Improved Simulation of Nondeterministic Turing Machines

Subrahmanyam Kalyanasundaram[a,*], Richard J. Lipton[a], Kenneth W. Regan[b,*], Farbod Shokrieh[c]

[a] *College of Computing, Georgia Tech, Atlanta, GA 30332*
[b] *Department of Computer Science and Engg., University at Buffalo, Buffalo, NY, 14260*
[c] *School of Electrical and Computer Engg., Georgia Tech, Atlanta, GA 30332*

Abstract

The standard simulation of a nondeterministic Turing machine (NTM) by a deterministic one essentially searches a large bounded-degree graph whose size is exponential in the running time of the NTM. The graph is the natural one defined by the configurations of the NTM. All methods in the literature have required time linear in the size S of this graph. This paper presents a new simulation method that runs in time $\tilde{O}(\sqrt{S})$. The search savings exploit the one-dimensional nature of Turing machine tapes. In addition, we remove most of the time-dependence on nondeterministic choice of states and tape head movements.

Keywords: Turing machines, nondeterminism, determinism, simulation, complexity, algorithms.

1. Introduction

How fast can we deterministically simulate a nondeterministic Turing machine (NTM)? This is one of the fundamental problems in theoretical computer science. Of course, the famous P $\neq$ NP conjecture, as most believe, would answer that we cannot hope to simulate nondeterministic Turing machines very fast. However, the best known result to date is the famous theorem of Paul, Pippenger, Szemerédi, and Trotter [13] that NTIME($O(n)$) is not contained in DTIME($o((n \log^* n)^{1/4})$). This is a beautiful result, but it is a long way from the current belief that the deterministic simulation of a nondeterministic Turing machine *should* in general take exponential time.

We look at NTM simulations from the opposite end: rather than seeking better lower bounds, we ask how far can one improve the upper bound? We suspect even the following could be true:

*Corresponding Author

Email addresses: subruk@cc.gatech.edu (Subrahmanyam Kalyanasundaram), rjl@cc.gatech.edu (Richard J. Lipton), regan@buffalo.edu (Kenneth W. Regan), farbod@ece.gatech.edu (Farbod Shokrieh)

For any $\varepsilon > 0$,

$$\mathrm{NTIME}(t(n)) \subseteq \mathrm{DTIME}(2^{\varepsilon t(n)}).$$

To our knowledge, this does not contradict any of the current strongly held beliefs. This interesting question has been raised before, see e.g., [4].

Our main theorem is:

Theorem 1. *Any k-tape NTM N with tape alphabet size a that runs in time $t(n)$, can be simulated by a deterministic Turing machine in time*

$$a^{kt(n)/2} \cdot \eta_N^{\sqrt{t(n)\log t(n)}},$$

up to polynomial factors, and where η_N is a constant that depends only on a and k.

Our bound has two key improvements. First, all nondeterminism arising from the choice of the next state or tape head movements is subsumed into the factor $\eta_N^{\sqrt{t(n)\log t(n)}}$ with much smaller time dependence, compared to the main exponential term. Second, while N may write any of $S = a^{kt(n)}$ strings nondeterministically on its k tapes, our simulator needs to search only $\sqrt{S}$ of that space. Thus, we search the NTM graph in the *square-root* of its size.

There is no general deterministic procedure that can search a graph of size S in $\sqrt{S}$ time, even if the graph has a simple description. Hence to prove our theorem we must use the special structure of the graph: we must use that the graph arises from an NTM. We use several simple properties of the operation of Turing tapes and the behavior of guessing to reduce the search time by the square root.

We believe that while the actual theorem is interesting, the techniques that are used to prove the theorem may be of use in other problems. We speculate that our methods may be extended to lower the exponent further.

In section 5, we consider NTMs with limited nondeterminism, and prove:

Theorem 2. *Suppose $t(n) = nr(n)$, where $r(n)$ is constructible in unary in $O(n)$ time, and fix an input alphabet of size b. Then for any NTM N that runs in time $t(n)$ with $o(n)$ nondeterminism and computes a function f, there exist circuits C_n of size $O(t(n)\log r(n))$ that compute f correctly on $b^{n-o(n)}$ inputs.*

2. Model & Problem Statement

We use a standard model of a nondeterministic multitape Turing machine, in which nondeterminism may arise through characters written, head motions on the tapes, and/or the choice of next state. Heads may stay stationary as well as move one cell left or right in any step. We stipulate that an NTM N runs in time $t(n)$ if all branches of computations on inputs of length n halt within $t(n)$ steps. Since our results involve bounds $t(n)$ that are fully time and space

constructible, this is equivalent to definitions that apply the time bound only to accepting computations. Throughout this paper, we use q for the number of states, k for the number of tapes, and a for the alphabet size of N. We also use the shorter notation $t = t(n)$ for simplicity. Our results hold for all sufficiently large input lengths n.

