A monotone counterexample to a question on distributional complexity

P. Graham
https://orcid.org/0009-0007-8493-1505

October 24, 2026

Abstract

For a Boolean function under a product measure, distributional algorithmic complexity is the minimum expected number of adaptive queries needed to determine its value. Distributional subcube partition complexity is the minimum expected number of fixed coordinates in a monochromatic subcube partition. Köhler-Schindler and Steif asked in 2024 whether these quantities are equal for every monotone Boolean function. We answer their Question 2.1 in the negative with the explicit function $f(x_0, \dots, x_4) = x_1x_2x_3 \vee x_0x_4(x_1 \vee x_2 \vee x_3)$. For independent fair input bits its algorithmic complexity is 13/4, while its subcube partition complexity is 51/16. A finite recurrence proves the algorithmic lower bound, and an explicit partition gives the strict inequality. An exact linear-programming dual certificate proves the partition value is optimal. The strict inequality is formalized in Lean; independent exact checkers verify the additional dual certificate.

1 Introduction

A deterministic adaptive algorithm evaluates a Boolean function by asking for input bits, with each query depending on previous answers. A subcube partition instead assigns each input to a unique partial assignment that determines the function. Every decision tree gives such a partition through its leaves, but not every partition comes from a decision tree. We compare their average costs when the bits are independent fair coins.

Following Laurin Köhler-Schindler and Jeffrey E. Steif [1], write π_p for the product measure with independent Bernoulli(p) coordinates. For $g : \{0,1\}^n \to \{0,1\}$, let $a(g, \pi_p)$ be the minimum expected number of adaptive queries until g is determined. A subcube fixes some coordinates and leaves the others free; its codimension is the number fixed. A partition is monochromatic for g if g is constant on each cell. Write $\mathrm{sc}(g, \pi_p)$ for the minimum expected codimension of the cell containing the random input, over all such partitions. The decision-tree partitions imply $a(g, \pi_p) \geq \mathrm{sc}(g, \pi_p)$.

Köhler-Schindler and Steif's paper was first posted on August 23, 2024 [1]. We quote their Question 2.1 from arXiv version 1:

> Does $a(f, \pi_p) = sc(f, \pi_p)$ hold for every monotone Boolean function f and for every $p \in (0, 1)$?

Here monotone means nondecreasing in each input coordinate. The following example answers this question.

The proofs, the Lean formalization, the search code, the certificate checkers and the text were produced by AI models (from OpenAI and Anthropic), working under the author's direction.

Theorem 1. *The Boolean function*

$$f(x_0, x_1, x_2, x_3, x_4) = x_1x_2x_3 \vee x_0x_4(x_1 \vee x_2 \vee x_3) \tag{1}$$

is monotone and satisfies

$$a(f, \pi_{1/2}) = \frac{13}{4} > \frac{51}{16} = \mathrm{sc}(f, \pi_{1/2}).$$

Products in (1) mean Boolean conjunction, and $\vee$ means disjunction.

The contribution is this small monotone example. Köhler-Schindler and Steif already gave strict distributional gaps for the nonmonotone ALL-EQUAL function on three inputs and the four-input function h [1, arXiv version 1, Example 2.4 and Section 2.3]. The additional task is to preserve monotonicity while making a partition cheaper than every adaptive query strategy. For (1), the main lower-bound obligation is to rule out all first queries and their possible continuations, rather than compare only a few natural algorithms. The finite recurrence below does so. We make no claim of minimality in the number of inputs, of a gap at other values of p, or of an asymptotic separation.

Earlier separations between decision trees and subcube partitions concern other costs. Kothari, Racicot-Desloges and Santha [2] established an asymptotic separation of randomized decision-tree complexity from deterministic subcube partition complexity using iterated quaternary majority. Those are worst-input costs, not the product-measure expectations here. Arunachalam, Chakraborty, Koucký, Saurabh and de Wolf studied average unambiguous certificate complexity [3, version 2, Section 2, unambiguous certificate complexity]; after relabeling $\{-1, 1\}$ as $\{0, 1\}$, their quantity is exactly $\mathrm{sc}(g, \pi_{1/2})$. Our definitions and question come from Köhler-Schindler and Steif, and these earlier works provide the surrounding comparison framework.

