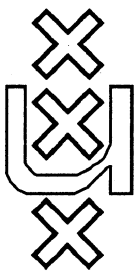
Institute for Language, Logic and Information

ON SPACE EFFICIENT SIMULATIONS

Ming Li
Herman Neuféglise
Leen Torenvliet
Peter van Emde Boas

ITLI Prepublication Series
For Computation and Complexity Theory CT-89-03



University of Amsterdam

Abstract

In this paper we show that for various nondeterministic machine models space is a more powerful resource than time. The main results are that nondeterministic $T(n)$ time-bounded single-tape Turing machines can be simulated by $\sqrt{T(n)}$ space-bounded Turing machines in $T(n)$ time, and that $T(n)$ time bounded nondeterministic queues can be simulated by $\sqrt{T(n)}\log T(n)$ space-bounded single-tape Turing machines in time $T(n)\sqrt{T(n)}\times \log^2 T(n)$. The second result is a consequence of the planarity of the computation graph (which we also show) and a special version of the planar separator theorem. Therefore similar results can easily be derived for closely related models.

1 Introduction

Determining the relationship between recognizing power of resource-bounded machine models is a central and long-standing open problem in computational complexity theory. The study of different machine models for sequential computation is essential to obtain a model-independent complexity theory, or to translate results obtained for one model directly to other models without proving them again.

Many theorems have been proved on the relation between different variants of the *Turing machine model*. For a Turing machine (TM for short) time is measured by the number of transitions in a computation and space is measured by the number of different cells visited by the head(s) during the computation. As far as space is concerned all different models can simulate each other with constant factor overhead. Results concerning time complexity show that all different *multitape* TMs can be represented by two-tape TMs if one allows for a logarithmic time overhead (See theorem 12.6 in [4]). The single-tape TM forms an exception here since if the Turing machine does not have a separate input tape there is a quadratic lower bound known for the recognition of palindromes (See theorem 10.7 in [3]). W. Maass [13], M. Li and P. Vitányi [10], Z. Galil, R. Kannan, and E. Szemeredi [1] proved following more general lower bounds for Turing machines with an extra input tape, viz.,

- There is a language that is accepted by a deterministic two work-tape TM in linear (even real) time but which requires time $\Omega(n^2)$ on any deterministic one work-tape TM [13] [10].
- There is a language that is accepted by a deterministic two work-tape TM in linear (even real) time but which is not accepted in time $n^2/\log^2 n \log\log n$ by any nondeterministic one work-tape TM [13]. This result is minorly improved by [10] to $\Omega(n^2/\log n \log\log n)$ and significantly improved by Galil, Kannan, and Szemeredi [1] to $\Omega(n^2/\underbrace{\log\ldots\log}_{s} n)$ for any positive integer s.
- There is a language that is accepted by a deterministic TM with 2 pushdown stores or one queue but which requires $\Omega(n^2)$ on any one work-tape deterministic TM [10].

For a survey of above results, see [11]. The present paper investigates the relation between time and space for some machine models. It is obvious that any TM that consumes $S(n)$ space to perform a computation must use up $\Omega(S(n))$ time. On the other hand:

> Any language accepted by a $T(n)$ time bounded multitape TM can be accepted by a $T(n)/\log T(n)$ space bounded multitape TM

as is shown in [6]. For *single*-tape TMs even sharper bounds are known [5].

- Any language accepted by a single-tape TM in time $T(n) \geq n^2$ can be accepted by a single-tape TM in space $\sqrt{T(n)}$.
- Any language accepted by an off-line TM in time $T(n) \geq n^2$ can be accepted by an off-line TM in space $\sqrt{T(n)\log n}$.

In their paper [7] Ibarra and Moran showed that these simulations (which up till then required exponential time) can also be time-efficient; in particular:

1. Any language accepted by a single-tape TM in time $T(n) \geq n^2$ can be accepted by a single-tape TM in space $\sqrt{T(n)}$ and time $T^2(n)$.

2. Any language that is accepted by an off-line TM in time $T(n) > n$ can be accepted by an off-line TM in space bounded by $\sqrt{T(n)\log n}$ and time bounded by $\sqrt{T^3(n)}\left(\sqrt{T(n)} + \frac{n}{\sqrt{\log n}}\right)$.

3. Any language accepted by a single-tape nondeterministic TM (NTM) in time $T(n) \geq n^2$ can be accepted by a single-tape NTM in space $S(n)$ and time $\frac{T^2(n)}{S(n)}$ for any $\sqrt{T(n)} \leq S(n) \leq T(n)$.

4. Any language accepted by an off-line NTM in space $S(n)$ and time $T^2(n)\left(1 + \frac{n}{S(n)}\right)$ for any $\sqrt{T(n)\log n} \leq S(n) \leq T(n)$.

