Branching Programs with Variables, and Implications for Time-Space Tradeoffs

Denis X. Charles*
University at Buffalo

Kenneth W. Regan†
University at Buffalo

Abstract

We introduce branching programs augmented with the ability to write to and read from variables other than the inputs. This is a substantial strengthening of the model. We show, however, that Ajtai's size lower bounds for linear-time multi-way branching programs solving a Hamming distance problem [Ajt99a] carry over to the stronger model. This indicates an obstacle to extending this and related results to a general $TS = \Omega(n^2)$ tradeoff for running times T above $n \log n$, yet also indicates "slack" in these results that might be exploited for better lower bounds in the original model. We also define and study an intermediate model in which the variable registers are write-once-only.

1 Introduction

A *branching program* B is an idealization of a random-access machine M running on inputs x of some length n over an alphabet Σ. B is a directed acyclic graph with one source `start` and sinks labeled 0 or 1. Each non-sink node of B bears a label "x_i" $(1 \leq i \leq n)$ meaning to poll the ith character of x, and has an outgoing edge for each value in Σ. An input $x \in \Sigma^n$ defines a unique path P_x through B, with result (accept or reject) given by the label of the sink reached by P_x. Intuitively, each node in P_x represents a configuration of M in its computation on x. The number s of nodes in B is thus the number of different configurations needed by M on inputs of length n. The logarithm of s was advanced by Cobham [Cob64] as a measure of the space S required by M. The time complexity T of B, corresponding to the worst-case running time of M, is the maximum length of P_x over all inputs x. Extending the model by giving certain edges labels $\langle j, b\rangle$ for "the j-th character of $f(x)$ is b" allows branching programs to compute functions $f : \Sigma^n \to \Sigma^m$ rather than recognize subsets of Σ^n.

Branching programs abstract away any notions of "contiguity" of successive configurations by which (for instance) Turing machines are considered more constrained than random-access machines. Hence lower bounds on T and S for branching programs are considered to carry over to any reasonable machine model (see argument in [BC82, Bea91]). However, every subset of or function on Σ^n is computed by a trivial branching program B of size $2^{n+1} - 1$ and depth n, giving $S = \Theta(n)$ (for fixed Σ) and $T = n$. This B is simply a full $|\Sigma|$-ary tree that reads the n input characters and maintains a separate configuration for all possibilities. Hence no

*Supported in part by NSF grant CCR-9821040
†Supported in part by NSF grant CCR-9821040. Corresponding author—contact: regan@cse.buffalo.edu

super-linear lower bounds can be proved for branching-program time or space in isolation, nor anything higher than $ST = \Omega(n^2)$ for the time-space product. However, so-called $ST = \Omega(n^2)$ *time-space tradeoffs* have long been known for functions f such as sorting or computing $ab+c$ in finite fields or any mapping $F(h,x) = h(x)$ with h chosen from families of universal$_2$ hash functions (see [Bea91, MNT93]). Ajtai made the breakthrough of obtaining time-space tradeoffs for languages, first with $|\Sigma|$ growing as n [Ajt99a] and then with $\Sigma = \{0,1\}$ [Ajt99b]. His bounds have the form that if $T = O(n)$, i.e. $T \leq kn$ for some fixed $k > 0$ and all large enough n, then $S = \Omega(n \log |\Sigma|)$, i.e. the branching programs must have size $2^{\epsilon n \log |\Sigma|}$ for some $\epsilon > 0$ depending on k and all large enough n. Beame et al. [BSSV00] obtain cases with $T = O(n\delta(n))$ and $S \approx n/\delta(n)$ for certain functions $\delta(n) = o(\sqrt{\log n})$. It is natural to ask whether these results extend for super-linear time bounds to a general $ST = \Omega(n^2)$ time-space tradeoff in the manner of [BC82, Bea91, MNT93], or at least with $T \geq n \log n$. This paper provides evidence to the contrary.

Our main result is that Ajtai's above-cited results carry over to an extension of the BP model that is far stronger than its inventors contemplated. The extension adds to the input variables $x_1, \ldots, x_n$ a set $y_1, \ldots, y_m$ of read-write variables, where m may depend on n. The syntax for writing a value $b \in \Sigma$ to variable y_j is the same edge-label $\langle j, b\rangle$ as in the branching-program model for functions, while reads are handled by node labels in $y_1, \ldots, y_m$ rather than $x_1, \ldots, x_n$. This extension makes space essentially free for the computation: for each y_j to contribute one unit to S the size s of the branching program should double, but instead the occurrences of y_j are merely added to s. More concretely, any linear time Turing machine can be simulated with $T = O(n \log n)$ and $S = O(\log n)$ in this "RW-BP" model, even with *oblivious* computation. Hence any $ST = \Omega(n^2)$ or even $ST = \Omega(n^{1+\epsilon})$ time-space tradeoff is ruled out in the extended model. Insofar as Ajtai's techniques are impervious to this extension, they absorb this limitation.

