MOVE RULES AND TRADE-OFFS IN THE PEBBLE GAME

Peter van Emde Boas and Jan van Leeuwen

RUU-CS-78-4

April/August 1978

Rijksuniversiteit Utrecht

Vakgroep Informatica

Budapestlaan 6
Postbus 80.012
3508 TA Utrecht
Telefoon 030-531454
The Netherlands

MOVE RULES AND TRADE-OFFS IN THE PEBBLE GAME

Peter van Emde Boas
Department of Mathematics
University of Amsterdam
Roetersstraat 15, 1018 WB Amsterdam
the Netherlands

and

Jan van Leeuwen
Department of Computer Science
University of Utrecht
P.O. Box 80.012
3508 TA Utrecht, the Netherlands

Technical Report RUU-CS-78-4

April / August 1978

Department of Computer Science
University of Utrecht
P.O. Box 80.012
3508 TA Utrecht, the Netherlands

all correspondence to:

Dr. Jan van Leeuwen
Department of Computer Science
University of Utrecht
Budapestlaan 6, P.O. Box 80.012
3508 TA Utrecht
the Netherlands

MOVE RULES AND TRADE-OFFS IN THE PEBBLE GAME

Peter van Emde Boas* and Jan van Leeuwen**

Abstract. The pebble game on directed acyclic graphs is commonly encountered as an abstract model for register allocation problems. The traditional move rule of the game asserts that one may "put a pebble on node x once all its immediate predecessors have a pebble", leaving some question as to whether x must get a pebble from the free pool (the strict interpretation) or one from an immediate predecessor. We show that precisely one pebble can be saved over the strict interpretation when the latter possibility is included. On the other hand, we show that this saving may be obtainable only at the cost of squaring the time needed to pebble the graph! It follows that one must very carefully state what interpretation of the move rule is used, as it may seriously affect the meaningfulness of one's results. By an extension of our argument, a much more extreme time-space trade-off result of the same kind is obtained if we pebble according to any one fixed interpretation: there is an infinite family of graphs (with indegrees bounded by 2) such that any strategy for pebbling a graph of this family with a minimum number of pebbles is exponentially worse than the fastest strategy for pebbling the same graph when one pebble more than the required minimum is available.

1. Introduction

It is wellknown that straight-line algorithms for evaluating sets of expressions can be represented as directed acyclic graphs or dags (see e.g. Aho and Ullman [1], Ch. 12). Input nodes (i.e. the nodes with no incoming arcs) correspond to constituent variables, and output nodes (i.e. the nodes with no outgoing arcs) correspond to goal-expressions, with a label at each internal node x indicating what algebraic operation must be performed on the immediate predecessors of x. The register allocation problem for "evaluating" a dag G can be modelled by a pebble game, traditionally played according to the following rules:

*Author's address: Department of Mathematics, University of Amsterdam, Roetersstraat 15, 1018 WB Amsterdam, the Netherlands.

**Author's address: Department of Computer Science, University of Utrecht, P.O. Box 80.012, 3508 TA Utrecht, the Netherlands.

(1) one can always put a pebble on an input node,

(2) one can always remove a pebble from a node,

(3) one can put a free pebble on an empty node x if and only if all immediate predecessors of x have a pebble.

The goal of the pebble game is, starting with G empty, to get a pebble on each output node at least once. Typically, one is interested in pebbling G such that the number of pebbles needed is minimized. Under the same token it may be of interest to minimize the "time" (i.e. the number of moves of types (1) and (3)) needed to pebble G as a function of the number of pebbles available at the start. A version of the pebble game for trees occurs in Paterson and Hewitt [5], but the first reference where the pebble game for general dags is mentioned seems to be Walker [11] (cited in Walker and Strong [12]). The recent revival of the pebble game originated in a completely different area, and is largely due to an intriguing application by Hopcroft, Paul and Valiant [3].

Following a strict interpretation of the rules as stated, it takes at least n+2 pebbles to play the game on a perfect binary tree T of height n (with the $N = 2^n$ leaves as input nodes and the root as a single output node). It is in apparent contradiction with the result usually cited in the litterature, which asserts that n+1 pebbles will do for the same tree T. The authors were reminded of this discrepancy during a "live" demonstration of the pebble game by J. Savage at the 1977 Fachtagung on Complexity Theory in Oberwolfach, Germany (using authentic Schwarzwalder pebbles). The reason is that rule (3) is often phrased in more liberal terms as follows (see Paul, Tarjan and Celoni [7], Pippenger [8], Savage and Swamy [9], Paul and Tarjan [6], or Lingas [4]):

if all immediate predecessors of an empty node x have a pebble, then one can put a pebble on x

which ostensibly allows one to "move" a pebble from an immediate predecessor of x to x directly. Observe, for instance, that 3 pebbles are required to pebble the simple tree of fig. 1.a according to the original interpretation, whereas 2 pebbles suffice if we allow the latter type of moves (see fig. 1.b).

figure 1

Rather than allowing rule (3) to be taken ambiguously, we recognize the new interpretation as an additional move-rule for the pebble-game:

(4) if all immediate predecessors of an empty node x have a pebble, then one can move a pebble from one of these nodes to x.

This rule has certainly been stated explicitly before. One can find it in Cook [2], but Sethi [10] seems to have been the first (and only?) author to explicitly distinguish between the two kinds of moves. Whereas everyone seems to take rule (4) for granted, we shall demonstrate in this paper that it can have unexpected consequences for the complexity of pebbling dags. Note that the problem is intimately related to a fundamental issue in designing instruction sets for computers. Should machine-instructions always deliver their result in a non-operand register (as implied by rule (3)), or do we allow an operand to be overwritten by the result of an operation (rule (4))? In all instruction sets we know the latter is allowed, but apparently no one ever studied why it should be one way or the other.

