# LINEAR TIME SIMULATION OF MULTIHEAD TURING MACHINES WITH HEAD - TO-HEAD JUMPS*

Walter J. Savitch**
Paul M.B. Vitányi
Mathematisch Centrum
Amsterdam, The Netherlands.

ABSTRACT

The main result of this paper shows that, given a Turing machine with several read-write heads per tape and which has the additional one move shift operation "shift a given head to the position of some other given head", one can effectively construct a multitape Turing machine with a single read-write head per tape which simulates it in linear time; i.e. if the original machine operates in time T(n), then the simulating machine will operate in time cT(n), for some constant c.

*'Ik zing en spring, de maan die heeft een ring.*
*Quisque sibi proximus ... .'*
*In: Marten Toonder, De Liefdadiger.*

## 1. INTRODUCTION

A model frequently used in the analysis of the time complexity of algorithms is that of a multitape Turing machine. This model allows only serial access to the information held in storage. Since many algorithms are most naturally stated assuming immediate access to several locations in memory, it is useful to have models which allow more flexible memory access and which yield run times which are equivalent to multitape Turing machine run times. A number of such models have been shown to be equivalent to the basic multitape Turing machine model in this sense. In [3] Fisher and Rosenberg proved that a multitape Turing machine, which is permitted a single fixed *reset* square on each of its tapes and is given the ability to execute a jump (fast rewind) to the reset square in a single machine step regardless of the distance of the tape head to that square, can be simulated in real time by an ordinary multitape Turing machine. With a similiar motivation of handling files requires several points of immediate two-way read-write access Fisher, Meyer and Rosenberg [2] showed that a Turing machine with several multihead tape units can be simulated in real time by an ordinary multitape Turing machine. Leong and Seiferas [6] have recently improved this simulation so that it can be performed with fewer tape units. Their simulation requires 4k - 4 tapes to simulate one k-head tape unit. Stoss [7] proved

*) This research was supported by NSF grant MCS-74-02338A01 and is registered by the Mathematical Centre as report IW 79/77.
**) On leave from the Computer Science Division, Dept. APIS, University of California, San Diego.

earlier that, given a multihead Turing machine with storage consisting of a 2 head tape unit, there exists a multitape Turing machine with 2 storage tapes which simulates it in linear time. Additional motivation for studying these problems can be found in the cited references. There the question arises whether the real time simulation of an instantly rewindable tape unit can be generalized to multihead tape units or to more general reset operations. Here we extend the capabilities of both multitape Turing machines with fast rewind to a reset square [3] and multihead Turing machines [2] as follows. We consider k head tape units with the ability to execute a jump from one head to the position of another head in a single machine step regardless of the distance between the heads concerned. Our main result shows that a k head tape unit with the ability to execute such jumps can be simulated in linear time by 8k - 8 ordinary single head tape units.

This main result represents a substantial time savings over the obvious simulation algorithm which requires square time when the simulated unit has more than two heads. In the obvious simulation algorithm jumps are simulated by simply moving a head to its target square one square at a time. This yields a multihead tape unit without jumps which can, in turn, be simulated in real time by single head units by means of the results cited above. A brief look at the obvious simulation algorithm will reveal the problems that arise in attempting to get an efficient simulation.

Consider first the obvious simulation algorithm for the two head case. Since the heads are assumed to be together at the origin at time 0 and have moved apart at most $2t_1$ squares at the first time $t_1$ when a jump from one head to the other is performed, it takes the jumping head at most $2t_1$ time to run over to the target head in the simulation. Therefore, altogether less than or equal to $3t_1$ time units are consumed by the simulating device when the jump is executed and both heads are together again. Similarly, with the next jump at time $t_2$ it takes at most $2(t_2-t_1)$ time units to execute the jump and the total time is less then or equal to $3t_2$, and so on. Hence, a two head ordinary tape unit needs at most 3t steps to simulate t steps of a two head tape unit with head-to-head jumps. A straight forward generalization to the k head case does not work. E.g., with 3 heads the two outer-most heads may run apart at full speed while the remaining head keeps jumping between them. The simulation would cost about $t^2$ steps for t steps of the simulated device. We will show, however, that a k head tape unit with head-to-head jumps can be simulated with ordinary multihead tape units at at most the cost of increasing the running time by a constant multiplicative factor.