While our read-write extension is too powerful for practical relevance, the intermediate "WORM-BP" model obtained by limiting writes to once per variable seems interesting in its own right. Irani, Naor, and Rubinfeld [INR92] studied WORM versions of RAM models, and observed that logspace (or constant space) plus a polynomial amount of WORM memory characterizes polynomial time. Beame, Saks, and Thathatchar [BST98] began by giving polynomial-time versus log-space as a primary motivation for studying branching programs; the added power of WORM-BPs shows up in erasing that distinction. We observe that for running times kn our syntactically defined RW-BPs and WORM-BPs are equivalent with regard to size. Our paper raises some interesting open problems about the power of WORM-BPs and their relation to nondeterminism and to the closure of small-BP functions under composition.

2 Branching programs with variables

Formal definitions of the standard branching-program models for decision problems and functions are readily available in the major references [Bea91, MNT93, Ajt99a, Ajt99b, BSSV00]. In this paper we consider only branching programs for decision problems. We define our extensions formally.

Definition 2.1. A *branching program with read-write variables* (RW-BP) over an alphabet Σ is a one-source DAG B whose non-sink nodes have labels in the *input set* $\{x_1, \ldots, x_n\}$ and the *variable set* $\{y_1, \ldots, y_m\}$. Those with labels in $\{y_1, \ldots, y_m\}$ have an extra integer tag as in (a) below. Sinks are labeled 0 or 1 as usual, and non-sink nodes have $|\Sigma|$-many outgoing edges labeled by the elements in Σ. Certain edges may have additional labels $\langle j, b \rangle$ meaning "$y_j := b$," with $1 \le j \le m$ and $b \in \Sigma$. We enforce the following syntactic restrictions:

(a) Every node with a variable y_j is tagged also with an integer $\ell \ge 1$ standing for the number of times y_j has been previously written to. Along every path from the source to y_j (not just legal paths P_x for some input x) there must be exactly ℓ-many edge-labels of the form $\langle j, - \rangle$. In order down from the source, these labels can be denoted by $\langle j, b_1 \rangle, \ldots, \langle j, b_\ell \rangle$.

(b) An individual edge can have at most one $\langle j, b \rangle$ label. That is, at most one y_j variable can be assigned in any one step.

The unique path P_x for an input $x \in \Sigma^n$ is determined by taking the out-edge labeled x_i at a node labeled x_i, and for a node labeled y_j with extra tag ℓ, taking the out-edge labeled b_ℓ (from the closest preceding edge label $\langle j, b_\ell \rangle$ in P_x). The output $B(x)$ is the label 0 or 1 of the sink reached by P_x.

Definition 2.2. A *BP with write-once variables* (WORM-BP) is the case of Definition 2.1 in which all y_j nodes have tag $\ell = 1$.

The two syntactic restrictions are mainly for convenience, and are met in our Theorem 2.2 on simulations of Turing machines. One can entertain "semantic" variants without the extra integer tags ℓ in (a) and/or the restriction to one write per edge in (b), but we defer questions about their power to the concluding section. One aspect of (b) is that we can associate each write $y_j := b$ with a unique separate node by inserting a new node into the edge, with all out-arcs from that node (reading, say, x_1) going to the original target node of that edge. Doing so multiplies the size by a factor of $|\Sigma| + 1$, which is negligible when $|\Sigma|$ is independent of n but may be noticeable when $\Sigma = \{0, \ldots, \alpha\}$ with $\alpha = n^{O(1)}$, as in [Ajt99a]. Alternately we can associate the write with the original source or target node of the edge—we revisit this matter when we get to the technical details of time-intervals as defined by Ajtai. We associated the write to an edge rather than a node in our formal definition for clarity and consistency with the standard definition of functional BPs.

A branching program is *leveled* if its nodes can be partitioned into "level sets" $S_0, \ldots, S_t$ with S_0 comprising the unique source node, S_t the sinks, and with all edges from S_k ($0 \le k \le t-1$) going to S_t. The *width* of a leveled BP is the maximum size of S_k over all k.

