TWO-DIMENSIONAL TURING MACHINES
WITH UNINITIALIZED WORKSPACES

Robert Melville

TR 80-434

Department of Computer Science
Cornell University
Ithaca, New York 14853
Two-dimensional Turing Machines
with uninitialized workspaces

1-Problem Statement

The conventional two-dimensional TM model[HU] provides an infinite two-dimensional workspace assumed to be blank at the start of the computation. In other words, the machine assumes that any cell, not previously visited in a computation, contains some distinguished symbol b, an element of the workplane alphabet. As an example of a computation making essential use of this assumption, consider the two dimensional self-crossing problem. The machine processes on-line an input sequence over the alphabet \( \{N, S, E, W\} \). Interpreting each symbol as a unit step in the corresponding compass direction, the machine must report whenever the path so described "crosses" itself, i.e. when the current endpoint of the path coincides with some previous point of the path.

A solution by a two dimensional TM is easy if we assume the workplane to be blank initially. The input alphabet of the machine is \( \{N, S, E, W\} \) and the workplane alphabet is \( \{b, 1\} \). The machine reads input symbols and moves its head accordingly. If a move is to a blank cell, the machine prints a 1; if the move is to a cell containing a 1, the machine reports a self-crossing. The time to process n input symbols is clearly \( O(n) \).

In this paper we criticize the above solution as being rather cheap — at least it is not quite fair to assign this algorithm only linear running time. Suppose the sequence of the first \( n/2 \) input symbols is \( (NE)^{n/4} \). The machine records a path as in figure 1. The second half of the input could cause a self-crossing at any of the \( O(n^2) \) cells inside the dotted square, so these cells must be initially blank. A physical device implementing two-dimensional
storage would have to initialize these cells to blank before they are visited. This observation casts some doubt on the linear running time of the simple solution given above.

We propose a more realistic model for multi-dimensional TM storage: a cell of the work space, not previously visited during the computation, may contain any symbol from the workspace alphabet. This stipulation invalidates the simple solution to the self-crossing problem since the machine could now get confused reading spurious 1's.

If a (multi-dimensional) Turing Machine M reading input data I preforms exactly the same computation on I for any initial contents of its workspace, we say the machine is initialization-independent. The construction of this paper shows that two-dimensional Turing Machines may be made initialization-independent with only a slight loss of speed. A computation of n steps in a blank workplane is simulated in time $O(n(\log n)^r)$ for some fixed $r$. 

![Figure 1](image-url)
-3-

Shönhage[SH] calls bounds of this form "quasi-linear" and we propose the notation $Q(f(n))$ to denote $O(f(n)(\log n)^r)$ for any fixed $r$. In particular, arithmetic operations on integers of size polynomial in $n$ can be done by a TM in time $Q(1)$, analogous to the $O(1)$ cost for such operations on a RAM[AHU]. We restate our result in this notation:

**Theorem**

The first $n$ steps of the computation of a one-head two-dimensional Turing Machine can be simulated in $Q(n)$ steps by an initialization-independent two-dimensional machine with two heads.

**Construction**

The only difficult part of the simulation is the representation of the simulated workplane and head in uninitialized storage. I/O manipulation and state transitions can be implemented in any of several ways, all affecting the running time by only a multiplicative constant.

In the sequel we speak as if $n$, the number of steps to be simulated, is known in advance. This is not possible in an on-line simulation, but we can achieve the same effect by the following device. For simulation steps $2^t$, ..., $2^{t+1}-1$ the machine uses the construction with $n'' = 2^{t+1}$. The simulating machine also records the history of operations modifying the simulated workspace. Before simulating step $2^{t+1}$, the machine will spend time $Q(2^{t+1})$ to repeat steps $1, 2, ..., 2^{t+1}$ but with $n'' = 2^{t+2}$. Now steps $2^{t+1}, ..., 2^{t+2}-1$ can be simulated on the updated workspace representation. Since $2^{t+2} - 2^{t+1} = 2^{t+1}$ we can charge the cost of the updating against the cost of simulation steps $2^{t+1}, ..., 2^{t+2}$ and the simulation is slowed down by only a constant factor.
Let \( n = 2^t \). The simulating machine will organize its workplane as a grid of records, each record having two fields able to store an integer in the range \( 1 \ldots t \). Therefore a record is \( \sqrt{2 \log t} \) on a side and the whole workplane is magnified by this factor. If a record occupies locations

\[
\{ i \sqrt{2 \log t}, \ldots, (i+1) \sqrt{2 \log t} \} \times \{ j \sqrt{2 \log t}, \ldots, (j+1) \sqrt{2 \log t} \}
\]

we say it has coordinates \((i, j)\).

The simulating machine will obey the following conventions:

1. a symbol \( \sigma \) written by the simulated machine at location \((x,y)\) on its workplane will be stored in record \((2x+1, 2y+1)\) of the simulating machine - that is, a whole record is used to contain a single symbol. Call these "data records".

2. a record with one odd and one even coordinate is called a border record; it is either uninitialized or contains some status information.

3. a record with even coordinates is uninitialized.

Figure 2 shows a portion of the workplane for \( t=4 \). The important point is that before moving to any data record the head must cross a border record.

