Electronic Colloquium on Computational Complexity, Report No. 22 (2001)

On separators, segregators and time versus space

Rahul Santhanam,
Department of Computer Science,
University of Chicago.
E-mail:rahul@cs.uchicago.edu

Abstract

We give the first extension of the result due to Paul, Pippenger, Szemeredi and Trotter [24] that deterministic linear time is distinct from nondeterministic linear time. We show that $NTIME(n\sqrt{log^*(n)}) \neq DTIME(n\sqrt{log^*(n)})$. We show that if the class of multi-pushdown graphs has $(o(n), o(n/log(n)))$ segregators, then $NTIME(nlog(n)) \neq DTIME(nlog(n))$. We also show that atleast one of the following facts holds - (1) $P \neq L$, (2) For all polynomially bounded constructible time bounds t, $NTIME(t) \neq DTIME(t)$.

We consider the problem of whether $NTIME(t)$ is distinct from $NSPACE(t)$ for constructible time bounds t. A pebble game on graphs is defined such that the existence of a "good" strategy for the pebble game on multi-pushdown graphs implies a "good" simulation of nondeterministic time bounded machines by nondeterministic space-bounded machines. It is shown that there exists a "good" strategy for the pebble game on multi-pushdown graphs iff the graphs have sublinear separators.

Finally, we show that nondeterministic time bounded Turing machines can be simulated by Σ_4 machines with an asymptotically smaller time bound, under the assumption that the class of multi-pushdown graphs has sublinear separators.

1 Introduction

$P \stackrel{?}{=} NP$ and $P \stackrel{?}{=} PSPACE$ are two of the central problems of complexity theory. As of now, we lack the techniques to solve these problems. Hence, it is natural to consider restrictions of these problems to specific time and space bounds, and attempt to solve the restricted versions.

ISSN 1433-8092

A natural restriction of $P \stackrel{?}{=} NP$ is $DTIME(t) \stackrel{?}{=} NTIME(t)$ for polynomially bounded t. The major result in this connexion was the separation of nondeterministic linear time and deterministic linear time by Paul, Pippenger, Szemeredi and Trotter [24], building on work by [12] and [25]. They show how to simulate deterministic Turing machines with Σ_4 machines that use less time, and then use a collapse lemma and a hierarchy theorem to prove their result. Gupta [10] showed that the simulation could in fact be done by Σ_2 machines. Unfortunately, these proofs, and specifically the collapse lemma that is common to them, cannot be used to separate $NTIME(t)$ and $DTIME(t)$ for t superlinear. We show how to refine the lemma and use a different hierarchy theorem to improve their result. Actually, we show something more general - a relation between the miminimum size of segregators for multi-pushdown graphs and the size of bounds t for which we can prove $NTIME(t) \neq DTIME(t)$.

Though we cannot prove that $P \neq L$ or that $DTIME(t) \neq NTIME(t)$ for polynomially bounded t, we are able to prove that the disjunction of these two statements holds. Such a result lends support to our intuition that both of these statements are true. Our technique is similar to that of Kannan in [15], where he proved that $P \neq L$ or $DTIME(n) \neq NTIME(n)$. Since we now know that the second of these hypotheses is true, this theorem lacks content for sequential Turing machines. We show how to apply our new collapse lemma and a standard hierarchy theorem to generalize the second hypothesis.

A natural restriction of $P \stackrel{?}{=} PSPACE$ is $DSPACE(t) \stackrel{?}{=} DTIME(t)$. This problem was solved by Hopcroft, Paul and Valiant [12] who showed that $DTIME(t) \subseteq DSPACE(t/logt)$ for all constructible bounds t. The separation follows from this simulation and the well-known hierarchy theorem for deterministic space. For the simulation, [12] defined a pebble game on graphs that models the space requirements of a deterministic time-bounded machine and showed that there is a "good" strategy for the pebble game on directed acyclic graphs of bounded degree, and therefore on the computation graphs of deterministic Turing machines. This pebble game is applicable only to deterministic computations, and the problem $NTIME(t) \stackrel{?}{=} NSPACE(t)$ is still open. We make some progress towards the solution of this problem by defining a new pebble game such that a "good" strategy for this game implies a fast simulation of non-deterministic time by nondeterministic space

and hence a separation, by the nondeterministic space hierarchy of [14]. Unfortunately, we are not able to prove that there exists a "good" strategy for this game on the class of pushdown graphs. Instead, we show that the existence of a "good" strategy is equivalent to a longstanding open problem - the existence of sublinear separators for the class of multi-pushdown graphs. The best results known for the latter problem are due to [9], and show that any upper bound on the size of separators has to be very close to linear. We also show that there exists a fast simulation of nondeterministic time by Σ_4 time under the hypothesis that multi-pushdown graphs have sublinear separators. Thus, if our technique is used to separate $NTIME(t)$ and $NSPACE(t)$, it will also separate $NTIME(t)$ and $\Sigma_4 - TIME(t)$.