Lemma 2.1 *For every RW-BP B of size s and maximum depth d over Σ, we can create an equivalent leveled RW-BP B' of depth $2d$ and width at most s and with a new RW-variable y_0, whose levels alternate reading from $\{x_1, \ldots, x_n\}$ and from $\{y_0, y_1, \ldots, y_m\}$.*

Proof. First apply the standard leveling process to B, inserting extra nodes reading from $\{x_1, \ldots, x_n\}$, which creates an equivalent BP of depth d and width at most s. All edges from the root write 0 to y_0, this being the only time that y_0 is written to, and go to dummy nodes that read from y_0. Then for each level, segregate nodes reading from $\{x_1, \ldots, x_n\}$ and those reading from $\{y_1, \ldots, y_m\}$ into two levels, inserting dummy reads to x_1 and to y_0 (the latter with tag 1 always) as needed. $\square$

Note that this says nothing about leveling the *writes*. It is possible to associate each write with a node rather than an edge, by inserting extra levels of nodes that do dummy-reads, but this idea is not needed here.

A BP is *oblivious* if it is leveled and all nodes in each level have the same label (x_i or y_j, ℓ in the case of a RW-BP). Unlike leveling, this condition is far from innocuous—no better general way of achieving it is known than multiplying the depth by the number of variables, which forces $T \geq n^2$. Some lower bounds are known for oblivious BPs but not general ones, such as those in [AM88]. Of main interest to us is the following upper bound.

Theorem 2.2 *A Turing machine M running in time $t(n)$ and space $s(n)$ can be simulated by oblivious RW-BPs B_n with $m = O(s(n))$ read-write variables, constant width, and depth $T = O(t(n) \log t(n))$.*

Proof. By Pippenger-Fischer's extension [PF79] of the Hennie-Stearns theorem, M can be simulated by an oblivious 2-tape Turing machine M' in time $O(t(n) \log t(n))$ and space $m = O(s(n))$. Here "oblivious" for Turing machines means that the sequence of positions of the input tape head and the two worktape heads at a given time t depend only on the length n of the input x, not on the characters in x. Now let the m worktape cells used by M' become read-write variables of a RW-BP B_n. Since the sequence of visiting cells depends only on n, we can associate to each worktape cell y_j visited at some time-step t the ordinal ℓ of times y_j has been visited before in this sequence. The single time-step t of M' becomes 5 levels of B_n, one to read an input variable, then two to read the two worktape cells, and finally two to *write* the new values to the worktape cells. (Technical details in [PF79] and in our definition actually allow us to save two levels, but this doesn't matter.) The width of the fourth level in this sequence is bounded by $|\Sigma|^3 \cdot |Q|$, where Σ is the (fixed!) alphabet of M', and Q is the state set of M'. □

The family B_n thus obtained is also *uniform* in senses we need not elaborate here. When $t(n) = O(n)$, there exists a fixed $\delta > 0$ such that for all sufficiently large n, B_n uses δn read-write variables and runs in time $T = O(n \log n)$. For RW-BPs B_n that run in *linear* time $T = kn$, we can eliminate multiple writes:

Proposition 2.3 *RW-BPs B_n that run in time kn can be simulated by WORM-BPs B'_n that run in the same time and use at most kn variables.*

Proof. We have set the definitions so that this only requires replacing each label y_j, ℓ by a new variable y_{jl}. (*Addendum 6/6/01:* Not quite so from the above—we could strengthen Definition 2.1 to make this so while preserving Theorem 2.2 and its import, but the whole WORM idea is actually not needed for our proofs, as noted also below. It was our original motivating idea and was also a convenience for initially visualizing the proofs, but a closer look at the counting confirms that it is unnecessary, and the idea is *obsolete* for cases $T = \omega(n)$ that we are working on anyway.) □

Corollary 2.4 *Every language in deterministic Turing machine linear time can be accepted*

(a) by uniform oblivious RW-BPs with $m = O(n)$, $T = O(n \log n)$, and $S = O(\log n)$; and

(b) *by uniform oblivious WORM-BPs with* $m = O(n \log n)$, $T = O(n \log n)$, *and* $S = O(\log n + \log\log n) = O(\log n)$. $\square$

Likewise, every language in P has oblivious WORM-BPs with $T = n^{O(1)}$ and $S = O(\log n)$. Thus WORM-logspace captures P in this model, as in [INR92].

3 Essential notions and transition to RW-BPs

The strategy of Beame et al. [BSSV00] of "applying the space bound early in the argument," by initially fragmenting a BP B into an OR of ANDs of small-depth decision trees, seems at first sight to fail for RW-BPs, even for WORM-BPs, because of the dependence on the y_j variables across fragments. We believe it is possible to make our strategy resemble theirs by building the notion of *which write-once values are set during a time-interval* on top of a decision-tree fragment. However, this may upset the essential simplicity of this step in [BST98, BSSV00], making it less worthwhile. Hence we stay with Ajtai's side of the contrast drawn in [BSSV00], applying the space bound "later in the argument" after first getting control of the y_j dependences.