Proof overview

The formula for f makes monotonicity immediate. Section 2 evaluates the optimal adaptive cost by conditioning on the next query. Section 3 gives a monochromatic partition of smaller average codimension; this already answers Question 2.1. It then proves that the partition is optimal by assigning a signed weight to each input. The sum of the weights in every monochromatic subcube is at most that subcube's total codimension cost, so summing over any partition gives the required lower bound. Section 4 explains the finite checks and their precise formalization boundary.

This work has not yet been reviewed by experts.

2 The adaptive cost

We determine the best adaptive algorithm by a recurrence that considers every available next query. A restriction $r \in \{0, 1, *\}^5$ specifies some coordinates and leaves the starred coordinates free. For example, $00 * **$ fixes $x_0 = x_1 = 0$. Let $D(r)$ be the minimum expected number of additional queries for f, conditional on the input belonging to r, under $\pi_{1/2}$. All unqueried coordinates remain independent fair bits. An irrelevant coordinate can be omitted: its answer is independent of the remaining coordinates, and choosing the cheaper of the two continuation strategies without asking that query cannot increase the expected cost. Let $r^{i,b}$ be the restriction obtained by replacing a free coordinate i with $b \in \{0, 1\}$.

Lemma 2. *If f is constant on r, then $D(r) = 0$. Otherwise,*

$$D(r) = 1 + \min_{i : r_i = *} \frac{D(r^{i,0}) + D(r^{i,1})}{2}. \tag{2}$$

Proof. A query to an already fixed coordinate gives no information and can be omitted. If the value is not determined, the first remaining query must therefore be to a free coordinate i. Each answer has conditional probability $1/2$. The two continuations have expected costs at least $D(r^{i,0})$ and $D(r^{i,1})$. Conversely, choosing a minimizing i and optimal continuations attains the right side. Induction on the number of free coordinates proves the assertion and existence of the minima. □

The recurrence starts at fully specified inputs and works upward. Its values after the first query are shown in Table 1. The two kinds of row reflect the symmetries exchanging x_0 with x_4 and permuting x_1, x_2, x_3. Here $D(x_i = b)$ denotes $D(r)$ for the restriction with $r_i = b$ and all other coordinates free. Both rows give

First queried coordinate i	$D(x_i = 0)$	$D(x_i = 1)$
$0, 4$	$7/4$	$11/4$
$1, 2, 3$	$15/8$	$21/8$

Table 1: Optimal remaining query costs under each first answer. All other coordinates are free.

$$1 + \tfrac{1}{2}(7/4 + 11/4) = 1 + \tfrac{1}{2}(15/8 + 21/8) = 13/4.$$

Here is a short derivation of the table. For $k \geq 1$, the conjunction AND_k of k distinct fair bits, and likewise their disjunction OR_k, has optimal expected cost $\sum_{j=0}^{k-1} 2^{-j}$. Indeed, after any first query one answer determines the value and the other leaves the same problem on $k-1$ bits. Thus the costs for two and three bits are $3/2$ and $7/4$.

Two three-variable functions will also occur: $u(v \vee w)$ and $u \vee vw$. Both have optimal cost $7/4$. Querying u first leaves either a constant or an OR of two bits (respectively an AND of two bits), so costs $1 + \frac{1}{2}(3/2) = 7/4$. Querying v or w first instead leaves a single bit in one branch and an AND or OR of two bits in the other, for cost $1 + \frac{1}{2}(1 + 3/2) = 9/4$.

If $x_0 = 0$, the function is $x_1x_2x_3$, of cost $7/4$. If $x_0 = 1$, write $(a, b, c, d) = (x_1, x_2, x_3, x_4)$; the function is $abc \vee d(a \vee b \vee c)$. Querying d first leaves AND_3 or OR_3, giving cost $1 + 7/4 = 11/4$. Querying any of a, b, c first leaves the two three-variable types just considered, each of cost $7/4$, and also gives $11/4$. This proves the first row.

