac{x}{\ln x} + \frac{x^2}{(\ln x)^2} + \frac{x^2}{2 \ln x} \pm O\left(\frac{x^3}{(\ln x)^2}\right).$$

Hence,

$$2\beta - \beta^2 = 1 - \frac{x^2}{(\ln x)^2} \pm O\left(\frac{x^3}{(\ln x)^2}\right).$$

From this we can conclude that for $x = 1/N$, we have

$$2\alpha - \alpha^2 - 1 = -\frac{1}{N^2 (\ln N)^2} \pm O\left(\frac{1}{N^3 (\ln N)^2}\right),$$

and our claim follows. ■

[8] This is because the harmonic numbers $H_n = \sum_{i=1}^{n} 1/i$ converge to $\ln N + \gamma$ for the Euler–Mascheroni constant $\gamma \approx 0.577$.

Bibliography

[1] Scott Aaronson. Complexity zoo. URL `https://complexityzoo.uwaterloo.ca/Complexity_Zoo`.

[2] Manindra Agrawal and Vikraman Arvind. Geometric sets of low information content. *Theoretical Computer Science*, 158(1-2):193–219, 1996.

[3] Eric Allender, Lane A. Hemachandra, Mitsunori Ogiwara, and Osamu Watanabe. Relating equivalence and reducibility to sparse sets. *SIAM Journal of Computing*, 21(3):521–539, 1992.

[4] Eric Allender, Harry Buhrman, and Michal Koucký. What can be efficiently reduced to the Kolmogorov-random strings? *Annals of Pure and Applied Logic*, 138:2–19, 2006.

[5] Eric Allender, Harry Buhrman, Michal Koucký, Dieter van Melkebeek, and Detlef Ronneburger. Power from random strings. *SIAM Journal on Computing*, 35:1467–1493, 2006.

[6] Eric Allender, Harry Buhrman, Luke B. Friedman, and Bruno Loff. Reductions to the set of random strings: the resource-bounded case. In *Proceedings of the 37th MFCS*, pages 88–99, 2012.

[7] Eric Allender, George Davie, L. Friedman, Samuel B. Hopkins, and Iddo Tzameret. Kolmogorov complexity, circuits, and the strength of formal theories of arithmetic. Technical Report TR12-028, 2012. Submitted for publication.

[8] Eric Allender, Luke B. Friedman, and William Gasarch. Limits on the computational power of random strings. *Information and Computation*, 222:80–92, 2013. ICALP 2011 special issue.

[9] Dana Angluin. Queries and concept learning. *Machine Learning*, 2(4): 319–342, 1987.

[10] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009.

[11] V. Arvind, Y. Han, L. Hemachandra, J. Köbler, A. Lozano, M. Mundhenk, M. Ogiwara, U. Schöning, R. Silvestri, and T. Thierauf. Reductions to sets of low information content. In K. Ambos-Spies, S. Homer, and U. Schöning, editors, *Complexity Theory: Current Research*, pages 1–45. Cambridge University Press, 1993.

[12] Vikraman Arvind and Jacobo Torán. Sparse sets, approximable sets, and parallel queries to NP. In *Proceedings of the 16th STACS*, pages 281–290, 1999.

[13] Vikraman Arvind, Johannes Köbler, and Martin Mundhenk. On bounded truth-table, conjunctive, and randomized reductions to sparse sets. In *Proceedings of the 12th FSTTCS*, pages 140–151, 1992.

[14] Vikraman Arvind, Johannes Köbler, and Martin Mundhenk. Lowness and the complexity of sparse and tally descriptions. In *Proceedings of the 3rd ISAC*, pages 249–258, 1992.

[15] Vikraman Arvind, Johannes Köbler, and Martin Mundhenk. Bounded truth-table and conjunctive reductions to sparse and tally sets. Technical report, University of Ulm, 1992. Technical Report Ulmer Informatik-Berichte 92–01.

[16] Vikraman Arvind, Johannes Köbler, and Martin Mundhenk. Hausdorff reductions to sparse sets and to sets of high information content. In *Proceedings of the 18th MFCS*, pages 232–241, 1993.

[17] Vikraman Arvind, Johannes Köbler, and Martin Mundhenk. Upper bounds for the complexity of sparse and tally descriptions. *Theory of Computing Systems*, 29:63–94, 1996.

[18] Vikraman Arvind, Johannes Köbler, and Martin Mundhenk. Monotonous and randomized reductions to sparse sets. *Theoretical Informatics and Applications*, 30(2):155–179, 1996.