An important feature of the techniques we use is that they are non-relativizing. Hence there is a possibility that they can be extended to prove deeper results. By results of Moran([20]), $NTIME(t) \stackrel{?}{=} DTIME(t)$ cannot be decided by relativizing techniques for time-constructible bounds t.

2 Preliminaries

We assume the standard definitions of deterministic and nondeterministic Turing machines and of time and space bounded complexity classes [13]. We also assume the definitions of alternating Turing machines making a bounded number of alternations [2]. A computation of an alternating Turing machine on an input is any correct finite sequence of configurations of the machine beginning with the initial configuration and ending with an accepting or rejecting configuration. The time taken by the computation is the length of the sequence of configurations minus one.

The concept of a computation graph [12] is critical to our investigation. This is defined for any fixed computation of a deterministic or alternating Turing machine. The computation is assumed to be block respecting, without loss of generality, for some block size b (which will depend on the context), i.e. the tapes of the machine are divided into blocks of size b and heads cross block boundaries only at times that are integral multiples of b. If the computation takes time t, the computation graph has t/b vertices numbered $1 \ldots t/b$ representing time intervals of size b, during which no block boundary is crossed. There is an edge from vertex i to vertex j if $j = i + 1$ or if the machine scans some block of some tape during time interval j that was last scanned during time interval i. We say a block is associated with a

vertex i of the graph if it is scanned during time interval i.

We also define the class of multi-pushdown graphs [24], which is relevant to many of our conjectures. A graph $G = (V, E)$ with $\mid V \mid = n$ is in $H_1(n)$ if $V = \{1 \ldots n\}$ and $E = S \cup R$ where $S \subseteq \{(i, i+1) \mid i \leq n-1\}$, every vertex in V has atmost one predecessor in R, and no edges cross in R, i.e if $(i_1, j_1), (i_2, j_2)$ are in R and $i_1 < i_2 < j_1$, then $j_2 \leq j_1$. A graph is in $H_r(n)$ if it is the union of r graphs, each of which is in $H_1(n)$. A family of graphs is in the class of multi-pushdown graphs if there is a fixed r such that each graph in the family of order n is in $H_r(n)$. The importance of this class of graphs is that the family of computation graphs of any alternating Turing machine with a fixed number of tapes is in the class of multi-pushdown graphs. Specifically, if the machine has r tapes, all the computation graphs corresponding to it are in H_{2r}.

Apart from the standard notation for deterministic and alternating time classes, we use L to refer to the class of languages accepted by logarithmic space bounded deterministic Turing machines, SC to denote the class of languages accpeted by deterministic Turing machines using polylogarithmic space and deterministic time and $DTISP(t, s)$ to denote the class of languages accepted by deterministic Turing machines operating in time t and space s. $NTIME(n)$ is referred to as $NLIN$ at times, and $DTIME(n)$ as $DLIN$.

All time and space bounds used in this paper are assumed to have appropriate constructibility properties. The time bounds are assumed to be $\Omega(n)$.

3 Extension of $DLIN \neq NLIN$

One of the key problems of complexity theory is to determine the relationship between deterministic and non-deterministic complexity classes with the same time bound. In their seminal paper [24], Paul, Pippenger, Trotter and Szemeredi proved that deterministic linear time is strictly contained in non-deterministic linear time. Unfortunately, their methods suffice to prove a separation only when the time bound is linear. We extend their techniques to get the slightly stronger result in Theorem 2.5.

[24] actually proves that $DTIME(t)$ is contained in $\Sigma_4(o(t))$. A collapse

lemma is then used to prove that, if $DTIME(n) = NTIME(n)$, then $DTIME(t) = \Sigma_4(t)$ for any time-constructible t. The simulation and the collapse lemma together yield a contradiction to the hierarchy theorem for alternating time classes, on the hypothesis that $DTIME(n) = NTIME(n)$. We modify the translation lemma to make the hypothesis stronger and prove a different hierarchy theorem that yields our result.

For the purposes of this section, we assume that the state set of alternating Turing machines is divided into existential, universal and deterministic states. Any computation of an ATM is considered to be divided into disjoint "phases", which are maximal time intervals during which either the machine is in an existential or deterministic state throughout (existential phase) or it is in an universal or deterministic state throughout(universal phase). We define classes of languages accepted by ATMs that make a bounded number of guesses in all but the last phase.