Our question is, in terms of a, k, q, what is the most efficient simulation of N by a deterministic Turing machine (DTM)? We identify three basic strategies:

1. *Tracing the computation tree:* Since we do not limit N to be binary-branching, individual nodes of the tree may have degree as high as $v = a^k 3^k q$, where the "3" allows each head on each tape to move left, right, or stationary. This is reflected in classic theorem statements of the form

   Theorem 3. *Any NTM N with time complexity $t(n)$ can be simulated by a DTM M in time $c(N)^{t(n)}$, where $c(N)$ is a constant depending on N.*

   According to proofs such as that in [12], $c(N)$ depends on q as well as k and a. There is thus a factor q^t in the running time of M. Since q, the number of states, is not a tangible feature of the NTM and since it can be quite large, it will be our goal to eliminate such a factor.
2. *Enumerating a witness predicate:* That is, one finds a predicate $R(x, y)$ that is deterministically efficient to decide, such that for all x, N accepts x iff for some y of a given length, $R(x, y)$ holds. Then one tries all possible y. This may be specific to the language $L(N)$ rather than simulate N in the direct sense of strategy 1. However, when $R(x, y)$ is the predicate "y codes an accepting path in the computation tree" it is the same as strategy 1.
3. *Searching the configuration graph:* A *configuration* of a Turing machine is an encoding of the current state, the non-blank contents of the tapes, and current position of the tape heads. Configurations form a directed graph where there are directed edges from a configuration to a valid successor configuration, with sources being the initial configurations I_x on given inputs x and sinks being accepting configurations I_a (perhaps including non-accepting halting configurations too). When N uses at most space s on any one tape, the number S of nodes in the graph (below I_x) is at most

   $$S = a^{ks} s^k q.$$

   Notice that $s \leq t$ holds trivially, where t is the running time of N. Using a look up table for simulating the transition function of the machine N, the dominant term in the running time is

   $$O(Sv \cdot \log(Sv) \cdot \log S) = (3at)^k a^{kt} q^2 \mathsf{poly}(\log q, k, t, a).$$

   Where $v = a^k 3^k q$ is the maximum possible degree of the computation tree, as defined in strategy 1. Note that the dependence on q is at most q^2, not q^t.

The classic tradeoff between strategy 1 and strategy 3 concerns the space requirement. Tracing the tree requires storing only the current path from the root and some local information, though it may waste time by re-computing nodes that are reached by multiple paths when the computation is treated as a graph. Breadth-first search of the graph avoids redundant expansion at the expense of storing the whole list of visited nodes. In this paper we find that by judicious mixing-in of strategy 2, there is also mileage to be gained on the running time alone. The following preliminary result illustrates the basic idea:

Proposition 4. *Any NTM N with time complexity $t(n)$ can be simulated by a DTM M in time $c(N)^{t(n)}$, where the constant $c(N)$ depends on the alphabet size a and the number of tapes k of N, but is* ***independent*** *of q.*

Proof. We define a *weak trace* as comprising the move labels on an accepting path in the computation tree, but *omitting* the next-state information. There are only $(a^k 3^k)^t$ such potential witnesses to enumerate. We call a path "compatible with the weak trace y" if it adds states $q_0, \ldots, q_t$ to the parts specified by y to make a legal computation. Below, we show that each of these weak traces can be verified in time $a^{2k}3^k q^2 \mathsf{poly}(\log q, k, t, a)$.

For each step j in the computation, define Q_j to be the set of states N can be in at step j on some full path that is compatible with y. Initially $Q_0 = \{ q_0 \}$, the start state of N. Given Q_{j-1}, to compute Q_j we take each state r in Q_{j-1} and look up all possible next states r' in a pre-computed lookup table based on the transition relation of N. After computing each Q_j, M needs to sort and remove the duplicate states in Q_j, else the size could explode by the end of the simulation. The simulation finally accepts if and only if Q_t contains the accepting state q_a, which we may suppose persists for each step once it is entered.

Our deterministic machine M has $k+3$ tapes, k to re-create the tapes of N guided by the weak trace, one to code the transition function of N serially as a lookup table, plus two for bookkeeping. The lookup table rows are indexed by the current state, the k symbols currently read, the k symbols that would be written, and k directions (left, right or stay) in which the tape heads shall move. The entries give all possible next-states for N in such a transition. The lookup table is stored in a serial fashion in a single tape. There are $q(3a^2)^k$ rows, and each row can have at most q states. The cost of a serial lookup is upper-bounded[1] by $O\Big(q(3a^2)^k \cdot [k \log(3a^2) + \log q + q \log q]\Big)$.

