ZASTOSOWANIA MATEMATYKI
APPLICATIONES MATHEMATICAE
XIX, 3–4 (1987), pp. 569–577

W. RYTTER (Warszawa)

REMARKS ON PEBBLE GAMES ON GRAPHS

Abstract. We briefly survey some applications of pebble games to main problems in the complexity theory. In the second part of the paper we analyse a new pebble game introduced by the author and related to the complexity of parallel computations. This new game changes the structure of the graph during the game, adding some extra edges, while previously known games are played on static graphs.

1. Introduction. Directed acyclic graphs (dags, for short) can be used to describe the structure of a computation. Each node of the graph can represent a piece of information, initially unknown, which should be computed. Instead of saying that this information for a given node x is *computed* we say that x is *pebbled*. Such an abstraction allows us to forget about many technical details and to concentrate on the structure of the computation graph.

Dags can be viewed as generalizations of trees. Such notions as root, leaves, father, sons are defined for dags in the same way as for trees. We assume throughout the paper that the number of sons is bounded by a constant.

The dag G models a computation associated with its root if initially the information asociated with its leaves only is known. To compute information related to a node x we need information associated with its sons. This implies the following rules of the *black pebble game* (*pebble game*, for short). At any point in the game some nodes will have pebbles on them (one pebble per node). If all sons of a node have pebbles on them, a pebble may be placed on that node (hence a leaf can always be pebbled); a pebble may be removed from any node. The goal of the game is to pebble the root.

We show how the pebbles are related to variables in straight-line programs. Such a program is a sequence of assignment statements

$$x_1 := W_1, \quad x_2 := W_2, \quad \ldots, \quad x_n := W_n,$$

where W_i are some algebraic expressions involving $O(1)$ number of variables x_k (for a given W_i, $k < i$). W_i involves zero variables iff it is a constant. The computation graph for the straight-line program is constructed as follows.

The nodes are variables x_k. The root is x_n and the sons of x_i are all variables occurring in W_i. The minimum number of pebbles needed to pebble the graph corresponds to the minimum number of variables needed to compute x_n. Hence the space complexity corresponds here directly to the number of pebbles. The natural combinatorial question is: what is the minimum number of pebbles with respect to n, where n can be interpreted as the time of the original program? However, if we decrease the number of variables (pebbles), then computation time can increase. This corresponds to another fundamental problem in complexity theory: time-space trade-off.

2. Pebble games and question in complexity theory. The presentation in this section is informal. We shall refer to the bibliography for many details. The main problems in complexity theory concern relations between time and space, and between types of computations: deterministic, nondeterministic, parallel, sequential.

Let $X \gg Y$ mean that X is, computationally, essentially stronger than Y, where X and Y are resources (time, space) or types of computation. We consider the following problems:

(1) space $\gg$ time (see [3]);

(2) time-space trade-off; less space more time is needed (see [6]);

(3) nondeterministic time $\gg$ deterministic time (see [9]);

(4) nondeterministic space $\gg$ deterministic space (see [4] and [6]);

(5) parallel time $\gg$ sequential time (see [1]).

Graph-theoretic results about pebbling gave affirmative answers to problems (1), (3) and (5), and gave some insight into problems (2) and (4).

The black pebble game defined in the Introduction corresponds to deterministic sequential computations. Another game, called a *white-black pebble game*, was introduced to model nondeterministic computations. Besides black pebbles we have here also white pebbles, which correspond to a nondeterministic guessing of information (such guessing should be verified later). The rules for black pebbles are as before. The rules for white pebbles are: a white pebble can be placed on any node; a white pebble may be removed from a node if all its sons are pebbled. The goal of the game is to place a black pebble on the root or to place a white pebble, which later is removed (verified), starting and ending with no pebbles.

Yet another pebble game, called a *two-person game*, was introduced to model parallel computations. The two-person pebble game is played between two players, called the *Challenger* and the *Pebbler*. The Challenger begins by placing his token, called the *challenge*, on some node. The Pebbler responds with placing some of his tokens, called *pebbles*, on some set of nodes. The pebbles are never removed. In each succeeding round, the Challenger may leave the challenge where it is or may move it to a node pebbled by the

Pebbler in the immediately preceding round. If, in the move of the Challenger, each node which can be challenged has all its sons pebbled, the Pebbler wins. It is clear that the Pebbler always has the winning strategy, e.g., pebbling all nodes in the first move. We say that the Pebbler *wins in R rounds and time T* if he has a strategy that ensures that he wins after making at most R moves and placing a total of T pebbles.

Now counterparts of problems (1)–(5) in the pebbling theory can be formulated as follows:

(1′) Every dag G of size n can be black-pebbled using $O(n/\log n)$ pebbles (see [3]). n corresponds here to time and the number of pebbles to space. There are graphs which require such a number of pebbles, they are constructed using difficult graphs called *superconcentrators* (see [10] and [2]).