If $x_1 = 0$, the remaining function has the form $uv(w \vee z)$. Querying u or v first leaves a constant or a function of cost $7/4$, giving $1 + \frac{1}{2}(7/4) = 15/8$. Querying w or z first leaves AND_3 or AND_2, giving $1 + \frac{1}{2}(7/4 + 3/2) = 21/8$. Thus the minimum is $15/8$. If $x_1 = 1$, the remaining function has the form $uv \vee wz$. Every first query is equivalent by symmetry and leaves AND_2 in one branch and $u \vee vw$ in the other. Its cost is therefore $1 + \frac{1}{2}(3/2 + 7/4) = 21/8$. This proves the second row.

Thus $D(*****) = a(f, \pi_{1/2}) = 13/4$. The full restriction table in `data/certificate.json` also certifies these values, as described in Section 4.

3 The partition cost

We first exhibit a partition cheaper than any algorithm, then give a matching lower bound for all monochromatic partitions. In Table 2, each string lists coordinates in the order x_0, x_1, x_2, x_3, x_4. The cells are pairwise disjoint and cover the cube, and direct substitution in (1) gives the indicated constant value. Here $|C|$ is the number of inputs in a cell C. Its total codimension cost is $|C| \operatorname{codim} C$, the sum of that cell's cost over its inputs. Consequently its expected codimension is

Cell C	Value of f	$\lvert C\rvert$	$\operatorname{codim} C$
`00***`	0	8	2
`01*0*`	0	4	3
`*111*`	1	4	3
`0101*`	0	2	4
`1*0*0`	0	4	3
`110*1`	1	2	4
`101*0`	0	2	4
`10*11`	1	2	4
`1*101`	1	2	4
`10001`	0	1	5
`11100`	0	1	5

Table 2: A monochromatic partition with total codimension cost 102.

k	0	1	2	3	4	5	6	7
y_k	-1	1	1	3	1	3	3	5
k	8	9	10	11	12	13	14	15
y_k	1	3	3	5	3	5	5	3
k	16	17	18	19	20	21	22	23
y_k	1	5	3	5	3	5	5	3
k	24	25	26	27	28	29	30	31
y_k	3	5	5	3	5	3	3	1

Table 3: An integer dual certificate for the partition lower bound.

$$\frac{1}{32}\sum_{C}|C|\operatorname{codim} C=\frac{2\cdot 8+3\cdot 12+4\cdot 10+5\cdot 2}{32}=\frac{51}{16}. \tag{3}$$

Together with the adaptive lower bound, this proves the strict inequality.

The partition also makes the distinction from a decision tree visible. Every first query would have to fix its coordinate in every leaf cell. Here x_0 is free in `*111*`, x_1 is free in `1*0*0`, and x_2, x_3, x_4 are free in `00***`. Thus no coordinate can be the first query of a tree inducing this partition. This observation alone does not prove the gap: another tree might still cost 51/16, and the adaptive lower bound rules that out.

For completeness, the partition is optimal. Encode a point by the integer $k=\sum_{i=0}^{4}2^i x_i$, and give it weight y_k from Table 3. These weights sum to 102 and satisfy

$$\sum_{x\in C} y_{\sum_i 2^i x_i} \;\le\; |C|\operatorname{codim} C \quad \text{for every monochromatic subcube } C. \tag{4}$$

The inequalities are finite integer comparisons; both exact checkers enumerate all restrictions and check (4) on each monochromatic one. No optimization solver is needed to verify them. Summing (4) over any monochromatic partition $\mathcal{C}$ counts every input once, and gives

$$102=\sum_{k=0}^{31} y_k \le \sum_{C\in\mathcal{C}}|C|\operatorname{codim} C.$$

The negative weight at $k = 0$ causes no difficulty: the cells form an exact partition, not a cover with multiplicities. Division by 32, together with (3), proves $\mathrm{sc}(f, \pi_{1/2}) = 51/16$ and completes the proof of Theorem 1.

4 Exact checks and formalization

This section describes the finite evidence for the displayed calculations and distinguishes the Lean theorem from the additional partition lower bound. The accompanying result archive contains the Lean project in `lean/`, the certificate in `data/`, and the checkers in `checks/`. File paths here are relative to that archive's result directory; its `README.md` gives the build commands and a fixed-version citation. We obtained the example using our own Python code to enumerate monotone functions on five inputs up to coordinate permutations. It computes adaptive costs by exact rational dynamic programming and minimizes partition costs with Google OR-Tools' CP-SAT solver [4]. This discovery code is not included in the result archive; the search and its optimizer are not needed to check the result.