After the lookups, we need to sort and remove duplicates from a set (of states) which could be potentially q^2 in size. This takes $O(q^2 \log q)$ comparisons by standard sorting algorithms, where each comparison costs $\log q$, yielding a

[1] One can remove the $q \log q$ inside the brackets by organizing the rows in canonical order of the subsets of states they produce, and having M count special aliased dividers up to 2^q in binary as it scans serially, to determine which subset goes with a given row. A final $q \log q$ outside the brackets can be for writing out the indexed subset as a list of states. However, this extra efficiency does not matter to our results.

running time of $q^2 \log^2 q$. Multiplying the whole expression by t, we get that the running time per weak trace is

$$O\Big([q(3a^2)^k \cdot [k \log(3a^2) + \log q + q \log q] + q^2 \log^2 q] \cdot t\Big),$$

which can be upper bounded by

$$h(a, q, k, t) = a^{2k} 3^k q^2 \mathsf{poly}(\log q, k, t, a).$$

The overall running time is $(3^k a^k)^t$ multiplied by the function h. The factor h is majorized by $(1 + \delta)^t$ for any $\delta > 0$ as t becomes sufficiently large. The whole time is thus bounded by $(3^k a^k + \delta')^t$, where $\delta' = 3^k a^k \delta$. Note that δ' is independent of q and can likewise be made arbitrarily small when a and k are fixed. Hence the deterministic simulation time is asymptotically bounded by $c(N)^{t(n)}$ where $c(N)$ is independent of q.

Finally, we observe that M does not need to know t beforehand. M can start simulating N using weak traces of length $1, 2, 3, \ldots$ till it observes that all computation branches halt. M would stop at t and this only adds a polynomial overhead. □

Our further improvements come from (a) slimming witnesses y further, (b) more-sophisticated precomputation, and (c) trading off strategies 1 and 3 according to the space actually used by N on the one-dimensional Turing tapes.

3. Block-Trace Simulation

We begin the push for faster simulations by breaking computations by NTMs N into "blocks" of d steps, where d will be specified later.

Definition 1. A *segment of size* d for a k-tape NTM N with alphabet of size a is a sequence of 4-tuples

$$\tau = [(r_1, f_1, \ell_1, u_1), \ldots, (r_k, f_k, \ell_k, u_k)]$$

where for each tape j, $1 \le j \le k$:

- $r_j \in \{\, 0, \ldots, d \,\}$ stands for the maximum number of cells to the right of its starting position the tape head will ever be over during the next d steps,
- $f_j \in \{\, 0, \ldots, d - r_j \,\}$ is the number of cells left of the position of r_j that the tape head ends up after the d-th step, and
- $\ell_j \in \{\, 1, \ldots, d \,\}$ is the number of distinct cells that shall be accessed during the next d steps on tape j. For a given r_j and f_j we have the bound $\ell_j \le d - \min\{\, r_j, f_j \,\}$.
- u_j is a string of length ℓ_j, which is interpreted as the final contents of those cells.

Technically ℓ_j can always be set to the stated bound, but we keep it separate for clarity.

Definition 2. A *block trace of block-size* d, for an NTM N, is a sequence of segments of size d.

Definition 3. An accepting full path is *compatible* with a block trace if the latter has $\lceil t/d \rceil$ blocks where t is the total number of steps in the path, and in every block each 4-tuple (r_j, f_j, ℓ_j, u_j) correctly describes the head locations after the corresponding d steps of the full path, *and* every character in u_j is the correct final content of its cell after the d steps.

Our witness predicate now asserts the existence of a block trace y with which some accepting computation is compatible. Clearly every accepting computation gives rise to such a y, so the predicate is correct. The running time of the resulting simulation is a consequence of the following lemmas. Notice that the above definition includes all the possible head movements of N over the next d steps.

Lemma 5. *The number B of valid segments is at most $(32a^d)^k$. Hence the number of potential block trace witnesses is at most $B^{\lceil t/d \rceil} = a^{kt}32^{k\lceil t/d \rceil}$.*

Proof. We first bound the number of 4-tuples per each tape. We note that for ℓ cells affected for a particular segment, there are a^ℓ possible strings u. We sum over all the possible values of ℓ – ranging from d to 1. Direct calculation gives us that for $\ell = d$, there are at most 6 possible sets of (r, f), for $\ell = d - 1$ at most 14, etc. An upper bound for the number of possible sets for $\ell = d + 1 - i$ is given by $6, 14, 24, \ldots$ for $i = 1, 2, 3, \ldots$. This can be simply written as $i^2 + 5i$. A total number of distinct 4-tuples is upper bounded by

$$\sum_{\ell=d}^{1}[(d+1-\ell)^2 + 5(d+1-\ell)]a^\ell = a^d \cdot \sum_{i=1}^{d}(i^2+5i)/a^{i-1} \leq 32a^d$$

where the last inequality follows by the worst case value $a = 2$. Since we have k tapes, we obtain $B \leq (32a^d)^k$. (In fact, we can get $B \leq (C_a a^d)^k$ where $C_a \longrightarrow 6$ as $a \longrightarrow \infty$, but this tighter counting does not result in any notable improvement in the eventual simulation.) □