Definition: A language L is in $\Sigma_k(\Pi_k) - TIGU(t(n), g(n))$ if there is a $\Sigma_k(\Pi_k)$ machine M accepting L that, given an input of length n, operates in time $t(n)$ and makes atmost $g(n)$ guesses in any computation path, excepting the last phase. We will be interested in the classes corresponding to $g(n) = O(n)$, which are called $\Sigma_k(\Pi_k) - TIGU(t(n), LIN)$

We now prove a hierarchy theorem for guess-bounded machines. The technique used is diagonalization, but first we need a tape reduction theorem, which is stated in Lemma 2.1. We use the superscript r in our notation for a complexity class if the machines defining the class are restricted to have atmost r tapes apart from the read-only input tape.

Lemma 2.1 For any k, $\Sigma_k - TIGU(t(n), LIN) \subseteq \Sigma_k^2 - TIGU(t(n), LIN)$.

Proof Our simulation of r tapes by 2 tapes is analogous to the standard simulation [3]. Given an r-tape guess-bounded machine M that makes $k-1$ alternations, we build a machine M' to simulate it. M' first guesses a computation of M on one tape. It then verifies the computation by using its second tape to simulate each of the r tapes of M, in turn. The verification is deterministic, so M' doesn't waste any guesses on it. Also, any guesses of deterministic steps of M are made existentially if the last phase of M is existential and universally if the last phase of M is universal, with acceptance criteria being modified accordingly. It is easy to see that M' makes only a linear number of guesses in all but its last phase, makes $k-1$ alternations

and accepts the same language as M.

Lemma 2.2 $\Pi_k - TIGU(t(n), LIN) \not\subseteq \Sigma_k - TIGU(o(t(n)), o(n))$

Proof By diagonalization. The proof in [25] goes through with the additional observation that the number of guesses is preserved by the simulating machine.

Lemma 2.3 If $t(n)$ is polynomially bounded and $NTIME(t(n)) = DTIME(t(n))$, then $\Pi_2 - TIGU(t(n), LIN) \subseteq co - NTIME(t(n)$

Proof Let L be a language in $\Pi_2 - TIGU(t(n), LIN)$. Let M be a Π_2 machine accepting L and making only a linear number of guesses in all but its last phase. We shall show the existence of a co-nondeterministic machine $M^{'}$, under the given hypothesis, such that $M^{'}$ operates in time $t(n)$ and accepts L.

Corresponding to M, there is a predicate $R(x, y, z)$ computable in deterministic time $t(n)$ such that
$x \in L \equiv \forall^{O(|x|)} y \exists^{O(t(|x|))} z R(x, y, z)$
Clearly, the predicate $Q(x, y) = \exists^{O(t(|x|))} z R(x, y, z)$ is computable in nondeterministic time $t(\mid x \mid)$ and therefore in nondeterministic time $t(\mid x \mid + \mid y \mid)$. By the hypothesis, there is a predicate $Q^{'}(y, z)$ that is computable in deterministic time $t(\mid x \mid + \mid y \mid)$ such that
$x \in L \equiv \forall^{O(|x|)} y Q^{'}(x, y)$
The co-nondetermistic machine $M^{'}$ works as follows: it universally guesses a string y of length $c \mid x \mid$ (where the constant c depends on M) and computes the predicate $Q^{'}(x, y)$. This only takes time $O(t(\mid x \mid))$ as we are dealing only with polynomially bounded time bounds t. Clearly, $M^{'}$ accepts an input iff M does, and $M^{'}$ runs in time $O(t(n))$.

Proposition 2.4([10],[24]) For every time-constructible function t with $t(n) \geq n$, $DTIME(t(n)) \subseteq \Sigma_2 - TIGU(n, t(n)/(log^*(t(n))))$.

Theorem 2.5 For all functions t such that $t(n) = o(nlog^*(n)), DTIME(t(n)) \neq NTIME(t(n))$

Proof Assume the contrary. Then, $DTIME(t(n)) = co - NTIME(t(n))$ and by Lemma 2.3, we have
$\Pi_2 - TIGU(t(n), LIN) \subseteq DTIME(t(n))$

Also, by Proposition 2.4, since $t(n) = o(nlog^*(n))$
$DTIME(t(n)) \subseteq \Sigma_2 - TIGU(n, o(n))$
Thus $\Pi_2 - TIGU(nlog^*(n), LIN) \subseteq \Sigma_2 - TIGU(n, o(n))$, which is a contradiction to Lemma 2.2. $\square$

Corollary 2.6 $DTIME(n\sqrt{log^*(n)}) \neq NTIME(n\sqrt{log^*(n)})$ Our result is more interesting for the technique used to prove it than for the truth it states, since we do not even know if there any languages in $DTIME(n\sqrt{log^*(n)}) - DTIME(n)$. On the other hand, a proof that $DTIME(nlog(n)) \neq NTIME(nlog(n))$ would indeed be interesting. We sketch a graph-theoretic hypothesis below that would lead to this result.

