MOVE RULES AND TRADE-OFFS IN THE PEBBLE GAME

Peter van Emde Boas
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 it open whether the pebble to be placed on x should be taken from some predecessor of x or from the free pool (the strict interpretation). We show that allowing pebbles to slide along an edge as a legal move enables one to save precisely one pebble over the strict interpretation. However, in the worst case the saving may be obtained only at the cost of squaring the time needed to pebble the dag. It shows that one has to be very careful in describing properties of pebblings; the interpretation of the rules can seriously affect the results. As a main result we prove a linear to exponential time trade-off for any fixed interpretation of the rules when a single pebble is saved. There exist families of dags with indegrees ≤ 2, with the property that they can be pebbled in linear time when one more pebble than the minimum needed is available but which require exponential time when the extra pebble is dropped.

1 Introduction

The pebble game has received interest in the theory of computational complexity both for practical and more theoretical goals (register alloc, network complexity, time-space trade-offs). The oldest references are Paterson & Hewitt [6] and Walker [14] (cited in [15]). The revived interest for pebbling arose from an application to Turing machine complexity by Hopcroft, Paul & Valiant [3].

The pebble game is played on directed acyclic graphs (dags). The nodes in the graph without incoming edges are called the inputs of the dag. Some other nodes are designated as the outputs of the dag. A position in the game is described by the subset of pebbled nodes. The size of this subset is the number of pebbles used in this position. Starting from an empty dag, the aim of the game is to move pebbles around according to the move rules specified below, in such a way

that eventually all outputs get pebbled at least once. This should be achieved using as few pebbles as possible, or, when the number of pebbles is fixed, using as few moves as possible.

Traditionally the moves are controlled by the following rules:

(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 pebble on node x provided all immediate predecessors of x have a pebble.

The formulation of rule (3') leaves open where the pebble to be placed on x has to come from. As stated rule (3') apparently allows us to slide a pebble from a predecessor of x to x, a liberal interpretation most often used in the literature For example, the well-known result that the complete binary tree T of height n with 2^n leaves (inputs) requires $n+1$ pebbles is valid only if the liberal interpretation is used ; otherwise $n+2$ pebbles are required. 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 (using authentic Schwarzwalder pebbles).

Instead of following the established practice of allowing the above ambiguity in rule (3') (cf. [5, 7, 8, 9, 11]) we recognise the liberal interpretation as an additional move rule. Hence we replace (3') by the pair :

(3) one can put a free pebble on node x provided all immediate predecessors of x have a pebble

(4) if all predecessors of an empty node x have a pebble then one can slide one of these pebbles to x.

Rule (4) has been stated by Cook [2], but to our knowledge only Sethi [12] explicitly distinguished between (3) and (4) before.

We shall demonstrate that (3) and (4) should not be equivalenced ; it can have a serious impact on the complexity of pebbling whether rule (4) is allowed or not. We show that when rule (4) is allowed, then it is possible to save precisely one pebble over the minimum needed if the strict interpretation, i.e. rule (3), is used. However, in the worst case this saving may be obtained only at the price of squaring the number of moves needed.

Clearly the problem mentioned above is related to a fundamental issue in the design of machines. Should machine-instructions always deliver their result in a non-operand register (rule (3)), or should we allow that one of the operands is overwritten (rule (4)). Our result shows that the usual architectures permitting overwriting instructions may save precisely one register in the register allocation problem, at a price which has to be considered a considerable loss of speed.

The argument used to obtain the quadratic increase in time can be extended to obtain an extreme time-space trade-off result for any fixed pebbling strategy. Paul and Tarjan [7] obtained an infinite class of graphs (with indegrees ≤ 2) such that the saving of some constant fraction of the pebbles may force the time required for pebbling a graph to blow up exponentially. Lingas [5] recently obtained a similar result by saving only 2 pebbles. We show that such an explosion may even occur when just a single pebble is saved. The results of this paper are spelled out in greater detail in [13] .

The observation that rule (4) allows one to save precisely one pebble has been made independently by Gilbert and Tarjan [4]. However, their proof overlooks the crucial case (iii) below (which is responsible for the squaring of the time needed) and seems therefore incomplete. Sethi [oral comm.] has conjectured that the dags he used for the NP-hardness construction in [12] may provide examples of trade-offs similar to the one described in the paper but no specific claims have been made. Our results are unrelated to the trade-offs recently announced by Reischuk [10] and Tarjan [oral comm.].

2 Some definitions and the saving of a pebble

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

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.