Lemma 6. *Whether there is an accepting computation that is compatible with a given block trace witness can be decided by a deterministic Turing machine in time $a^{3kd}q^2\mathsf{poly}(\log q, k, t, a, d)$.*

Proof. We generalize the ideas in Proposition 4, we use a similar strategy to check a block trace witness using a $(k+3)$-tape deterministic machine. We are given a block trace witness, i.e., $\lceil t/d \rceil$ segments of size d each. The idea is to maintain the set Q_i of states that N on input x can possibly be in, this time after the i-th segment of d steps in some computation path. We precompute a lookup table T_d whose values are sets of states, and whose rows are indexed by the following information:

- An initial state p entering the segment of d steps.
- Strings w_j of length at most $2d-1$ indicating the true contents in the cells surrounding the head on tape j. The cases where a segment of cells on the right or left are blank (through never having been visited before) are handled by adjoining integers b_j indicating such cells.
- The string u_j and integers r_j, f_j for each tape j, representing a segment in a block trace.

The lookup table is the d-length segment equivalent of the lookup table in Proposition 4. There are $qa^{(3d-1)k}d^{2k}$ rows of the table, the length of each index in binary being thus asymptotic to $\log_2 q + (3d-1)k\log_2 a + 2k\log_2 d$. The cost of each lookup is thus upper bounded by $O\Big(qa^{3kd}d^2(\log q + 3kd\log a + 2k\log d + q\log q)\Big)$. The last $q\log q$ term in the above expression is because of the fact that there are at most q states that each index can have, each of which requires $\log q$ storage. By including the time for sorting the states, and multiplying by the running time of $\lceil t/d\rceil$ segments, we get

$$O\Big([qa^{3kd}d^2(\log q + 3kd\log a + 2k\log d + q\log q) + q^2\log^2 q]\cdot t/d\Big).$$

which is upper bounded by

$$a^{3kd}q^2\mathsf{poly}(\log q, k, t, a, d).$$

□

Theorem 7. *A nondeterministic k-tape TM with q states and alphabet size a can be simulated by a multi-tape deterministic TM in time*

$$a^{kt}C_N^{\sqrt{t}}\cdot q^2\mathsf{poly}(\log q, k, t, a),$$

where C_N is a constant that depends only on a and k.

Proof. This follows from Lemmas 5 and 6. We have a $(k+3)$-tape simulator machine that tries out all the possible block witnesses, with a running time

$$a^{kt+3kd}32^{k\lceil t/d\rceil}q^2\mathsf{poly}(\log q, k, t, a, d).$$

The two factors in the above expression that depend on d in a big way are a^{3kd} and $32^{k\lceil t/d\rceil}$. We can choose d to be such that these the product of these two factors are minimized. Direct calculation gives us that this happens when $d = \sqrt{5t/(3\log_2 a)}$. Setting $C_N = 2^{2k\sqrt{15\log_2 a}}$, we get a running time of

$$a^{kt}C_N^{\sqrt{t}}\cdot q^2\mathsf{poly}(\log q, k, t, a).$$

Like in the proof of Proposition 4, note that simulator machine does not need to know t beforehand. □

4. Main Theorem

We have seen two simulations of an NTM where the dominant term in the running time is a^{kt}. One is strategy 3, searching the configuration graph, discussed in Section 2, with a running time of $(3at)^k a^{kt} q^2 \mathsf{poly}(\log q, k, t, a)$. The other is the block trace method, with a running time of $a^{kt} C_N^{\sqrt{t}} \cdot q^2 \mathsf{poly}(\log q, k, t, a)$. Even though the time bounds seem similar, the approaches are quite different – a difference that we shall exploit in this section.

Our goal in this section is to reduce the exponent of the simulation time by half. In the graph search method, the dominating part in the running time is caused by the number of configurations. There are at most $a^{kt} t^k q$ of them. If the NTM used only a tape space of $kt/2$ over all the k tapes, then the dominating part in counting the number of configurations would have reduced. We have only a maximum possible $a^{kt/2}$ combinations of tape contents. This would lead to a simulation which requires $(3at)^k a^{kt/2} q^2 \mathsf{poly}(\log q, k, t, a)$ time.

But of course, not all NTM simulations use less than $kt/2$ tape space. Here we will use the block trace method to exploit an interesting property of the Turing machines. We make the following observation: the last time we visit a location in the NTM tape, we need not write any character there. This is because the tape head is not going to return to that position. If the NTM visits at least $kt/2$ locations on all tapes together, then there are at least $kt/2$ locations visited for a last time. Now, when we consider block traces, we do not need to have a symbol to write down, if we are visiting a tape location for a last time. We could potentially save on a factor of $a^{kt/2}$ on the running time. This brings down the main factor in the running time in Theorem 7 to $a^{kt/2}$ as well.