[19] László Babai, Lance Fortnow, and Carsten Lund. Non-deterministic exponential time has two-prover interactive protocols. *Computational Complexity*, 1:3–40, 1991.

[20] Eric Bach and Jeffrey Shallit. *Algorithmic Number Theory I: Efficient Algorithms*. Cambridge University Press, 1996.

[21] J. L. Balcázar, J. Días, and J. Gabarró. *Structural Complexity I*. Springer-Verlag, 1988.

[22] Ziv Bar-Yossef, T. S. Jayram, Ravi Kumar, and D. Sivakumar. An information statistics approach to data stream and communication complexity. *Journal of Computer and System Sciences*, 68(4):702–732, 2004. Special issue of FOCS 2002.

[23] Boaz Barak, Mark Braverman, Xi Chen, and Anup Rao. How to compress interactive communication. In *Proceedings of the 42nd STOC*, pages 67–76, 2010.

[24] David A. Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1. *Journal of Computer and System Sciences*, 38(1):150–164, 1989.

[25] Janis Barzdin. Complexity of programs to determine whether natural numbers not greater than n belong to a recursively enumerable set. *Soviet Mathematics Doklady*, 9:1251–1254, 1968.

[26] Richard Bellman. Bottleneck problems and dynamic programming. *Proceedings of the National Academy of Sciences of the USA*, 39(9):947–951, 1953.

[27] Leonard Berman and Juris Hartmanis. On isomorphisms and density of NP and other complete sets. In *Proceedings of the 8th STOC*, pages 30–40, 1976.

[28] Mark Braverman. Interactive information complexity. In *Proceedings of the 45th STOC*, pages 505–524, 2012.

[29] Mark Braverman and Anup Rao. Information equals amortized communication. In *Proceedings of the 52nd STOC*, pages 748–757, 2011.

[30] Joshua Brody, Harry Buhrman, Michal Koucký, Bruno Loff, Florian Speelman, and Nikolay Vereshchagin. Towards a reverse Newman's theorem in interactive information complexity. In *Proceedings of the 23rd CCC*, pages 24–33, 2013.

[31] Nader H. Bshouty, Richard Cleve, Ricard Gavaldà, Sampath Kannan, and Christino Tamon. Oracles and queries that are sufficient for exact learning. *Journal of Computer and System Sciences*, 52(3):421–433, 1996.

[32] Harry Buhrman and Elvira Mayordomo. An excursion to the Kolmogorov random strings. *Journal of Computer and System Sciences*, 54:393–399, 1997.

[33] Harry Buhrman, Bruno Loff, and Leen Torenvliet. New hardness results for knapsack problems. Submitted.

[34] Harry Buhrman, Lance Fortnow, Ilan Newman, and Nikolay Vereshchagin. Increasing Kolmogorov complexity. In *Proceedings of the 22nd STACS*, pages 412–421, 2005. ISBN 978-3-540-24998-6.

[35] Harry Buhrman, Michal Koucký, and Nickolay Vereshchagin. Randomized individual communication complexity. In *Proceedings of the 23rd CCC*, pages 321–331, 2008.

[36] Harry Buhrman, Lance Fortnow, Michal Koucký, and Bruno Loff. Derandomizing from random strings. In *Proceedings of the 25th CCC*, pages 58–63, 2010.

[37] Harry Buhrman, Lance Fortnow, John Hitchcock, and Bruno Loff. Learning reductions to sparse sets. In *Proceedings of the 38th MFCS*, pages 243–253, 2013.

[38] Samuel R. Buss. The boolean formula value problem is in ALOGTIME. In *Proceedings of the 19th STOC*, 1987.

[39] Samuel R. Buss, Stephen A. Cook, Arvind Gupta, and Vijaya Ramachandran. An optimal parallel algorithm for formula evaluation. *SIAM Journal on Computing*, 21:755–780, 1992.

[40] Jin-Yi Cai. $S_2^p \subseteq \mathrm{ZPP}^{\mathrm{NP}}$. *Journal of Computer and System Sciences*, 73 (1):25–35, 2002.

[41] Jin-Yi Cai, Ashish Naik, and D. Sivakumar. On the existence of hard sparse sets under weak reductions. In *Proceedings of the 13th STACS*, pages 307–318, 1996.

[42] Amit Chakrabarti, Yaoyun Shi, Antony Wirth Wirth, and Andrew Chi-Chih Yao. Informational complexity and the direct sum problem for simultaneous message complexity. In *Proceedings of the 42nd FOCS*, pages 270–278, 2001.

[43] Ashok K. Chandra, Dexter Kozen, and Larry J. Stockmeyer. Alternation. *Journal of the ACM*, 28(1):114–133, 1981.