A first, and quite important result is that the situation demonstrated in fig. 1. is no exception: precisely one pebble can be saved over the minimum needed in the strict interpretation, if we allow rule (4). It means that one can always do just as much as before with one register less, if instructions are allowed "which overwrite operand-registers". There is a catch, though. We will show that the saving of one pebble by allowing rule (4) may be achievable only at the cost of essentially squaring the time needed to pebble the dag! The conclusion for minimizing register-use of straight-line code, written in a modern instruction set, is self-evident.

The result shows that one must state very carefully what move-rule in the pebble game one allows, as it can seriously affect the meaningfulness of one's time or space estimates for a given dag.

The construction we develop to show the impact of rule (4) can be extended to obtain an extreme time-space trade-off result if we pebble dags by any fixed (i.e., either strict or non-strict) set of rules. Recall that Paul and Tarjan [6] have demonstrated that for some infinite class of dags (with indegrees ≤ 2) the saving of some constant fraction of the pebbles may force the time required to pebble the dags to blow up exponentially. In fact very recently Lingas [4] observed independently of us that saving just a constant number of pebbles (2, in Lingas' case) can do the same. We shall prove that a similar phenomenon can occur when even one pebble gets saved.

2. Some definitions and the saving of a pebble

Given a convention for the type of moves allowed, we shall be counting the number of moves during the game in which a pebble gets placed (or "moved", if permitted), i.e. we count all moves which are described by rules (1), (3) and (4) (the latter, again, if permitted). Let G be an arbitrary dag.

Definition.

$S(G)$ = the minimum number of pebbles required for pebbling G according to rules (1), (2) and (3).

$S'(G)$ = the minimum number of pebbles required for pebbling G according to rules (1), (2), (3) and (4).

$T_k(G)$ = the minimum number of counted moves required for pebbling G according to rules (1), (2) and (3) when $S(G) + k$ pebbles may be used.

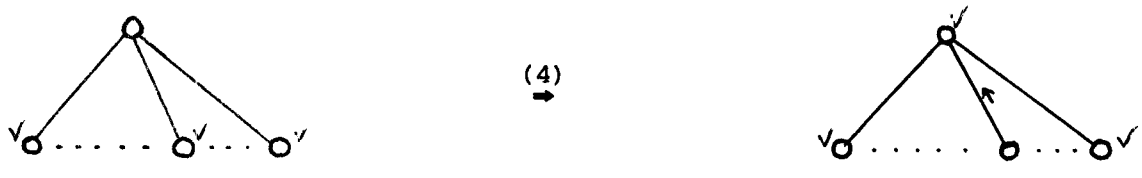
$T'_k(G)$ = the minimum number of counted moves required for pebbling G according to rules (1), (2), (3) and (4) when $S'(G) + k$ pebbles may be used.

Note that S and T_k are quantities related to the "strict" interpretation of the game, S' and T'_k are the corresponding quantities if the extended move-policy is implemented. T_k and T'_k measure the "time" required for pebbling a dag if one is given k more pebbles than the minimum needed. In particular, T_0 and T'_0 measure the time required to pebble a dag with the smallest possible number of pebbles.

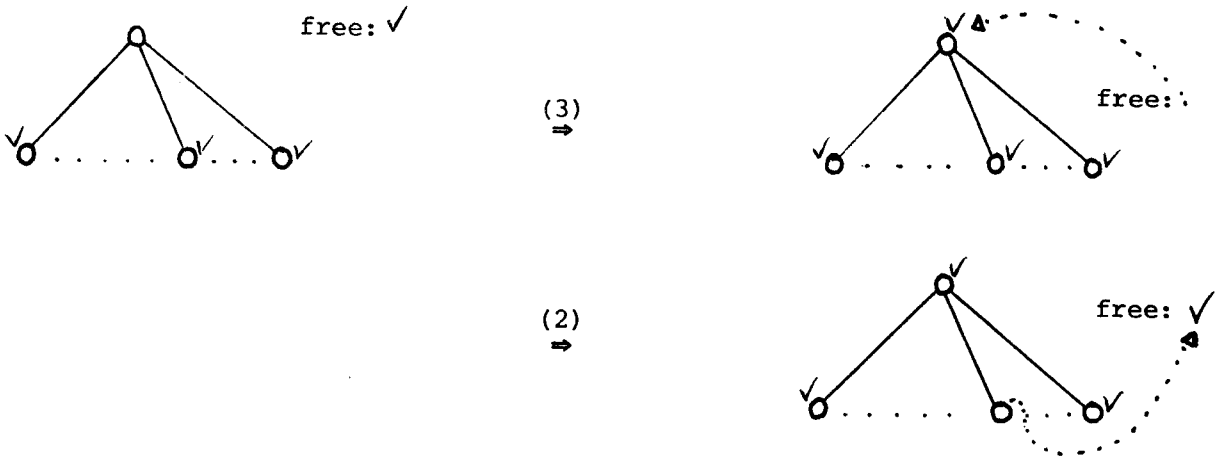
It is quite easy to see that for all dags G:

$$S(G) \geq S'(G) \geq S(G) - 1. \qquad (2.1)$$

The first part follows trivially, the second part (which may be read as $S(G) \leq S'(G) + 1$) follows by observing that each application of rule (4) may be simulated by rules (3) and (2) if one extra pebble is provided from the start: moves like



can be replaced throughout by



Note that the number of counted moves is not changed in the simulation.

Our first result is that rule (4) always enables one to save exactly one pebble over S(G).

Theorem A. For dags G with at least one edge, $S'(G) = S(G) - 1$.

Proof.

By (2.1) it is sufficient to show that $S'(G) \leq S(G) - 1$.

Consider any strategy W for pebbling G according rules (1), (2) and (3), using S(G) pebbles. Let W(t) denote the game-configuration of G at time t. We shall transform W "step by step" by taking rule (4) into account, leading to a complete strategy in which at any one time no more than S(G) - 1 pebbles are on the dag.