One sense in which our simulation algorithm is weaker than related previous work is that we obtain only a linear time simulation rather than a real time simulation. A closer inspection of the proof shows however, that it is not really very much weaker. To simulate a Turing machine with head-to-head jumps by an ordinary, multitape Turing machine we proceed in two steps. First, we simulate a multihead Turing machine with jumps by a multihead Turing machine without jumps in linear time. Then

we appeal to known results for a real time simulation of the multihead machine by a machine with but one head per tape. This first linear time simulation is accomplished with only a negligable increase (one symbol) in the tape alphabet. The previous results concerning real time simulation by multitape Turing machines of tape units with added capabilities, like in [2], [3] and [6], were obtained by a large increase in the tape alphabet. This means, that when we keep the alphabet fixed to that of the simulated machine, all of the above algorithms need be implemented such that they run in linear and not in real time. So, when viewed in the (more realistic) framework of machines with a fixed tape alphabet, the linear time result is not really that much weaker than the "real time" results obtained previously. Of course, since we also use one of these real time results in our simulation, our simulation can not be any faster and indeed is slower by a constant multiplicative factor which is easily computed from the proofs.

## 2. PRELIMINARY DEFINITIONS

We assume that the reader is familiar with the definition of a multitape Turing machine as a given in, for example, Hopcroft and Ullman [5]. All our models will be variants of this basic model. As in [2], we will assume that our machines have a one-way, read-only input tape and a one-way write-only output tape. Our definition of simulation is the same as that given in [2], namely "black box" simulation of the input/output behaviour of the simulated device

We say that a machine $M_2$ *simulates* a machine $M_1$ if the input/output (I/O) behaviour of both machines is identical for all words in the input alphabet of $M_1$; that is , for a given input string, the ouput strings are the same for $M_1$ and $M_2$, and for each i, the output strings produced by the time the i-th input symbol is read are also the same for both machines. The simulation is said to be a *linear time* simulation provided there is a fixed constant c such that: if $M_1$ performs an I/O operation at time t, then $M_2$ will perform the same occurrence of that I/O operation at time ct or sooner. Using Hartmanis-Stearns speedup[4] we can frequently show that linear time simulations can be improved to simulations "without delay". The simulation is said to be *without delay* provided it satisfies the definition of linear time simulation with c set equal to one. The notion of simulation without delay is slightly weaker than the notion of real time simulation given in [3]. For real time simulation we require that if $M_1$ performs an I/O operation at time t then $M_2$ performs the same occurrence of that I/O operation at time exactly t. For simulation without delay, we merely require that $M_2$ perform the I/O operation at time t or sooner. In most contexts, the distinction between real time simulation and simulation without delay is unimportant.

We also apply the concept of real time to computations of an individual machine. A machine *runs in real time* if it reads a new input symbol in every step (until the input, if any, is exhausted) and writes an output symbol in every step of the computation. To accept a language in real time the machine must run in real time and must

indicate acceptance (when appropriate) as soon as the input has been read. Unless specifically stated to the contrary, all machines are deterministic and have a storage tape alphabet of at least two symbols. In order to establish out terminology, we repeat the following informal definition taken from [2].

A Turing machine (TM) consists of a finite state control unit, special one-way (one-head) read-only and write-only devices for *input* and *output*, and *storage*. For a *multihead* TM the latter consists of a two-way linear tape, unbounded at both ends and divided into an infinite number of discrete squares and a fixed number $k \geq 1$ of read-write heads. The term *tape* will always refer to the storage tape of the TM and never to an I/O device. Initially all heads on a tape are scanning the same tape square. A *step* in a computation of a multihead TM is uniquely determined by the state of the control unit, the ordered set of symbols scanned by the storage tape heads and the input head, and a partition of the set of heads into classes of heads which are scanning the same tape square. (In other words, the control unit can detect which heads are coincident). In one step the control unit may cause each head to write a symbol on the tape square scanned (several heads scanning the same tape-square must write the same symbol), independently shift each head at most one square in either direction, and enter a new state. As part of this single step it also has the options to advance the input head and/or output a symbol. The extension to a *multitape- multihead* TM is straight forward, and a *multitape* TM is such a device with but one head per tape. Note that each multitape-multihead TM can always be simulated by a multihead TM in real time by dividing the single tape in tracks, one for each tape, on which the appropriate heads compute. Now we introduce a device, called a jump TM, which is a TM with head-to-head jumps.