[44] Pierluigi Crescenzi and Viggo Kann. A compendium of NP optimization problems, 2005. URL `http://www.nada.kth.se/~viggo/problemlist/compendium.html`.

[45] Marek Cygan, Holger Dell, Daniel Lokshtanov, Dániel Marx, Jesper Nederlof, Yoshio Okamoto, Ramamohan Paturi, Saket Saurabh, and Magnus Wahlström. On problems as hard as CNF-SAT. In *Proceedings of the 27th CCC*, pages 74–84, 2012.

[46] Uriel Feige, David Peleg, and Prabhakar Raghavan. Computing with noisy information. *SIAM Journal on Computing*, 23(5):1001–1018, 1994.

[47] Lance Fortnow and Adam Klivans. NP with small advice. In *Proceedings of the 20th CCC*, pages 228–234, 2005.

[48] P. Hall. On representatives of subsets. *Journal of London Mathematical Society*, 1(10):25–30, 1935.

[49] Ryan Harkins and John M. Hitchcock. Dimension, halfspaces, and the density of hard sets. In Guohui Lin, editor, *Computing and Combinatorics*, volume 4598 of *Lecture Notes in Computer Science*, pages 129–139. Springer, 2007.

[50] Johan Håstad. Some optimal inapproximability results. *Journal of the ACM*, 48(4):798–859, 2001.

[51] John M. Hitchcock. Online learning and resource-bounded dimension: Winnow yields new lower bounds for hard sets. *SIAM Journal on Computing*, 36(6):1696–1708, 2007.

[52] Sorit S. Hochbaum and David B. Shmoys. A polynomial approximation scheme for machine scheduling on uniform processors: using the dual approach. *SIAM Journal on Computing*, 17:539–551, 1988.

[53] Ellis Horowitz and Sartaj Sahni. Computing partitions with an application to the subset-sum problem. *Journal of the ACM*, 20(2):277–292, 1974.

[54] Russel Impagliazzo. Hard-core distributions for somewhat hard problems. In *Proceedings of the 36th FOCS*, pages 538–545, 1995.

[55] Russel Impagliazzo and Ramamohan Paturi. The complexity of k-SAT. In *Proceedings of the 14th CCC*, pages 237–240, 1999.

[56] Russel Impagliazzo and Avi Wigderson. $P = BPP$ if E requires exponential circuits: Derandomizing the XOR lemma. In *Proceedings of the 29th STOC*, pages 220–229, 1997.

[57] Russel Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences*, 63(4):512–530, 2001.

[58] Rahul Jain and Ashwin Nayak. The space complexity of recognizing well-parenthesized expressions in the streaming model: the Index function revisited, 2010. URL `http://arxiv.org/abs/1004.3165`.

[59] Rahul Jain, Jaikumar Radhakrishnan, and Pranab Sen. A direct sum theorem in communication complexity via message compression. In *Proceedings of the 30th ICALP*, pages 300–315, 2003.

[60] Rahul Jain, Attila Pereszlenyi, and Penghui Yao. A direct product theorem for bounded-round public-coin randomized communication complexity. In *Proceedings of the 53rd FOCS*, pages 167–176, 2012.

[61] Stasys Jukna. *Extremal Combinatorics*. Texts in Theoretical Computer Science. Springer, New York, 2001.

[62] Jim Kadin. $\mathrm{P}^{\mathrm{NP}[O(\log n)]}$ and sparse Turing-complete sets for NP. *Journal of Computer and System Sciences*, 39(3):282–298, 1989.

[63] Ravi Kannan. Circuit-size lower bounds and non-reducibility to sparse sets. *Information and Control*, 55(1-3):40–56, 1982.

[64] Richard Karp and Richard Lipton. Some connections between nonuniform and uniform complexity classes. In *Proceedings of the 12th STOC*, pages 302–309, 1980.

[65] Johannes Köbler and Osamu Watanabe. New collapse consequences of NP having small circuits. In Zoltán Fülöp and Ferenc Gécseg, editors, *Automata, Languages and Programming*, volume 944 of *Lecture Notes in Computer Science*, pages 196–207. Springer, 1995.

[66] Martin Kummer. On the complexity of random strings. In Claude Puech and Rüdiger Reischuk, editors, *Proceedings of the 13th STACS*, volume 1046 of *Lecture Notes in Computer Science*, pages 25–36. Springer, 1996.

[67] Troy Lee and Andrei Romaschenko. Resource bounded symmetry of information revisited. *Theoretical Computer Science*, 345:386–405, 2005. Special Issue of MFCS 2004.