The start (and inspiration) of the present paper is a sharpening of their result 3. We show for machines operating within reasonably well-behaved time bounds that:

Theorem 1 *Any language accepted by a single-tape NTM in time $T(n)$ can be accepted by a single-tape NTM in space $\sqrt{T(n)}$ and time $T(n)$ provided that $T(n) \geq n^2$ and $\sqrt{T(n)}$ is fully time- and space constructible.*

2 On single-tape Turing machines

2.1 The standard model

The key idea in the space-efficient simulation for Turing machines is that instead of replaying the entire computation chronologically, the computation can be divided into chunks of moves during which the tape head is in a certain block on the tape. Dividing the tape in blocks of more or less the same length (but $\leq \sqrt{T(n)}$) we can replay the computation block by block, and accept if in some block an accepting configuration is reached. Consistency between blocks is guarded by writing down de direction and state of the Turing machine upon entering and leaving the block. The fact that this administration of states and directions and the storage of the tape block needs about the same space then makes the entire simulation space efficient. Thus there are two important concepts which we will describe briefly: The partition of the tape, and the "trace of the computation".

Definition 1 A sequence $P = P_1, P_2, \ldots$ is a *legal partition* of a semi-infinite tape F if for each i:

- P_i is a block of a finite number of consecutive cells of F.
- P_{i+1} is directly to the right of P_i.
- each cell of F belongs to a uniqe P_i.

Now for a division of the tape into blocks and a computation of machine M we can define the trace of a computation. The concept is simple: Each time the head crosses the border between two consecutive blocks, the direction and state of the machine are denoted

in a pair (d, q). Furthermore we keep the trace ordered so that the entire sequence of crossings pertaining to block i is denoted (chronologically) *before* the sequence pertaining to block j, whenever $i < j$. Separator symbols may make discrimination between pairs belonging to P_i and $P_{i\pm1}$ possible. Now:

Definition 2 Let M be a single-tape TM and w be in Σ^*. Let P be a legal partition of the tape of M. Then for each nonnegative integer n the *trace of M on input w with respect to P after n steps* (denoted by TRACE(M, w, P, n)) is a finite sequence $(d_1^1, q_1^1), \ldots, (d_k^1, q_k^1), \ldots, (d_l^m, q_l^m)$ where $d \in \{\leftarrow, \rightarrow\}$ and q is a state of M and is defined as follows:

1. TRACE$(M, w, P, 0)$ is the empty sequence.
2. If M's head does not cross a border between two consecutive blocks of P during the n'th move then TRACE(M, w, P, n)= TRACE$(M, w, P, n-1)$.
3. If M's head *does* cross a boundary during the n'th move then the two blocks in the trace corresponding to the two blocks in P change. In *both* blocks the pair (d, q) is added to the sequence. (Note that entrance or exit of the tape head is determined uniquely by parity. Then e.g. $\leftarrow$ is interpreted as "the tape head enters coming from the block immediately to the right" in the case of an entrance pair and as "the tape head exits to the left" in the case of an exit pair).

As the trace is actually a double administration of events, we define the *length* of a trace to be the number of pairs in the trace divided by 2 and rounded upwards. Since the first entrance pair is always $(\rightarrow, q_0)$ it may wlog be omitted.

It is easy to see that for a computation of length $T(n)$ the length of a computation trace corresponding to a partition in which all blocks have length $\sqrt{T(n)}$ is about $\sqrt{T(n)}$. We can get this precise if we allow the first block to have a different length.

Definition 3 Let $s \geq 1$ and $1 \leq j \leq s$. Then the j'th legal partition of size s is the partition (P_0, P_1, P_2) defined by:

- P_0 consists of cells $1, 2, \ldots, j$.
- For $t > 0$, P_t consists of cells $j + (t-1)s + 1$ through $j + t \times s$.

Now from [7] we borrow the following lemma:

Lemma 1 *Let M be of time complexity $T(n)$. Let w be in Σ^*, $|w| = n$. For each j, $1 \leq j \leq \sqrt{T(n)}$ let P^j be the j'th legal partition of size $\sqrt{T(n)}$ and let ℓ_j be the length of TRACE(M, w, P^j). Then there is a j_0 such that $1 \leq j_0 \leq \sqrt{T(n)}$ and $\ell_{j_0} \leq \sqrt{T(n)}$.*

Proof: The proof is an elementary application of the pigeon hole principle, and is left to the reader. □

Now we are ready to present the simulation. A Turing machine M operating in time $T(n)$ is simulated by a Turing machine M' operating in time $T(n)$ and space $\sqrt{T(n)}$. (Constant factors are swept frequently under the carpet in the sequel by grace of speedup and

compression theorems.) The idea is quite simple. First the tape is divided into two blocks of size $\sqrt{T(n)}$. (This can be done since $\sqrt{T(n)}$ is fully time- and space constructible.) The first block will be used to simulate the actions of the Turing machine. The second contains a guessed computation trace as described in definition 2. If the guessed trace threatens to become longer than $\sqrt{T(n)}$ then M' halts and rejects. The machine accepts if the guessed trace ends in an accepting configuration and the trace can be verified. The verification consists of two parts.