Definition An M-segregator for an acyclic directed graph G is a set J of vertices of G such that every vertex in $G - J$ has atmost M predecessors in $G - J$. A family F of graphs is said to have $(p(n), q(n))$ segregators if every graph $G \in F$ with n vertices has a $p(n)$ segregator of size $q(n)$.

[24] shows that the class $H_r(n)$ of multi-pushdown graphs has $(n/(log^*(n)), n/(log^*(n)))$ segregators, using path-compression techniques due to [6] and [28]. This result is the cornerstone of their proof that deterministic linear time is distinct from nondeterministic linear time. Their speed-up of deterministic time by Σ_2 time, and hence our result also, by the dependence through Proposition 2.4, will be improved if it can be proved that there are smaller segregators for the class of multi-pushdown graph.

Theorem 2.7 If, for each r, the class $H_r(n)$ of graphs has $(o(n), o(n/log(n)))$ segregators, then $DTIME(nlog(n)) \neq NTIME(nlog(n))$
Proof Analogous to proof of Theorem 2.5.

It is known [24] that there are families of pushdown graphs that do not have $(o(n/log(n), o(n/log(n)))$ segregators. We are interested in the extent to which the current upper bound can be improved.

We now describe another application of our technique. In [15], Kannan proved that atleast one of the two statements $P \neq L$, $DLIN \neq NLIN$ must hold. We show how to generalize the second hypothesis to $DTIME(t(n)) \neq NTIME(t(n))$, for any t.

Proposition 2.8 (Kannan) $DTISP(n^k, (logn)^k) \subseteq \Sigma_{2k} - TIME(n)$

Theorem 2.9 $P = L \implies$ For all polynomially bounded $t, NTIME(t) \neq DTIME(t)$

Proof Suppose, on the contrary that for some polynomially bounded t, $NTIME(t) = DTIME(t)$. By a simple generalization of Lemma 2.3, for any k, $\Sigma_k - TIGU(t(n), LIN) = NTIME(t(n)) = DTIME(t(n))$. By Proposition 2.8, a given language in L is in $\Sigma_k - TIME(n)$ for some k. Hence it is in $\Sigma_k - TIGU(t(n), LIN)$ and in $DTIME(t(n)$, by Lemma 2.3. Thus $L \subseteq DTIME(t(n))$, which implies $L \neq P$ by the time hierarchy theorem for deterministic time. □

Note that essentially the same proof shows that we can replace the hypothesis $P \neq L$ by $P \neq SC$.

4 Time vs space

Another core problem of complexity theory is whether space is more powerful than time. This question was answered in the affirmative by Hopcroft, Paul and Valiant in their seminal paper [12], where they proved that, for constructible t, $DTIME(t(n)) \subseteq DSPACE(t(n)/log(t(n)))$. By the deterministic space hierarchy theorem, it follows that space is more powerful than time for deterministic Turing machines. The main theorem of [12] was extended to RAMs by [26] and to pointer machines by [11]. But the related question for nondeterministic machines remains open.

The proof in [12] proceeds by defining a pebble game on graphs, such that a good strategy for the pebble game implies a good simulation of deterministic time-bounded machines by space-bounded machines. Unfortunately, this game cannot be used to model nondeterministic computations. In this paper, we define a different pebble game that models the simulation of nondeterministic time-bounded machines by space-bounded machines. We are not able to prove that a good strategy exists for this game on the class of graphs(multi-pushdown graphs) we are interested in; we show instead that the existence of a good strategy is equivalent to a graph-theoretic property, i.e the property of having good "separators".

We first define the game. The BWR pebble game is played with three kinds of pebbles(black, white and red) on directed acyclic graphs and is defined by the following rules-

1. A white pebble may be placed on a vertex of the graph at any time.
2. A white pebble on a vertex may be replaced by a red pebble if the immediate predecessors of the vertex all have white or red pebbles placed on them.
3. A red pebble on a vertex may be replaced by a black pebble if all the immediate successors of the vertex have red pebbles placed on them.