For the final theorem, we need one more definition.

Definition 4. A *directional segment of size d* for a k-tape NTM N with alphabet size a is a segment of size d, omitting the strings u_j, that is

$$\tau = [(r_1, f_1, \ell_1), \ldots, (r_k, f_k, \ell_k)]$$

where r_j, f_j, ℓ_j are defined as in Definition 1.

A *directional trace* of block size d, is a sequence of directional segments of size d.

Lemma 8. *The number of directional segments of block size d is upper bounded by d^{3k}, for $d \geq 6$. The number of potential directional trace witnesses is at most $(d^{3k})^{\lceil t/d \rceil}$.*

Proof. The calculations are similar to those in the proof of Lemma 5. The difference here is that we do not need to count the number of possible strings u for each tape. This bounds the number of directional segments to $\sum_{i=1}^{d}(i^2 + 5i) = \frac{1}{3}d(d+1)(d+8) \leq d^3$ per tape, for $d \geq 6$. Since we have k tapes, the bound is d^{3k}. The bound on directional traces follows. □

We are now ready to prove the main theorem.

Theorem 1 (Restated). *A nondeterministic k-tape TM N with q states and alphabet size a can be simulated by a multi-tape deterministic TM in time*

$$a^{kt(n)/2}\eta_N^{\sqrt{t(n)}\log t(n)} \cdot q^2\mathsf{poly}(\log q, k, t(n), a),$$

where $t(n)$ is the running time of N and η_N is a constant that depends only on a and k.

PROOF. We assume that we know an upper bound $t = t(n)$ as a function of the input length n. (If not, one can run the simulations for $t = 1, 2, 3, \cdots$, and this will introduce a multiplicative factor $t(t-1)/2$, which is polynomial in t anyway.)

The simulation consists of three parts. First, preprocessing the directional traces. Second, running the block trace simulation for those traces which have tape usage $\geq kt/2$. And third, running the graph search simulation restricting the tape usage to $kt/2$.

1. In the preprocessing stage, the simulator lists down all the possible directional traces. The value of $d = \sqrt{5t/(3\log_2 a)}$ as optimized in Theorem 7 is used. For t large enough such that $d \geq 6$, there are $d^{3k\lceil t/d\rceil}$ such traces by Lemma 8. We get that the number of traces is $(\sqrt{t})^{O(\sqrt{t})}$ or $\eta_N^{\sqrt{t}\log t}$, where η_N is a constant that depends on only a and k.
   Using the directional trace, the simulator calculates the total tape usage of N. In particular, the simulator decides if the total tape usage is $\leq kt/2$ or $\geq kt/2$. The simulator also calculates the time of the last visit to each of the tape locations. This data is stored in a lookup table, which is stored in another tape of the simulator. All of the above operations can be performed in time $\mathsf{poly}(k, t)$ per directional trace.
2. If the total tape usage is $\geq kt/2$ for a given directional trace, the block trace simulation is performed. All the block traces which match the (r, f, ℓ) parts of the directional trace are generated—with a twist. For those time instances for which the tape head is visiting the location for the last time, the block trace is generated with a $\llcorner$ character in the corresponding location. The preprocessed data from the directional traces would be used to determine if the location is being visited for the last time or not.
   There are at least $kt/2$ locations visited for the last time, so the number of block traces that correspond to a given directional trace is $\leq a^{kt/2}$. So the total number of relevant block traces here is upper bounded by $\eta_N^{\sqrt{t}\log t}a^{kt/2}$.
   The running time in the Lemma 6 holds essentially by the following observation. The lookup table could be expanded (slightly) to accommodate one more symbol in the alphabet, the '$\llcorner$' symbol. The set of states that are possible in the lookup table after a doing block trace move with a $\llcorner$ are the union of the set possible states after a move with the block trace with one of the original a characters in place of the $\llcorner$.
   The running time contribution of this stage is $a^{kt/2}\eta_N^{\sqrt{t}\log t} \cdot q^2\mathsf{poly}(\log q, k, t, a)$.

3. For the cases when the total tape usage is $\leq kt/2$, the directional trace is discarded. For all such cases combined, one call to the graph search simulation is enough. The simulator needs to keep track of the configurations, and reject a branch as soon as the tape usage exceeds $kt/2$. This gives a running time of $a^{kt/2}(3at)^k q^2 \mathsf{poly}(\log q, k, t, a)$.