The simulating machine maintains a recursive partition of its workplane into frames, sub-frames, etc. that delimit areas not yet visited. Delimiting frames are constructed in time proportional to the perimeter rather than the area and this is the essential reason for the efficiency of the construction. Because the head movements of the simulated machine form a path, the head must cross the perimeter of a frame before accessing any cell inside the frame.
Figure 2

An n x n frame surrounds the workplane at level 0. In general, a level i frame has a perimeter of $4 \times 2^{t-i}$ border records and four sub-frames at level i+1. A level i frame is marked by border records containing the level number i in one field. A border record could be common to two adjacent frames; hence the need for two fields. If two frames are adjacent vertically (respectively horizontally) the records forming the common border are type "vertical" (respectively "horizontal"). Moreover, some fields maybe marked as "corner" to indicate the limits of a frame. These coding details are illustrated in figure 3.

The second head performs two functions during the simulation:
(1) record workplane contents for re-organization described above. This could be a sequence over the alphabet

\{\text{shift}_\delta \mid \delta \in \{N,S,E,W\}\} \cup \{\text{print}_\sigma \mid \sigma \in \text{workspace alphabet}\}

(2) arithmetic on integers in the range 1...n or 1...t.

The arithmetic is used to sub-divide frames and to move the first head in increments of \(\sqrt{2 \log t}\) so that it respects record boundaries. The arithmetic is carried out in some fixed number of registers of length \(t\). The workplane description and registers are written as shown in figure 4. This data can be safely stored and manipulated in uninitialized storage because of its one dimensional arrangement. Moreover, any operation by the second head will be done in time \(O(1)\).

Figure 5 is the program of the simulating machine. The operation of the subroutine "enter-frame" is explained more fully in figure 6.

3-Runtime time analysis
\[ O(n) \quad Q(1) \]

\[
\begin{array}{|c|c|}
\hline
\text{workplane description} & \text{registers} \\
\hline
\end{array}
\]

\[ ^\wedge \text{Head 2} \]

\[
\begin{array}{|c|}
\hline
\wedge \\
\hline
\text{Head 1} \\
\hline
\end{array}
\]

\[ n\sqrt{2} \log t \]

\[ n\sqrt{2} \log t \]

\[ \text{Figure 4} \]
construct $n \times n$ level 0 frame;

enter-frame(1, 0, "E");

**do** simulated machine not in final state **+**

determine next move from transition table;

**if** state change **+**

something $O(1)$

**[]** printing move **+**

print symbol in data record
currently under head 1 and
record move with head 2

**[ ]** shift $\delta$ **+**

enter-frame( $i$, $j$, $\delta$ ) where

$(i,j)$ is the border record crossed
by the shift;

record move by head 2

**fi**

**od**

enter-frame( $i$, $j$, $\delta$ ) :

let $k$ be the level of the frame entered by crossing border
record $(i,j)$;

**do** $k < t$ **+**

trace perimeter of frame entered by crossing
record $(i,j)$ to increment level numbers then
sub-divide as in figure 6

**od**

Figure 5 - simulation algorithm
Figure 6
This illustrates the sub-division operation upon frame entry. From the type of the border record (in this case vertical) and the shift $\delta$ the machine can determine how to trace the perimeter of the frame. The machine calculates the length of the side and counts this length off from the four original corners. The perimeters of the sub-frames are initialized with the proper level values and the head returns to record $(i,j)$.

The key to the running time analysis is the following observation about paths.
Lemma
Let the plane be marked off in a grid of frames of side \( c \). A path of length \( n \) can visit at most \( \frac{9n}{c} \) different frames.

Proof
Let \( F \) be the set of frames visited by the path. Construct \( G \subset F \) by the following "thinning" procedure:

To start, set \( G = F \). As long as \( G \) contains an element \( g \) such that one of the eight immediate neighbors of \( g \) is also in \( G \), delete from \( G \) all immediate neighbors of \( g \).

Upon termination \( \#G \geq \#F/9 \).

Let \( g_1, g_2, \ldots, g_p \) be the sequence of members of \( G \) in the order in which they are visited by the path.

\( p \geq \#G \) since every member of \( G \) is visited at least once

\( p \leq n/c \) since \( c \) steps of the path are needed to get from any \( g_k \) to \( g_{k+1} \).

Collecting:
\[
\frac{n}{c} \geq p \geq \#G \geq \#F/9 \Rightarrow \#F \leq \frac{9n}{c}
\]
Q.E.D.

The cost to enter a level \( i \) frame is the cost to trace the perimeters of frames at level \( i \), level \( i+1 \), \ldots, level \( t \). This is:

\[
\sum_{k=i}^{t} Q(2^{t-k}) = Q(2^{t-i}) \quad (3.1)
\]

We apply the lemma to each of the \( t \) levels. The cost associated with level \( i \) is the number of level \( i \) frames entered times the cost to enter a level \( i \) frame. This is:

\[
\frac{9n}{2^{t-i}} \times Q(2^{t-i}) = Q(n) \quad (3.2)
\]

Since there are \( Q(1) \) levels, the entire cost is \( Q(n) \).