Arguing inductively, suppose W actually is a pebbling strategy for G which uses rules (1) to (4) and no more than S(G) - 1 pebbles up to some time s and which still uses rules (1) to (3) only and up to S(G) pebbles from time s+1 and onwards. In the beginning we have such a strategy for s=1. Let t be the first time after s at which all S(G) pebbles available are on the dag. (Such a t must exist, or else W is already of the desired form.) We may assume without loss of generality that $s = t-1$, as we could suitably move s to it otherwise.

One can have all pebbles on the dag at time $s+1 = t$ only if at this time a last free pebble got placed on some node x. As moves according to rule (4) are still prohibited here, the next move must be the removal of a pebble from some node y. (If the pebbling of x was the last move of the game, just pretend there is one more step removing the pebble from x and continue the argument.) The following possibilities can arise at time t+1. In each case we shall argue how the transition from $W(s)\{= W(t-1)\}$ to W(t+1) can be achieved with a saving of one pebble and applying rule (4) when necessary.

Case (i). y is an immediate predecessor of x.

- Pebble x (at time t) by moving the pebble of y according to rule (4), rather than using a new pebble -

Case (ii). y is not an immediate predecessor of x, and $y \neq x$.

- Interchange the moves at times t and t+1. Thus, we first remove the pebble from y and then use it, rather than another free pebble, to cover x. Note that apart from the transposition no new moves are introduced -

Case (iii). y=x, but x is not an output node.

- Eliminate the moves at times t and t+1 alltogether, as the pebbling of x obviously served no purpose -

Case (iv). y=x, and x is an output node.

- If x happens to have no predecessors, then we just pick ("borrow") a pebble from elsewhere in the dag (there must be one) and use it on x. Take all pebbles from the dag and repeat W up to stage s to re-establish the configuration at time t-1, and jump to W(t+1). If x does have predecessors, then we follow a rather similar procedure: pebble x by moving (!) a pebble from one of its immediate predecessors according to rule (4) instead, clear all pebbles off the dag, re-establish W(t-1) by repeating the game as played up to stage s and continue pebbling by the original strategy from W(t+1) on -

In all cases we succeeded eliminating the need for having a full supply of S(G) pebbles at stage t, obtaining a modified pebbling strategy for G whose "cut" s has come closer to the end of the game. Repeatedly applying this procedure will eliminate all stages which had the maximum of S(G) pebbles on the dag.

□

We should point out that the re-pebbling of portions of the dag, called for in case (iv) of the given proof, may cause a substantial increase in the time for pebbling G. The next result puts a bound on the number of extra moves needed.

Proposition B. Let G be a dag with m outputs. Then $T_0'(G) \leq m \cdot T_0(G)$

Proof.

The estimate can be derived by carefully analysing what moves we actually need to repeat, according to the construction of theorem A. Let the original pebbling strategy, using rules (1) to (3) and up to S(G) pebbles, be represented as a string

$$\sigma_1\sigma_2\sigma_3\cdots \qquad \cdots\sigma_{T_0(G)},$$

where σ_i $(1 \leq i \leq T_0(G))$ codes the move at time i. The transformation of the

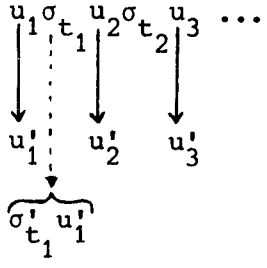
string to save one pebble, as outlined in theorem A, usually involves only minor changes of the σ_i's. Extra steps are required only at the "few" times when a last free pebble is used to cover an output node. Let the moments at which this occurs be $t_1, t_2, \ldots, t_l$. (Obviously $1 \leq l \leq m$.) The string of original moves has a corresponding decomposition

$$u_1 \sigma_{t_1} u_2 \sigma_{t_2} \quad \cdots \quad u_l \sigma_{t_l} u_{l+1} \tag{2.2}$$

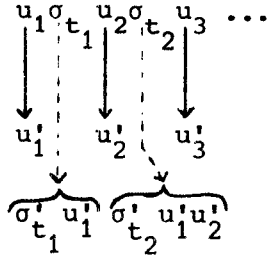
The proof of theorem A shows that the stretches $u_1, u_2, \ldots, u_{l+1}$ can be transformed independently from the special actions needed to eliminate the extra pebble at times $t_1, t_2, \ldots, t_l$. In fact, each u_i transforms into a string u_i' without the need for extra intermittent moves:

$$|u_i'| \leq |u_i| \tag{2.3}$$

Recall that σ_{t_1} (like any σ_{t_i}) concerns the pebbling of an output node x by means of the last free pebble. To eliminate this pebble, theorem A suggested to replace σ_{t_1} by a "shift" (call it σ'_{t_1}) and to repeat u_1' to recreate the configuration on the dag just before x was pebbled:



For σ_{t_2} a similar transformation is applied, but now we must repeat $u_1'u_2'$ to recreate the configuration just before σ_{t_2}:



It is easy to see that in fact each move σ_{t_i} gets replaced by a sequence $\sigma'_{t_i} u_1' u_2' \ldots u_i'$. It holds even when the output node pebbled at time t_i has no predecessors, which was a special case in theorem A. The new strategy for pebbling G, using rules (1) to (4) but no more than S(G) - 1 pebbles, corresponds to the string

$$u_1' \sigma_{t_1} u_1' u_2' \sigma'_{t_2} u_1' u_2' u_3' \ldots u_1' u_2' \ldots u_1' \sigma'_{t_1} u_1' u_2' \ldots u_1' u_{l+1}' \tag{2.4}$$