The theorem follows by observing that if the NTM has an accepting computation path, at least one of the two simulations, the block trace, or the graph search method would yield an accepting path. Also note that like in Theorem 7, we need a $(k+3)$-tape deterministic Turing machine to perform the directional trace simulation. The graph search simulation can be performed with a machine with a constant (independent of k) number of tapes, so the whole simulator needs $k+3$ tapes. The running time is

$$T(n) = a^{kt/2}\eta_N^{\sqrt{t}\log t} \cdot q^2 \mathsf{poly}(\log q, k, t, a).$$

□

4.1. Uniform Simulation

In this section we illustrate that a similar bound applies in a uniform simulation, meaning a single DTM that takes an NTM N and its input x as arguments.

Theorem 9. *We can construct a deterministic two-tape Turing machine that given any k-tape NTM N running in time $t(n)$ and binary string x of length n as input, simulates $t(n)$ steps of $N(x)$ in time*

$$a^{kt(n)/2}\eta_N^{\sqrt{t(n)}\log t(n)} \cdot q^4 \mathsf{poly}(\log q, k, t(n), a), \tag{1}$$

where a is the alphabet size of N, q is the number of states of N, and η_N is a constant that depends only on a and k.

Proof. We first extend Theorem 1 to show that we can build a $k+3$-tape deterministic TM M, which takes in the description of the NTM N and an input string x as input, and performs a uniform simulation in the following running time as in Theorem 1.

$$t'(n) = a^{kt/2}\eta_N^{\sqrt{t}\log t} \cdot q^2 \mathsf{poly}(\log q, k, t, a).$$

To build M we need to show how to perform each action in Theorem 1 with a universal TM. We go through the three parts, as listed in the proof of Theorem 1, and explain how each of the parts can be performed. Like in Theorem 1 we assume that we know an upper bound $t = t(n)$ as a function of the input length n.

1. The preprocessing stage is a set of calculations which are independent of the machine N. This can be performed with no knowledge of the transition function of N.

2. The block trace simulation requires the lookup table T_d which provides the successor state to each state as in Lemma 6. Once the DTM has this table, it can perform the simulation. This can be computed from the description of the NTM N, which contains the description of the transition function δ of N.
3. Here we need to perform the graph search simulation. We require the ability to compute the configuration(s) which are successor(s) to a given configuration. This too can be computed from the description of the transition function of the NTM N.

Notice that the running time remains the same as that in Theorem 1, up to a polynomial factor. The number of tapes that is required is $k+3$ as in Theorem 1, we need k tapes to recreate the tapes of N, one store the lookup table and two for other computations.

Now we apply the Hennie-Stearns construction [5] to M to obtain the required 2-tape DTM which simulates M. Here the second tape serves only to copy "blocks" on the first tape to adjacent locations. The 2-tape TM thus runs in time at worst $O(t'(\log t' + |M|))$. Using the the above expression for t', we get that the running time is at most

$$t' \cdot O(kt/2 \log a + \sqrt{t} \log t \log \eta_N + \text{lower terms}) + t' \cdot |M|$$

where $|M|$ is the program size of M. It is $O(t'(\log t' + |M|))$ not $O(t' \log t' \cdot |M|)$ because the part of the second tape storing the program needs to be consulted only once for each step simulated. The multiplier inside the $O(\ldots)$ is absorbed into the poly term of (1), so we are left only to bound and absorb the term $t' \cdot |M|$, The proof of Theorem 1 constructs the program size $|M|$ of M to be $O(|N| + kt \log \eta_N)$ plus lower-order terms. The multiplier $kt \log \eta_N$ of t' is likewise absorbed into the poly term, leaving just $|N| \approx a^{2k} 3^k q^2$ to deal with. The first part converts the multiplier q^2 into q^4, while the rest can be absorbed into the $\eta_N^{\sqrt{t(n)} \log t(n)}$ term, by increasing η_N slightly. □

5. Sub-linear nondeterminism and small circuits

Now we consider NTMs N that have $o(n)$ nondeterministic steps in any computation path on inputs of length n, where the inputs are over an alphabet Σ of size b. For each n, it follows that some nondeterministic choice string α_n is used for a set of at least $b^{n-o(n)}$ strings. When N is a language acceptor, the computation on α_n also gives the correct answer for all rejected strings, so we add them when defining S to be the set of inputs on which N-with-α_n works correctly. When N computes a partial multi-valued function f, S includes all strings not in the domain of f, and for all other $x \in S$, N with α_n outputs a legal value of $f(x)$. We can hard-wire α_n into deterministic circuits C_n that work correctly on S. The main theorem of [16] gives C_n size $O(t(n) \log t(n))$. We show that for $t(n)$ near linear time we can improve the size of C_n considerably.