1. *The local verification*

   This is the verification per block worktape. It starts by making the first part of the worktape blank, except when the *first* and maybe the second block are simulated. (This is the input which is of size $\leq T(n)$ and is divided between the first and second block of simulation depending on the guessed j for which the j'th legal partition was guessed[1])

   Now repeat until the block of trace items pertaining to the present block is exhausted:

   - Fetch a next entrance pair (d, q) from the trace part.
   - Simulate M's program starting from the left or the right side of the block (as indicated by d) in state q, until M attempts to leave the tape block.
   - Check that the exit state and direction correspond to the exit pair written down in the trace part.

   Obviously the starting conditions for each block can be realized in time $\sqrt{T(n)}$. So the total time spent does not exceed $T(n)$. The simulation of M's program totalled over all blocks cannot exceed $T(n)$ for obvious reasons. Each time a new entrance pair is fetched or and exit pair is verified the head has to travel a distance of at most $2\sqrt{T(n)}$. As there are only $\sqrt{T(n)}$ pairs to be checked this cannot cost more that $T(n)$ steps.

2. *The global verification*

   This part checks that the computation trace is a consistent log of the movements of the tape head between blocks.

   Starting with the first pair (d, q) in the the trace (which is the first exit pair of the input block). It marks a pair, goes to the next unmarked pair (d', q') of the block given by d in this pair (which must be the first pair of block 2 for the pair (d, q)) and checks that the direction and state of the two pairs (remembered exit pair and found entrance pair) compare. Then the next exit pair is found immediately to the right of the just checked entrance pair, and the corresponding entrance pair can be found in the block given by d. In this way the entire trace is checked for consistency, and the machine accepts if it finds an accepting state during this process[2]. The distance between two pairs does not exceed $\sqrt{T(n)}$ and this distance has to be crossed at

[1] we don't move the symbols belonging to the second block since this may take $\Omega(T(n))$ time and complicate the proof. Instead the extension of the input over two blocks may require assigning $j \leq \sqrt{T(n)}$ extra cells of worktape to the simulation, but this will disappear in the constant factor

[2] As the global verification is in chronological order it can immediately halt on this event

most $\sqrt{T(n)}$ times. Therefore the total time spent in the global verification is also at most $T(n)$.

This concludes the proof of theorem 1

Note: The local and global verification described in the proof above can easily be intertwined without loosing correctness. Each time an exit pair is verified, an entrance pair can be checked off in one of the adjacent blocks of the trace. As noted before the discrimination between entrance and exit pairs can be made by parity, the entrance pairs in the adjacent block pertaining to the presently checked block are determined uniquely by d, and they are stored in chronological order.

Note: The simulation above is obtained for weakly time-bounded machines. By inserting a Martin Führer-like clock [2] the result can also be obtained for strongly time-bounded models. This is in fact what is done by Lorys and Liskiewicz who obtained the result of theorem 1 independently (but later) in their paper [8].

2.2 Sublinear-space models

The reason for having the restriction $T(n) \geq n^2$ in theorem 1 is of course that the input string (which has length n) is found on the tape at the start of the computation. Hence all cells occupied by the input have to be charged to the computation. On the standard Turing machine model therefore one can never talk about memory occupation less than n.

For this reason space on (single-tape) Turing machines is usually measured on a slightly different machine model. It is a standard one-tape Turing machine machine with an extra read-only tape on which initially the input is written. Cells occupied on this input tape are not charged for in counting space usage.

These "single"-tape machines come in two flavors. There are machines with a head on this input tape which can move in two directions—these are called *off-line* Turing machines—and there are machines with a head on this tape that can only move to the right—these are called *on-line* Turing machines.

It is easy to see that on these machines a simulation of the standard model does not need the restriction $T(n) \geq n^2$. We will show this for the weakest of the two models, the on-line TM.

Theorem 2 *Any language accepted by a single-tape NTM in time $T(n)$ can be accepted by an on-line NTM in space $\sqrt{T(n)}$ and time $T(n)$ provided that $\sqrt{T(n)}$ is fully time- and space constructible.*

Proof: Use the same simulation as in the proof of theorem 1 with the exception of the initialization of the blocks. If the input is not yet exhausted (i.e. if the symbol under the input head isn't a blank) a next chunk of $\sqrt{T(n)}$ cells is read off the input tape and stored in the simulating block of the work tape. □

Note that this theorem is by no means a sharpening of the results mentioned in the introduction since it is a result on simulation between two different machine models whereas the results of Ibarra and Moran are about simulations of on(off)-line machines *on* on(off)-line machines. It is of course more interesting to have results on these type of simulations

and therefore we will investigate the meaning of the simulation described in theorem 1 for these models. We begin with the off-line model.