Initially, there are no pebbles on any of the vertices. The game ends when all vertices have black pebbles placed on them. A strategy is a correct sequence of moves from the initial configuration to the final configuration. The space used by a strategy is the maximum number of red and white pebbles placed on vertices of the graph after any move. An optimal strategy for a graph is a strategy that uses the minimum space. A graph is said to require space s in the BWR game if an optimal strategy for this graph uses space s.

Intuitively, the red and white pebbles represent guesses of configurations during a computation. The white pebbles represent unverified guesses, the red pebbles represent guesses that have been verified in the sense that the configuration associated with such a guess is compatible with the guesses of previous configurations on which this configuration is directly dependent. Black pebbles represent guesses that do not have to be stored any longer.

Theorem 4.1 Let f be a function on the positive integers. If, for each r and n, no graph in $H_r(n)$ requires more than $f(n)$ space in the BWR game, then $NTIME(t(n)) \subset NSPACE(t(n)^{1/3} f(t(n)^{2/3}))$ for time-constructible functions t.

Proof Deferred to Appendix.

Definition An acyclic directed graph G on n vertices has an S-separator if there is a set of vertices C of the graph of size S, such that $G - C$ is not weakly connected and each component of $G - C$ has size $\leq 2n/3$. A graph G on n vertices has a $(f(n), g(n))$-separator if G has a $f(n)$-separator C such that each component of $G - C$ has atmost $g(n)$ vertices. The above definitions extend in the natural way to a family F of graphs.

Proposition 4.2 If a family F of graphs, closed under subgraphs, has $o(n)$-separators, it has $(o(n), o(n))$-separators.

Proof Refer to [22]

Lemma 4.3 If a graph $G = (V, E)$ with $\mid V \mid = n$ has a strategy for the BWR game that uses space $f(n)$, then G has an $f(n)$-separator.

Proof Deferred to Appendix.

Lemma 4.4 If a graph $G = (V, E)$ with $\mid V \mid = n$ has a $((f(n), g(n))$-separator, then there is a strategy for G for the BWR game that uses space $O(f(n) + g(n))$.

Proof Deferred to Appendix.

Theorem 4.5 A family of graphs, closed under subgraphs, has $o(n)$ separators iff for each graph in the family, there is a strategy for the BWR game that uses $o(n)$ space.

Proof Immediate from Proposition 4.2, Lemma 4.3 and Lemma 4.4.

It is mentioned in [22] that the existence of sublinear separators for multi-pushdown graphs implies that nondeterministic space is more powerful than nondeterministic time. 2-pushdown graphs are planar and these have sublinear(in fact, $\mathrm{O}(\sqrt{n})$) separators [18]. The question of existence of sublinear separators for 3-pushdown graphs is equivalent to the question for general r-pushdown graphs, by a result of Kannan [16]. The question is still open, the most recent attack on it being the [9] paper in which they prove nearly linear lower bounds on the sizes of separators of certain pushdown graphs. It might have been hoped that a weaker hypothesis than the existence of sublinear separators for multi-pushdown graphs would imply the separation of nondeterministic time and nondeterministic space. Theorems 4.1 and 4.5 show that proof techniques analogous to those of [12] need the hypothesis of existence of sublinear separators to work in the nondeterministic domain.

5 Separators and Simulation

In the previous section,we saw that the existence of sublinear separators for multi-pushdown graphs implies that nondeterministic time is distinct from nondeterministic space. In this section, we prove that it implies something stronger - that $NTIME(t)$ is strictly contained in $\Sigma_4 - TIME(t)$, for con-

structible t. Thus, by the remarks at the end of the previous section, any "simulation" proof analogous to that of [12] showing nondeterministic time is distinct from nondeterministic space would also show that nondeterministic time is distinct from Σ_4-time.

Our proof technique is similar to that of [10] and [24]. We simulate a computation of a nondeterministic machine by one of a Σ_4 machine. The existence of sublinear separators in the computation graph enables us to break up the computation into a number of independent subcomputations, each of which takes time asymptotically less than the time required for the entire computation. Since each of these subcomputations can be performed in parallel, the Σ_4 machine can save time over the nondeterministic machine. The hypothesis of existence of sublinear separators is essential here, since if we had only sublinear segregators as in [24], the subcomputations would not be independent and we cannot ensure that the various guesses we make in the simulation are consistent.

Let us assume, without loss of generality, that $g(n) \geq f(n)$ and $g(n) \geq \sqrt{n}$ for all n.

Theorem 5.1 If the class of multi-pushdown graphs has $(f(n), g(n))$-separators, then $NTIME(t) \subseteq \Sigma_4 - TIME(t^{1/3}g(t^{2/3}))$, for time-constructible t.