DEFINITION A *jump* TM is a multihead TM which at each instant of time can perform a move of the following form.

(i) A regular move as described above for a multihead TM followed by

(ii) A redistribution of the tape heads over a nonempty subset of the currently scanned tape squares.

All this constitutes a single one step move of the jump TM. The moves are uniquely determined as in the case of the ordinary multihead TM's.

## 3. MAIN THEOREM

The key result of this paper is the following:

THEOREM 1. *Let M be a jump* TM *with* k *heads and* s *tape symbols. Then M can be simulated by an ordinary multihead* TM $M'$ *with* $s + 1$ *tape symbols,* k *tapes with two heads apiece and one tape with* $k - 1$ *heads such that* $M'$ *needs at most* $(2k - 1)t$ *steps to simulate* t *steps of a computation by* M.

The proof of Theorem 1 proceeds by a series of Lemmas. First we introduce the

notion of a RJTM ("restricted jump Turing machine"). This device is the same as a jump TM except that when it redistributes heads, it *must leave at least one head at each scanned tape square.*

LEMMA 2. *A* k *head* RJTM $M$ *with* s *tape symbols can be simulated in real time by an ordinary multihead* TM $M'$ *with* s + 1 *tape symbols and* k *tapes with two heads per tape.*

PROOF Indicate the leftmost and rightmost tape squares ever visited during the previous computation of $M$ by a marker \$. Let $M'$ be a k tape TM with two heads per tape. On k - 1 tapes of $M'$ we maintain the k - 1 tape segments between two consecutive heads of $M$. On the remaining tape of $M'$ we maintain the tape segments between the rightmost head and the right marker, and the leftmost head and the left marker of $M$. The situation is depicted in Figure 1.