Theorem 2 (Restated). *Suppose $t(n) = nr(n)$, where $r(n)$ is constructible in unary in $O(n)$ time, and fix an input alphabet of size b. Then for any NTM N that runs in time $t(n)$ with $o(n)$ nondeterminism and computes a function f, there exist circuits C_n of size $O(t(n) \log r(n))$ that compute f correctly on $b^{n-o(n)}$ inputs.*

The size improves on [16] when $r(n) = n^{o(1)}$. When $r(n) = (\log n)^{O(1)}$, meaning $t(n)$ is *quasi-linear* time, this reduces the size of C_n to $t(n) \log \log t(n)$. When $t(n) = O(n)$, this says we can reduce the overhead to any constructible slow-growing unbounded function, in a sense getting the circuit size as close to linear as desired. Of course the circuits C_n work only on a sizable fraction of the inputs—on other $x \in dom(f)$ they may incorrectly fail to output a legal value.

The proof employs Wolfgang Paul's notion of a *block respecting* Turing Machine, from his paper with Hopcroft and Valiant [6] separating time and space. The result of [6] were later extended to multi-dimensional and tree-structured tapes in [15] and [14]. The notion of block-respecting Turing machines has been used a number of times to prove other results, e.g. in [10]. We refer the reader to [17] for a discussion on the results of [6].

Proof. Given n, take $B = r(n)^2$. Let the Turing machine N computing f have k tapes, and regard those tapes as segmented into "blocks" of length B. By the block-respecting construction in [6], we can modify N into N' computing f in time $t'(n) = O(t(n))$ such that on all inputs of length n, all tape heads of $N'(x)$ cross a block boundary only at time-steps that are multiples of B.

For all length-n strings x, and nondeterministic choice strings α_n, we define the "block-respecting graph" $G_{x,\alpha}$ to have vertices $V_{\ell,i}$ standing for the ith block on tape ℓ, and W_j for $0 \leq j < t'(n)/B$—note also $i < t'(n)/B$ since N' runs in $t'(n)$ space. We use the notation $i(j, \ell)$ to denote the block that N is on the ℓth tape, during the time block from $(j-1)B$ to jB. For all time steps jB, if the heads before that step were in blocks $V_{\ell,i(j-1,\ell)}$ and are in blocks $V_{\ell,i(j,\ell)}$ afterward, then $G_{x,\alpha}$ has edges from all $V_{\ell,i(j-1,\ell)}$ to W_j and from W_j to the nodes $V_{\ell,i(j,\ell)}$. Because there are at most 3 choices of next-block per tape at any j, there are at most $R(n) = (3^k)^{t'(n)/B}$ different block-respecting graphs. By the choice of B, $R(n) = b^{O(n/r(n))}$. There are also $A(n) = |A|^{o(n)}$-many possible α_n. Hence, by the pigeonhole principle, there is some block-respecting graph G_n that equals G_{x,α_n} for at least $b^n/R(n)A(n) = b^{n-O(n/r(n))-o(n)} = b^{n-o(n)}$-many x's.

Now from G_n we define the circuits g_n as a cascade of $t'(n)/B$-many segments S_j. Each S_j represents a time-B computation whose input x_j is the current contents of the r-many blocks $V_{\ell,i(j,\ell)}$, with output written to those blocks. By the result of [16], S_j needs circuit size only $O(B \log B)$. So the entire circuit has size $O\left(\frac{t'(n)}{B}\right) B \log B = O(t(n) \log r(n))$.

To finish the proof, we note that there are also junctures between segments that represent any cases head on tape crossing a block boundary at time jB. If in fact the head does not cross the boundary, then the juncture generates a null value '$*$', which then propagates through all remaining segments to produce a

rejecting output. The sizes for the junctures are negligible, so the above bound on the size of the circuits holds. □

6. Conclusions

We have shown techniques by which we can search the computation tree of an NTM in time square root of the size of the graph. It would be interesting to see if these techniques can be used to push the running time even lower. Also, it would be interesting to see lower bounds for the problem, i.e., to understand the limitations of determinism as compared to nondeterminism.

6.1. Some related work

The only separation of nondeterministic from deterministic time known is $\mathrm{DTIME}(n) \neq \mathrm{NTIME}(n)$ proved in [13], which is also specific to the multi-tape Turing machine model. It is also known that nondeterministic two-tape machines are more powerful than deterministic one-tape machines [8], and nondeterministic multi-tape machines are more powerful than deterministic multi-tape machines with additional space bound [9]. Limited nondeterminism was analyzed in [4], which showed that achieving it for certain problems implies a general subexponential simulation of nondeterministic computation by deterministic computation. In [20] an unconditional simulation of time-$t(n)$ probabilistic multi-tape Turing machines Turing machines operating in deterministic time $o(2^t)$ is given.

For certain NP-complete problems, improvements over exhaustive search that involve the constant in the exponent were obtained in [19], [18], [1], and [2], while [11] and [7] also found NP-complete problems for which exhaustive search is not the quickest solution. Williams [21] showed that having such improvements in all cases would collapse other complexity classes. Drawing on [20], Williams [21] showed that the exponent in the simulation of NTM by DTM can be reduced by a multiplicative factor smaller than 1. The NTMs there are allowed only the string-writing form of nondeterminism, but may run for more steps; since the factor is not close to 1/2, the result in [21] is incomparable with ours.