Observe that $u'_i \ldots u'_1$ need to be repeated after σ'_1 only if there are any outputs remaining to be pebbled during u_{1+1}. Clearly this won't happen when $1=m$. Hence we can estimate $T'_0(G)$ differently depending on the value of 1:

if $1 < m$, then

$$T'_0(G) \le |u'_1\sigma'_{t_1}u'_2\sigma'_{t_2}\ldots u'_1\sigma'_{t_1}u'_{1+1}| + |u'_1| + |u'_1u'_2| + \ldots + |u'_1\ldots u'_1| \le$$

$$\le (1+1)\,|u'_1\sigma'_{t_1}u'_2\ldots u'_1\sigma'_{t_1}u'_{1+1}| \le$$

$$\le m.\ T_0(G)$$

if $1 = m$, then

$$T'_0(G) \le |u'_1\sigma'_{t_1}u'_2\ldots u'_1\sigma'_{t_1}u'_{1+1}| + |u'_1| + \ldots + |u'_1\ldots u'_{1-1}| \le$$

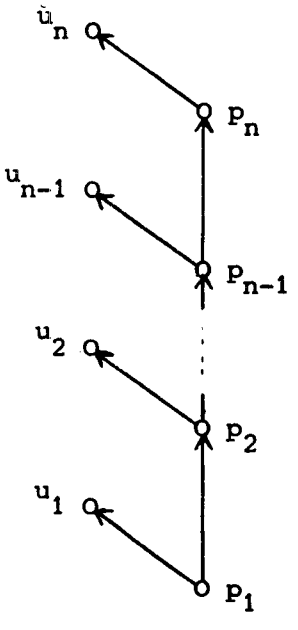
$$\le m.\ T_0(G)$$

which proves our result.

□

Proposition B shows that the loss of time we risk, when saving one pebble in favor of rule (4), stays within reasonable limits as long as the number of outputs of a dag is small. In general, m can be as large as $O(T_0(G))$ (whereas clearly $m \le T_0(G)$) and proposition B learns that in worst case a squaring of the pebbling time may occur. A simple example shows that the worst case can occur and that the bound of proposition B is best possible.

Consider the dag $E_{n,1}$ defined as



(2.5)

The reader easily verifies that $S(E_{n,1}) = 2$, $T_0(E_{n,1}) = 2n$ and $S'(E_{n,1}) = 1$, but

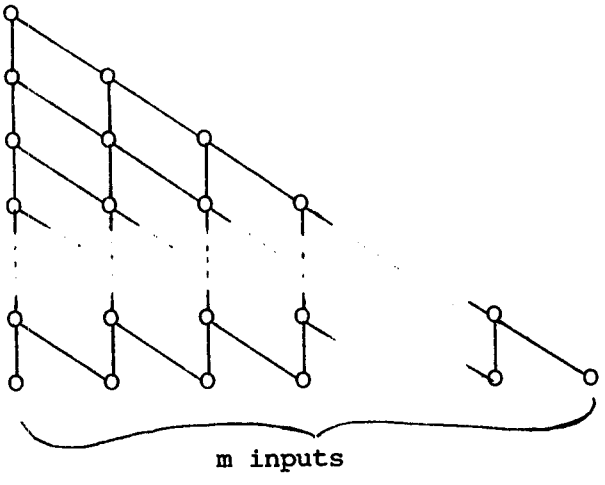
$$T'_0(E_{n,1}) = 2 + 3 + \ldots + (n+1) = \theta(n^2)$$

We conclude

Theorem C. Saving a pebble by allowing rule (4) in worst case squares the (order of magnitude of the) pebbling time in the strict interpretation.

The dags $E_{n,1}$ are admittedly simple, but more examples for theorem C can readily be obtained. Actually, the $E_{n,1}$'s are the simplest of a family of dags $\{E_{n,m}\}$ exhibiting the same behavior. As these dags will be very useful later on, we digress here to study some of the properties of a general $E_{n,m}$.

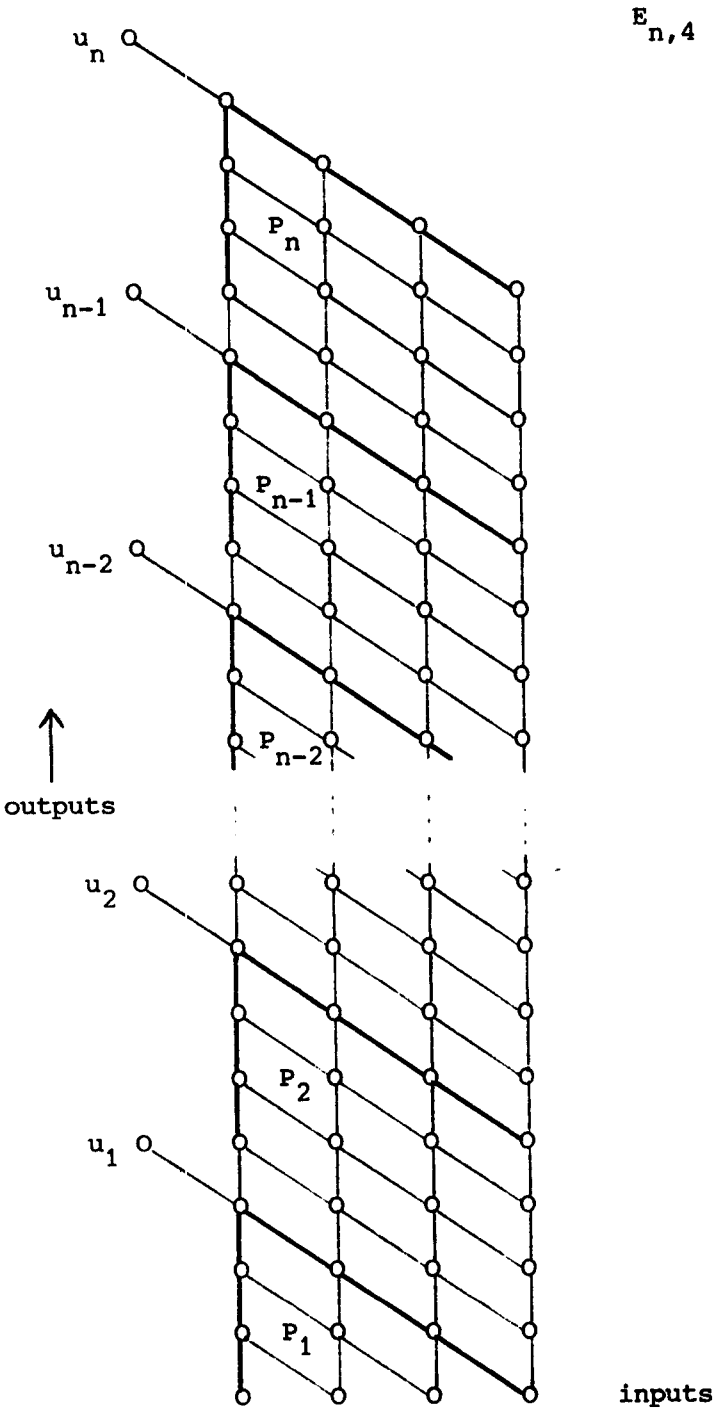
Before defining $E_{n,m}$ it is useful to recall the structure of Cook's pyramidal dags C_m of width m [2]:

$C_m =$ 

(2.6)

The structure of $E_{n,m}$ is obtained by vertical translation of a pyramid C_m over unit distance (n-1).m times, leading to a "staircase" of width m and heigth n.m tapering off as a pyramid at the top. Special (unary) output nodes $u_1, \ldots, u_n$ are added on to the left side of the staircase, with u_i connected to the node at height i.m. It follows that each u_i is connected to the top of an embedded copy of C_m, denoted by P_i. Observe that the base of P_i is located at exactly one level above P_{i-1}. The structure of $E_{n,m}$ must be evident from (2.7), where $E_{n,4}$ is shown.

It is easy to verify that $S(E_{n,m}) = m+1$, and if m+1 pebbles are available then one can pebble $E_{n,m}$ in time equal to its size (in fact, regardless of whether rule (4) is used or not), thus without ever pebbling a node more than once:



(2.7)

$$T_0(E_{n,m}) = T_1'(E_{n,m}) = \theta(n.m^2) \qquad (2.8)$$

This is no longer true if we pebble $E_{n,m}$ with m pebbles (which we can) when rule (4) is allowed, no matter how we try. Although it isn't the strongest result we can prove, the following proposition will be sufficient for our later purposes. It suggests that before a "next" output can be pebbled a good deal of repebbling must be performed.

Proposition D. Let node u_i of $E_{n,m}$ be pebbled, using rules (1) to (4) and m pebbles, without pebbling any input more than once. At the time u_i gets pebbled, there must be a pebble-free path from each of the remaining outputs to some input.

[... thus at least one input and some higher nodes must be repebbled before another output can be pebbled]

Proof.

Let a configuration of pebbles on $E_{n,m}$ be called proper if each of the following conditions is satisfied:

(i) each column of $E_{n,m}$ contains a pebble (hence, all available pebbles are in use and occupy different columns),

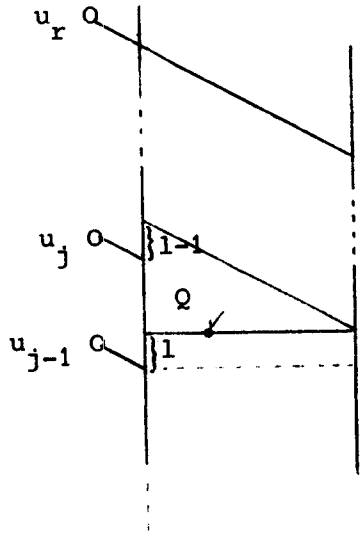
(ii) each pebble resides at the same level or one higher than the pebble in the column immediately to its right.

Clearly there is a way to pebble u_i such that all configurations on the way are proper. As we must consider arbitrary strategies, we shall first argue that properness is a meaningful concept for $E_{n,m}$ in general. It is important to note that we do not allow inputs to be pebbled more than once in our approach here. On the other hand, since u_i is initially unblocked from all inputs one must eventually pebble each input at least once. It is easy to see that at the time the last input gets its pebble assigned the configuration on the dag must have become proper, or else there is no way to even get a pebble in column one past P_1's top.

Suppose we pebble towards u_i, maintaining a proper configuration for a while. When u_i gets its pebble the configuration cannot be proper anymore: the (at most) m-1 pebbles left on the "body" of $E_{n,m}$ can impossibly cover all m columns. Suppose that the properness condition gets disturbed for the first time through the move at time t, i.e. the configuration at time t+1 is no longer proper but the configurations at time t and earlier (until the last input got pebbled) were.

Since the configuration at time t is proper, its "line" of m pebbles is entirely contained in some embedded pyramid Q. (It isn't necessarily a P_j, but it is one of the many translates of P_1 in the dag.) Let the base of Q be at level $(j-1)m+1$, some $1 \le l \le m$. See (2.9). Without loss of generality we may assume that the node in column m of Q's base is pebbled.

Observe that all pebbles have moved past the top-level of P_{j-1} and, as inputs are never pebbled again, no pebble will ever return below this level. It follows that all u_r with $1 \le r \le j-1$ have pebble-free paths to the inputs at time t and later, including the time u_i gets pebbled.



(2.9)

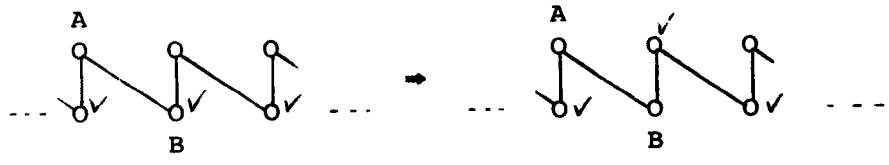
level $(j-1)m+1$, $1 \le l \le m$

Output u_j must be connected to W, and there isn't much we can say about it (in a moment it will be clear that u_j must be u_i). Consider any output u_r with $r > j$. Since u_r is located well above the top of Q, the entire edge of P_r is disjoint from Q and (hence) pebble-free. Let us consider how properness can be disturbed through a move at time t and observe the consequences in each case:

Case a. A pebble is removed or shifted "along a diagonal" to the left, leaving a pebble-free column c.