The paper [Ajt99a] contains a first sequence of lemmas that provide lower bounds on a Hamming distance problem over alphabet $\Sigma = \{ 0, \ldots, \alpha - 1 \}$ with $\alpha = n^{O(1)}$, and then a stronger sequence giving lower bounds on the classical element-distinctness problem. To date we have successfully carried over the first sequence. This culminates in the following extension of Ajtai's first main theorem.

Definition 3.1. Given any $\gamma < 1/2$ and $c > 0$, and for any input length n, define α_{cn} to be the next power of 2 above n^c, and $\Sigma_{cn} = \{ 0, \ldots, \alpha_{cn} - 1 \}$ thought of as the set of binary strings of length $\ell = \log_2(\alpha_{cn})$. Then define the Hamming distance set $HD_n(\gamma, c)$ to be the set $\{x \in \Sigma^n$: there exist distinct characters x_i, x_j of x whose corresponding binary strings differ in no more than $\gamma\ell$ places$\}$.

Theorem 3.1 (cf. [Ajt99a], Theorem 1 on p7) *For all positive* $\gamma < 1/2$ *there exists* $c > 0$ *such that for all integers* $k, r \geq 1$ *there exists* $\epsilon > 0$ *such that for all sufficiently large* n, *every* n*-input RW-BP* M *of depth* kn *and size at most* $2^{\epsilon n \log n}$, *using alphabet* Σ_{nc} *and* rn *variables, fails to recognize* $HD_n(\gamma, c)$.

We will perforce also obtain the abstraction of this statement to arbitrary relations on Σ (in place of the Hamming distance condition) that are "λ-full" and "$(1/2, n)$-sparse," with $\lambda = n^{-\delta}$ for arbitrary positive $\delta < c$, which corresponds to Theorem 2 of [Ajt99a], p7. This implies Theorem 3.1 via a combinatorial argument that is postponed to page 16 of [Ajt99a] and has nothing to do with branchine programs. Our version of the abstracted theorem follows from our parallels to the sequence Lemma 1–Lemma 8 in [Ajt99a] in the same manner as there. We give detailed proofs only for those parts where the argument requires modification for RW-BPs in this extended abstract, putting other information in prose text with reference to [Ajt99a] (page numbers are for the revised version) itself.

First, we include the rest of Ajtai's notation with important additions and changes noted. We preserve his use of M as synonymous with B for branching programs, and "state" for node; while we pluralize his `right` to `rtimes` and `rstate` to `rstates` to emphasize that these functions do not return a single timestep or node when their argument is not a single time interval. By results in the last section, we may assume that M is leveled and *write-once*, the latter since M runs in linear time,

Notation, Terms, and Givens

The parameters c and γ on the Hamming-distance problem are fixed. The large constants k on the linear running time of BPs M and r (new, can be chosen equal to k) on the number $m = rn$ of variables they may use are givens. The following quantities are chosen and manipulated in the proofs:

1. σ: a small real number depending on k.
2. $\mathcal{I}$: a partition of the timesteps $0, \ldots, kn-1$ into intervals of duration between σn and $2\sigma n$. Subject to this, it does not matter how $\mathcal{I}$ is chosen.
3. ϵ: depends on k and choice of σ, and defines a size bound of $2^{\epsilon n \log n}$ for M.
4. x: an element of Σ^n given as input to M.
5. P_x: the unique path in M induced by x (added notation). Often x and P_x are interchangeable.
6. $\texttt{out}(x)$: the 0-1 output value of the sink reached by P_x.
7. $\texttt{defvals}(u, x)$: the defined values of $y_1, \ldots, y_m$ at node u in P_x, including '$\star$' for an undefined value (new).
8. $\texttt{state}(t, x)$: the t-th node in the path P_x.
9. η: a "partial input," formally a function from a subset D of $\{1, \ldots, n\}$ to $\Sigma^{|D|}$.
10. $x \wr \eta$: the modified input obtained by changing the character in any place $u \in \mathit{dom}(\eta)$ to $\eta(u)$.
11. T: any subset of $\{0, \ldots, kn\}$, regarded as a set of timesteps.
12. I: a set of timesteps that forms an interval.
13. $\texttt{rtimes}(T)$: the set of $t \notin T$ such that $t - 1 \in T$. If T is an interval this is a single timestep.
14. $\texttt{rstates}(T, x) = \{\, \texttt{state}(x, t) : t \in \texttt{rtimes}(T) \,\}$.
15. $\texttt{wvals}(T, x)$: a function $g : T \to (\{1, \ldots, m\} \times \Sigma) \cup \{\star\}$ such that for all $t \in T$, $g(t) = (j, b)$ if value b is written to y_j at that timestep along P_x, $g(t) = \star$ if no value is written in that step.
16. $\texttt{reg}(T, x)$: the set of registers in $\{x_1, \ldots, x_n\}$ *and* $\{y_1, \ldots, y_m\}$ accessed in $\texttt{state}(t, x)$ for $t \in T$ (changed).
17. $\texttt{Ireg}(T, x) = \texttt{reg}(T, x) \cap \{x_1, \ldots, x_n\}$: the literal equivalent of $\texttt{register}(T, x)$ in [Ajt99a], but our extended $\texttt{reg}(T, x)$ above plays the corresponding role in the proofs.