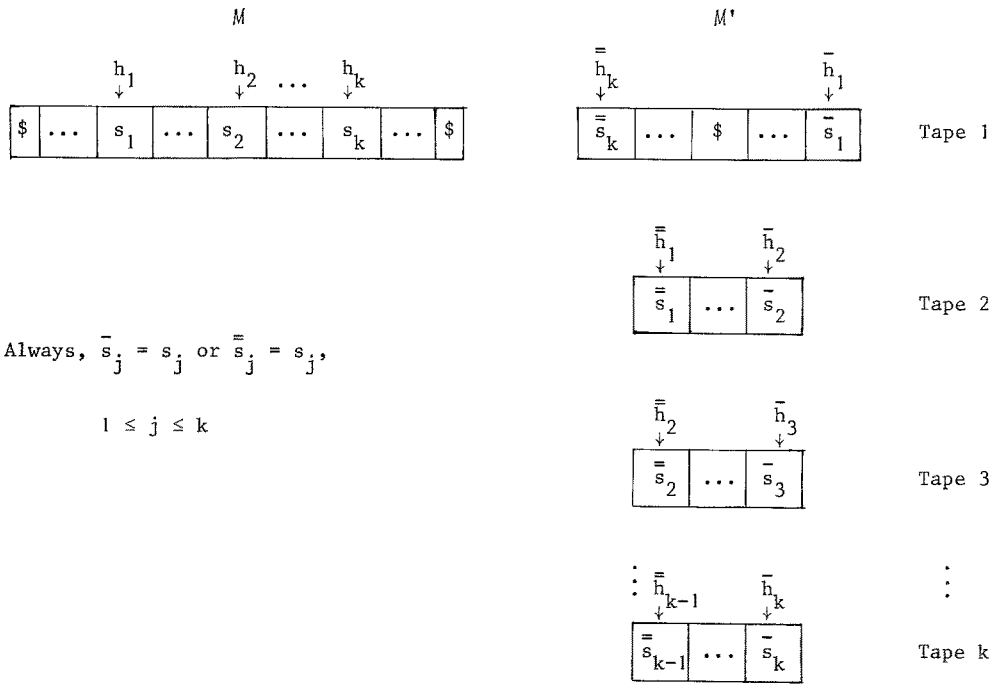


Figure 1. RJTM simulation

A move to the right of a head $h_\ell$, $1 \le \ell < k$, in $M$ is simulated in $M'$ by moving both copies of $h_\ell$ to the right. The copy of $h_\ell$ which is leftmost (i.e. the one on the $\ell + 1$ -th tape simulating the tape segment $(h_\ell, h_{\ell+1})$) leaves a blank square, and the copy of $h_\ell$ which is rightmost (i.e. the one on the $\ell$-th tape simulating the tape segment $(h_{\ell-1}, h_\ell)$ or $(\$, h_\ell)$ for $\ell = 1$) leaves the symbol it ought to have printed in the just scanned square. Similarly, for left moves of $h_\ell$, $1 < \ell \le k$. Tape 1 of $M'$ simulates both the left- and the rightmost tape segments of $M$ and when a head reads \$ the finite control remembers whether this was a left- or rightmost head and for all further

moves to the left or right, respectively, of this head assumes that it reads blank symbols and keeps the head copy on tape 1 immobile. Hence, at all times one head copy of $h_\ell$ in M' is scanning the symbol $h_\ell$ scannes in M. We leave it to the reader to verify that heads passing each other present no difficulties. It is also clear that if i heads coincide we have i - 1 tapes in M' on which the heads coincide, and a redistribution of the heads of M over the scanned tape squares just means a renaming of the heads in the simulating M'. This manipulation of head identities is performed by M' 's finite control unit. □

Suppose we want to simulate a jump TM M by an RJTM M'. To simulate a jump of $h_i$ to head $h_j$ in M all M' has to do is to have head $h_i$ run to one of its neighbors, and then, disappear and reappear in one step at the position of head $h_j$. All this costs the least time if the head which wants to jump runs to its nearest neighbor. To be able to do so, we equip our RJTM with a k - 1 head tape unit which keeps track of the lengths of the k - 1 tape segments induced by the k heads on M's tape. Each of these k - 1 heads keeps track of the length of one segment by positioning itself d squares to the right of an origin to record the fact that the segment has length d. In order to be able to tell which of two segments are shorter, the finite control keeps track of the left to right order of the k - 1 heads. The fact that the lengths of tape segments might be increased or decreased by two in one step is accomodated by giving each square a value of two and using the finite control to remember the remainder for odd lengths. All this can be done with but two tape symbols. (With the aid of the other tapes used in the simulation, it can even be done with just one tape symbol). By using this tape unit, the control unit can decide which head is a head's nearest neighbor, the right or the left one. By Lemma 2 we have therefore:

LEMMA 3 *A* k *head* RJTM M *capable of deciding whether the nearest neighbor of a head is the left or the right one can be simulated in real time by an ordinary multihead* TM M' *with* k *two head tape units and one* k - 1 *head tape unit. Furthermore, if* M *has* s *tape symbols, then* M' *need have only* s + 1 *tape symbols.*

LEMMA 4 *Given a* k *head jump* TM M, *we can construct a* k *head* RJTM M' *of the type described in Lemma 3 such that* M' *simulates* M *in at most* (2k - 1) *times the time used by* M. *Furthermore,* M' *may be taken to have the same tape alphabet as* M.

PROOF We will make a few simplifying assumptions about M, without losing any generality. We assume that the heads of M are numbered 1,2,...,k. If in a single move head i jumps to the square occupied by head j, then for this move, i is called a *jump head* and j is called a *target head*. By modifying the finite control of M so that it renames heads in a judicious manner, we can assume that, in every move, no head is both a jump head and a target head. Furthermore, we can assume that each jump head has a unique target head.