The certificate in `data/certificate.json` contains the truth table, a rational lower-bound table $L(r)$ for the $3^5 = 243$ restrictions, an attaining decision tree, the partition, and the dual weights. Each of two independently written Python checkers checks constancy and the inequalities

$$L(r) \le 0 \quad \text{if } f \text{ is constant on } r, \qquad L(r) \le 1 + \frac{L(r^{i,0}) + L(r^{i,1})}{2}$$

for every free coordinate of every nonconstant restriction. Induction using Lemma 2 gives $L(r) \le D(r)$, in particular $D(*****) \ge 13/4$. Both checkers validate the decision tree and the partition, check all the dual inequalities, and use exact rational arithmetic. The second checker also recomputes the adaptive optimum without relying on the lower-bound table. Corruption controls alter the truth table, lower bounds, tree, partition, and dual weights; both checkers must reject each altered certificate.

The Lean project proves the strict inequality independently of the Python checkers and certificate. Its theorem `KS24.answer` states that f is monotone; every correct adaptive decision tree has total query cost at least 104 on the 32 inputs; some such tree attains 104; and some monochromatic partition has total codimension 102. The theorem `KS24.gap` states that this partition costs strictly less than every correct tree. Dividing the sums by 32 gives precisely $a(f, \pi_{1/2}) = 13/4$ and $\mathrm{sc}(f, \pi_{1/2}) \le 51/16$. The lower bound $\mathrm{sc}(f, \pi_{1/2}) \ge 51/16$ is established by the separate dual certificate and the summation argument above; it is not part of the Lean theorem.

The formal model counts queries only until the value is first determined. It allows repeated queries, and proves its lower bound by induction on an arbitrary finite decision tree. A deterministic algorithm truncated at determination gives such a tree; conversely a tree can be extended by querying all remaining bits without changing the cost up to determination. A partition cell is represented by a bit mask recording which coordinates are fixed, together with their values. Every input must lie in exactly one cell, and codimension counts the fixed coordinates among the five inputs. Discarding empty cells or mask bits outside those coordinates leaves the same ordinary subcube partition. These identifications connect the formal statement to the definitions of [1].

Lean checks the finite calculations by reducing expressions inside its proof-checking kernel, rather than trusting an external computation. Build checks called axiom guards also fail if the final theorems acquire assumptions beyond the listed logical axioms: `propext` (propositional extensionality) and `Quot.sound` (soundness of quotients). Sections 2 and 3 already give a written proof of the strict inequality. Its independent Lean verification depends on Lean's kernel and the

correspondence of the finite definitions with the mathematical ones. The exact partition optimum additionally depends on the two Python checkers, Python's integer and rational arithmetic, and the weak-duality argument above. The public README gives the standard build and certificate-checking commands.

References

[1] Laurin Köhler-Schindler and Jeffrey E. Steif, *A study of distributional complexity measures for Boolean functions*, ACM Transactions on Computation Theory (2026), published online May 19, 2026. https://doi.org/10.1145/3814613. First posted as arXiv:2408.12995 on August 23, 2024, https://arxiv.org/abs/2408.12995v1. Question and example numbering cited from arXiv version 1, the only version listed in the arXiv history. The open author manuscript linked from Steif's publications page retains Question 2.1 with the same wording and number: https://www.math.chalmers.se/~steif/p74.pdf. ACM's final typeset text was not accessible to us.

[2] Robin Kothari, David Racicot-Desloges, and Miklos Santha, *Separating Decision Tree Complexity from Subcube Partition Complexity*, Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2015), LIPIcs **40** (2015), 915–930. https://doi.org/10.4230/LIPIcs.APPROX-RANDOM.2015.915.

[3] Srinivasan Arunachalam, Sourav Chakraborty, Michal Koucký, Nitin Saurabh, and Ronald de Wolf, *Improved bounds on Fourier entropy and Min-entropy*, arXiv:1809.09819, first posted September 26, 2018; version 2, September 17, 2021. https://arxiv.org/abs/1809.09819.

[4] Google OR-Tools contributors, *OR-Tools: CP-SAT solver*. https://developers.google.com/optimization/cp/cp_solver.