The main problem we have with the simulation of an off-line NTM on an off-line NTM is keeping track of the position of the input head. While simulating the computation on a single block of the partition, the input head may move to the left or to the right. But between two steps in the trace during the local verification of the trace, the input head may have in the simulated computation visited several other cells and as a consequence may not be in a position adjacent to the position taken upon previously leaving the block. The only solution we see at present to this situation is storing in the computation trace the position of the input head upon entering and leaving the block for a cost of another $\log n$ tape cells per pair. For the local verification this means that for each fetched entrance pair the input head must be positioned. This means traveling a distance of $\leq n$ steps while keeping and updating a counter of size $\log n$. This can be done in time $n \log n$ time. As there are $\leq \sqrt{T(n)}$ entrance pairs, keeping $n \leq \sqrt{T(n)}$ this means that the total time for the local verification kan be kept below $T(n) \log n$. The global verification is again done by checking pairs, but now the tape head has to travel a distance of $\sqrt{T(n)} \log n$ (Note that the comparison of input head positions can be done bit by bit, so there's no need for keeping a counter.) This means that the total cost of the global verification also amounts to $T(n) \log^2 n$. From this we get:

Theorem 3 *Any language accepted by an off-line NTM in time $T(n) \geq n^2$ can be accepted by an off-line NTM in space $\sqrt{T(n)} \log n$ and time $T(n) \log^2 n$.*

For the on-line model the situation is just a little more complicated. Of course the head can move between exiting and reentering the simulated block but all reads performed in the local verification between two consecutive fetches of entrance pairs can be found in a consecutive block of cells on the input tape since the head of the simulated machine can move only to the right. Moreover these "blocks of reads" are consecutive if the entrance pairs are placed in chronological order, as is done in the global verification. Therefore we propose expanding the entrance pairs in the computation with a (guessed) string of symbols that represent the reads that take place between entering and leaving the block. As no more than n reads can take place (and $T(n) \geq n^2$), the length of the computation trace is increased by at most a factor 2. On the other hand obtaining the (guessed) read symbols during the local verification may imply a travel of the tape head over a distance of $2 \times \sqrt{T(n)}$ cells, but this event can occur only n times in the *entire* local verification. Therefore it presents $O(T(n))$ extra work, and this vanishes.

During the global verification the guessed strings must be checked against the input. At the beginning of the global verification the input head is still positioned on the first cell of the input tape. Each time a new entrance pair occurs, its string is matched against the next block of input tape cells by simultaneously moving the input- and worktape head.

Altogether we obtain:

Theorem 4 *Any language accepted by an on-line NTM in time $T(n) \geq n^2$ can be accepted by an on-line NTM in space $\sqrt{T(n)}$ and time $T(n)$.*

The door is now open for a number of inter-model simulations. (On-line on off-line and vice versa, On-line on standard model etc.) We don't however see any nice new insights

other than the ones given above to speed up these simulations or make them more space efficient then the ones above so we abstain from describing these for the moment.

3 On the planar separator theorem

Lipton and Tarjan [12] proved an interesting theorem on planar graphs. It turns out that in any in graph G having n nodes that can be embedded in the plane, a subset S of $\sqrt{n}$ nodes can be singled out with the property that this subset partitions the remaining nodes of G into two subsets G_1 and G_2 both of size about $\frac{1}{2}n$ such that any path going from a node in G_1 to a node in G_2 runs through a node of S.

For our purposes we need a special form of this planar separator theorem. Basically we want to have a division of G into chunks of size about $\sqrt{n}$, with separators of size $\sqrt{n}$ between the chunks. This makes it possible to verify a guessed graph of size n on a single-tape Turing machine in about $\sqrt{n}$ space. It is not clear that any given planar graph has this property. However for any given planar graph G there is graph G' having this property and is "close enough" to G to fit our purposes.

Theorem 5 *For any planar graph G with n nodes of maximal degree c there is a planar graph G' with the property that:*

1. *G' can be divided into $2\sqrt{n}-1$ subsets $\{V_1,\ldots,V_{\sqrt{n}}\}$ and $\{S_1,\ldots,S_{\sqrt{n}-1}\}$ of size $O(\sqrt{n})$ such that:*
   (a) *There are no edges between nodes in V_i and nodes in V_j unless $i=j$.*
   (b) *There are no edges between nodes in V_i and S_j unless $i=j$ or $i=j+1$.*
   (c) *There are no edges between nodes in S_i and S_j unless $i=j$ or $j=i+1$.*
2. *G' is constructed from G by (repeatedly) transforming edges into paths of length 3.*

Proof: Start with G. Find a planar separator S of size $\sqrt{n}$ dividing G in W_1^ℓ and W_1^r both of size about $\frac{1}{2}\times n$.