M' performs a step by step simulation of M. That is, if an M computation passes

through a certain sequence of storage configurations, then the simulating $M'$ computation will pass through exactly the same storage configurations. $M'$ will, however, require a number of steps to change its storage configuration in a way corresponding to a single step of $M$. In order to simulate a single step of $M$, $M'$ first performs a a direct simulation of the regular multihead TM part of $M'$s move (clause (i) in the definition of a move) and then $M'$ redistributes the tape heads. To redistribute the tape heads $M'$ moves the heads in numerical order, first repositioning head 1 if necessary, then repositioning head 2 and so forth. A jump head is repositioned by having it run to its nearest neighbor and then appear again at the position on the tape of the target head. Hence, to simulate t steps of $M$, $M'$ needs a number of steps equal to t plus the combined distances to nearest neighbors involved in simulating jumps $M$ executed during its t step computation. We formalize this by defining for a particular t step computation of $M$, the functions jump and cost. The domain of jump is the set integer time instances $\tau$, $1 \leq \tau \leq t$. If $1 \leq \tau \leq t$, then $\text{jump}_\tau$ denotes the value of jump at $\tau$; $\text{jump}_\tau$ is itself a partial function from K to K, where $K = \{1, 2,\ldots,k\}$. If head i is a jump head at time $\tau$ then $\text{jump}_\tau(i) = j$ where j is the target head for i; otherwise $\text{jump}_\tau(i)$ is undefined. The function cost has two arguements: a time instance $\tau$ and a head number i. If i is a jump head at time $\tau$, then $\text{cost}(\tau,i)$ is the distance from head i (of $M'$) to its nearest neighbor at the time $M'$ simulates the jump by head i; otherwise $\text{cost}(\tau,i) = 0$. So in this notation, $M'$ simulates a t step computation of $M$ in time $t' = t + \Sigma_{\tau=1}^{t} \Sigma_{i=1}^{k} \text{cost}(\tau,i)$ and we must show that $t' \leq (2k-1)t$. So it will suffice to show.

CLAIM

$$\sum_{\tau=1}^{t} \sum_{i=1}^{k} \text{cost}\ (\tau,i) \leq 2(k-1)t$$

PROOF OF CLAIM We depict the jumping pattern of $M$ by a tree-like diagram. Each arc of the tree is labeled by some i, $1 \leq i \leq k$, identifying the arc as corresponding to head i. The vertical $\tau$-coordinate measures elapsed run time starting at $\tau = 1$ above and ending with $\tau = t + 1$ below. For each integer coordinate $\tau$ there will be exactly k arcs, labelled $1,2,\ldots,k$, immediately above ($\tau > 1$) and immediately below coordinate $\tau$. The diagram is constructed as follows. The root is at $\tau = 1$; k arcs eminate from the root and are labeled $1,2,\ldots,k$. Suppose we have constructed the diagram up to some integer coordinate $\tau$. It is extended to $\tau + 1$ as follows.

(i) If at time $\tau$ head i is neither a jump nor a target head, then the arc labelled i is extended to $\tau + 1$.

(ii) If j is a target head and $i_1,i_2,\ldots,i_\ell$ are all i such that $\text{jump}_\tau(i) = j$, then the arcs labeled $i_h (1 \leq h \leq \ell)$ are terminated by a node at level $\tau$. The arc labelled with j is terminated at level $\tau$ by a node from which emanate $\ell + 1$ arcs labelled $i_1,i_2,\ldots,i_\ell,j$ which are extended to level $\tau + 1$.

Hence the diagram depicts the jumping history of $M$ up to time $t + 1$. A sample

diagram is given in Figure 2.