18. $\texttt{core}(T, x) = \texttt{reg}(T, x) \setminus \texttt{reg}(\sim T, x)$, where $\sim T$ is the somplement of T in $\{ 0, \ldots, kn - 1 \}$. These are the registers accessed *only* at timesteps in T. This set can be empty, but the goal is to find many inputs x for which it is large for certain T.

19. $\texttt{Icore}(T, x) = \texttt{core}(T, x) \cap \{ x_1, \ldots, x_n \}$: again the literal but not figurative equivalent of Ajtai's $\texttt{core}(T, x)$. The last two differences actually do not matter in defining:

20. R_x: The partition of $\{ x_1, \ldots, x_n \}$ *only* (i.e., no y_j registers) defined by $u \equiv_x v$ if for every interval $I \in \mathcal{I}$, $u \in \texttt{Ireg}(I, x) \iff v \in \texttt{Ireg}(I, x)$. (Recall that $\mathcal{I}$ is fixed and its members I have adjustably short height σn to $2\sigma n$.)

21. $R'_x \subset R_x$: those classes of R_x such that some (equivalently, all) $u \in R'_x$ are accessed in at most $2k$ intervals on $\mathcal{I}$.

22. $\Gamma_x \subset R'_x$: those classes of R_x that have more than $n/(4|R'_x|)$ registers.

23. $\texttt{set}(C, x)$: for a class C in Γ_x, the set of intervals in which some (equivalently, each) register of C is accessed.

The distinction between $\texttt{Icore}$ versus $\texttt{core}$ and between $\texttt{Ireg}$ versus $\texttt{reg}$, is *not* needed for our main proof here, but is important to bear in mind and seems to be important for current attempts to extend Ajtai's second main result in [Ajt99a], as discussed in the conclusions section.

4 Proof of the Main Theorem

Lemma 1 from [Ajt99a] is unchanged here: it states that if A_1 and A_2 are sets of partial inputs so that for all $\eta_1 \in A_1$ and $\eta_2 \in A_2$, $dom(\eta_1) \cap dom(\eta_2) = \emptyset$, and the numbers of modified inputs obtainable as $x \wr \eta$ for $\eta \in A_1$ respectively $\eta \in A_2$ are both at least α/n^δ, then there exist $\eta_1 \in A_1$ and $\eta_2 \in A_2$ with $\texttt{out}_{kn}((x \wr \eta_1) \wr \eta_2)$. This connects with the structure of the Hamming distance problem (or any defined analogously via an $n^{-\delta}$-full relation Q) to provide the contradiction that proves Theorem 3.1 in the same way as in [Ajt99a]. Our first adjustment is to Lemma 2 on p11 of [Ajt99a].

Lemma 4.1 (cf. Lemma 2 of [Ajt99a]) *Let x be an input, let η_1, η_2 be partial inputs, and let $T_1, T_2 \subseteq \{0, 1, \cdots, kn - 1\}$. Suppose the following hold:*

1. *$dom(\eta_1) \cap dom(\eta_2) = \emptyset$;*

2. *$T_1 \cap T_2 = \emptyset$;*

3. *$dom(\eta_1) \subseteq \texttt{core}(T_1, x)$ and $dom(\eta_2) \subseteq \texttt{core}(T_2, x)$.*

4. *$\texttt{rstates}(T_1, x) = \texttt{rstates}(T_1, x \wr \eta_1)$. and $\texttt{rstates}(T_2, x) = \texttt{rstates}(T_2, x \wr \eta_2)$.*

