On Superlinear Lower Bounds in Complexity Theory

Kenneth W. Regan[*]
State University of New York at Buffalo

Abstract

This paper first surveys the near-total lack of superlinear lower bounds in complexity theory, for "natural" computational problems with respect to many models of computation. We note that the dividing line between models where such bounds are known and those where none are known comes when the model allows *non-local communication with memory* at *unit cost*. We study a model that imposes a "fair cost" for non-local communication, and obtain modest superlinear lower bounds for some problems via a Kolmogorov-complexity argument. Then we look to the larger picture of what it will take to prove really striking lower bounds, and pull from ours and others' work a concept of *information vicinity* that may offer new tools and modes of analysis to a young field that rather lacks them.

1 The Problem of Superlinear Lower Bounds

When the subject of the NP-complete problems comes up, people think about the NP $\neq$? P question: whether they have super-polynomial time lower bounds. But the gap in our knowledge is much wider than that: for all but a scant few of these problems, there is currently no super-*linear* time lower bound, not even for a deterministic Turing machine (DTM) with two tapes. All of the twenty-one problems in Karp's foundational paper [Kar72] extending Cook's Theorem [Coo71] belong to nondeterministic TM linear time (NLIN) under reasonable encoding schemes, but none—not *SAT*, not *Clique*, nor *Hamiltonian Path* nor *Subset Sum*, has been proved to lie outside DTM linear time (DLIN).

We do have the theorem of Paul, Pippenger, Szemerédi and Trotter [PPST83] that NLIN $\neq$ DLIN. When unwound, the proof of this theorem provides a lower bound of $\Omega(n \cdot (\log^* n)^{1/4})$ on the running time of any deterministic TM that accepts a certain language in NLIN (whose construction goes through Σ_4-alternating TMs). This barely-superlinear bound also applies to all languages that are complete for NLIN under DLIN many-one reductions ($\leq_m^{lin}$). That such languages exist is not a simple matter of defining $L = \{\langle N, x, 0^m \rangle$: the NTM N accepts x within m steps$\}$, because the NTMs N may have any number of tapes. However, Book and Greibach [BG70] showed that every language in NLIN is accepted in real time (i.e., time $n+1$) by some NTM N' with two worktapes. The alphabet of N' may be arbitrarily large, but at the cost of losing the real time, we can replace N' by a linear-time N'' with work alphabet $\{0,1\}$. Now define $L' := \{\langle N'', x, 0^{m|N''|} \rangle : N''$ accepts x in m steps, where N'' has the form above$\}$. Then L' is NLIN-hard under $\leq_m^{lin}$, and also $L' \in$ NLIN.

However, the most efficient versions of Cook's Theorem known to date (see [Rob91, BG93, Sch78]) transform a time-$t(n)$ NTM N into formulas that have $O(t(n) \log t(n))$ variables, so while *SAT* belongs to NLIN it may not be NLIN-hard. Grandjean [Gra88, Gra90b] proved that a few NP-complete problems are NLIN-hard under $\leq_m^{lin}$, including just one listed in [GJ79], called "Reduction to Incompletely Specified Automaton":

> Given a DFA M with some arcs marked, and an integer k, is there a way to redefine the marked arcs to obtain a DFA M' such that M' is equivalent to a DFA M'' that has at most k states?

None of the problems identified by Grandjean is known to belong to NLIN, but at least they definitely do not belong to DTIME$[o(n(\log^* n)^{1/4})]$.

Still, if one moves to a machine model that is slightly richer than the standard TM, even the lower bound of [PPST83] goes away. Deterministic TMs with planar tapes may still accept NLIN-complete languages in linear time. The linear-time classes

[*]This work was supported in part by NSF Grant CCR-9409104. Author's current address: Computer Science Department, 226 Bell Hall, UB North Campus, Buffalo, NY 14260-2000. Email: *regan@cs.buffalo.edu*, tel.: (716) 645–3180x114, fax: (716) 645–3464.

$\mathrm{DTIME}^d[O(n)]$ for TMs with d-dimensional tapes (d-TMs) form a hierarchy that many suspect to be proper. To simulate a linear-time d-TM M by a standard TM M', the best known time is $O(n^{2-1/d})$. If one requires that the simulation be *on-line*, meaning in general that every t steps of M are simulated by t' corresponding steps of M', then a lower bound that matches this upper bound was proved early on by Hennie [Hen66]. If M is a *tree-computer* (TC); i.e., a TM with binary tree-structured tapes, then this time is $O(n^2)$, again with a matching lower-bound for the on-line case [Hen66] (see also [PR81, Lou81, Lou83, Lou84, LL92]). However, this does not prove that the language classes $\mathrm{DTIME}^d[O(n)]$ and TC-TIME$[O(n)]$ are distinct from DLIN or from each other. Moreover, none of these classes above DLIN is known to differ from its nondeterministic counterpart. The upshot is that DLIN may be as much as quadratically weaker than these other reasonable notions of linear time, making our inability to prove bounds against DLIN for most NP-complete problems all the more flabbergasting.

One can, of course, construct languages and functions that are not in these linear time classes by diagonalization. But these methods are intuitively "artificial." Adachi and Iwata [AI84] (see also [KAI79]) proved $\Omega(n^k)$ lower bounds on certain pebble games in P, but these are tied closely to TM simulation and diagonalization.[1] What we are most interested in, besides the major NP-complete problems, are "natural" computational tasks of a simpler kind: sorting, finding elements with duplicates on a list, arithmetic in finite fields, Fast Fourier Transform, matrix transpose, matrix multiplication, to name a few. All but the last belong to $\mathrm{DTIME}[n \log n]$; the best known time to multiply two $n \times n$ integer or Boolean matrices is $n^{2.376\cdots}$ [CW90], which gives time $N^{1.188\cdots}$ when $N = n^2$ is regarded as the input length. The first three have been the focus of several efforts to prove super-linear lower bounds on TMs; these efforts have been neatly summarized by Mansour, Nisan, and Tiwari [MNT93], and their technique is a major topic below. The two simplest *languages* that have attracted similar efforts are the language of lists with no duplicate elements, and the language of undirected graphs that have a triangle. The former can be solved with one call to sorting, but the best known time to solve the latter (even on a unit-cost RAM) is $N^{1.188\cdots}$ by calculating $A^2 + A$, where A is the adjacency matrix of the graph.

Note that the famous $\Omega(n \log n)$ lower bound on sorting applies to a model where the only operation allowed on numbers is to compare them. The lower bound is preserved when numbers can also be added, subtracted, and multiplied [PS84], but is not known when division or bitwise Boolean operations (at log-cost; i.e., per-op cost proportional to the bit-length of the numbers) are allowed. Allow a TM to get its mitts on the bits on a list of m r-bit numbers (say $r \approx 2 \log n$), and no one has shown that the TM can't sort them in $O(n)$ time, where $n = mr$ is the true bit-length of the list. For more in this line, see [FW90, FW93]. Aggarwal and Vitter [AV88] called the task of extending their lower bounds to models that "allow arbitrary bit-manipulations and dissections of records" a "challenging open problem." They do not offer such a model, and the related work of [AACS87, ACS87, AC88, ACF90, ACS90, Vit91, VN92] still treats integers and records as indivisible units.

The sequential-time lower bounds in the last-mentioned papers are "modest," by which we mean $\Omega(n \log n)$ or $\Omega(n \log\log n)$ and the like. Let us call a bound of $\Omega(n^{1+\epsilon})$, for some fixed $\epsilon > 0$, *strong*. A strong lower bound puts a problem out of reach of the *quasilinear* time class $\mathrm{DQL} = \mathrm{DTIME}[qlin]$ for TMs, where $qlin = n \cdot (\log n)^{O(1)}$. Schnorr [Sch78] proved (as mentioned above) that *SAT* is complete for nondeterministic *qlin* time (NQL) under DQL-reductions, and the catalogued results of Dewdney [Dew81, Dew82, Dew89] extend this to many other problems in [GJ79]. Time *qlin* on the TC may still be quadratically more powerful than DQL in the above sense. However, it is "robust" insofar as it equals time *qlin* on a wide variety of "reasonable RAM" models: the *log-cost RAM* of Cook and Reckhow [CR73], the *successor RAM* (SRAM) and its relatives (see [WW86]), the random-access TMs of Gurevich and Shelah [GS89] (the hub paper for the robustness), the "pointer machines" of Schönhage [Sch80, Sch88], and the models of Grandjean and Robson [GR91], Jones [Jon93], and Grandjean [Gra93, Gra94b, Gra94a]. A strong lower bound against these models also puts a problem out of reach of the "$\bigcap_{\epsilon>0} time\ n^{1+\epsilon}$" class of Graedel [Gra90a], which is robust for these RAMs and also for TMs that can have tapes of arbitrary dimen-

[1] S. Buss [Bus94] has recently proved that the language $\{\langle \phi, n \rangle : \phi$ has a proof in first-order logic that is less than n symbols long$\}$ requires time $\Omega(2^N)$ infinitely often on a DTM, and time $\Omega(2^N/N)$ on an NTM. Here n is written in binary and the result holds even when ϕ is restricted to have length $N = O(\log n)$ when encoded over a finite alphabet. But this is up at the level of complete sets for (nondeterministic) exponential time. Buss also shows that when n is written in unary and ϕ can have size $O(n)$, the problem is NP-complete. His techniques hold some promise for progress on whether *SAT* is NLIN-complete.

sion. But even modest lower bounds are hard enough to get, enough to prompt some writers to say that the lowly standard TM is "intractable to analyze for lower bounds."