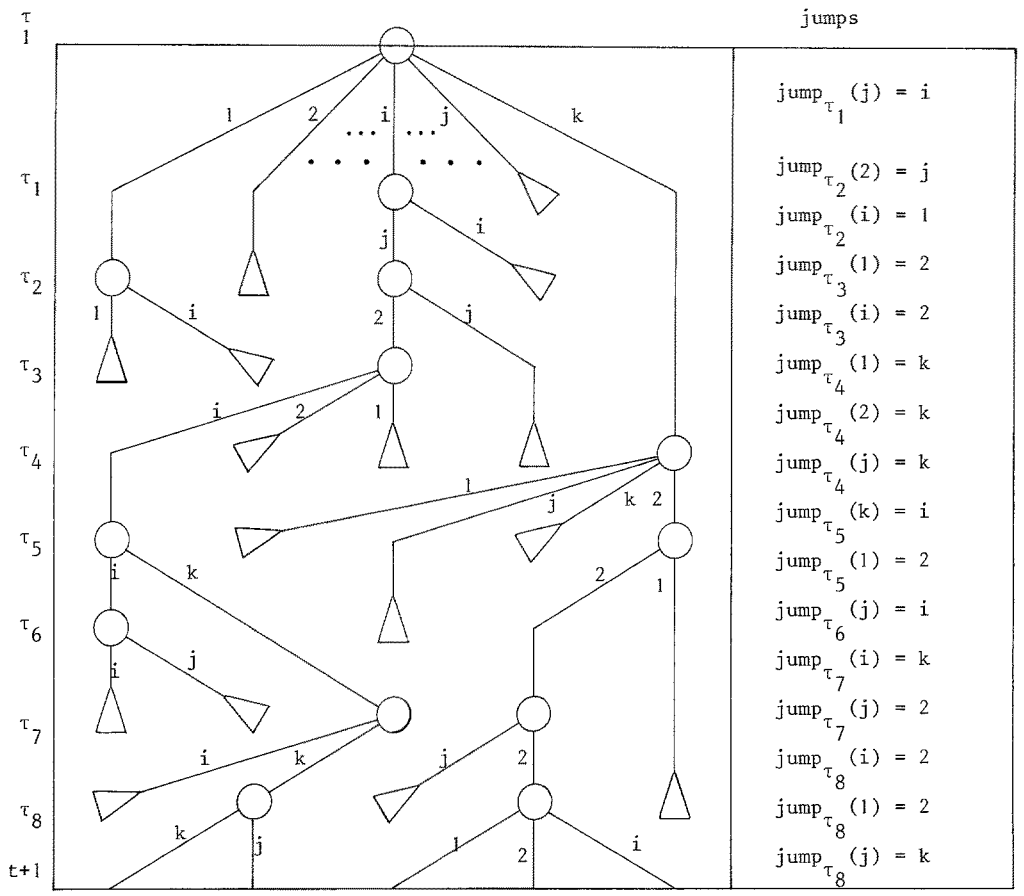


Figure 2. Tree-like Diagram

We now describe a pruning procedure for the diagram that associates with each terminal node an ancestor splitting node. If a terminal node represents a jump by head i at time $\tau$ and the ancestor node associated with it is at level $\tau'$, then, at the time $M'$ simulates this jump, head i will have a neighbor head at a distance of at most $2(\tau - \tau')$ squares away. So $\text{cost}(\tau,i) \leq 2(\tau - \tau')$. Thus the pruning procedure will allow us to estimate the values of the $\text{cost}(\tau,i)$ and ultimately the value of t'. The terminal nodes are pruned off for levels $\tau = 1,2,\ldots,t$ in that order and within each level $\tau$ in the numerical order of labels on arcs terminating at these nodes. In other words, each terminal node represents a jump in M and these jumps are pruned off in the order in which they are simulated by $M'$. To prune a node, simply remove the node and the branch leading up to its nearest splitting ancestor. This nearest splitting ancestor is the ancestor node associated with the terminal node. If a terminal node represents a jump by head i at time $\tau$ and the associated splitting node is at level $\tau'$, then define $f(\tau,i) = \tau'$. Note that the act of pruning may

change some splitting nodes to nonsplitting nodes. After each pruning the tree is changed and the pruning procedure for the next node to be pruned is performed on this changed tree. The pruning for the diagram in Figure 2 is illustrated in Figure 3.