Proof Deferred to Appendix.

Proposition 5.2([25]) For any k, $\Sigma_k - TIME(t) \not\subseteq \Pi_k - TIME(o(t))$.

Corollary 5.3 If the class of pushdown graphs has sublinear separators, for all t, $NTIME(t) \neq \Sigma_4 - TIME(t)$

Proof By Theorem 5.1, under the given hypothesis, $NTIME(t) \subseteq \Sigma_4 - TIME(o(t))$. Assume, contrary to the statement of the corollary, that $NTIME(t) = \Sigma_4 - TIME(t)$. Then $\Sigma_4 - TIME(t) = \Sigma_4 - TIME(o(t))$. Hence, also $\Pi_4 - TIME(t) = \Pi_4 - TIME(o(t))$. But $NTIME(t) \subseteq \Pi_4 - TIME(t)$, which implies $\Sigma_4 - TIME(t) \subseteq \Pi_4 - TIME(o(t))$, a contradiction to Proposition 5.2.

References

[1] J.L.Balcazar, J.Diaz and J.Gabarro. Structural Complexity 1. Volume 11 of EATCS Monographs in Theoretical Computer Science, Springer Verlag, 1988.

[2] J.L.Balcazar, J.Diaz and J.Gabarro. Structural Complexity 2. Volume 22 of EATCS Monographs in Theoretical Computer Science, Springer Verlag, 1990.

[3] R.V.Book, S.A.Greibach and B.Wegbreit. Time- and Tape-Bounded Turing Acceptors and AFLs. *Journal of Computer and System Sciences*, 4(6):606-621, 1970.

[4] A.K.Chandra, D.C.Kozen and L.J.Stockmeyer. Alternation. *Journal of the ACM*, 8(1):114-133, 1981.

[5] P.W.Dymond and M.Tompa. Speedups of deterministic machines by synchronous parallel machines. In *Proceedings of the 24th Annual IEEE Symposium on the Foundations of Computer Science*, 336-343, 1983.

[6] P.Erdos, R.L.Graham and E.Szemeredi. On sparse graphs with dense long paths. Computers and Mathematics with applications, Pergamon, 1976.

[7] L.Fortnow. Nondeterministic polynomial time versus nondeterministic logarithmic space:Time-space tradeoffs for satisfiability. In *Proceedings of the 12th Annual IEEE Conference on Computational Complexity*, pages 52-60. IEEE, Los Alamitos, 1997.

[8] L.Fortnow and D.van Melkebeek. Time-space Tradeoffs for Nondeterministic Computation. In *Proceedings of the 15th IEEE Conference on Computational Complexity*, pages 2-13, 2000.

[9] Z.Galil, R.Kannan and E.Szemeredi. On nontrivial separators for k-page graphs and simulations by nondeterministic one-tape Turing machines. In *Journal of Computer and System Sciences*, 38(1):134-149, 1989.

[10] S.Gupta. Alternating Time Versus Deterministic Time: A Separation. *Mathematical Systems Theory*, 29(6):661-672, 1996.

[11] J.Y.Halpern, M.C.Loui, A.R.Meyer and D.Weise. On time versus space 3. *Mathematical Systems Theory*, 19:13-28, 1986.

[12] J.Hopcroft, W.J.Paul and L.Valiant. On time versus space. *Journal of the ACM*, 24:332-337, 1977

[13] J.E.Hopcroft and J.D.Ullman. Introduction to Automata Theory, languages and Computation. Addison-Wesley, 1979.

[14] N.Immerman. Nondeterministic space is closed under complement. *Siam Journal of Computing*, 17:935-938, 1988.

[15] R.Kannan. Towards separating nondeterminism from determinism. *Mathematical Systems Theory*, 17(1):29-45, April 1984.

[16] R.Kannan. Unraveling k page graphs. *Information and Control*, 1985.

[17] T.Lengauer. Black-White Pebbles and Graph Separation. *Acta Informatica*, 16:465-475, 1981.

[18] R.Lipton and R.E.Tarjan. Applications of a planar graph separator theorem. *Siam Journal of Computing*,9(3):615-627, 1980.

[19] R.Lipton and A.Viglas. On the complexity of SAT. In *Proceedings of the 40th Annual Symposium on Foundations of Computer Science*, pages 459-464. IEEE, New York, 1999.

[20] S.Moran. Some results on relativized deterministic and nondeterministic time hierarchies. *Journal of Computer and System Sciences*,22(1):1-8, 1981.

[21] N.Pippenger. Pebbling. In *Proceedings of the 5th IBM Symposium on Mathematical Foundations of Computer Science*, 1980.