In search of lower bounds, researchers have studied models that are weaker than the standard TM. Among several sources of super-linear lower bounds for these models, we mention [DGPR84, MS86, LV88, DMS91, LLV92, DM93]. The surveys [WW86, vEB90] give a window onto the vast literature of machine models of all powers. In an attempt to find a *unifying principle* that divides (1) all the models for which such lower bounds are currently known from (2) those on which we've come up empty (or fourth-root of log-star from empty), we offer the following observation:

> Does your model permit communication between remote parts of memory at unit cost? Does it operate on bits? Then it is in category (2).

For example, the tight lower bounds of $n^{3/2}$ and $n^{5/4}$ (up to some log factors) for matrix transpose and sorting proved in [DMS91, DM93] apply to TMs with only one worktape. A second worktape, or a second head on the one tape, allows remote communication at unit cost and blows away the bound. (See also [FMR72, Kos79, LS81, JL93, JSV94].) VLSI and systolic models by definition allow only local communication, and there has been much success on lower bounds and *area-time* tradeoffs for them (see [MC80, CM85, Gru90, HKMW92]).

The approach proposed here is a logical response to the above observation:

> Let us allow remote communication, but charge a "fair cost" for it, and then study the effect of this charge on the running time.

Section 2 describes a machine model, called *Block Move*, which carries out this motivation, and which binds practical elements such as *latency*, *pipelining*, and *stream transductions* that the older models lack. Section 3 proves "modest" lower bounds in this model for some string-editing and permutation problems. Section 4 reviews the $\Omega(n^2)$ time-space tradeoff lemmas for functional branching programs (BPs) due to Mansour, Nisan, and Tiwari [MNT93]. The general idea is that if time t on your model translates to time-space $o(t^2)$ for BPs, then any function with the tradeoff—this includes sorting and finite-field arithmetic [MNT93]—has a superlinear time lower bound. We point out that these lemmas apply also to *nondeterministic* BPs computing *multivalued* functions in the "NPMV" sense of Selman (see [Sel94]). Section 5 shows how these upgraded lemmas combine with the Kolmogorov-complexity argument of section 3 to yield a lower-bound technique; however, this leads to interesting combinatorial problems about BPs that we have so far been unable to solve. Finally, Section 6 raises the greater goal of proving *strong* lower bounds. In considering what of greater general value can be learned from *Block Move* and the "modest" lower bounds, we offer a notion of *information vicinity* that extends a recent treatment by Feldman and Shapiro [FS92], and that at least attempts to get beyond the notorious *relativization results* that have cast a pall on many efforts to prove strong lower bounds.

Sources and Acknowledgements. Sections 1 and 2 supplement my earlier survey article [Reg93]. The theorem in Section 3, which solves a problem left open in [Reg93], appeared with only a "Proof Sketch" in [Reg94a] owing to a 6-page limit; here we give more details. The modification of [MNT93] in Section 4 and everything that comes afterward is entirely new. I am grateful to all who have given me comments on those earlier papers, and in particular to Etienne Grandjean and Sam Buss for discussions of their recent work.

2 String Editing and Block Moves

The main idea of the Block Move model can be expressed as a one-person game in which the Player (P) edits a tape that stands for a long sequential file. Let the tape initially hold a string w over $\{0,1\}$ in cells 0 through n–1, where $n = |w|$, and let P have at her disposal a finite work alphabet Γ that includes $\{0,1\}$ and the *blank* B. The Player is given a goal string x, and seeks the least costly way, starting from w, to produce a tape whose first $|x|$ cells hold x. The cost of each editing move by P is calibrated by a function $\mu : \mathbf{N} \to \mathbf{N}$ that grades the communication time with memory. The intent is that low-numbered cells are like a fast-memory cache, while high locations figuratively reside on a large but slow disk drive. For instance, if P changes the single character in some cell e, the cost is $\mu(e)$ time units. The principal μ functions studied here, as earlier in the *Block Transfer* model of [ACS87] (which is integer-based rather than bit-based, and in other respects weaker than ours), are defined by $\mu_d(e) = \lceil e^{1/d} \rceil$, where $d \geq 1$ is figuratively the dimension of the memory.

The distinctive idea of the game is that if several edits of a similar kind can be done in one block $[a \dots b]$ of consecutive locations in the file, then P should profit from this *spatial locality* by being charged $\mu(a)$ or $\mu(b)$ only for the initial access, and unit time per edit thereafter. The notion of "similar kind" is that the edits can be done in one stream that is pipelined through a finite-state machine. Many commands in the UNIX$^{(R)}$ stream editor *sed* are representable this way. The particular model of finite-state machine we use is the *deterministic generalized sequential machine* (DGSM), as formalized in [HU79] or [HKL92]. The Player P has some finite set $\mathcal{S}$ of DGSMs available to her.

Rule 1. In any move, P may mark locations a, b, c, d, and select a DGSM S. Let z be the string held in locations $[a \dots b]$. Then $S(z)$ is written to locations $[c \dots d]$. The *μ-time* for the move is $|z| + \mu(a')$, where $a' = \max\{ a, b, c, d \}$.

We require that the intervals $[a \dots b]$ and $[c \dots d]$ be disjoint, and that the output $S(z)$ exactly fills the target block. One can have $a < b$ and/or $c < d$; in particular, substrings can be reversed on the tape. $S(z)$ overwrites any previous content of the target block $[c \dots, d]$, except for the following provision:

Rule 2. The blank B may be an output character of GSMs, and every B in the output stream $S(z)$ leaves the previous symbol in its target cell in $[a_2 \dots b_2]$ unchanged.

This rule is proved in [Reg94b] to have the same effect as making the write of $S(z)$ completely destructive, but adding an instruction that shuffles two equal-sized blocks into a third. Having Rule 2 enables us to describe all actions by P as a sequence of *block moves* from Rule 1, viz.:

$$\begin{array}{lcl} S_1[a_1 \dots b_1] & \textit{into} & [c_1 \dots d_1] \\ S_2[a_2 \dots b_2] & \textit{into} & [c_2 \dots d_2] \\ & \vdots & \\ S_R[a_R \dots b_R] & \textit{into} & [c_R \dots d_R], \end{array}$$

where $S_1, \dots, S_R$ belong to the fixed finite set $\mathcal{S}$ of DGSMs at P's disposal.

This editor does not have an insert/delete mode or allow the familiar form of "cut and paste" where the file is joined together after the cut. The realistic file-system model of Willard [Wil92] has similar restrictions. Note that changing a single character in a cell a' is subsumed by a block move with $c = d = a'$. The *μ-time* of the program is the sum of the μ-times of the block moves.

The above defines a natural *straight-line program* (SLP) model, similar in form to other non-uniform SLP models based on machines, on arithmetical formulas (see e.g. [BF90]), on Boolean circuits (see e.g. [Wig93]), or on bounded-width branching programs (see [BT88, Bar89, BIS90, BS91]). To build a uniform machine around the above, we need to specify control structures and appropriate "hardware." The main results of [Reg94b] show that under any of the μ_d cost functions, the model is *linear-time robust* under just about any choice borrowed from familiar machines: A form with just one (large) DGSM and one tape can simulate a form with any finite number k of tapes and k-input DGSMs, or a form with random-access addressing of the tapes, with only a constant-factor time slowdown. The above restrictions on overlap and size of the target block can be enforced or waived; it makes no difference. For definiteness, we use a form with four "fingers" and some finite number (8 is enough [Reg94b]) of "markers," initially placed on cells 0 and n–1, such that after every block move: (1) each marker on a cell i may be moved to cell $\lfloor i/2 \rfloor$, $2i$, $2i$+1 at a charge of $\mu(i)$—or left where it is at no charge, (2) the fingers "a,b,c,d" for the next move are (re-)assigned to markers, and (3) control branches according to the character now scanned by finger "a."

In the SLP form, of course, we don't have to worry about control or which a, b, c, d can legally follow the previous move, and we can have a separate program P for each input length n—or even for each individual w of length n. It is nice that the relevant robustness results carry over, particularly that a one-tape Player can simulate a k-tape Player in linear time. Next we establish lower bounds on certain "non-uniform" problems for the SLPs, with an eye toward using them as ingredients for lower bounds on the natural problems described in Section 1, for the uniform machines.

3 A Kolmogorov Lower Bound Argument

Given two strings w and x of the same length n, define their *edit distance* $E_\mu(w, x)$ to be the least t such that the Player can change w to x in μ-time t. Define $e_\mu(n) := \max\{ E_\mu(w, x) : |w| = |x| = n \}$. For all $d \geq 1$, let E_d and e_d stand for the above functions under μ_d.