[68] Michael Lewin, Dror Livnar, and Uri Zwick. Improved rounding techniques for the MAX 2-SAT and MAX DI-CUT problems. In *Proceedings of the 9th ICPO*, pages 67–82, 2002.

[69] Ming Li and Paul Vitányi. *An Introduction to Kolmogorov Complexity and Its Applications*. Graduate Texts in Computer Science. Springer, New York, third edition, 2008.

[70] Maciej Liśkiewicz and Rüdiger Reischuk. The complexity world below logarithmic space. In *Proceedings of the 9th CCC (then known as Structures in Complexity Theory)*, pages 64–78, 1994.

[71] Daniel Lokshtanov and Jesper Nederlof. Saving space by algebraization. In *Proceedings of the 42nd STOC*, pages 321–330, 2010.

[72] Carsten Lund, Lance Fortnow, Howard Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. *Journal of ACM*, 39:859–868, 1992.

[73] Wolfgang Maass and György Turán. How fast can a threshold gate learn? In *Workshop on Computational Learning Theory and Natural Learning Systems*, pages 381–414, 1994. ISBN 0-262-58126-4.

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

[75] Donal A. Martin. Completeness, the recursion theorem, and effectively simple sets. *Proceedings of the AMS*, 17(4):838–842, 1966.

[76] Joseph S. Miller, Mingzhong Cai, Rachel Epstein, and Steffen Lempp. Private communication.

[77] Ketan Mulmuley. Lower bounds in a parallel model without bit operations. *SIAM Journal on Computing*, 28(4):1460–1509, 1999.

[78] Saburo Muroga, Iwao Toda, and Satoru Takasu. Theory of majority decision elements. *Journal of the Franklin Institute*, 271(5):376–418, 1961.

[79] Daniil Musatov. Improving the space-bounded version of Muchnik's conditional complexity theorem via naive derandomization. *Theory of Computing Systems*, pages 1–14, 2012.

[80] Daniil Musatov, Andrei Romaschenko, and Alexander Shen. Variations on Muchnik's conditional complexity theorem. *Theory of Computing Systems*, 49(2):227–245, 2011.

[81] Alexei Myasnikov, Andrey Nikolaev, and Alexander Ushakov. Knapsack problems in groups, 2013. URL `http://arxiv.org/abs/1302.5671`.

[82] Jesper Nederlof, Erik Jan van Leeuwen, and Ruben van der Zwaan. Reducing a target interval to a few exact queries. In *Proceedings of the 37th MFCS*, pages 718–727, 2012.

[83] Ilan Newman. Private vs. common random bits in communication complexity. *Information processing letters*, 39(2):67–71, 1991.

[84] Noam Nisan and Avi Wigderson. Hardness vs randomness. *Journal of Computer and System Sciences*, 49(2):149–167, 1994.

[85] Mitsunori Ogiwara and Osamu Watanabe. Polynomial-time bounded truth-table reducibility of NP sets to sparse sets. *SIAM Journal on Computing*, 20(3):471–483, 1991.

[86] Denis Pankratov. Direct sum questions in classical communication complexity. Master's thesis, University of Chicago, 2012.

[87] Desh Ranjan and Pankaj Rohatgi. On randomized reductions to sparse sets. In *Proceedings of the 7th STOC*, pages 239–242, 1992.

[88] Ran Raz and Shmuel Safra. A sub-constant error-probability low-degree test, and a sub-constant error-probability PCP characterization of NP. In *Proceedings of the 29th STOC*, pages 475–484, 1997.

[89] Renato Renner and Stefan Wolf. Simple and tight bounds for information reconciliation and privacy amplification. In *Proceedings of the 11th ASIACRYPT*, pages 199–216, 2005.

[90] Sandeep Sen. The hardness of speeding-up knapsack. Technical report, BRICS, 1998.

[91] Adi Shamir. IP = PSPACE. *Journal of ACM*, 39:869–877, 1992.

[92] David Slepian and Jack K. Wolf. Noiseless coding of correlated information sources. *IEEE Transactions on information Theory*, 19:471–480, 1973.

[93] Larry J. Stockmeyer and Albert R. Meyer. Word problems requiring exponential time. In *Proceedings of the 5th STOC*, pages 1–9, 1973.

[94] Luca Trevisan and Salil Vadhan. Pseudorandomness and average-case complexity via uniform reductions. In *Proceedings of the 17th CCC*, pages 129–138, 2002.

[95] Vijay V. Vazirani. *Approximation Algorithms*. Springer, 2004.

[96] Wikipedia. Multiplication algorithm. URL `http://en.wikipedia.org/wiki/Multiplication_algorithm`.