5. *$\texttt{wvals}(T_1, x) = \texttt{wvals}(T_1, x \wr \eta_1)$ and $\texttt{wvals}(T_2, x) = \texttt{wvals}(T_2, x \wr \eta_2)$.*

6. *$dom(\eta_1) \cap \texttt{reg}(T_2, x \wr \eta_2) = \emptyset$ and $dom(\eta_2) \cap \texttt{reg}(T_1, x \wr \eta_1) = \emptyset$.*

Then $\texttt{out}_{kn}(x) = \texttt{out}_{kn}((x \wr \eta_1) \wr \eta_2)$ *and* $\texttt{defvals}(\texttt{state}(kn, x), x) = \texttt{defvals}(\texttt{state}(kn, (x \wr \eta_1) \wr \eta_2), (x \wr \eta_1) \wr \eta_2))$.

Item 5 is a strong condition to impose, and we succeed only because it does not fatally dilute the counting lemma that comes last.

Proof. Let K_i be a set of disjoint intervals of minimum cardinality such that $\cup K_i = T_i$. Let $K = K_1 \cup K_2$. Order $K = \{I_1, I_2, \cdots, I_r\}$ so that $t_i \in I_i < t_j \in I_j$ if $i < j$. Let $x_1 = x \wr \eta_1$, $x_2 = x \wr \eta_2$ and $x' = (x \wr \eta_1) \wr \eta_2$. We will show that M on input x' acts like M on input x_i during times from intervals $I \in K_i$ and acts like M on input x elsewhere, which implies the result.

Let h_s be the unique element of $\mathtt{rtimes}(I_s)$. We *claim* that

$$\begin{aligned} \mathtt{state}_x(h_s) &= \mathtt{state}_{x_1}(h_s) \\ &= \mathtt{state}_{x_2}(h_s) \\ &= \mathtt{state}_{x'}(h_s). \end{aligned}$$

For the basis consider $s = 1$, and wlog. suppose that the interval I_1 belongs to K_1. Since $\eta_1 \subseteq \mathtt{core}(T_1, x)$ and $\eta_2 \subseteq \mathtt{core}(T_2, x)$, all values read before we enter this interval by M on all the four inputs are the same, so that the paths taken up to times in I_1 are the same and hence the write-once registers have the same values. During I_1, by using the condition $dom(\eta_2) \cap \mathtt{reg}(T_1, x \wr \eta_1) = \emptyset$, we conclude that M does not access any input changed by η_2 (the write-once registers have identical values so reading them does not cause a problem), and so M behaves as though it operates on input x_1. However at the right end of the interval, $\mathtt{rstates}(T_1, x) = \mathtt{rstates}(T_1, x_1)$, which gives $\mathtt{state}_x(h_s) = \mathtt{state}_{x_1}(h_s)$. The write-once registers also have the same values by condition 5. Thus at the end of the first interval $M(x), M(x_1), M(x_2)$ and $M(x')$ are in identical configuration. The proof of the inductive step is similar. $\square$

Note that the paths taken by $M(x), M(x_1), M(x_2)$ and $M(x')$ may differ within the intervals considered, but since by condition 4 the first three end at the same node closing the interval, and since by condition 5 the same values were written during the interval (and at the same time instants—this doesn't matter here) to the y_j variables, the paths of all four reconvene after the interval.

The next two lemmas in [Ajt99a] are counting lemmas that relate the number of strings obtainable as $x \wr \eta$ over $\eta \in A$ (there called $s = |\bigcup_{\eta \in A} \mathtt{range}(\eta)|$) to the size of the set $D \subset \{1, \ldots, n\}$ for which A is a set of η of common domain D. Lemma 3 concludes that $s^{|D|} \geq |A|$, so that if $|A| \geq (\alpha/n^\delta)^{|D|}$ then $s > \alpha/n^\delta$. (Here s does not mean BP size.) Lemma 4 states that if M, x, A_1, A_2, T_1, T_2 are such that:

- The $\eta \in A_1$ have a common domain W_1 and the $\eta \in A_2$ have a common domain W_2;
- For all $\eta_1 \in A_1$ and $\eta_2 \in A_2$ the conditions of (our) Lemma 2 are satisfied by $x, \eta_1, \eta_2, T_1, T_2$; and
- $|A_1| > (\alpha/n^\delta)^{|W_1|}$ and ditto for A_2, W_2,

then there exist $\eta_1 \in A_1$, $\eta_2 \in A_2$, and distinct $i, j \in \{1, \ldots, n\}$ such that with $x' = (x \wr \eta_1) \wr \eta_2)$ we have $\mathtt{out}_{kn}(x) = \mathtt{out}_{kn}(x')$ and the required Hamming-distance or λ-full relation holds between x_i and x_j. This carries over transparently and connects directly with the (converted) Lemmas 1–3.