The idea of the lower bounds is that the time for a block move under μ_d is asymptotically greater than the number of bits required to write the move down. The latter is bounded above by $C + 4\log_2(a')$, where

C is a constant that depends only on the size of the fixed $\mathcal{S}$, and a' is the maximum address involved in the move. Indeed, the lower bounds work for any finite set of operations, not just DGSMs, and ignore the time to read and write the addressed blocks after the initial $\mu_d(a')$ access charge The *matching* upper bounds require only that $\mathcal{S}$ contain the DGSM *copy* and the two DGSMs S_0 and S_1, which run for one step only and write a single 0 or 1. This we tacitly assume in stating:

Theorem 3.1 ([Reg94a]) *For any fixed $\mathcal{S}$, $e_1(n) = \Theta(n \log n)$, and for all $d > 1$, $e_d(n) = \Theta(n \operatorname{loglog} n)$.*

Proof. For the upper bounds, it suffices to bound $\mathrm{E}_d(0^n, x)$ for all x of length n. In the case $d = 1$, we may suppose $n = 2^k$. The procedure is:

1. Generate the right half of x in cells $0 \dots 2^{k-1} - 1$,
2. *copy* $[0 \dots 2^{k-1} - 1]$ *into* $[2^{k-1} \dots 2^k - 1]$,
3. Generate the left half of x in cells $0 \dots 2^{k-1} - 1$.

The basis is writing 0 or 1 to cell 0, and the μ_1-time taken is $O(n \log n)$. Note, moreover, that the moves are *oblivious*, insofar as every x of length n uses the same sequence of address 4-tuples (a_i, b_i, c_i, d_i).

For integral $d > 1$, the recursion works on the intervals from $n_{k-1} = 2^{d^{k-1}}$ to $n_k = 2^{d^k}$, chosen so that $n_k/n_{k-1} = n_{k-1}^{d-1}$. For each j, $1 \le j < n_k/n_{k-1}$, it recursively generates the required substring in the first n_{k-1} cells and executes *copy* $[0 \dots n_{k-1} - 1]$ *into* $[jn_{k-1} \dots (j+1)n_{k-1} - 1]$. The charges $\mu_(a)$ for these *copy* steps are bounded by $D n_k$, where D depends only on d. This gives the recursion $T(n_k) = (n_{k-1})^{d-1} T(n_{k-1}) + O(n_k)$, with solution $T(n) = O(n \operatorname{loglog} n)$. This much is similar to the upper bound methods for the"Touch Problem" in [ACS87]. For non-integral but rational $d > 1$, the necessary lemmas for computing interval endpoints efficiently may be found in section 4.2 of [Reg94b].

For the *lower bounds*, we give full detail for the case $d = 1$, and a start on the argument for $d > 1$. Let $g(n)$ be such that for all but finitely many n, $e_1(n) \le ng(n)$. We will show that $g(n)$ must be $\Omega(\log n)$. Let n be given, and let $k = \lceil \log_2 n \rceil$.

Now let x be any string such that the *conditional Kolmogorov complexity* $K(x|0^n)$ is at least n (see [LV93]). Let P be an SLP that consists of the sequence of moves used to generate x in μ_1-time $ng(n)$. Note that P itself is a description of x. We will convert P into a "modified" program P'' that generates x from 0^n, and is such that if $g(n) = o(\log n)$, then P'' has length $o(n)$, contradicting the choice of x.

For each i, $1 \le i \le k$, call the tape interval $[2^{i-1} \dots 2^i - 1]$ "region i." Cell 0 itself forms "region 0." The portion of the tape from cell 2^k onward is also usable by P; for the purpose of the proof, it is enough to consider it also part of region k. Say that a block move (or marker movement) is "charged in region i" if the memory-access charge $\mu(a')$ recorded for the move is for some cell a' in region i. Note that any move charged in region i is independent of any information in regions $i+1$ and above. To simplify some calculations, without affecting μ_1 time by more than a factor of 2, we suppose that the charge for region i is exactly 2^i, and regard n as equal to 2^k. Now we make an important observation that embodies the connection between μ_1 and the choice of regions.

Claim. P can be modified to an equivalent program P' such that for each step charged in some region i, the next step is charged in region $i-1$, i, or $i+1$, and the μ_1-time of P' is at most 3 times the μ_1 time of P.

The proof of this is straightforward: if P wants to jump from region i to region j, let P' make dummy moves in the regions in-between. Now we refer to P' and ignore the constant 3. For each i, $1 \le i \le k$, define $N(i)$ to be the number of steps in P' charged in region i. Then

$$\text{for some } i,\ 2^i N(i) \le ng(n)/k. \tag{1}$$

Choose the *greatest* such i. We have $N(i) \le 2^{k-i} g(n)/k$, and rewriting (1) another way, $N(i) \le ng(n)/2^i \log n$. Also since $N(i) \ge 1$, $i \le k - \log(\log(n)/g(n))$.

By the choice of i, we have that for each $j > i$, $N(j) > 2^{k-j} g(n)/k$. Hence at least $2^{k-i} g(n)/k$ moves are charged above region i. Call these $2^{k-i} g(n)/k$ moves "required" moves. The total μ_1-time charged for these required moves is at least $(k-i) 2^k g(n)/k$. Since $n = 2^k$ and the total μ_1 time is assumed to be $ng(n)$, the total μ_1-time available for all other moves is at most

$$B = ng(n)(1 - (k-i)/k) = ng(n)i/k.$$

By the adjacency condition imposed on P', all the moves charged in regions i and above fall into at most $N(i)$ segments of the program P', which we refer to as "high segments."

Now the following is a description of x: For each high segment l, $1 \le l \le N(i)$, give

(a) The contents w_l of cells $[0 \ldots 2^{i-1}-1]$ prior to the first move of the segment, and

(b) The instructions executed by P' in that segment.

Finally, after the last high segment, append the first 2^{i-1} bits of x. This finishes P''.

Per remarks before Theorem 3.1, each block move instruction charged in region j can be written down using Cj bits, where the constant C depends only on $\mathcal{S}$. Now we can place upper bounds on the length of the description:

- For intervals between high segments: at most $C_0 2^{i-1} \cdot (N(i)+1) < C_0 ng(n)/(\log n)$ bits, where C_0 depends on the work alphabet size.
- For the required high moves: at most $\sum_{j=i+1}^{k} 2^{k-j} Cjg(n)/(\log n)$ bits, which is bounded above by $(g(n)/\log n) \cdot C \cdot (\log_2 e)^2 i 2^{k-i}$.
- For moves charged in region i, at most $N(i)Ci \leq (g(n)/\log n) \cdot Ci 2^{k-i}$.
- For the "other" moves charged in regions $j > i$, let $N'(j) = N(j) - 2^{k-j} g(n)/\log n$. For the bound we need to maximize

  $$\sum_{j=i+1}^{k} CjN'(j)$$

  subject to

  $$\sum_{j=i+1}^{k} 2^j N'(j) \leq B.$$

  Elementary calculation gives an upper bound of $CiB/2^{i+1} = (g(n)/\log n) \cdot 2^{k-i-1} Ci^2$.

(Intuitively, the maximum is achieved when $N'(i+1)$ is maximum and $N(j) = 0$ for all $j > i+1$, so as to minimize the disparity between the description-length of Cj and the μ_1 time charge of 2^j for each move.) Putting this all together gives an upper bound on description length of

$$\frac{g(n)}{\log n}[C_0 n + C_1 i 2^{k-i} + C_2 i^2 2^{k-i}]$$

for appropriate constants C_0, C_1, and C_2. Finally, we calculate that the second term inside the $[\ldots]$ is bounded above by $C_1(n \log_2 e)/e$, and the third by $4C_2 n[(\log_2 e)/e]^2$. Thus

$$\text{description length} = \frac{g(n)}{\log n}\Theta(n),$$

and we conclude that $g(n)$ must be $\Omega(\log n)$.

For $d > 1$, we use "regions" of the form $[n_{k-1} \ldots n_k)$, where $n_k = 2^{d^k}$, as in the upper bound. As before, the lead point in the proof is that one of these loglog n-many regions receives no more than the "average" $ng(n)/\operatorname{loglog} n$ share of the total μ-time. The rest follows a similar pattern to the above. $\square$

Proposition 3.2 *There are permutations of* $(1, \ldots, n)$ *(written in binary) that cannot be realized in linear* μ_d*-time, for any* d*.*

Proof. Since there are $2^{\Theta(n \log n)}$ permutations of $(1, \ldots, n)$, some π have conditional Kolmogorov complexity $K(\pi|n) = \Theta(N)$, putting $N = n \log n$. The above proof, with the initial "0^n" replaced by the input list $(1, \ldots, n)$, shows that every SLP computing π requires time $\Omega(N \log N)$ under μ_1 and $\Omega(N \operatorname{loglog} N)$ under μ_d for $d > 1$. (Matching upper bounds follow as before.) Since N serves as the input length of the encoding, the result follows. $\square$

Compared to the corresponding result in [ACS87] (see also [AV88]), we have obtained the lower bound in a memory-hierarchy model that allows (arbitrary) bit operations.

The above bounds, however, hold merely because of individual strings (of high Kolmogorov complexity). For the problems mentioned in Section 1, we are really interested in larger-scale properties of the *mapping* of all input strings $w \in \{0,1\}$ to their outputs. In the next section, we put the Block Move model completely aside to review and improve some lemmas of [MNT93] about mappings with a certain "property of randomness," which also provide a place to insert the technique of Theorem 3.1 for larger ends.