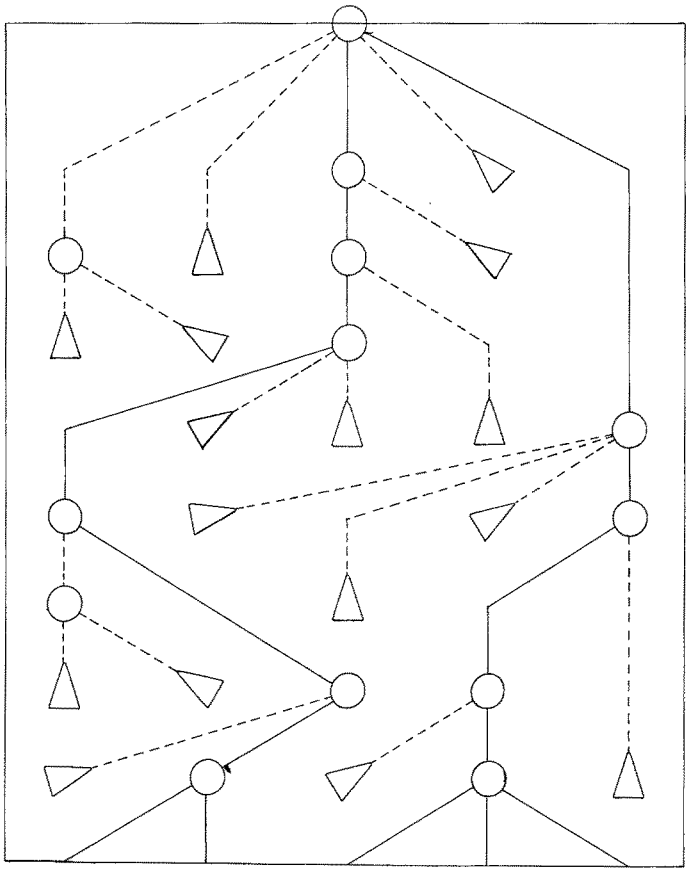

Figure 3. Pruned Tree-like Diagram

If a terminal node represents a jump by head i at time $\tau$ and the associated splitting node is at level $\tau'$, then the splitting node indicates that head i and some other head were together at time $\tau'$. Because of the order of the pruning, it also follows that this other head did not jump between times $\tau'$ and $\tau$. So when $M'$ wishes to simulate this jump by head i at time $\tau$, head i and the head which was coincident with i at time $\tau'$ can be at must $2(\tau - \tau') = 2(\tau - f(\tau,i))$ squares apart. So $\text{cost}(\tau,i) \leq 2(\tau - f(\tau,i))$ as promised. Noting that $(\tau - f(\tau,i))$ is equal to the length of the corresponding pruned off branch, we can easily estimate the sum mentioned in the claim.

Let T be the sum of the ($\tau$-coordinate) lengths of all arcs in the original tree. Let R be the sum of the ($\tau$-coordinate) lengths of all the arcs after the tree

has been pruned. Since the tree has width at most k at each point, $T \leq kt$. Since the heads end up some place after t time units, there must be a path of length t left in the tree after pruning. so $R \geq t$. Now,

$$\sum_{\tau=1}^{t} \sum_{i=1}^{k} \text{cost}(\tau,i) \leq \sum_{\tau=1}^{t} \sum_{i=1}^{k} 2(\tau - f(\tau,i)) \leq 2(T - R) \leq 2(k-1)t$$

and the claim is proven. □

From the proof we see that in fact the order in which the jumps are simulated at each level does not matter, and neither does it matter if they are all simulated in parallel. This follows since T - R is fixed. The maximum value in cost of $2(k-1)t$ can only be reached if all heads jump together at time t and only if, on the physical tape, each head is able to aquire certain distances to all other heads which is impossible except for 2 heads on a linear tape.

PROOF OF THEOREM 1 Immediate from Lemmas 3 and 4. □

## 4. EASY COROLLARIES

By using Hartmanis-Stearns speedup we can reduce the "cost" factor in the claim in the proof of Lemma 4 to less than or equal to $\varepsilon t$, for an arbitrary small constant $\varepsilon$. Therefore the run time of $(2k - 1)t$ in Theorem 1 can be reduced to $(1 + \varepsilon)t$, provided we are willing to expand the tape alphabet. In [6] it is shown that a k head tape unit (without jumps) can be simulated in real time by $4k - 4$ tape units with one read-write head per tape. Combining these two observations we get,

THEOREM 5 *Let* M *be a jump* TM *with* k *heads. Then* M *can be simulated by an ordinary multitape* TM M' *with* 8k - 8 *single head tapes such that* M' *need at most* $(1 + \varepsilon)t$ *steps to simulate* t *steps of a computation by* M. *Here* $\varepsilon$ *is an arbitrary small, non-zero constant.*

If the machine to be simulated does not run too slowly then the above linear time simulations can be improved to simulations without delay.