Lemma 5 involves the definitions of R_x, R'_x, Γ_x, and $\mathtt{set}(C, x)$ above. It states:

Lemma 4.2 (cf. Lemma 5 in [Ajt99a], p12) *For any input x:*

1. $\forall C \in \Gamma_x \ : \ |C| \geq \frac{1}{4}(\frac{\sigma}{k})^{2k} n.$
2. $|\cup_{C \in \Gamma_x} C| \geq \frac{1}{4} n.$
3. *There are $W_1, W_2 \in \Gamma_x$ with $\mathtt{set}(W_1, x) \cap \mathtt{set}(W_2, x) = \emptyset$.*

The proof in [Ajt99a] carries through because our definitions of R_x, R'_x, Γ_x, and $\mathtt{set}(C, x)$ are the same as in [Ajt99a] involving only the input registers, and only counting based on the definitions is used. The counting does not care that the levels reading from $y_1, \ldots, y_m$ act as "dead weight" here, the same as if they were "dummy reads" of x_1. Any alteration of constants that one could hope to obtain by taking this into account would be absorbed by the freedom to choose σk arbitrarily small anyway.

Our critical juncture is the next lemma. We need to allow for all possibilities of when values are *written* to the y_j registers along a path P_x, and *which* values are written. This is on top of Ajtai's idea of joining up pieces of paths between endpoints of successive intervals of $\mathcal{I}$: we may splice such pieces only when the written y_j values also agree. What we count in addition are all the different possibilities for functions g_0 to apply at timesteps t via $g_0(t) = \mathtt{wvals}(\mathtt{state}(t, x), P_x)$. Note that given a set R of states along the path P_x we can also define $g(R) = \{\, \mathtt{wvals}(u, P_x) : u \in R\,\}$. Owing to the write-once condition, these sets must "telescope" nicely (but in fact, our counting here does not need this).

Lemma 4.3 (cf. Lemma 6 of [Ajt99a], p13) *For any k there exist $\sigma, \epsilon > 0$ such that for all large enough n and BPs M of size at most $2^{\epsilon n \log n}$ and depth kn, with a partition $\mathcal{I}$ of short time intervals defined as above: there are a set H of inputs, sets $W_1, W_2 \subseteq \{1, \cdots, n\}$, a value $v \in \{0, 1\}$, $J_1, J_2 \subseteq \mathcal{I}$, and functions f_1, f_2, g_1, g_2 so that for all $x \in H$ we have:*

1. $|H| \geq 2^{-9k\epsilon n \log n} \alpha^{(1-\theta)n}$*, where θ can be made arbitrarily small by choosing σ.*
2. $W_1, W_2 \in \Gamma_x$
3. $J_i = \mathtt{set}(W_i, x)$ *for* $i = 1, 2$
4. *If $T_i = \cup_{I \in J_i} I$ then $f_i = \mathtt{rstates}(T_i, x)$ and $g_i = \mathtt{wvals}(f_i, x)$.*
5. $J_1 \cap J_2 = \emptyset$ *and* $W_1 \cap W_2 = \emptyset$
6. $\mathtt{out}_{kn}(x) = v.$

Proof. As in [Ajt99a], since M outputs either 1 or 0, we start with H' chosen to be the larger of the two sets $M^{-1}(0)$ or $M^{-1}(1)$. Assume that $x \in H'$ is fixed. By the previous lemma there exists $W_1, W_2 \in \Gamma_x$ such that $W_1 \cap W_2 = \emptyset$. Let $J_i = W_i, T_i = \cup_{I \in J_i} I$, $f_i = \mathtt{rstates}(T_i, x)$ and $g_i = \mathtt{wvals}(f_i, x)$. We will upper bound the number of possible choices for the sequence $\langle W_1, W_2, J_1, J_2, f_1, f_2, g_1, g_2 \rangle$.

The number of choices for the pair W_1, W_2 is at most 2^{2n}. The number of choices for the pair J_1, J_2 is at most $|\mathcal{I}|^{4k}$. Since $|\mathcal{I}| \leq \sigma^{-1} k$ this gives an upper bound of $\sigma^{-4k} k^{4k}$. The functions f_i have domains that have at most $2k$ elements and ranges of size at most $2^{\epsilon n \log n}$, so the number of choices for the pair f_1, f_2 is at most $2^{4k\epsilon n \log n}$.