4 Functional BPs and Time-Space Tradeoffs

As described in [MNT93], a *functional branching program* (here, just "BP") with domain in $\{0,1\}^n$ and range in $\{0,1\}^m$ is a one-source DAG B with labels on its nodes and some of its edges. Each non-sink node is labeled by an integer i, $0 \leq i \leq n-1$, and has two outarcs, one for "bit $x_i = 0$" and one for "bit $x_i = 1$." Each sink is labeled "accept" or "reject." Some edges of P are labeled by pairs (j, b), where $0 \leq j \leq m-1$ and $b \in \{0,1\}$. Traversing such an edge is intended to signify that the jth bit of $f(x)$ equals b.

It follows that every $x \in \{0,1\}^n$ determines a unique path from the source of P to a sink. The path is valid, according to [MNT93], provided that there are exactly m edge labels along the path, one for each j, and that the last node of the path is labeled "accept." Then the corresponding bits b form the output value $P(x)$. The BP computes a (possibly partial) function $f : \{0,1\}^n \to \{0,1\}^m$ if for every $x \in dom(f)$, the path for x is valid and outputs $f(x)$. We can relax the definition in [MNT93] somewhat by allowing more than m labels along the path, so long as every j appears at least once, taking the output bit j to be the last "mind change" made by P along the path.

As usual, when speaking of computing a function f defined on all of $\{0,1\}^*$, we use a family $[P_n]$ of BPs, one for each input length. The techniques in this section do not require any uniformity properties of the families under consideration. The BP *time complexity* $T(n)$ is the longest length of a valid path in P_n, and the *capacity* $C(n)$ is the number of nodes in P_n. It is customary to write $S(n) = \log C(n)$ and call this the *space* used by P_n; these notions of capacity and space go back to Cobham [Cob64]. The point of (functional) BPs is that all of the machines surveyed in Section 1, together with their associated time measures $t(n)$ and space measures $s(n)$ (see [vEB90]), can be translated into BPs with $T(n) = O(t(n))$ and $S(n) = O(s(n))$. Thus the BPs have been termed a "general sequential model of computation" [BC82, Bea91]. Read-only input tapes are *not* counted against the space bound of the machines; they are handled directly be the node labels of the BPs.

Now we consider a nondeterministic extension of this model. For convenience, we define it in a normal form that is analogous to requiring an NTM to make all of its nondeterministic guesses first and then behave deterministically.

Definition 4.1. A *nondeterministic branching program* (NBP) is a disjoint union of some number I of *pieces*, which are ordinary BPs.

Here $\log(I)$ corresponds to the number of bits of nondeterminism used by a nondeterministic machine. Without the normal form, where one has nondeterministic internal nodes (see [Mei89, Mei90a, Mei90b]), one has to be careful to tally this quantity separately from the space.

Of itself, an NBP P computes a *partial multivalued function* F on $\{0,1\}^n$. Then following Selman (see [Sel94]), say that an ordinary (partial) function f is a *single-valued refinement* of F if $dom(f) = dom(F)$, and for all $x \in dom(f)$, $f(x)$ is one of the values of $F(x)$. Below we restrict attention to total functions f on $\{0,1\}^n$, but the counting arguments can be modified so long as the domain is sufficiently large.

In adapting notions and results of Mansour, Nisan, and Tiwari [MNT93], we use their notation, with "l" in place of "n," but we will have $\ell, m = \Theta(n)$ anyway for the functions we are interested in. Define a set B of strings of length m to be a *k-cylinder* if there are k indices $1 \le i_1 \le i_2 \le \ldots \le i_k \le m$ and a string u of length k such that $B = \{x \in \{0,1\}^m : (\forall j, 1 \le j \le k)\, x_{i_j} = u_j\}$.

Definition 4.2 ([MNT93]).
A function $f : \{0,1\}^l \to \{0,1\}^m$ has the *property of randomness with parameters* (n, α, β) if for every n-cylinder $A \subseteq \{0,1\}^l$ and k-cylinder $B \subseteq \{0,1\}^m$, $\Pr_{x\in A}[f(x) \in B] \le 2^\beta / 2^{\alpha k}$.

Lemma 4.1 ([MNT93]) *If f has the property of randomness with parameters (n, α, β), then for every ordinary branching program P of depth at most n, and every $k \le n$, the proportion of inputs for which P outputs k correct bits of $f(x)$ is at most $2^\beta/2^{\alpha k}$.*

Now we carry over what comes next in [MNT93] to the case of NBPs.

Lemma 4.2 *Let $f : \{0,1\}^l \to \{0,1\}^m$ be a function that has the property of randomness with parameters (n, α, β). If f is a single-valued refinement of an NBP P, then P must have $(S + \log I + \beta)(T + n) \ge \alpha mn$, where S is a bound on the space of each individual piece.*

Proof. We basically mimic [MNT93]. Since $T \ge n$ and $S \ge \log n$, we can make each individual piece of P into a DAG that is *leveled*, meaning that all paths through P have the same length. This is done by adding "dummy nodes," at an additive cost of $O(\log n)$ to S. Then each piece is broken into $\lceil T/n \rceil$ *blocks*, where each block consists of n consecutive levels. For every x, there is some $i \le I$ and some block P_{ix} in the ith piece that outputs at least $k := m/\lceil T/n \rceil$ correct bits of $f(x)$. Note that $k \ge mn/(T + n)$. Let c_{ix} stand for the source node in P_{ix} that is traversed by x in the ith piece, together with all of its descendents in P_{ix}; call this a "cone." Since each piece has at most 2^S nodes, there are at most $I2^S$ different cones. For each cone c, let

$$A(c) = \{x : c(x) \text{ produces} \ge k \text{ correct outputs}\}.$$

By Lemma 4.1, $|A(c)| \le 2^{-\alpha k+\beta} 2^l$. Since $\{0,1\}^l \subseteq \cup_{c_{ix}} A(c_{ix})$, we have

$$2^l \le \sum_{c_{ix}} 2^{-\alpha k+\beta+l} \le 2^S I 2^{-\alpha k+\beta+l},$$

and so $2^{S+\log I-\alpha k+\beta} \geq 1$. Taking logs, $S+\log I+\beta \geq \alpha k \geq \alpha mn/(T+n)$, from which the conclusion follows. $\square$

It is interesting that this counting argument works even though other pieces are allowed to output spurious values—Lemma 4.1 is needed only for a particular "cone," which is deterministic. If P has enough pieces to cover all possible values, then $\log I$ is already as high as advertised!

Corollary 4.3 *Let $\{ f_l : \{0,1\}^l \rightarrow \{0,1\}^m \}$ be a family of functions that have the property of randomness with parameters (n, α, β), where $l, m = \Theta(n)$ and α, β are constants independent of n. Then every family of NBPs P_l that have f_l as single-valued refinements must have $(S+\log I)T = \Omega(n^2)$.*

For examples of functions shown by [MNT93] to have the property of randomness with suitable parameters, let H be a family of hash functions $h : \{0,1\}^n \rightarrow \{0,1\}^{m(n)}$ that is *universal*$_2$ ([CW79]), meaning that the random variables $r_x = h(x)$-over-$h \in H$ are pairwise independent. Further suppose that there exists $c > 0$ such that for all $h \in H$ and all n, the cardinality of $h(\{0,1\}^n)$ is at least $2^m/c$. Now define f_H by $f_H(h,x) = h(x)$. Then, as proved in [MNT93], f_H has the property of randomness with constant α and β. Via known ways of computing universal$_2$ hash functions, they show that computing $ab+c$ in a finite field and computing the *convolution* of two strings have the $\Omega(n^2)$ tradeoff. Beame [Bea91] proved $ST = \Omega(n^2)$ tradeoffs on essentially the same model (with alphabets of variable size R) for sorting and for finding all uniquely-occurring elements on a list, and [MNT93] observe that these have the same property of randomness.

Corollary 4.4 *Let N be a nondeterministic Turing machine (or TC) that, on every input list $(a_1, \ldots, a_m)$ of binary numbers $a_i = m^{O(1)}$, has at least one nondeterministic branch that outputs the correctly sorted list. Then either the number of bits of nondeterminism or the space used by N must be $\Omega(n)$, where $n = m \log m$.*

The same goes for N that find all the uniquely-occurring elements in the list along some branch, or that have $ab+c$ in GF(2^n) as a single-valued refinement.

What this *says* is that for computing the above functions, adding nondeterminism will not save you any space, unless you use a *lot* of nondeterminism. It is tempting to try to find counterexamples to this last result, even just to knock the space and nondeterminism down to $O(m)$, but it will not yield them. What it does yield, however, is a funny strategy for possibly using nondeterminism to prove super-linear lower bounds for *functions* f against deterministic machines M:

(1) Show that if M runs in linear time, then M can be simulated by BPs with sub-quadratic ST product.

(2) Nondeterminism may help you get $(S+\log I)T = o(n^2)$ instead!

(3) Then show that f has the "property of randomness" with appropriate parameters, or has some other suitable "mixing" property.

The main inherent limitation of this scheme is that it can only ever achieve "modest" lower bounds. This is because every function f on $\{0,1\}^n$ already has a "simple direct" BP that computes it with $ST = n^2$: make a binary tree of depth n that branches for each bit of x, so that each x goes to a unique leaf, and then burp out all the bits of $f(x)$ below that leaf. Thus already the above $O(n \log n)$-time functions are "maximally complex" for this technique. However, a modest lower bound is better than none. We attempt to apply this strategy to the Block Move model, and then speculate on its use for Turing machines.