- Clearly we can now move from u_r along the edge of P_r to column c and then straight down to an input, without getting blocked. Hence, a pebble-free path is created in this case -

Case b. All columns remain covered, but the pebble in some column c is moved up past the level of the pebble in column c-1:



- This time we have a pebble-free path from u_r down the diagonal of P_r to column c-1, then down to node A and across to B, continuing in column c towards an input node -

It is easy to see that there are no other legal moves by which properness can be disturbed. We conclude that after the move at time t each u_r for $r > j$ is unblocked from at least one input. Since repebbling of inputs is explicitly forbidden, these outputs will remain unblocked for the remainder of the game.

We conclude that all u_r with $r \neq j$ must be unblocked when u_i gets pebbled. It means that $i=j$, and proposition D holds.

□

3. Extreme time-space trade-offs

We have noted before that rule (4) can be simulated by a combination of rules (3) and (2) without changing the number of counted moves, provided one extra pebble is made available. Together with the result of theorem C, we conclude that for any dag G:

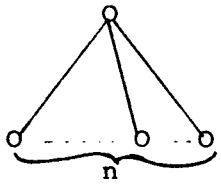
$$T_0(G)^2 \geq T_0'(G) \geq T_0(G) \geq T_1'(G) \geq T_1(G) \geq \ldots \qquad (3.1)$$

Examining the two underscored inequalities, we will discover here that there can be very large (exponential) gaps between the quantities on the left-hand and right-hand sides in both. Our main goal shall be to prove the following time-space trade-off result, stated informally as

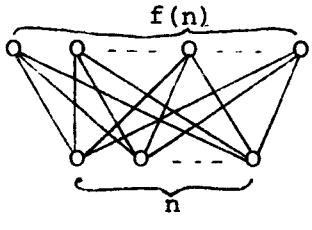
Theorem E. There is an infinite family of dags H_n ($n \geq 1$), with indegrees bounded by 2, such that $T_0'(H_n)$ is exponentially worse than $T_1'(H_n)$.

Because $T_0(G) \geq \sqrt{T_0'(G)}$ and, on the other hand, $T_1(G) \leq T_1'(G)$ by (3.1), the same family of dags suffices to show that $T_0(G)$ can be exponentially worse than $T_1(G)$ uniformly. Thus, we shall only pursue the details of the result when rule (4) is allowed here. Note that it substantiates an earlier claim that, in any interpretation of the rules, the saving of a single pebble can blow up the pebbling time exponentially.

It will require a bit of "engineering" to keep the indegrees of all nodes in H_n bounded by 2. We shall ignore this constraint for the moment, so as not to obscure the idea of the construction. Let $f: N \to N$ be some function to be chosen later. Define the following auxiliary graphs:

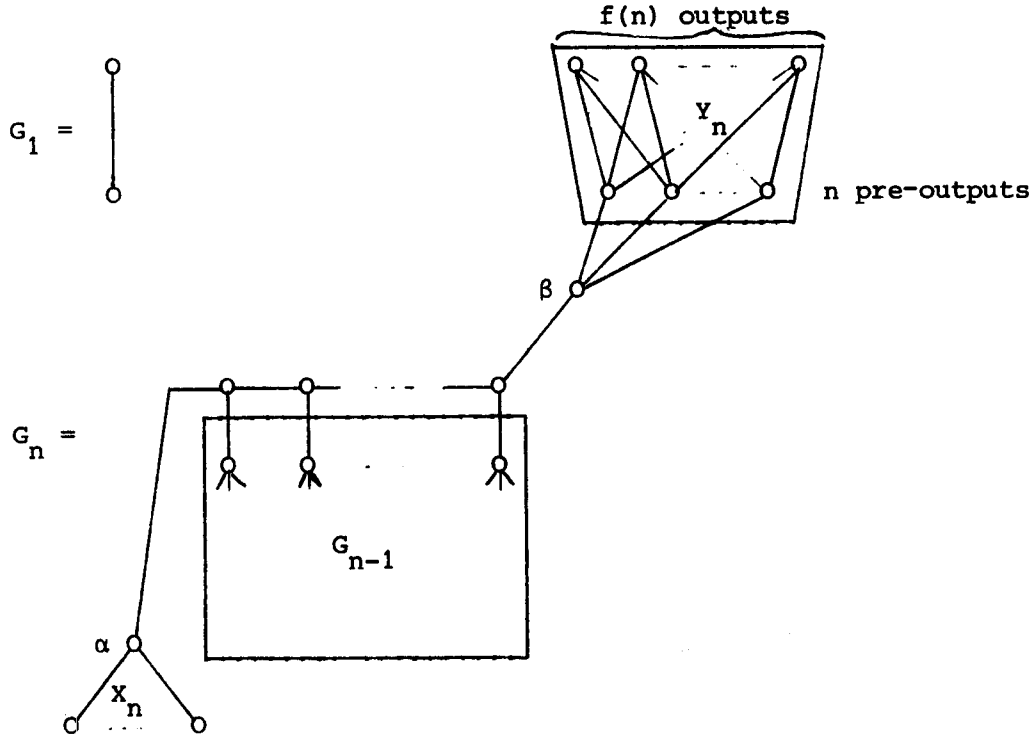
$X_n =$ 

(a node with n immediate predecessors)

$Y_n =$ 

(a bipartite graph)

Now consider the family of dags G_n $(n \geq 1)$, defined inductively as follows:



The size of G_n satisfies the recursion

$$\text{size } (G_1) = 2$$

$$\text{size } (G_2) = \text{size } (G_{n-1}) + 2(n+1) + f(n) + f(n-1) \qquad \text{for } n \geq 2$$

and it follows that size $(G_n) \leq 2 \cdot \sum_{2}^{n} f(i) + \theta(n^2)$. Likewise one can easily verify that G_n has $\theta(n^2)$ input nodes and, obviously, exactly f(n) output nodes.