[22] N.Pippenger. Advances in pebbling. In *Proceedings of the International Colloquium on Automata, Languages and Programming*, pages 407-417, 1982.

[23] W.J.Paul, E.Prauss and R.Reischuk. On alternation. *Acta Informatica*, 14:243-255,1980.

[24] W.Paul, N.Pippenger, E.Szemeredi and W.Trotter. On determinism versus non-determinism and related problems. In *Proceedings of the 24th IEEE Symposium on Foundations of Computer Science*, pages 429-438. IEEE, New York, 1983.

[25] W.J.Paul and R.Reischuk. On alternation 2. *Acta Informatica*, 14:391-403, 1980.

[26] W.J.Paul and R.Reischuk. On time versus space 2. *Journal of Computer System Sciences*, 22:312-327, 1981.

[27] R.Szelepcsenyi. The method of forced enumeration for nondeterministic automata. *Acta Informatica*, 26:279-284, 1988.

[28] L.Valiant. Graph-theoretic properties in computational complexity. *Journal of Computer and System Sciences*, 13:278-285, 1976.

6 Appendix

Proof of Lemma 4.3 Consider a strategy for the pebble game on G that uses space $f(n)$. Since atmost one pebble is placed on a vertex of the graph in any move, there is some stage after which exactly $n/2$ vertices have been pebbled black. Let A be the set of vertices pebbled black at this stage, B the set of vertices on which no pebbles have been placed and C the set of vertices pebbled red or white. We claim that C is a $f(n)$-separator for G. Clearly, $\mid A \mid < 2n/3$, $\mid B \mid < 2n/3$ and $\mid C \mid = f(n)$. Also, there are no edges joining a vertex in A to a vertex in B since all the immediate successors and predecessors of a vertex pebbled black must be pebbled (an easy consequence of the rules of the BWR game).

Proof of Lemma 4.4 Let S be an $(f(n), g(n))$ separator of G. We show that there is a strategy for the BWR game on G that uses space $O(f(n) + g(n))$.

The strategy is as follows: First place white pebbles on all the vertices of S. Place white pebbles on all the immediate predecessors of vertices in S. Replace all the white pebbles placed on vertices in S with red pebbles. Then pebble each component of $G - S$, in some order. Since the vertices in any component of $G - S$ have their neighbours either in S or the component, all the vertices in the component can be pebbled black using atmost $g(n)$ pebbles. When all the vertices in $G - S$ have been pebbled, replace the red pebbles on vertices in S with black pebbles, thus completing the pebbling of G. The space used by the strategy is upper bounded by $df(n) + g(n)$, where d is the maximum degree of a vertex in S. Since we are considering bounded-degree graphs, this is $O(f(n) + g(n))$.

Proof of Theorem 4.1 Let L be a language in $NTIME(t(n))$ and M a block-respecting nondeterministic Turing machine with block size $t(n)^{2/3}$

accepting it. We show how to simulate M by a nondeterministic space-bounded Turing machine $M^{'}$, under the given hypothesis.

Fix an input x of length n and let $t = t(n)$. Given this input, $M^{'}$ first guesses a computation graph G of size $t^{1/3}$ corresponding to a block-respecting computation of M on x. This can be done in space $t^{2/3}$. Then $M^{'}$ simulates an optimal strategy (i.e a strategy using $f(t^{1/3})$ space) for the BWR game on the graph G. Note that, since there are atmost $4^{f(t^{1/3})}$ possible configurations of pebbles on G, the next move in an optimal strategy can be computed in deterministic space $t^{2/3}$ by an application of Savitch's theorem as in [12].

Corresponding to a move in an optimal strategy for the BWR game on G, $M^{'}$ acts as follows - When a white pebble is placed on a vertex i of the graph, $M^{'}$ guesses and stores the ending configurations of all blocks associated with i, i.e it guesses the state, tape head positions and block inscriptions at the end of the time interval i. If i is the last vertex of the graph and $M^{'}$ guesses that M rejects, then $M^{'}$ rejects. Since the size of each block is $t^{2/3}$ and the number of blocks associated with a vertex is bounded above by a constant (the number of tapes of M), the additional space used is $O(t^{2/3})$.

When a white pebble on a vertex i is replaced by a red pebble, $M^{'}$ verifies its guesses for that vertex by running M during time interval i. A red pebble can be placed on a vertex only when all immediate predecessors have white pebbles on them, so $M^{'}$ knows the inscriptions of all associated blocks as well as the state and tape head positions at the beginning of time interval i. $M^{'}$ checks if the configurations it has guessed for i can be accessed in exactly $t^{2/3}$ steps. If this check fails, it rejects. The space required for the entire procedure is $O(t^{2/3})$ since this is the total size of configurations at any point of tiem in the check.