5 Block Move and BPs

Let f be a function with a time-space tradeoff $S(n)T(n) = \Omega(n^2)$ for NBPs. We may suppose that $|f(x)|$ is linear in $|x|$. We sketch the lower bound idea for $d = 1$. Suppose M computes f in μ_1-time $ng(n)$. The proof strategy is to build a small NBP for f, so as to show that $g(n)$ must be $\Omega(\log n)$.

Following the proof of Theorem 3.1, let $k = \lceil \log n \rceil$. Divide the tape into "regions" $[2^{i-1} \ldots 2^i - 1]$; as before, we may suppose that successive moves by M visit adjacent regions. Fix an input $x \in \{0,1\}^n$ for the moment. Then there is an SLP P_x that describes the actions of M on input x. For each i, $1 \leq i \leq k$, let $N_x(i)$ be the number of times M on input x is charged in region i. Fix the greatest i such that $N_x(i) \leq 2^{k-i} g(n)/\log n$. By the same calculations as before, P_x can be changed to a "modified SLP" P''_x that consists of moves of the following kinds:

(a) *Arbitrary$_i$*: change cells $[0 \ldots 2^{i-1}-1]$ of the tape only.

(b) S $[a \dots b]$ *into* $[c \dots d]$, where S is a DGSM, and $[c \dots d]$ contains cells in region i or above. Call these "high moves."

We claim that, analogous to the way the "contents of cells $[0 \dots 2^{i-1} - 1]$ just prior to a high move" were written verbatim into the description in the proof of Theorem 3.1, we can build the mapping in each $Arbitrary_i$ step by the "simple direct" tree-branching idea at the end of the last section. But the BP components for the high moves will still have to be filled in.

The main point is that, by the calculations in Theorem 3.1, every such SLP P''_x has bit-length $(g(n)/\log n) \cdot \Theta(n)$. Unless $g(n) = \Omega(\log n)$, the total number $I(n)$ of these programs, over all x of length n, is much less than 2^n. If we allocate one NBP "piece" for each one, then we will have:

- For each x, at least one piece correctly computes $f(x)$.
- $\log I(n) = (g(n)/\log n) \cdot \Theta(n) << n$, so that $\log I(n)$ effectively drops out of the picture, and we need only bound the space $S(n)$ of each individual piece.

Hence, it remains to bound the capacity/space needed to code the high moves by BPs. By analogy with a high move charged in region j having description complexity $O(j)$, it would be nice to bound the additional space for the move by $O(j)$. Letting $N = 2^j$ figure as the input length for that move, we're talking about achieving $ST = O(N \log N)$ for a DGSM trandsuction on input length N, *and preserving a comparable bound when we sequence these transductions.*

Although the high moves are intermixed with the "low" moves, the above treatment of the $Arbitrary_i$ moves "factors through" in a way that allows us to pretend that all the high moves form one sequence. This brings us to the following

Problem. Find the best general bound on ST for BPs that simulate R block moves with total μ_d-time t on a tape of n cells, where each block move has a minimum charge of m, and the DGSMs read w bits in total.

In our context with $d = 1$, the high moves we need to simulate have minimum charge 2^i. This problem is interesting by itself without the "where" clause, but this last clause tries to take advantage of what we have already done in the above proof strategy.

Now we can simulate a single block move $S[a \dots b]$ *into* $[c \dots d]$ by a BP in the form of a $|b-a| \times |d-c|$ grid. Each entry in the grid has q nodes, one for each state of S. The main invariant is that each node in entry (i, j) signifies that S has so far read i symbols, written j symbols, and is in the corresponding state. For each input character c, the edge from this node on c goes to the node in row $i+1$ that maintains this invariant, and has the edge labels for whatever S outputs in this step. If S in that step outputs a (single) B, then we may insert an extra node that is labeled "x_{c+j}" (or "x_{c-j}" if $d < c$) in order to preserve the character that previously occupied the corresponding cell. S for this grid is roughly $\log|b-a| + \log|d-c|$, which is fine and dandy so far.

However, things already become thorny when we compose *two* block moves. The actual inputs to the second move may come from outputs of the first one, and we have to "cascade" through to the inputs of the first move. Two factors that make doing this difficult are (1) the shuffling effect of blank outputs, and (2) the fact that the time step in which a DGSM writes its jth output may depend heavily on the input. We can both simplify our task and crystallize these difficulties by exploiting another "robustness result" of the Block Move model. First, two definitions that are more-or-less standard:

Definition 5.1. Given integral $d > 0$ and $e \geq 0$, a *generalized homomorphism* (gh) with *ratio* $d : e$ is a finite function from Σ^d to Σ^e that is extended homomorphically to all strings in Σ^* whose length is a multiple of d. For strings of other lengths, we allow any fixed "patching" at the end—this makes no difference.

For $c \in \Sigma$, the *erasing homomorphism* (eh) Er_c erases all occurrences of c in its argument.

Lemma 5.1 *Every BM M can be simulated, with constant-factor overhead under any given μ_d, by a BM M' each of whose DGSMs is a fixed-ratio gh or an eh. This holds also for the SLPs.*

Proof Sketch. The idea is first to pad a DGSM S used by M by a new character c so that every step by S outputs the same number of characters. Then the method of "parallel prefix sum" from [Ofm63, LF80] can be applied to compute $S(z)$. Since each "sweep" in this method does local computation only, it can be simulated efficiently by fixed-ratio gh's. A final erasing homomorphism Er_c removes the padding. The technical details needed to implement this idea may be found in [Reg94b]. $\square$

Now, we can also require $a \leq b$ and $c \leq d$ in every block move if we give M' an instruction *reverse*

$[a \ldots b]$, and we can remove the convention on B if we give M' an instruction *shuffle* $[a \ldots b]$, $[c \ldots d]$ *into* $[e \ldots f]$, subject to the first two intervals having equal length that is half that of the third.

The BP for a gh does not even need the grid of the above example, because the jth output always comes when the dj/e-th input is read. However, an eh still poses the problem that the timing of the outputs is "unpredictable." Note that in the context of our proof strategy, we may suppose that if the ith block move in P''_x is an erasing one Er_c $[a_i \ldots b_i]$ *into* $[c_i \ldots d_i]$, then x and d are such that the output of this step exactly fills $[c_i \ldots d_i]$. However, even this convenience does not tell us "when" outputs *midway* through the target block are produced. So the square grid approach is the best we know, even here. But it and the "shuffle" feature combine to cause headaches when we try to work backward from outputs to inputs in a composition of block moves. Even with these simplifications, we have not been able to obtain bounds on ST for this part that are subquadratic, let alone $O(N \log N)$...

...And, here is a clear indication that $N \log N$ is sheer wishful thinking, that something like $ST \leq N^2/\log N$ is the best we can hope to eke out even here. A BM, even under μ_1, can simulate a time-$t(n)$ Turing machine T in time $t(n) \log t(n)$, by carrying out the *Hennie-Stearns construction* [HS66] (see also [HU79, WW86]) applied to T. Hence a BM can sort in μ_1-time $O(n \log^2 n)$. If there were steep time-space savings to be had in translating a sequence of block moves to functional BPs, then this would contradict the very tradeoff proved for sorting!

The knotty **Problem** we are left with is not an isolated problem. Rather, it is next-in-line to a sequence of problems relating to branching programs that have been recently posed and solved. These concern *input-oblivious* BPs, which are leveled BPs such that every node at the same level is labeled by the same input bit. Alon and Maass [AM88] proved exponential lower bounds on the size of input-oblivious BPs that recognize the "sequence-equality" language $\{\, x, y \in \{\, 0, 1, 2 \,\}^* : Er_2(x) = Er_2(y) \,\}$. This language is decidable by a short sequence of block moves, including one eh move and one shuffle. These results were extended in [KMW89, KW91, HKMW92] for related problems.

The "grid" BP above for an individual DGSM computation is input-oblivious *provided* that either (1) S follows some $d : e$ "steady pace" (see the general notion of Manacher [Man82]; here S need not be a gh) or (2) the convention on B is not used. The simplifications to gh and eh, with-or-without the exchange for *shuffle* and *reverse* instructions, push the problem of maintaining "some degree of" input-obliviousness into the compositions. On the basis of the last-cited results and the earlier work in this section, we have:

Theorem 5.2 *A BM in which every DGSM is a gh (or is steady-paced) requires time $\Theta(n \log n)$ to recognize the sequence-equality language, and time $\Omega(n \log n)$ to sort, find unique elements, or compute $ax + b$ in finite fields (etc.).*

There is some interest in the BM with only gh and *shuffle*, since $O(\log^k n)$ block moves and polynomial unit-cost runtime gives an exact characterization of (DLOGTIME-uniform) NC^k, for each $k \geq 1$ [Reg94c]. But we really want something much better than this partial result.

If known strategies for making time-t Turing machines "block-respecting" (see [HPV75, PPST83] and note the implicit nondeterminism in the proof-size bound of [Bus94]) can be improved, they may yield (N)BPs with $TS = O(t^2/\log t)$. This would suffice to answer the conjecture at the end of [MNT93] that the sorting and hashing-related functions require super-linear (indeed, $\Omega(n \log n)$) time on TMs.