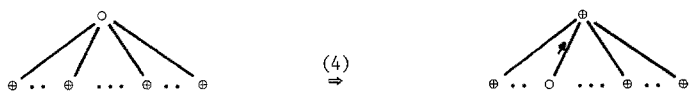
$T'_k(G)$ = the minimum number of counted moves required for pebbling G according to rules (1), (2), (3) and (4) when $S'(G) + k$ pebbles may be used.

Note that S and T_k are quantities related to the "strict" interpretation of the game, S' and T'_k are the corresponding quantities for the extended move-policy. 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.

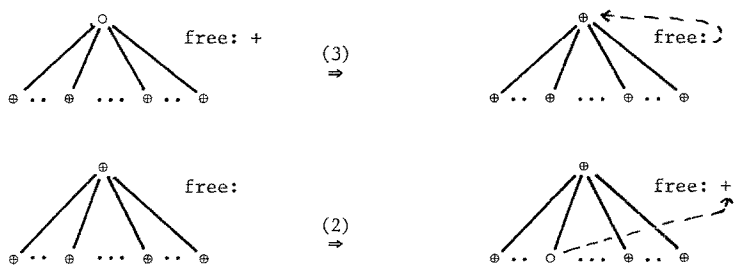
It is quite easy to see that for all dags G :

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

The first part is trivial, 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). Observe that if G contains no edges then clearly G is pebbled by pebbling its nodes succesively using a single pebble; thus $S(G) = S'(G) = 1$.

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

Proof.

It suffices to prove that $S'(G) \leq S(G) - 1$. Consider a strategy $W = W_0, W_1, \ldots, W_N$ which uses $k = S(G)$ pebbles, with W_0 the empty position and each W_j obtained from W_{j-1} by an application of rule (1), (2) or (3). Consider the W_j where exactly k pebbles are used. In the next move some pebble must be removed, as otherwise k+1 pebbles would be present in W_{j+1}. The following possibilities arise:

(i) The pebble removed in the move W_j, W_{j+1} is not removed from a predecessor of the node pebbled during the preceding move, or from this node itself. In this situation the order of the two moves may be interchanged, thus eliminating the position involving k pebbles.

(ii) The pebble removed in the move W_j, W_{j+1} is taken from a predecessor of the node pebbled during the preceding move. In this situation the two moves may be replaced by an application of rule (4) thus eliminating the position involving k pebbles.

(iii) The pebble removed in the move W_j, W_{j+1} is the pebble which was placed during the preceding move.

Only this third case requires a non-local transformation in order to eliminate the position using k pebbles. Note that the move makes sense only if the node pebbled is an output (otherwise W_j and W_{j+1} may be eliminated alltogether). We replace W_j by a shift, provided the pebbled node has some predecessor. Otherwise we take some arbitrary pebble from the dag and use this pebble instead. In both cases the position involving k pebbles has been eliminated. However, in order to regenerate position W_{j+1}, which equals W_{j-1} in this case, it no longer suffices to take the pebble just placed from the dag ; instead we take all pebbles from the dag and repeat the entire pebbling strategy upto W_{j-1}, in this way restoring configuration W_{j+1}.

It is not hard to obtain a complete proof based upon the above transformation [13] . Always taking W_j as the first position involving k pebbles, we can use complete induction based on the number of positions in a pebbling strategy which have k pebbles on the dag. □

We should point out that the re-pebbling of portions of the dag, called for in case (iii) 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.

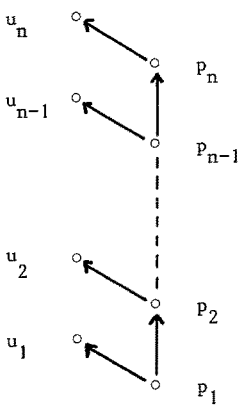
Proposition B. Let G be a dag with m outputs. Then $T_0'(G) \leq m.T_0(G)$

Proof

The argument before shows that no time is lost if the dag contains only one output. If there are m outputs, then split the pebbling strategy into m strategies, starting from empty dags and each involving a single output (costing together at most $m.T_0(G)$ moves). Apply the transformation from the proof of theorem A to each strategy individually. □

Proposition B shows that the loss of time in saving one pebble with 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 $m \leq T_0(G)$) 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



(2.2)

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.

3 Extreme time-space trade-offs

A pebbling strategy is called a real-time pebbling of G in case no node in G gets pebbled twice during the game. Assuming that G does not contain useless nodes (i.e. nodes which do not precede any output) this is equivalent to saying that the time needed to pebble G equals the size of G. Clearly each dag can be pebbled in real-time provided sufficiently many pebbles are available.