Clearly $S'(G_n) = n$. The following proposition makes some precise claims about the time needed to pebble G_n with n+1 or n pebbles.

Proposition F.

(i) $T_1'(G_n) = \text{size } (G_n) \leq 2 \cdot \sum_{2}^{n} f(i) + \theta(n^2)$

(ii) $T_0'(G_n) \geq \prod_{2}^{n} f(n)$

Proof.

(i) The simplest strategy to pebble G_n using n+1 pebbles proceeds as follows, using as an induction-hypothesis that the outputs of G_{n-1} can be pebbled (in consecutive order) using n pebbles in size (G_{n-1}) moves. First pebble α and move ("slide") its pebble along the chain to β while pebbling the outputs of the embedded G_{n-1} in consecutive order (indeed with exactly n free pebbles available to do it!) With a pebble on β we can place a pebble on each of the n pre-outputs of G_n, which will be fixed there. Now

take the pebble from β and use it to pebble each of the f(n) actual output nodes from left to right. The total number of moves adds up to exactly size (G_n), using our inductive assumption.

(ii) If only n pebbles are available one must proceed in a similar fashion initially, resorting to rule (4) more often now and using that G_{n-1} can be pebbled using n-1 pebbles as an inductive assumption. Once β is reached (i.e., pebbled) things will change and we are going to see the effect of having only n pebbles to play with, despite the freedom with rule (4). In order to pebble an output node of Y_n at all one must

(a) move a pebble to each pre-output node, which can be done only by committing all n-1 free pebbles and moving the pebble from β to the last pre-output still open,

(b) move ("slide") a pebble from any one pre-output node to the designated output.

To pebble any other output now, we are in deep trouble: we must get a pebble back on the one pre-output node which is now open. This requires that we get a pebble back on β first. The only way to repebble β is to pick up all n pebbles from the dag and to repebble the entire dag, including the embedded copy of G_{n-1}! So we must proceed for each output node again, and clearly

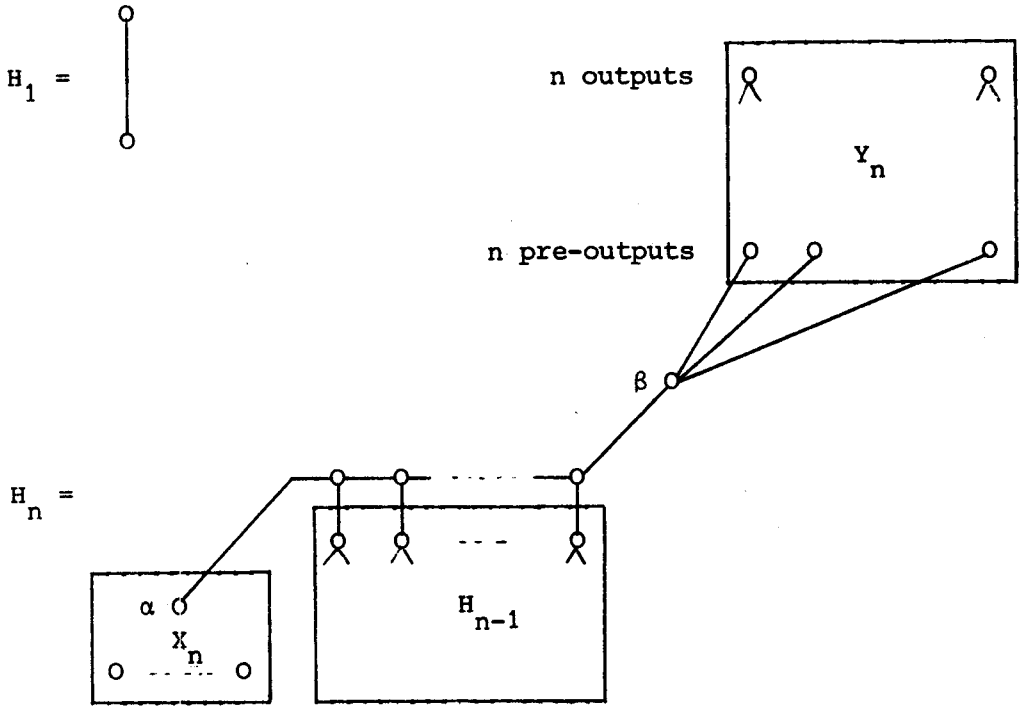
$$T_0'(G_n) \geq f(n) \cdot T_0'(G_{n-1})$$

which yields the desired estimate as stated. We note that the construction of G_n indeed forces the entire repebbling of the embedded G_{n-1}, because pebbles must appear on its outputs from left to right if we are to move a pebble along the "chain" at all.

□

Choosing f(n) = n we get a result as desired. The construction yields a family of dags G_n with size $(G_n) = \Theta(n^2)$ and $T_1'(G_n) = \Theta(n^2)$ but $T_0'(G_n) \geq \Theta(n!)$, an exponential blow-up by saving just one pebble! One should note, however, that the indegrees of nodes in G_n can be as large as n.

We shall now modify the construction to obtain a family H_n which exhibits the same behavior while indegrees remain bounded by 2. The idea is based on the same inductive scheme, but the sub-dags X_n and Y_n are still to be determined. We draw the diagram again, for the sake of clarity:



The dags X_n and Y_n should now be binary, choosen such that a similar argument as before will go through to get an analog of proposition F.