6 For Better Lower Bounds—?

Despite the impasse reached in Section 5, the ideas behind the machinery in Sections 3 and 4 are worthy of notice, also since they apply more generally than to the Block Move model. Let us now try to see the basic idea in a larger context: The whole dilemma with lower bounds is that one has to reason about what a *nonexistent* algorithm *cannot* do. One is given just blank tape to work with, a *tabula rasa* for a "black box." We want to analyze structures in how information must flow in order to solve a given problem in a given short amount of time, but we seem to have no structure to work with.

The argument in Section 3 begins with just a grain of structure: at least one region has to fall at-or-below the average for time use. The μ_d cost function determines the division into regions and introduces this grain, around which the rest of the argument crystallizes. This is like finding (somewhat non-constructively) a "pinch point" in the "black box." In this regard, we have something very like classical "crossing sequence" lower-bound arguments—these and related arguments using Kolmogorov complexity are described extensively in [LV93].

What one would really like is a larger-scale methodology that finds multiple major bottlenecks in information flow to solve a problem, rather than one "pinch point," and that doesn't rely so heavily on the particular one-dimensional geometry of tapes (or of pipelining in general). One would like the methodology to begin with, and apply to, *uniform* models such as machines—real obstacles to non-uniform circuit lower bounds have recent been raised by Razborov and Rudich [RR94], and striking contrasts to uniform lower bounds emerge from [AG92], and also [RSC95]. In search of a start, we offer the following formalization and generalization of ideas in [FS92] and references therein:

Take the notion of a machine *configuration* or *ID* as fundamental. The one general assumption we need is that we can break an ID into two components: one for the "status" of the machine and its hardware, and the other for the actual bit-values of stored data. With a Turing machine, the former includes the current state of the finite control and the current positions of its tape heads, while the latter stands for the actual contents of the tapes. The one choice we have to make, which is reflected below, is whether a read-only input tape is counted as part of the "machine status" or part of the "data." For a RAM, one can postulate that the current instruction and the current value of the main (ALU) register are status information, and the remaining registers hold data. For a PRAM, the "status" includes the current state and local memory of each processor, while the "data" is the current contents of the shared memory.

This notion can also accomodate models in which data is not stored at referenced locations but is mobile, such as the cellular machine of Feldman and Shapiro [FS92] in which data bits propagate through free space; this indeed is the setting of their informal definition of "vicinity." To make the definition formal, we just have to be able to talk about "data points" as distinct from the values, 0 or 1, that each data point currently has. We need to be able to single out subsets S of the data points in an ID I, and talk about an ID J that results from changing the actual bit-values of some data points in S, or in the complement $\widetilde{S}$ of S. For RAMs, we specify some way of talking about individual bits of registers. Most interesting in the present context, we can accomodate *oracle machines* by identifying the answer to an oracle query w as the bit stored in location w. Thus the IDs of an OTM include the oracle; in general, the IDs we talk about are infinite objects.

Definition 6.1. Given IDs I and J, write $I \approx J$ if I and J have the same "machine status" (i.e., same head positions, same current instruction, and possibly the same read-only input). Given a set S of data points, write $I \approx_S J$ if they have the same machine status and agree on the values of data points in S. Finally, write $I \sim_S J$ if they agree on S, but may have different head positions and so on.

Now for an ID I of a deterministic machine M, and an integer $r \geq 0$, write $I(r)$ for the unique ID J such that $I \vdash_M^r J$. (If M halts in those steps, let J be the halting ID; if M "crashes," take $J = \perp$.) If M is nondeterministic, then write $I(r)$ for the set of IDs than can occur after r steps, and write $I(r) \approx_S J(r)$ to mean that there is a 1-1 correspondence between IDs $I' \in I(r)$ and $J' \in J(r)$ such that $I' \approx_S J'$. Now we can define the main notion.

Definition 6.2 (compare [FS92]). The *vicinity function* v_H of an ID H is defined for all $r \geq 1$ to be the minimum size k of a set S of data points such that for all IDs $I, J \approx H$ (meaning: I and J have the same state and head positions as H):

$$I \approx_S J \Longrightarrow I(r) \approx_S J(r) \;\wedge\; I \sim_{\widetilde{S}} I(r).$$

This says that the evolution of the computation of M over the next r steps is completely determined by the state and head positions and by the data values in the set S, and that no data outside S can be affected in these r steps.

Definition 6.3. (a) The *vicinity function* V_M of a machine M is defined by $V_M(r) =$ the maximum of $v_H(r)$ over all IDs H of M.

(b) The vicinity function $V_{\mathcal{M}}$ of a machine *model* or a class of machines $\mathcal{M}$ is defined by similarly taking the maximum over all $M \in \mathcal{M}$.

(c) The *relative vicinity* is defined by $v'_H(r) = v_H(r)/v_H(1)$, and so on for V'_M and $V'_{\mathcal{M}}$.

Relative vicinity is the sensible notion for a PRAM and for any other machine whose hardware scales upward with input length. It is also the notion to use when we talk about machine models with variable hardware, such as "any finite number of tapes" for TMs. For example,

- The vicinity function of a k-tape Turing machine M is $V_M(r) = (2r-1)k$.
- Same for a TM with k heads on one tape, etc.
- A TM with d-dimensional tapes has vicinity $O(r^d)$.

- A Block Move machine with cost parameter μ_d also has vicinity $O(r^d)$.

- The relative vicinity of the d-TM model, and of the Block Move model under μ_d, is $O(r^d)$, regadless of the number of tapes and heads.

- The tree computer has vicinity $\Theta(2^r)$. So do the log-cost RAM and the related machines in the "cluster" referred to in Section 1.

- The unit-cost RAM has undefined (or "infinite") vicinity.

- A "strongly polynomially compact RAM" M of Grandjean and Robson [GR91], where M runs in time $t(n)$, is defined so that the value $V_M(t(n))$ of its vicinity function is polynomial in $t(n)$. However, $V_M(r)$ may be (and generally is) exponential for smaller values of r. This exponential "local" vicinity is used for M to simulate efficiently the other models in the aforementioned cluster.

The fact that a time-$t(n)$ TC can be simulated by an ordinary TM in time $t(n)^2$, and a unit-cost RAM in time $t^3(n)$, means that exponential vicinity gives no more than a polynomial speedup on memory *that has been initialized to zero*. Oracle machines, however, use memory that has been initialized to something else. We can identify an oracle A with an extra tree tape such that cell w on the tree tape initially holds $A(w)$.

Now, the heart of the matter is that for every oracle set A that is known to make $\mathrm{NP}^A = \mathrm{P}^A$, the P^A-machines M accepting SAT^A all have exponential vicinity, viz. $2^{\Theta(r)}$. Indeed, oracle machines with *polynomial* vicinity turn out to be equivalent to notions already known in structural complexity, depending on whether the input x is considered a fixed part of the machine or part of the data, and on whether a given oracle is fixed. In the next result, "oracle machine" can mean an OTM or oracle TC or oracle RAM, although the intent of the result is more general than that. The classes P.ALL(D) and NP.ALL(D) were defined and studied by Book, Long, and Selman [BLS85] (see also [SMRB83, BLS85]), who showed "positive relativizations" for them.

Theorem 6.1 *Let "M" stand for a polynomial-time deterministic oracle machine, and "N" for a nondeterministic one. Let L be a language.*

(a) There exists an M of polynomial vicinity, that accepts L with some oracle, iff $L \in \mathrm{P}/poly$ (i.e., iff L has polynomial-sized circuits).

(b) There exists an M whose vicinity is polynomial when the input is regarded as machine status, that accepts L with some oracle D, iff L belongs to P.ALL(D).

(c) There exists an N of polynomial vicinity, that accepts L with some oracle, iff $L \in \mathrm{NP}/poly$.

(d) There exists an N of polynomial vicinity excluding input from data, that accepts L with some oracle D, iff L is in NP.ALL(D).

Proof. (a) For the forward direction, let $t(n)$ be a polynomial bound on the running time of M on inputs of length n. Let D be some oracle with which M accepts L. Let H be the initial ID of M^D on any such input. Then there exists a set S of size polynomial in $t(n)$, hence polynomial in n, that satisfies the condition in Definition 6.2. It doesn't matter whether S includes the input tape or not; what matters is that the premise $I \approx H$ in that condition does not restrict I to have the same input as H. Now fix all the bits in S that belong to D. This becomes a polynomial-size advice string for inputs of length n, yielding the characterization of P/*poly* from [KL80].

For the converse direction, let S comprise the (locations of the) "advice bits." We may either keep the advice on a tape, or treat the advice as an oracle that can be "random-accessed." In the latter case, however, in order to bound the vicinity function by a polynomial for all $r \leq t(n)$, we have to "slow the OTM down" by making it wait n steps between queries (or, by adopting the convention that an oracle query string is erased when it is submitted; see [WW86, BDG88]). For polynomial bounds, this slowdown does not matter.