We noted 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 :

$$T_0(G)^2 \geq T_0'(G) \geq T_0(G) \geq T_1'(G) \geq T_1(G) \geq \ldots \geq T_k(G) = T_k'(G) = \text{size } (G) \quad (3.1)$$

for some $k \geq 0$ (the last equality holding only if G contains no useless nodes). From theorem C we conclude that in general :

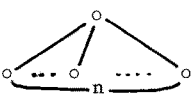
$$T_j(G)^2 \geq T'_j(G) \qquad (3.2)$$

Examining the two inequalities underscored in (3.1), we shall 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

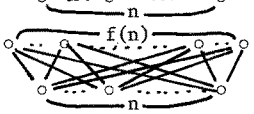
Theorem D. There is an infinite family of dags $H_n (n \geq 1)$, with indegrees bounded by 2, such that $T'_0(H_n)$ is exponentially worse than $T'_1(H_n)$ = size (H_n)

Because $T'_0(G) \leq T_0(G)^2$ and, on the other hand, $T_1(G) \leq T'_1(G)$, the same family of dags suffices to show that $T_0(G)$ can be exponentially worse than $T_1(G)$ uniformly. Thus, we need only pursue the details of the result when rule (4) is allowed. 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:

G_1 = 

G_n =

(3.3)

The size of G_n satisfies

$$\text{size } (G_1) = 2$$
$$\text{size } (G_2) = \text{size } (G_{n-1}) + 2(n+1) + f(n) + f(n-1) \qquad \text{for } n \geq 2$$

and it follows that size $(G_n) \leq 2 . \sum_{2}^{n} f(i) + \theta(n^2)$. 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 E.

(i) $T'_1(G_n) = \text{size } (G_n) \leq 2 . \sum_{2}^{n} f(i) + \Theta(n^2)$

(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 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) output nodes from left to right. This actually yields a real-time pebbling of G_n.

(ii) If only n pebbles are available one must initially proceed in a similar fashion, 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. 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) slide a pebble from any one pre-output node to the designated output.

To pebble any other output, 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

$$T'_0(G_n) \geq f(n) . T'_0(G_{n-1})$$

which yields the desired estimate as stated. 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 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 inductive scheme of (3.3), but the sub-dags X_n and Y_n will be changed. So X_n and Y_n should now be binary, chosen such that an argument as before will go through to get an analog of proposition E.

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. Moreover X_n and Y_n can be pebbled in real-time when one extra pebble is provided.

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 F. If X_n and Y_n satisfy conditions I and II and if the sequence of dags $\{H_n\}$ is defined according to (3.3), then:

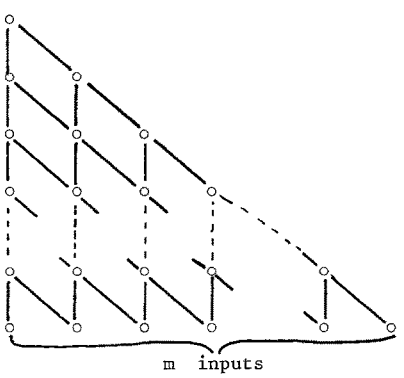
(i) $T'_1(H_n) = \text{Size}(H_n)$, so H_n can be pebbled in real time using one extra pebble

(ii) $T'_0(H_n) \geq \prod_{2}^{n} f(n)$.

Proof. Similar to the proof of proposition E (see [13]) .

At this time it is useful to recall the structure of Cook's pyramidal dags C_m of width m ([2]):

C_m =

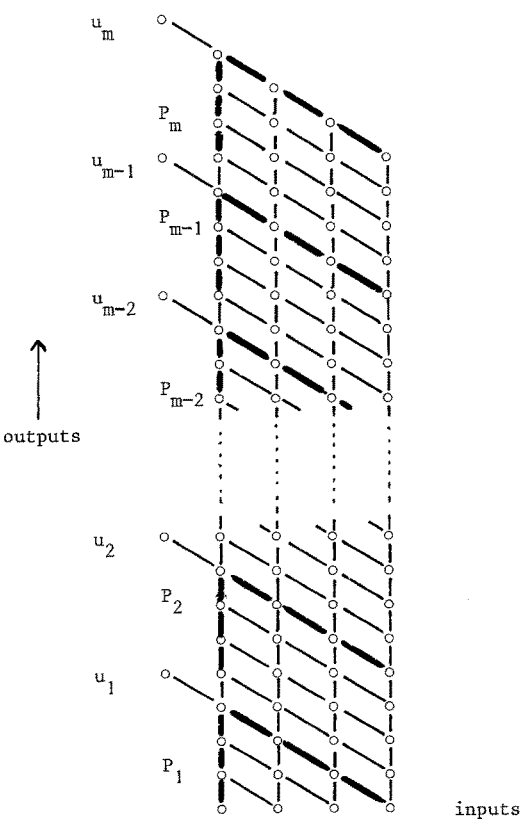


(3.4)