Finally there remains the question asked at the beginning: Is

$$\mathrm{NTIME}(t(n)) \subseteq \mathrm{DTIME}(2^{\varepsilon t(n)})$$

for all $\varepsilon > 0$? We have not found any "dire" collapses of complexity classes that would follow from a 'yes' answer, but it would show that nondeterminism is weaker than we think. David Doty [3] showed that there is an oracle relative to which the answer is no. Our techniques do not resolve this question as yet, but may provide new leads.

6.1.1. Acknowledgments.

We thank David Doty, Bart de Keijzer, Ryan Williams, and the anonymous referees for suggestions and helpful comments.

References

[1] Richard Beigel and David Eppstein. 3-coloring in time $O(1.3289^n)$. *J. Algorithms*, 54(2):168–204, 2005.

[2] Andreas Björklund. Determinant sums for undirected hamiltonicity. In *FOCS '10: Proceedings of the 51st annual symposium on Foundations of Computer Science*. IEEE, 2010. to appear.

[3] David Doty. An oracle a such that $\mathrm{NTIME}^A(t(n)) \not\subseteq \mathrm{DTIME}^A(2^{\varepsilon t(n)})$, via Kolmogorov complexity. private communication, 2009.

[4] Uriel Feige and Joe Kilian. On limited versus polynomial nondeterminism. *Chicago J. Theoret. Comput. Sci.*, pages Article 1, approx. 20 pp. (electronic), 1997.

[5] Fred C. Hennie and Richard E. Stearns. Two-tape simulation of multitape turing machines. *J. ACM*, 13(4):533–546, 1966.

[6] John Hopcroft, Wolfgang J. Paul, and Leslie Valiant. On time versus space. *J. Assoc. Comput. Mach.*, 24(2):332–337, 1977.

[7] Alon Itai and Michael Rodeh. Finding a minimum circuit in a graph. *SIAM J. Comput.*, 7(4):413–423, 1978.

[8] Ravi Kannan. Towards separating nondeterministic time from deterministic time. In *Foundations of Computer Science, 1981. SFCS '81. 22nd Annual Symposium on*, pages 235–243, Oct. 1981.

[9] Ravi Kannan. Alternation and the power of nondeterminism. In *STOC '83: Proceedings of the fifteenth annual ACM symposium on Theory of computing*, pages 344–346, New York, NY, USA, 1983. ACM.

[10] Richard J. Lipton and Anastasios Viglas. Non-uniform depth of polynomial time and space simulations. In *Fundamentals of computation theory*, volume 2751 of *Lecture Notes in Comput. Sci.*, pages 311–320. Springer, Berlin, 2003.

[11] Jaroslav Nešetřil and Svatopluk Poljak. On the complexity of the subgraph problem. *Comment. Math. Univ. Carolin.*, 26(2):415–419, 1985.

[12] Christos H. Papadimitriou. *Computational complexity*. Addison-Wesley Publishing Company, Reading, MA, 1994.

[13] Wolfgang J. Paul, Nicholas Pippenger, Endre Szemeredi, and William T. Trotter. On determinism versus non-determinism and related problems. In *Foundations of Computer Science, 1983., 24th Annual Symposium on*, pages 429–438, Nov. 1983.

[14] Wolfgang J. Paul and Rüdiger Reischuk. On time versus space. II. *J. Comput. System Sci.*, 22(3):312–327, 1981. Special issued dedicated to Michael Machtey.

[15] Nicholas Pippenger. Probabilistic simulations (preliminary version). In *STOC '82: Proceedings of the fourteenth annual ACM symposium on Theory of computing*, pages 17–26, New York, NY, USA, 1982. ACM.

[16] Nicholas Pippenger and Michael J. Fischer. Relations among complexity measures. *J. Assoc. Comput. Mach.*, 26(2):361–381, 1979.

[17] Rahul Santhanam. Relationships among time and space complexity classes, 2001.

[18] Richard Schroeppel and Adi Shamir. A $T \cdot S^2 = O(2^n)$ time/space tradeoff for certain NP-complete problems. In *20th Annual Symposium on Foundations of Computer Science (San Juan, Puerto Rico, 1979)*, pages 328–336. IEEE, New York, 1979.

[19] Robert Endre Tarjan and Anthony E. Trojanowski. Finding a maximum independent set. *SIAM J. Comput.*, 6(3):537–546, 1977.

[20] Dieter van Melkebeek and Rahul Santhanam. Holographic proofs and derandomization. *SIAM J. Comput.*, 35(1):59–90 (electronic), 2005.

[21] Ryan Williams. Improving exhaustive search implies superpolynomial lower bounds. In *STOC '10: Proceedings of the fortysecond annual ACM symposium on Theory of computing*, 2010.