(b) Here, the condition in Definition 6.2 applies only to IDs I that have the same input as the initial ID H. Hence one can have different sets S for different inputs x. This is what yields the same situation as in [BLS84]. Parts (c) and (d) are similar. □

Book, Long, and Selman [BLS84] proved the following "positive relativization": P = NP iff for all oracle sets D, P.ALL(D) = NP.ALL(D). From this it follows that if some oracle A making $\mathrm{P}^A = \mathrm{NP}^A$ does so with a polynomial-vicinity algorithm for SAT^A, then P = NP. The question of whether one can arrange $\mathrm{P}^A = \mathrm{NP}^A$ with *sub-exponential* vicinity is an interesting one, perhaps related to questions about the *power index* of (unrelativized) SAT raised by Stearns and Hunt [SH90].

However, we really have in mind going in the other direction, toward saying what we can do in *linear* vicinity. The NLIN $\neq$ DLIN separation of [PPST83] is not known to carry over to any model with superlinear vicinity. Here, the difference between bounding $V_M(t(n))$ and bounding $V_M(r)$ for all $r \leq t(n)$ should also matter, as reflected in the suspected difference between the linear time classes of a "strongly n^d-compact RAM" and a TM with d-dimensional tapes.

The first item to work on is whether we can find a general way to map machines of, say, quadratic vicinity into specific planar networks, of a kind that may bring "geometric" lower-bound arguments into play. The hope is that the simple "one-bottleneck" idea in Section 3 can be extended into a more-powerful kind of argument. One reason the extension via Section 4 turned out to have only "modest" power in Section 5 is that the "property of randomness" defined in [MNT93] depends only on the input/output behavior of the function in question, and combined with the branching-program model, which is non-uniform and has no internal data structure apart from the I/O labels at all, this "washes out" most of the dataflow structure we want to analyze. We conclude with the hope that this survey gives a better picture of the shape of the problem of lower bounds, and a direction for the challenge of proving them.

References

[AACS87] A. Aggarwal, B. Alpern, A. Chandra, and M. Snir. A model for hierarchical memory. In *Proc. 19th STOC*, pages 305–314, 1987.

[AC88] A. Aggarwal and A. Chandra. Virtual memory algorithms. In *Proc. 20th STOC*, pages 173–185, 1988.

[ACF90] B. Alpern, L. Carter, and E. Feig. Uniform memory hierarchies. In *Proc. 31st FOCS*, pages 600–608, 1990.

[ACS87] A. Aggarwal, A. Chandra, and M. Snir. Hierarchical memory with block transfer. In *Proc. 28th FOCS*, pages 204–216, 1987.

[ACS90] A. Aggarwal, A. Chandra, and M. Snir. Communication complexity of PRAMs. *Theor. Comp. Sci.*, 71:3–28, 1990.

[AG92] E. Allender and V. Gore. A uniform circuit lower bound for the permanent. Technical Report Tech. Report 92-30, DIMACS, 1992.

[AI84] A. Adachi and S. Iwata. Some combinatorial game problems require $\Omega(n^k)$ time. *J. ACM*, 31:361–376, 1984.

[AM88] N. Alon and W. Maass. Meanders and their application to lower bound arguments. *J. Comp. Sys. Sci.*, 37:118–129, 1988.

[AV88] A. Aggarwal and J. Vitter. The input-output complexity of sorting and related problems. *Comm. ACM*, 31:1116–1127, 1988.

[Bar89] D. Mix Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1. *J. Comp. Sys. Sci.*, 38:150–164, 1989.

[BC82] A. Borodin and S. Cook. A time-space tradeoff for sorting on a general sequential model of computation. *SIAM J. Comput.*, 11:287–297, 1982.

[BDG88] J. Balcázar, J. Díaz, and J. Gabarró. *Structural Complexity Theory.* Springer Verlag, 1988.

[Bea91] P. Beame. A general sequential time-space tradeoff for finding unique elements. *SIAM J. Comput.*, 20:270–277, 1991.

[BF90] L. Babai and L. Fortnow. A characterization of #p by arithmetic straight-line programs. In *Proc. 31st FOCS*, pages 26–34, 1990.

[BG70] R. Book and S. Greibach. Quasi-realtime languages. *Math. Sys. Thy.*, 4:97–111, 1970.

[BG93] J. Buss and J. Goldsmith. Nondeterminism within P. *SIAM J. Comput.*, 22:560–572, 1993.

[BIS90] D. Mix Barrington, N. Immerman, and H. Straubing. On uniformity within NC^1. *J. Comp. Sys. Sci.*, 41:274–306, 1990.

[BLS84] R. Book, T. Long, and A. Selman. Quantitative relativizations of complexity classes. *SIAM J. Comput.*, 13:461–487, 1984.

[BLS85] R. Book, T. Long, and A. Selman. Qualitative relativizations of complexity classes. *J. Comp. Sys. Sci.*, 30:395–413, 1985.

[BS91] D. Mix Barrington and H. Straubing. Superlinear lower bounds for bounded-width branching programs. In *Proc. 6th Structures*, pages 305–313, 1991.

[BT88] D. Mix Barrington and D. Thérien. Finite monoids and the fine structure of NC^1. *J. ACM*, 35:941–952, 1988.

[Bus94] S. Buss. On Gödel's theorems on lengths of proofs II: Lower bounds for recognizing k symbol provability, 1994.

[CM85] B. Chazelle and L. Monier. A model of computation for VLSI with related complexity results. *J. ACM*, 32:573–588, 1985.

[Cob64] A. Cobham. The intrinsic computational difficulty of functions. In *Proc. 1964 Congress for Logic, Mathematics, and Philosophy of Science*, pages 24–30, 1964.

[Coo71] S. Cook. The complexity of theorem-proving procedures. In *Proc. 3rd STOC*, pages 151–158, 1971.

[CR73] S. Cook and R. Reckhow. Time bounded random access machines. *J. Comp. Sys. Sci.*, 7:354–375, 1973.

[CW79] J. Carter and M. Wegman. Universal classes of hash functions. *J. Comp. Sys. Sci.*, 18:143–154, 1979.

[CW90] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetical progressions. *J. Symbolic Computation*, 9:251–280, 1990.

[Dew81] A.K. Dewdney. Fast Turing reductions between problems in NP. Technical report, Department of Computer Science, University of Western Ontario, London, Ontario, Canada N6A 5B9, 1981. Reports #68–#76, subtitled as chapters 1–9, the last four in 1982–1984.

[Dew82] A.K. Dewdney. Linear time transformations between combinatorial problems. *Intern. J. Comp. Math.*, 11:91–110, 1982.

[Dew89] A.K. Dewdney. Fast Turing reductions of combinatorial problems and their algorithms. In G. Bloom, R. Graham, and J. Malkevitch, editors, *Combinatorial Mathematics: Proceedings of the Third International Conference*, volume 555 of *Annals of the New York Academy of Sciences*, pages 171–180, 1989.

[DGPR84] P. Dūriš, Z. Galil, W. Paul, and R. Reischuk. Two nonlinear lower bounds for on-line computations. *Inform. and Control*, 60:1–11, 1984.

[DM93] M. Dietzfelbinger and W. Maass. The complexity of matrix transposition on one-tape off-line turing machines with output tape. *Theor. Comp. Sci.*, 108:271–290, 1993.

[DMS91] M. Dietzfelbinger, W. Maass, and G. Schnitger. The complexity of matrix transposition on one-tape off-line Turing machines. *Theor. Comp. Sci.*, 82:113–129, 1991.

[FMR72] P. Fischer, A. Meyer, and A. Rosenberg. Real-time simulations of multihead tape units. *J. ACM*, 19:590–607, 1972.

[FS92] Y. Feldman and E. Shapiro. Spatial machines: A more-realistic approach to parallel computation. *Comm. ACM*, 35:60–73, October 1992.

[FW90] M. Fredman and D. Willard. BLASTING through the information theoretic barrier with fusion trees. In *Proc. 22nd STOC*, pages 1–7, 1990.

[FW93] M. Fredman and D. Willard. Surpassing the information theoretic bound with fusion trees. *J. Comp. Sys. Sci.*, 47:nnn–nnn, 1993.

[GJ79] M. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* Freeman, 1979.

[GR91] E. Grandjean and J. Robson. RAM with compact memory: a robust and realistic model of computation. In *CSL '90: Proceedings of the 4th Annual Workshop in Computer Science Logic*, volume 533 of *Lect. Notes in Comp. Sci.*, pages 195–233. Springer Verlag, 1991.

[Gra88] E. Grandjean. A natural NP-complete problem with a nontrivial lower bound. *SIAM J. Comput.*, 17:786–809, 1988.

[Gra90a] E. Graedel. On the notion of linear-time computability. *Intern. J. Found. Comp. Sci.*, 1:295–307, 1990.

[Gra90b] E. Grandjean. A nontrivial lower bound for an NP problem on automata. *SIAM J. Comput.*, 19:438–451, 1990.

[Gra93] E. Grandjean. Sorting linear time and the satisfiability problem. Technical Report 17, Laboratoire d'Informatique de l/Université de Caen, September 1993.

[Gra94a] E. Grandjean. Invariance properties of RAMs and linear time. *Computational Complexity*, 4:62–106, 1994.

[Gra94b] E. Grandjean. Linear time algorithms and NP-complete problems. *SIAM J. Comput.*, 23:573–597, 1994.

[Gru90] J. Gruska. Synthesis, structure, and power of systolic computations. *Theor. Comp. Sci.*, 71:47–77, 1990.