Next find a separator T_1^ℓ in the planar graph W_1^ℓ of size $\sqrt{\frac{1}{2}\times n}$ dividing it into W_2^ℓ and W_2^r both of size $\frac{1}{4}\times n$. Continue this process until W_k^ℓ is of size $\leq\sqrt{n}$. Recursively we thus create a tree-like structure representing a partition of G into subsets. The leaves represent subsets of G of size $\leq\sqrt{n}$ and the internal nodes represent "separators" of size $\sqrt{\frac{n}{2^\ell}}$ in which $\ell\in O(\log n)$ stands for the level of the internal node in the tree.

To avoid confusion we will use the names 'separator' and 'leaf' for the nodes in the tree reserve the word 'node' for nodes in G and G'.

Now from the construction we may easily infer that there are no edges between nodes in subsets represented by separators and leaves on different paths in the tree. They have a common "ancestor-separator". Hence the only edges that can present a conflict with the situation we want to achieve run from "higher order" separators to "lower order" separators, or from separators to leaves. These edges may spoil the linear construction we may get from flattening the tree in an inorder, since they connect nodes that should be separated. The indicated repair for this situation is inserting extra nodes into the "lower order" separators crossed by these edges, break up the edge such that it runs through such a node, and argue that the number of thus inserted extra nodes is tollerable.

As the degree of any node in G is bounded by c, the number of edges emanating from separator T cannot exceed $c \times |T|$. Hence for any separator T_0 on a path in the tree the splitting up of edges that run across such a separator cannot mean an addition of more than $\sum_T c \times |T|$ extra nodes where the sum is taken over all ancestors of T_0. This means that for any separator there can be at most $\sum_{\ell=0}^{\log n} c \times \sqrt{\frac{n}{2^\ell}}$ extra nodes necessary. Hence any separator contains at most $O(\sqrt{n})$ nodes after this operation, and the conditions of the theorem are met. □

A *computation graph* of some machine model's computation on a given input is a graphical representation of this computation. E.g., for a single-tape Turing machine the computation can be represented by nodes in which tape symbols and the state of the machine are stored. The nodes in the graph represent tape cells and steps in time. A cell on the tape is represented by a node in the graph for each time it is visited by the tape head. There is an edge between nodes i and j if they represent the same tape cell c and j represents "the next time" that tape cell c is visited by the head, and there is an edge between i and j if j represents the tape cell that is visited immediately after visiting the tape cell represented by i. For locality of consistency check needed in corollary 1 below it seems necessary that there's also room for an integer in the node in which the number of the tape cell can be stored.

The computation graph for a Turing machine may be checked by pairwise comparison of nodes. It is consistent if and only if for all nodes that are connected by edges that mean that they represent the same tape cell—and hence bear the same number—the difference in symbol is consistent with the program (The head is present by definition, and the state of the machine is also information present in the node), and all nodes connected by time-edges must have the direction and state transition consistent with the program.

For machine models that have a *planar* computation graph for which consistency can be checked through pairwise comparison of nodes theorem 5 has the following implication.

Corollary 1 *Any computation which has a planar computation graph of n nodes, in which each node can be described using $S(n) \in O(\log n)$ tape cells and for which the consistency of the computation graph can be checked by pairwise comparison of nodes can be simulated on a single-tape NTM in space $\sqrt{n} \log n$ and time $n\sqrt{n} \log^2 n$*

Proof: According to theorem 5 the computation graph can be transformed to a planar graph with the structure $V_1, \ldots, V_{\sqrt{n}}, S_1, \ldots, S_{\sqrt{n}-1}$. Some nodes in S_i are not part of the original computation graph these nodes are used for "passing through information", i.e. two nodes v_i and v_j of the original computation graph are connected by a path of these nodes. If the segment that contains v_i comes before the segment that contains v_j then all of these intermediate nodes contain the information of v_i.

Now consistency of the computation graph can be verified by the following process:

1. Guess V_1, and S_1 and load them on a tape. Note that for each node a pointer giving the location of connected nodes takes up $O(\log n)$ space.

2. Verify the part loaded by pairwise comparison of nodes. This involves keeping and updating a counter of size $\log n$ under the head for a cost of $\log^2 n$ time, checking off pairs for a cost of the square of the number of tape cells occupied.

3. Guess a next part of the graph consisting of a new V_i and S_i if the tape contains a treated block that has no edges in common with part then this part of the tape can be reused. Hence there are never more than 3 blocks of size $\sqrt{n}\log n$ tape cells in use. Now repeat step 2 with the new blocks. Once $V_{\sqrt{n}}$ is thus verified the entire computation graph is verified and we can accept if and only if we have found an accepting configuration on the way.