When a red pebble on a vertex i is replaced with a black pebble, $M^{'}$ simply deletes all information associated with that vertex.

At the end of the simulation of the BWR game, when all vertices have been pebbled black, $M^{'}$ accepts. The amount of space required by $M^{'}$ is $O(f(t^{1/3}t^{2/3})$ since there can be atmost $f(t^{1/3})$ red or white pebbles on G at any time and each pebble costs space $t^{2/3}$. The space used for computing the next move in an optimal strategy or simulating M is specific to each

move and can be reused. We now need to prove that M accepts x if and only if $M^{'}$ does.

The 'only if' part is easy. If there is some accepting computation of M on x, $M^{'}$ accepts by guessing the computation graph for this computation correctly and guessing each configuration in this computation correctly. In this case, all checks go through and, by the assumption of a "good" strategy on the computation graph, $M^{'}$ accepts while using the stated amount of space.

We shall only give a sketch of the proof for the 'if' part. The idea is that $M^{'}$ only accepts if all its guesses are 'consistent', in some sense. Note that there is exactly one set of guesses of configurations made for each vertex in the computation graph- we need to prove that, if $M^{'}$ accepts, there is a computation of M in which each of these configurations actually occurs at the specified times. Since every vertex has a red pebble placed on it at some stage, consistency within a time interval(local consistency) is guaranteed. Global consistency is guaranteed by the consistency of all local checks and the correctness of the computation graph, which is verified implicitly during each local consistency check. The key point is that each local consistency check involves only a vertex and its immediate predessors. Therefore, once all the immediate successors of a vertex have been checked, the information associated with that vertex need no longer be stored, since it will not be accessed in further checks. This enables us to save on the space used.

Proof of Theorem 5.1 Let L be a language in $NTIME(t)$. Let M be a block-respecting nondeterministic machine that accepts L and operates in time t and block size $t^{1/3}$. We shall construct a Σ_4 machine $M^{'}$ that accepts L in time $t^{1/3}g(t^{2/3})$. On an input x of length n, $M^{'}$ first computes $t(n)$, $a(n) = t(n)^{2/3}$, $b(n) = t(n)^{1/3}$ and $e(n) = g(t(n)^{2/3})$. It then proceeds as follows.

Phase 1(existential): $M^{'}$ guesses the computation graph G of a computation of M on x. It does this implicitly by guessing the tape head movement at the end of each time interval. It then guesses a $(f(n), g(n))$-separator C of the computation graph. It also guesses the components of the graph formed by removing the separator from the computation graph (these guesses are verified in Phase 2.1). It then guesses the starting and ending configurations of each block associated with the time segments corresponding to the guessed separator vertices. For each vertex, it verifies that there is a computation

leading from the starting to the final configuration by simulating M. It also guesses the starting and ending configurations of the blocks associated with the last vertex of the computation graph and verifies that they are consistent. It also verifies that the ending configuration associated with the last vertex is an accepting configuration of M. If any of the above verifications fails, it rejects. It proceeds to Phase 2.1.

Phase 2.1(universal): $M^{'}$ checks that the size of the guessed separator C is atmost $f(n)$ and the size of each guessed component of $G - C$ is atmost $g(n)$. It universally guesses a vertex of G and verifies that it belongs either to C or to exactly one guessed component. It universally guesses an edge of G and verifies that either its endpoints belong to the same component or one of the endpoints is in S. It proceeds to Phase 2.2.

Phase 2.2(universal): $M^{'}$ universally guesses the index of a component of $G - C$. It then proceeds to Phase 3.

Phase 3(existential): $M^{'}$ guesses the starting and ending configurations of blocks associated with the vertices in the component whose index has been guessed in Phase 2.2. It checks consistency within blocks. If some check fails, it rejects. Otherwise, it proceeds to Phase 4.

Phase 4(universal): $M^{'}$ universally guesses a vertex in the component guessed in Phase 2.2. It checks that the guesses for the vertex in Phase 3 are consistent with the guesses for its immediate predecessors by comparing the starting configurations of the vertex with the ending configurations of its immediate predecessors. If all the checks succeed, it accepts, otherwise it rejects.

ECCC ISSN 1433-8092
http://www.eccc.uni-trier.de/eccc
ftp://ftp.eccc.uni-trier.de/pub/eccc
ftpmail@ftp.eccc.uni-trier.de, subject 'help eccc'