It is easy to see that by choosing $X_n = C_n$ the requirements of condition I are satisfied as far as X_n is concerned. To obtain the Y_n's we introduce a family of dags $E_{m,n}$. We encountered its members $E_{m,1}$ already in constructing a worst case example of the trade-off in section 2.

The structure of $E_{m,n}$ is obtained by vertical translation of a pyramid C_n over unit distance (m-1).n times, leading to a "staircase" of width n and heigth n.m tapering off as a pyramid at the top. Special (unary) output nodes $u_1,\ldots,u_m$ are added on to the left side of the staircase, with u_i connected to the node at height i.n. It follows that each u_i is connected to the top of an embedded copy of C_n, denoted by P_i. Observe that the base of P_i is located at exactly one level above P_{i-1}. The structure of $E_{m,n}$ must be evident from (3.5), where $E_{m,4}$ is shown.

It is easy to verify that $S(E_{m,n}) = n+1$, and if n+1 pebbles are available then one can pebble $E_{m,n}$ in real-time (in fact, regardless of whether rule (4) is used or not).



(3.5)

$$T_0(E_{m,n}) = T'_1(E_{m,n}) = \theta(m.n^2) \qquad (3.6)$$

We show that $E_{m,n}$ satisfies conditions I and II for the Y_n :

Proposition G. Let node u_i of $E_{m,n}$ 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.

Proof.

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

(i) each column of $E_{m,n}$ 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.

It is possible to pebble u_i in such a way that all intermediate configurations are proper.

The following observations can be made for an arbitrary pebbling strategy:

(a) Since we do not allow the repebbling of inputs in the pebbling of u_i , a configuration must occur in which the last input gets pebbled before u_i gets pebbled ; if this configuration is not proper, then it is impossible to pebble u_i .

(b) Before any u_i can be pebbled the properness condition must be disturbed.

(c) Once the properness condition is established and it gets disturbed some time later, a situation will arise in which all outputs except possibly one have a pebble-free path to some input.

The above observations together imply proposition G : in order to pebble u_i the properness condition is established at the time the last input gets pebbled; at a later stage the properness gets disturbed, and from that stage onwards the pebble-free paths from outputs to inputs remain pebble-free since no input gets repebbled. The proofs of the observations are tedious but straightforward (see [13]) □

Proof of theorem D.

Choose $X_n = C_n$ and $Y_n = E_{n,n_4}$. The reader easily verifies that the dags H_n constructed by (3.3) have size $\theta(n^4)$. Now $T'_1(H_n) = \theta(n^4)$ whereas $T'_0(H_n) \geq n!$ by lemma F, thus yielding the required exponential blow-up. □

Theorem D shows that the explosion of time in minimizing register use, first reported by Paul and Tarjan [7] in case some constant fraction of the registers gets saved, can occur already if just one register is saved. We note that Lingas [5], 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 $T'_2(G_n) = \text{size}\ (G_n) = \theta(n^3)$. 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, Journal Computer Systems Sciences 9 (1974) 308-316.

[3] Gilbert, J.R. and R.E. Tarjan, Variations of a pebble game on graphs. Rep. Stanford STAN-CS-78-661 (Sept. 1978).

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

[5] 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.

[6] 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.

[7] 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.

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

[9] 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).

[10] Reischuk, R., Improved bounds on the Problem of Time-Space Trade-off in the Pebble Game (Preliminary version), Conf. Record 19th Annual IEEE Symp. on Foundations of Computer Science, Ann Arbor, 1978, pp. 84-91.

[11] Savage, J.E. and S. Swamy, Space-Time Trade-offs in the FFT Algorithm, Techn. Rep. CS-31 (August 1977), Div. of Engineering, Brown University, Providence, 1977.

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

[13] van Emde Boas, P. and J. van Leeuwen, Move-rules and trade-offs in the pebble game, Techn. Rep. RUU-CS-78-4, Dept. of Computer Science, University of Utrecht, Utrecht, April/August 1978.

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

[15] Walker, S.A. and H.R. Strong, Characterizations of Flow-chartable Recursions, Journ. Computer System Sciences 7 (1973) 404-447.