□

An immediate corollary to this is that the single-tape Turing machine operating in time $T(n)$—and having therefore a computation graph of $T(n)$ nodes which can be stored in $O(\log T(n))$ cells per node—can be simulated on a single-tape Turing machine in space $\sqrt{T(n)}\log T(n)$, but in view of theorem 1 this is not surprising. It is however surprising that the corollary holds for any machine model for which the computation graph can be proven to be planar. In the next section we will show a particular nice example of this.

4 Example: The single-queue machine

In this section we will show that the computation graph belonging to a computation of a machine with a single queue data structure is planar, and therefore the single queue satisfies the conditions of corollary 1. A queue can be thought of as a Turing machine with an input tape and a single work tape. Instead of a single read-write head on the work tape however the queue has a separate read and a separate write head, initially positioned on the leftmost cell of the semi infinite work-tape. The queue can write a symbol on the tape, after which the write-head moves to the right, or it can read a symbol from the work-tape, after which the read head moves to the right. If the queue attempts to read when the positions of read and write head are equal, then the computation aborts.

M. Li [9] has shown that a single queue machine operating in time $T(n)$ can be simulated on a single tape in time $T(n)\sqrt{T(n)}\sqrt{\log(T(n)}$. Therefore theorem 1 means that a single que can be simulated on a single tape NTM in space $\sqrt{T(n)\sqrt{T(n)}\sqrt{\log T(n)}}$ in time $T(n)\sqrt{T(n)}\sqrt{\log(T(n))}$. However we can show that the single queue has a planar computation graph and that we can therefore obtain a simulation of a single queue on a single tape NTM which is substantially more space-efficient.

The *computation graph* of the single queue consists of three types of node.

1. Nodes in which a read (from the queue) is executed.
2. Nodes in which a write (to the queue) is executed.
3. Nodes in which neither a read nor a write is executed.

All three types of node can execute a read-from-the-input instruction. This corresponds to a change of position for the input head. Of course there are also the two flavors (on- and off-line) here. Nodes are connected via edges if they represent consecutive steps in time or a symbol is written on the queue in one node which is read from the queue in the second.

Consistency of the computation graph can be checked as follows.

1. Nodes that are connected via time edges must contain state descriptions that are consistent with the program. Positions of read and write head for the queue must be equal unless the a read-to-the-queue or write-to-the-queue is performed[3].

2. Nodes that are connected via read-write edges must contain the same queue-symbol. Moreover the position of the read head described in the read node must agree with the position of the write head in the write node.

3. Nodes that contain a read-from-the-input-tape instruction must have a change of position of the input head consistent with the program. Note that for this purpose, the position of the input head must be held in all nodes and changes in this position must also be checked during the verification of the time edges.

It is now clear that $O(\log T(n))$ tape cells are enough to describe nodes of a computation graph of a single queue operating in time $T(n)$ on a single-tape Turing machine, and that consistency can be checked via pairwise comparison of nodes. Depending on an on- or off-line situation an overall check for consistency with the input has to take place. This can cost an overhead of as much as $T(n) \times n \times \log^2 n$ in the off-line case. However since this overhead is already consumed by the time overhead of corollary 1 in case $n \leq \sqrt{T(n)}$ we can ignore this. To meet the conditions of corollary 1 we only need to show that the computation graph of a single queue is planar. For this we have a simple and elegant proof.

Theorem 6 *The computation graph of a single queue can be embedded in the plane.*

Proof: Divide the nodes of the computation graph into three sets.

1. V_1 is the set of nodes that execute a write-to-the-queue instruction.

2. V_2 is the set of nodes that execute a read-from-the-queue instruction.

3. V_3 is the set of remaining nodes.

We assume that all three sets are ordered chronologically. Now the entire computation graph can be embedded in the plane as follows.

- Nodes in V_1 are laid along the positive x-axis in chronological order towards $+\infty$.
- Nodes in V_2 are laid along the negative x-axis in chronological order towards $-\infty$.

Nodes in V_1 are connected to nodes in V_2 via read-write edges that run in the positive y-halfplane. It is quite easy to see that these edges can never cross. In the negative y-halfplane edges run that connect nodes in V_1 with nodes in V_1 or V_2 (and vice versa) according to their chronological order. It is also quite easy to see that these edges can never cross. Finally the nodes in V_3 are placed *on* the edges that run in the negative y-halfplane—thereby of course replacing such edges by an edge-node-edge triplet—in chronological order between the right pairs of nodes in V_1 and/or V_2. □

From theorem 6, the possibility of local verification and corollary 1 we then get.