[GS89] Y. Gurevich and S. Shelah. Nearly-linear time. In *Proceedings, Logic at Botik '89*, volume 363 of *Lect. Notes in Comp. Sci.*, pages 108–118. Springer Verlag, 1989.

[Hen66] F. Hennie. On-line Turing machine computation. *IEEE Trans. Electr. Comp.*, EC-15:35–44, 1966.

[HKL92] T. Harju, H.C.M. Kleijn, and M. Latteux. Deterministic sequential functions. *Acta Informatics*, 29:545–554, 1992.

[HKMW92] J. Hromkovic, M. Krause, C. Meinel, and S. Waack. Branching programs provide lower bounds on the areas of multilective deterministic and nondeterministic VLSI circuits. *Inform. and Control*, 96:168–178, 1992.

[HPV75] J. Hopcroft, W. Paul, and L. Valiant. On time versus space and related problems. In *Proc. 16th FOCS*, pages 57–64, 1975.

[HS66] F. Hennie and R. Stearns. Two–way simulation of multitape Turing machines. *J. ACM*, 13:533–546, 1966.

[HU79] J. Hopcroft and J. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison–Wesley, Reading, MA, 1979.

[JL93] T. Jiang and M. Li. K one-way heads cannot do string-matching. In *Proc. 25th STOC*, pages 62–70, 1993.

[Jon93] N. Jones. Constant factors do matter. In *Proc. 25th STOC*, pages 602–611, 1993.

[JSV94] T. Jiang, J. Seiferas, and P. Vitányi. Two heads are better than two tapes. In *Proc. 26th STOC*, pages 668–675, 1994. U. Rochester Comp. Sci. TR 493, March 1994.

[KAI79] T. Kasai, A. Adachi, and S. Iwata. Classes of pebble games and complete problems. *SIAM J. Comput.*, 8:576–586, 1979.

[Kar72] R. Karp. Reducibility among combinatorial problems. In R.E. Miller and J.W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–104. Plenum Press, 1972.

[KL80] R. Karp and R. Lipton. Some connections between uniform and nonuniform complexity classes. In *Proc. 12th STOC*, pages 302–309, 1980.

[KMW89] M. Krause, C. Meinel, and S. Waack. Separating complexity classes related to certain input-oblivious logarithmic space-bounded Turing machines. In *Proc. 4th Structures*, pages 240–249, 1989.

[Kos79] R. Kosaraju. Real-time simulation of concatenable double-ended queues by double-ended queues. In *Proc. 11th STOC*, pages 346–351, 1979.

[KW91] M. Krause and S. Waack. On oblivious branching programs of linear length. *Inform. and Comp.*, 94:232–249, 1991.

[LF80] R. Ladner and M. Fischer. Parallel prefix computation. *J. ACM*, 27:831–838, 1980.

[LL92] M. Loui and D. Luginbuhl. The complexity of on-line simulation between multidimensional Turing machines and random-access machines. *Math. Sys. Thy.*, 25:293–308, 1992.

[LLV92] M. Li, L. Longpré, and P. Vitányi. The power of the queue. *SIAM J. Comput.*, 21:697–712, 1992.

[Lou81] M. Loui. Simulations among multidimensional Turing machines. *Theor. Comp. Sci.*, 21:145–161, 1981.

[Lou83] M. Loui. Optimal dynamic embedding of trees into arrays. *SIAM J. Comput.*, 12:463–472, 1983.

[Lou84] M. Loui. Minimizing access pointers into trees and arrays. *J. Comp. Sys. Sci.*, 28:359–378, 1984.

[LS81] B. Leong and J. Seiferas. New real-time simulations of multihead tape units. *J. ACM*, 28:166–180, 1981.

[LV88] M. Li and P. Vitányi. Tape versus queue and stacks: the lower bounds. *Inform. and Comp.*, 78:56–85, 1988.

[LV93] M. Li and P. Vitányi. *An Introduction to Kolmogorov Complexity and its Applications.* Springer Verlag, 1993.

[Man82] G. Manacher. Steady-paced-output and fractional-on-line algorithms on a RAM. *Inf. Proc. Lett.*, 15(2):47–52, 1982.

[MC80] C. Mead and L. Conway, editors. *Introduction to VLSI Systems.* Addison-Wesley, Reading, Mass., 1980.

[Mei89] C. Meinel. *Modified Branching Programs and Their Computational Power*, volume 370 of *Lect. Notes in Comp. Sci.* Springer Verlag, Berlin, 1989.

[Mei90a] C. Meinel. Polynomial-size Ω-branching programs and their computational power. *Inform. and Control*, 85:163–182, 1990.

[Mei90b] C. Meinel. Restricted branching programs and their computational power. In *Proc. 15th MFCS*, volume 452 of *Lect. Notes in Comp. Sci.*, pages 61–75. Springer Verlag, 1990.

[MNT93] Y. Mansour, N. Nisan, and P. Tiwari. The computational complexity of universal hashing. *Theor. Comp. Sci.*, 107:121–133, 1993.

[MS86] W. Maass and G. Schnitger. An optimal lower bound for Turing machines with one work tape and a two–way input tape. In *Proc. 1st Annual Conference on Structure in Complexity Theory*, volume 223 of *Lect. Notes in Comp. Sci.*, pages 249–264. Springer Verlag, 1986.

[Ofm63] Y. Ofman. On the algorithmic complexity of discrete functions. *Soviet Physics—Doklady*, 7(7):589–591, 1963. English translation.

[PPST83] W. Paul, N. Pippenger, E. Szemerédi, and W. Trotter. On determinism versus nondeterminism and related problems. In *Proc. 24th FOCS*, pages 429–438, 1983.

[PR81] W. Paul and R. Reischuk. On time versus space II. *J. Comp. Sys. Sci.*, 22:312–327, 1981.

[PS84] W. Paul and J. Simon. Decision trees and random-access machines. In K. Mehlhorn, editor, *Sorting and Searching*, pages 85–97. Springer Verlag, 1984.

[Reg93] K. Regan. Machine models and linear time complexity. *SIGACT News*, 24:5–15, October 1993. Guest column, L. Hemachandra ed., “Compelxity Theory Column”.

[Reg94a] K. Regan. Linear time algorithms in memory hierarchies. In B. Pehrson and I. Simon, editors, *Proceedings, 13th IFIP World Computer Congress '94, Volume 1: Technology and Foundations*, volume A-51 of *IFIP Transactions*, pages 288–293. North-Holland, 1994.

[Reg94b] K. Regan. Linear time and memory efficient computation. Technical Report UB-CS-TR 94-18, Computer Science Dept., University at Buffalo, 1994. Revision of UB-CS-TR 92-28, accepted to *SIAM J. Comput.*

[Reg94c] K. Regan. A new parallel vector model, with exact characterizations of NC^k. In *Proc. 11th STACS*, volume 778 of *Lect. Notes in Comp. Sci.*, pages 289–300. Springer Verlag, 1994.

[Rob91] J. Robson. An $O(T \log T)$ reduction from RAM computations to satisfiability. *Theor. Comp. Sci.*, 82:141–149, 1991.

[RR94] A. Razborov and S. Rudich. Natural proofs. In *Proc. 26th STOC*, pages 204–213, 1994.

[RSC95] K. Regan, D. Sivakumar, and J.-Y. Cai. Pseudorandom generators, measure theory, and natural proofs. Technical Report UB-CS-TR 95-02, Computer Science Dept., University at Buffalo, January 1995.

[Sch78] C. Schnorr. Satisfiability is quasilinear complete in NQL. *J. ACM*, 25:136–145, 1978.

[Sch80] A. Schönhage. Storage modification machines. *SIAM J. Comput.*, 9:490–508, 1980.

[Sch88] A. Schönhage. A nonlinear lower bound for random-access machines under logarithmic cost. *J. ACM*, 35:748–754, 1988.

[Sel94] A. Selman. A taxonomy of complexity classes of functions. *J. Comp. Sys. Sci.*, 48:357–381, 1994.

[SH90] R. Stearns and H. Hunt III. Power indices and easier hard problems. *Math. Sys. Thy.*, 23:209–225, 1990.

[SMRB83] A. Selman, X. Mei-Rui, and R. Book. Positive relativizations of complexity classes. *SIAM J. Comput.*, 12:565–579, 1983.

[vEB90] P. van Emde Boas. Machine models and simulations. In J. Van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume A, pages 1–66. Elsevier and MIT Press, 1990.

[Vit91] J. Vitter. Efficient memory access in large-scale computation. In *Proc. 8th STACS*, volume 480 of *Lect. Notes in Comp. Sci.*, pages 26–41. Springer Verlag, 1991.

[VN92] J. Vitter and M. Nodine. Large-scale sorting in uniform memory hierarchies. Technical Report Technical Report CS–92–02, Department of Computer Science, Brown University, August 1992.

[Wig93] A. Wigderson. The fusion method for lower bounds in circuit complexity. In *Combinatorics, Paul Erdős is Eighty*, Bolyai Society Mathematical Studies, pages 453–467. North-Holland, Amsterdam, 1993.

[Wil92] D. Willard. A density control algorithm for doing insertions and deletions in a sequentially ordered file in a good worst-case time. *Inform. and Comp.*, 97:150–204, 1992.

[WW86] K. Wagner and G. Wechsung. *Computational Complexity*. D. Reidel, 1986.