For a general result we consider the following requirements for X_n and Y_n:

Condition I. $S'(X_n) = S'(Y_n) = n$, and n pebbles are actually required for pebbling any single output of Y_n above.

Condition II. If Y_n is pebbled using n pebbles (without repebbling any input), then at the time one of its outputs gets pebbled there must be a pebble-free path from each of the remaining outputs to an input.

The qualifier "without repebbling any input" may seem unnatural but really isn't, considering that Y_n is embedded in H_n and the repebbling of an "input" is not just a matter of applying rule (1).

Lemma G. If conditions I and II are satisfied, then any strategy for pebbling H_n using n pebbles must repebble the entire dag before a next output node can be pebbled.

Proof.

We leave it to reader to verify that, when conditions I and II are satisfied, H_n can be pebbled using n pebbles (and not with less)!

In order to pebble any output node at all we must first get a pebble to β. Once a pebble is there, we can begin pebbling Y_n using the (n-1) free pebbles. In order to pebble even a single output of Y_n we need one more pebble (see condition I), thus forcing the pebble of β to be moved. Note

that while pebbling on Y_n no "input" (i.e. pre-output node) will get repebbled, as it would require us to start all over to get a pebble to β: the only way to pebble β again is to repebble X_n which, by condition I, requires us to pick up all pebbles from Y_n and to undo all work there. Considering the pebbling of Y_n, condition II learns that once an output gets pebbled, all other output nodes will have an open path to some pre-output. Thus, to pebble a next output node at all we must pebble the pre-output node it is openly connected to first (some time). In order to do so we must get a pebble back on β which, in turn, is possible only by picking up all n pebbles and repebbling the entire dag from the start.

□

In order to have a proper trade-off result we should also require that X_n and Y_n can be pebbled in time, say, polynomial in n when n+1 pebbles are used, with all inputs pebbled at the start and no input to be repebbled ever after. We can now conclude our work.

Proof of theorem E.

Choose $X_n = C_n$ (the pyramid of order n, see (2.6)) and $Y_n = E_{n,n}$. The reader easily verifies, inductively, that for this choice of dags $T'_1(H_n) = \text{size}(H_n) = \Theta(n^4)$.

To estimate $T'_0(H_n)$ we argue first that X_n and Y_n satisfy conditions I and II. Condition I easily follows from known properties of the pyramid. Condition II, low and behold, is exactly the property of the graphs $E_{n,m}$ which we proved in proposition D. By lemma G we can infer that necessarily

$$T'_0(H_n) \geq n.\ T'_0(H_{n-1})$$

because the repebbling of H_n for each of the n distinct outputs forces the repebbling of all outputs of H_{n-1} from left to right. Hence $T'_0(H_n) \geq n!$, and once again an exponential blow up from $T'_1(H_n)$ to $T'_0(H_n)$ has been established. Clearly, each of the dags H_n has indegrees ≤ 2.

□

Theorem E shows that the explosion of time in minimizing register use, first reported by Paul and Tarjan [6] in case some constant fraction of the registers gets saved, can occur already if just one register is saved. We note that Lingas [4], independently, found a construction which yields a sequence of binary dags $\{G_n\}$ satisfying $S'(G_n) = 2n$, $T'_0(G_n) \geq 2^n$ and

$T_2'(G_n)$ = size $(G_n) = \Theta(n^3)$. The resulting trade-off is more extreme (because the dags are "smaller"), but one had to trade 2 pebbles to get it. An interesting problem might be to find a family of dags $\{G_n\}$ with $S'(G_n) = \Theta(n)$, such that the saving of some constant number of pebbles gives a jump from linear to exponential in pebbling time whereas size (G_n) is only $o(n^3)$.

5. References

[1] Aho, A.V. and J.D. Ullman, Principles of Compiler Design, Addison-Wesley Publ. Comp., Reading, Mass., 1977.

[2] Cook, S.A., An Observation on Time - Storage Trade Off, J. Comp. Syst. Sci. 9 (1974) 308-316.

[3] Hopcroft, J., W. Paul and L. Valiant, On Time versus Space, J.ACM 24 (1977) 332-337.

[4] Lingas, A., A PSPACE - complete Problem related to a Pebble Game, in: G. Aussiello and C. Böhm (eds.), Automata, Languages and Programming (Fifth Colloquium, Udine, 1978), Springer Lecture Notes in Computer Science 62, 1978, pp. 300-321.

[5] Paterson, M.S. and C.E. Hewitt, Comparative Schematology, Record of Project MAC Conference on Concurrent Systems and Parallel Computations (June 1970) 119-128, ACM, New Jersey, Dec. 1970.

[6] Paul, W. and R.E. Tarjan, Time - Space Trade-offs in a Pebble Game, in: A. Salomaa and M. Steinby (eds.), Automata, Languages and Programming (Fourth Colloquium, Turku, 1977), Springer Lecture Notes in Computer Science 52, 1977, pp. 365-369.

[7] Paul, W., R.E. Tarjan and J.R. Celoni, Space Bounds for a Game on Graphs, Math. Syst. Th. 10 (1976) 239-251.

[8] Pippenger, N, A Time - Space Trade-off, Computer Science Res. Rep. RC 6550 (#28265), IBM, Yorktown Heights, 1977 (also: J.ACM 25 (1978) 509-515).

[9] Savage, J.E. and S. Swamy, Space - Time Tradeoffs on the FFT Algorithm, Techn. Rep. CS-31 (August 1977), Div. of Engineering, Brown University, 1977.

[10] Sethi, R., Complete Register Allocation Problems, SIAM J. Comput. 4 (1975) 226-248.

[11] Walker, S.A., Some Graph Games related to Efficient Calculation of Expressions, Res. Rep. RC-3633, IBM, 1971.

[12] Walker, S.A. and H.R. Strong, Characterizations of Flow-chartable Recursions, J. Comp. Syst. Sci. 7 (1973) 404-447.