[3]If there are (as in the case of the Turing machine) more than one type of edges present in the nodes (each of which must be validated by pairwise comparison) this step may be repeated several (but always a constant number of) times

Corollary 2 *Any language accepted by a single queue in time $T(n) \geq n^2$ can be accepted by a single-tape NTM in space $\sqrt{T(n)}\log T(n)$ and time $T(n)\sqrt{T(n)}\log^2 T(n)$*

5 Conclusions

Corollary 1 may set the stage for numerous simulation results on a single-tape Turing machine. Once planarity of the computation graph is proven, and the graph can be checked locally for consistency the simulation is given. A first candidate may be the two-stack machine which also has a planar computation graph [14]. It is however annoying to say the least that theorem 1 presents a sharper result than can be obtained from the planarity of the computation graph and corollary 1. It may be possible to get rid of the polylog factors in time and space overhead. The overhead in space stems from the fact that the nodes of a computation graph for a single-tape Turing machine seem to need counters to keep track of the head position. The overhead in both time and space stems from the fact the pointers in a graph of $\sqrt{n}$ nodes need $\log n$ space. Now if an argument can be found for storage of the graph in such a way that consistency can be checked without keeping track of the head position, and nodes can be laid on the tape such that the relative addresses—i.e. pointers—cannot become too large (preferably of constant size) then the polylog overhead vanishes. The $\sqrt{T(n)}$ time overhead stems from the fact that pairwise comparison of nodes needs quadratic time. Therefore time $T(n)$ is spent in the pairwise verification of sections of the computation graph. To get the result of theorem 1 it seems that an observation is needed that allows a linear check of a section of the computation graph for a cost of $\sqrt{T(n)}$. Only if such obstacles are overcome then the general result can become as sharp as the result in theorem 1. Much work remains.

The question of whether the time overhead can be completely abolished in the nondeterministic case is particularly interesting since it either shows a difference between determinism and nondeterminism for the particular model—if the time overhead can not be abolished for the deterministic model—or it shows a real difference between time and space for the model, in case it can. It is at present unclear if the results in this paper have any deterministic counterparts. This therefore also remains an object of further research.

Bibliography

[1] Galil Z., R. Kannan & E. Szemeredi. *On nontrivial separators for k-page graphs and simulations by onep-tape Turing machines.* In Proc. of 18th ACM Symp. on Theory of Computing, pp. 39-49, 1986.

[2] Fürer M. *The tight deterministic hierarchy.* Report on the 1st GTI workshop Univ. of Paderborn (1983) pp96-104. In Proc. of 14th ACM Symp of Theory of Computing (1982) pp. 8-16.

[3] Hopcroft J.E. & J.D.Ullman. *Formal Languages and Their Relation to Automata.* Addison-Wesley (1969).

[4] Hopcroft J.E. & J.D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison Wesley (1979).

[5] Hopcroft J.E & J.D. Ullman. *Relations between time and tape complexities.* J. Assoc. Comput. Mach. **15** (1968) 414-427.

[6] Hopcroft J.E., W.J. Paul & L. G. Valiant. *On time versus space.* J. Assoc. Comp. Mach. **24** (1977) 322-337.

[7] Ibarra O.H. & S. Moran. *Some time-space tradeoff results concerning single-tape and off-line TMs.* SIAM J. Comput **12** (1983) 389-394.

[8] Lorys K. and M. Liskiewicz.*Two applications of Fürer's counter to one-tape nondeterministic TMs.* in M.Chytil, L. Janiga and V. Koubek (eds.), proc. Mathematical foundations of computer science, Springer Lecture notes in computer science **324** (1988) pp445-453.

[9] Li M. *Simulating two pushdown stores by one tape in n**1.5 time.* In proc. 26th IEEE Symp. of Foundation of Computer Science (1985) pp. 56-64. Also in Journal of Computer and System Sciences **37**:1 (1988) 101-116.

[10] Li M. & P.M.B. Vitányi. *Tape versus queue and stacks: the lower bounds.* Information and Computation, **78**:1(1988) 56-85.

[11] Li M. & P.M.B. Vitányi. *Two decades of Kolmogorov complexity.* In proc. 3rd Structure in Complexity Theory Conference (1988) pp. 80-101. Also available in Russian language in Uspekhi Math. Nauk (Russian Mathematical Surveys), **43**:6 (1988) 129-166.

[12] Lipton R.E & R.E. Tarjan *A separator theorem for planar graphs.* SIAM J. of applied mathematics **36**:2 (1979) 177-189.

[13] Maass W.*Combinatorial lower bound arguments for deterministic and nondeterministic Turing machines*. Trans. Amer. Math. Soc. **292** (1985) 675-693.