DEFINITION A machine M is said to operate in *at least linear time* if there is a constant $c > 1$ such that, in every computation M takes cn or more steps before reading the n-th input symbol.

All aspects of the above simulations, except for one move per input symbol read, can be modified so that the run time is multiplied by an arbitrarily small constant. This is accomplished by a standard application of Hartmanis-Stearns speedup. For machines that run in at least linear time, this speedup converts linear time simulations to simulations without delay. So, we get

THEOREM 6 *Let M be a jump* TM *with* k *heads and which operates in at least linear time. Then M can be simulated without delay by an ordinary multitape* TM *with* 8k - 8 *single head tapes.*

## 5. NONDETERMINISTIC COMPUTATIONS

A *nondeterministic* machine $M_2$ is said to *simulate* a nondeterministic machine $M_1$ if the possible I/O behaviours of $M_1$ and $M_2$ are the same. For time bounded simulations, we also require that, for every $M_1$ computation, there is an $M_2$ computation on the same input such that: the $M_2$ computation simulates the $M_1$ computation within the same time constraints as those given for deterministic machines. All the Turing machines discussed so far were deterministic. Everything above stays valid if we assume that the devices were all of the corresponding nondeterministic varities. Moreover, we can delete the extra k - 1 head tape unit from the RJTM in Lemma 3 because the nondeterministic variety of the RJTM can "guess" whether a head's nearest neighbor is the left or right one. Therefore, we have,

THEOREM 7 *Let M be a nondeterministic jump* TM *with* k *heads and let* $\varepsilon > 0$. *Then M can be simulated by a nondeterministic, multitape* TM M' *with* 4k *single head tapes such that M' needs at most* $(1 + \varepsilon)t$ *steps to simulate* t *steps of a computation by M. Furthermore, if M operates in at least linear time, then M' can simulate M without delay.*

## 6. ACCEPTING DEVICES

The above constructions were for on-line devices with an output tape. Clearly, the same constructions go through for the corresponding off-line devices viewed as acceptors. Hence , among other obvious corollaries, we get

THEOREM 8 *If a language* A *is accepted by a deterministic (respectively nondeterministic) jump* TM M *in time* T(n), *then* A *is accepted by an ordinary deterministic (respectively nondeterministic) multitape* TM M' *in time* $n + \varepsilon T(n)$, *for an arbitrarily small constant* $\varepsilon > 0$.

The term n in the run time of M', given in the last theorem, is truly important only if M runs in real time. Since for acceptors nondeterministic real time and nondeterministic linear time are equivalent [1], M' (in Theorem 8) can be taken to run in time T(n) in the nondeterministic case. This is true even if T(n) = n.

## REFERENCES

[1] BOOK, R.V. & S.A. GREIBACH, *Quasi-realtime languages*, Math. Systems Theory 4 (1970), 97-111.

[2] FISHER, P.C., A.R. MEYER & A.L. ROSENBERG, *Real time simulation of multihead tape units*, JACM 19 (1972) 590-607.

[3] FISHER, M.J. & A.L. ROSENBERG, *Limited random access Turing machines*, In: Proc. 9-th SWAT, IEEE, (1968) 356-367.

[4] HARTMANIS, J. & R.E. STEARNS, *On the computational complexity of algorithms*, Transactions of the AMS 117 (1965), 285-306.

[5] HOPCROFT, J.E. & J.D. ULLMAN, *Formal languages and their relation to automata*, (1969), Addison- Wesley, Reading, Mass.

[6] LEONG, B. & J. SEIFERAS, *New real-time simulations of multihead tape units*, Proc. 9th ACM Symposium on Theory of Computing (1977), (to appear).

[7] STOSS, H.J., *k-Band Simulation von k-Kopf Turing Maschinen*, Computing 6 (1970) 309-317.