Finally, the g_i have domains of size $|T_i| \leq 2k\sigma n$ and range $(\{1, \cdots, m\} \cup \{\star\}) \times \Sigma$. That is, at each time step in T_i we write at most one value. Even allowing for rewrites, the number of possible such functions is $\leq (m\alpha)^{4k\sigma n}$. Let $m = r\alpha$; then the above bound is $\leq 2^{(\log r)4k\sigma n}\alpha^{8k\sigma n}$. Thus there are at least $2^{-9k\epsilon n \log n}\alpha^{(1-8k\sigma)n}$ inputs for which this sequence is fixed. Picking H to be this set completes the proof. □

Lemmas 7 and 8 in [Ajt99a] are pure counting lemmas again that do not reference the operation of branching programs and connect the counting estimates in the conclusions of Lemma 6 with the hypotheses of Lemma 4. Looking at the difference between the above lemma and Ajtai's Lemma 6, the conclusions are the same except for the constant "9" and the extra $(1 - \theta)$ in the exponents, with $\theta = 8k\sigma$. However, the "9" can be absorbed into the freedom to choose ϵ, and the $(1 - \theta)$ into the freedom to choose any $\alpha > n^c$ granted in [Ajt99a]. Hence we obtain conclusions of equivalent strength, and they can be plugged in to complete the proof of the main theorem exactly as in pages 14–15 of [Ajt99a]. □

5 Conclusions and Open Problems

We have shown that Ajtai's size requirement for linear-time branching programs solving his Hamming-distance problem carries over to a model that is given much extra space "for free." Since $TS = \Omega(n^2)$ or even $TS = \Omega(n \log^3 n)$ tradeoffs are impossible in this extended model, we argue that the extensions of Ajtai's arguments used in [BSSV00] to push T near $n \log^{.5} n$ are getting close to a limit for such extensions. However, this may also point the way to arguments that do break under the extensions, and which may exploit the "slack" we have shown for stronger results.

The most immediate open problem is to get the element-distinctness result in the second half of [Ajt99a] to carry over. This is the pre-requisite for carrying over the 2-way branching program results of [Ajt99b]. The crux of our current state of this is in the difference between $\mathtt{core}(T, x)$ and $\mathtt{Icore}(T, x)$. We believe we can fairly straightforwardly modify the more-delicate counting lemmas in the second half of [Ajt99a], especially the pivotal Lemma 13 on page 26, to work when "$\mathtt{core}(T, x)$" includes the registers $y_1, \ldots, y_m$. However, a lower bound on the size of $\mathtt{core}(T, x)$ then does not guarantee that a large-enough number of *input* registers are included in this core. One needs the large number of input registers to obtain enough places to modify inputs x. (*Addendum 6/6/01:* It now appears that we can solve this by working directly with `Icore` and ignoring the concept of core for y_j registers.)

We also advance the concept of a WORM-BP as worthy of study in its own right. Which time-space tradeoffs hold for WORM-BPs? Insofar as WORM-BPs can be subsumed by nondeterministic BPs, bounds for those carry over, but we look for bounds that exploit the fact that the write-once registers are being written deterministically.

References

[Ajt99a] M. Ajtai. Determinism versus non-determinism for linear time RAMs with memory restrictions. Technical Report ECCC TR98-077, revision 1, Electronic Colloquium in Computational Complexity, January 1999. A preliminary version appeared in the proceedings of the 31st ACM STOC, 1999.

[Ajt99b] M. Ajtai. A non-linear time lower bound for Boolean branching programs. Technical Report ECCC TR99-026, Electronic Colloquium in Computational Complexity, July 1999. A preliminary version appeared in the proceedings of the 40th IEEE FOCS, 1999.

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

[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.

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

[BSSV00] P. Beame, M. Saks, X. Sun, and E. Vee. Super-linear time-space tradeoff lower bounds for randomized computation. Technical Report ECCC TR00-025, Electronic Colloquium in Computational Complexity, May 2000.

[BST98] P. Beame, M. Saks, and J. Thathatchar. Time-space tradeoffs for branching programs. Technical Report ECCC TR98-053, Electronic Colloquium in Computational Complexity, September 1998. Corrected version. Also appeared in the proceedings of the 40th IEEE FOCS, 1998, pages 254–263.

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

[INR92] S. Irani, M. Naor, and R. Rubinfeld. On the time and space complexity of computation using write-once memories, or Is pen really much worse than pencil? *Math. Sys. Thy.*, 25:141–159, 1992.

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

[PF79] N. Pippenger and M. Fischer. Relations among complexity measures. *J. Assn. Comp. Mach.*, 26:361–381, 1979.