[14] Neuféglise H. *Planar computation graphs*. Master's thesis, University of Amsterdam (1989)

The ITLI Prepublication Series

1986

86-01 The Institute of Language, Logic and Information

86-02 Peter van Emde Boas — A Semantical Model for Integration and Modularization of Rules

86-03 Johan van Benthem — Categorial Grammar and Lambda Calculus

86-04 Reinhard Muskens — A Relational Formulation of the Theory of Types

86-05 Kenneth A. Bowen, Dick de Jongh — Some Complete Logics for Branched Time, Part I Well-founded Time, Forward looking Operators

86-06 Johan van Benthem — Logical Syntax

1987

87-01 Jeroen Groenendijk, Martin Stokhof — Type shifting Rules and the Semantics of Interrogatives

87-02 Renate Bartsch — Frame Representations and Discourse Representations

87-03 Jan Willem Klop, Roel de Vrijer — Unique Normal Forms for Lambda Calculus with Surjective Pairing

87-04 Johan van Benthem — Polyadic quantifiers

87-05 Víctor Sánchez Valencia — Traditional Logicians and de Morgan's Example

87-06 Eleonore Oversteegen — Temporal Adverbials in the Two Track Theory of Time

87-07 Johan van Benthem — Categorial Grammar and Type Theory

87-08 Renate Bartsch — The Construction of Properties under Perspectives

87-09 Herman Hendriks — Type Change in Semantics: The Scope of Quantification and Coordination

1988

Logic, Semantics and Philosophy of Language:

LP-88-01 Michiel van Lambalgen — Algorithmic Information Theory

LP-88-02 Yde Venema — Expressiveness and Completeness of an Interval Tense Logic

LP-88-03 — Year Report 1987

LP-88-04 Reinhard Muskens — Going partial in Montague Grammar

LP-88-05 Johan van Benthem — Logical Constants across Varying Types

LP-88-06 Johan van Benthem — Semantic Parallels in Natural Language and Computation

LP-88-07 Renate Bartsch — Tenses, Aspects, and their Scopes in Discourse

LP-88-08 Jeroen Groenendijk, Martin Stokhof — Context and Information in Dynamic Semantics

LP-88-09 Theo M.V. Janssen — A mathematical model for the CAT framework of Eurotra

LP-88-10 Anneke Kleppe — A Blissymbolics Translation Program

Mathematical Logic and Foundations:

ML-88-01 Jaap van Oosten — Lifschitz' Realizabiility

ML-88-02 M.D.G. Swaen — The Arithmetical Fragment of Martin Löf's Type Theories with weak Σ-elimination

ML-88-03 Dick de Jongh, Frank Veltman — Provability Logics for Relative Interpretability

ML-88-04 A.S. Troelstra — On the Early History of Intuitionistic Logic

ML-88-05 A.S. Troelstra — Remarks on Intuitionism and the Philosophy of Mathematics

Computation and Complexity Theory:

CT-88-01 Ming Li, Paul M.B.Vitanyi — Two Decades of Applied Kolmogorov Complexity

CT-88-02 Michiel H.M. Smid — General Lower Bounds for the Partitioning of Range Trees

CT-88-03 Michiel H.M. Smid, Mark H. Overmars, Leen Torenvliet, Peter van Emde Boas — Maintaining Multiple Representations of Dynamic Data Structures

CT-88-04 Dick de Jongh, Lex Hendriks, Gerard R. Renardel de Lavalette — Computations in Fragments of Intuitionistic Propositional Logic

CT-88-05 Peter van Emde Boas — Machine Models and Simulations (revised version)

CT-88-06 Michiel H.M. Smid — A Data Structure for the Union-find Problem having good Single-Operation Complexity

CT-88-07 Johan van Benthem — Time, Logic and Computation

CT-88-08 Michiel H.M. Smid, Mark H. Overmars, Leen Torenvliet, Peter van Emde Boas — Multiple Representations of Dynamic Data Structures

CT-88-09 Theo M.V. Janssen — Towards a Universal Parsing Algorithm for Functional Grammar

CT-88-10 Edith Spaan, Leen Torenvliet, Peter van Emde Boas — Nondeterminism, Fairness and a Fundamental Analogy

CT-88-11 Sieger van Denneheuvel, Peter van Emde Boas — Towards implementing RL

Other prepublications:

X-88-01 Marc Jumelet — On Solovay's Completeness Theorem

1989

Logic, Semantics and Philosophy of Language:

LP-89-01 Johan van Benthem — The Fine-Structure of Categorial Semantics

Computation and Complexity Theory:

CT-89-01 Michiel H.M. Smid — Dynamic Deferred Data Structures

CT-89-02 Peter van Emde Boas — Machine Models and Simulations

CT-89-03 Ming Li, Herman Neuféglise, Leen Torenvliet, Peter van Emde Boas — On Space efficient Simulations

Other prepublications:

X-89-01 Marianne Kalsbeek — An Orey Sentence for Predicative Arithmetic

X-89-02 G. Wagemakers — New Foundations. a Survey of Quine's Set Theory