(2′) Let $S_J(n) = n/\log\log(n)$ (S_J is a space-jump function). Then there are constants c_1, c_2 such that if $S(n) \geqslant c_1 S_J(n)$, then every dag G of size n can be pebbled using $S(n)$ pebbles in polynomial time; if $S(n) \leqslant c_2 S_J(n)$, then there are graphs of size n that can only be pebbled with $S(n)$ pebbles in superpolynomial time (see [6]).

(3′) Let G be a computation graph of a multitape Turing machine (see [9] for the definition) or, more generally, let G be the dag with page number bounded by a constant. If n is the size of G, then the Pebbler can win in two rounds and time $T = O(n/\log^*(n))$, where $\log^*(n)$ is the iterated logarithm. T corresponds to the parallel time measured by the depth of alternating computations. The correspondence is very technical and we refer the reader to [9]. The main application to complexity theory here is the following result: nondeterministic linear time is essentially more powerful than deterministic linear time (this solves the problem similar to P = ?NP, where polynomial is replaced by linear).

(4′) There is a class of dags for which there are white-black pebbling strategies that use asymptotically less pebbles than black strategies (see [16]). If $S(n)$ white-black pebbles suffice for a given dag, then $O(S^2(n))$ black pebbles suffice (see [7]).

(5′) For every dag of size n there is a winning strategy for the Pebbler with $T = O(n/\log n)$ pebbles in the two-person game. This implies a parallel speedup by the factor $\log n$ for some models of parallel computations (see [1]).

3. A parallel pebble game on trees. As a model of parallel computation we consider the parallel random access machine without write conflicts (P-RAM, for short), see [12]. The action of the instruction

for $a \in S$ **parallel do** instruction(a)

consists of: assigning a processor to every $a \in S$, assigned processors have access to their value of a; executing each instruction(a) simultaneously. The

processors can use common memory; however, two different processors cannot attempt to write into the same memory location simultaneously.

Let T be a binary tree of size n, where by *size* we mean here the number of leaves. Denote by T_x the subtree of T rooted at x. Associate with each node x a node $\mathrm{cond}(x)$; initially cond is the identity function. We say that a node x is *activated* iff $\mathrm{cond}(x) \neq x$. We treat the pairs $(x, \mathrm{cond}(x))$, if x is activated, as the additional edges. We show later in examples how to compute the information associated with a node x (pebbling x) if we know some information corresponding to the edge $(x, \mathrm{cond}(x))$ and if the node $\mathrm{cond}(x)$ is pebbled (which means that information associated with $\mathrm{cond}(x)$ is known). The main role of additional edges is that we can pebble x iff $\mathrm{cond}(x)$ is pebbled. We define the operation activate, square and pebble as follows:

activate:

for each nonleaf node x **parallel do**
if x is not activated and one of its sons is pebbled, then set $\mathrm{cond}(x)$ to the other son; if both sons are pebbled, then one of them is (arbitrarily) chosen.

square:

for each nonleaf node x **parallel do** $\mathrm{cond}(x) := \mathrm{cond}(\mathrm{cond}(x))$.

pebble:

for each node x **parallel do** if $\mathrm{cond}(x)$ is pebbled then pebble x.

Let one pebbling move (move, in short) consist in executing the sequence of operations activate; square; square; pebble, in that order. Initially, all leaves are pebbled.

Remark (added in proof). This is related to Rake and Compress operations in [8]. Our game and a construction from [8] were introduced independently. In fact, the first version of the parallel pebble game (slightly different from the game introduced here and without proof) was presented by the author in 1984 (see [13]). The theorem below does not follow from the result in [8].

THEOREM 1. *Let T be a binary tree with n leaves, all initially pebbled. Then after $\lceil \log n \rceil$ moves the root of T is pebbled.*

Proof. We consider a modified pebbling move consisting of the following sequence: pebble; activate; square; square. It is enough to prove that after $\lceil \log n \rceil + 1$ such moves the root will be pebbled. Observe that if the node is pebbled after $k+1$ such modified moves, then it is pebbled after k originally defined moves, the first pebble operation and the last operations activate, square, square are redundant in this context. Let $\mathrm{size}(x)$ denote the number of leaves of T_x and

$$\mathrm{size}(x/y) = \mathrm{size}(x) - \mathrm{size}(y).$$

If y is a descendant of x, then $\text{size}(x/y)$ is the size of T_x with gap y. The modified moves are numbered from 0 to $\lceil \log n \rceil$.

CLAIM. *After the k-th modified move the following invariants are preserved:*

(I1) *if* $\text{size}(x) \leqslant 2^k$, *then* x *is pebbled;*

(I2) $\text{size}(x/\text{cond}(x)) \geqslant 2^k$ *or both sons of* $\text{cond}(x)$ *are not pebbled or* $\text{cond}(x)$ *is a leaf.*

The proof is by induction on k. After the move 0 all nodes of size 2^0 are pebbled, since they are leaves. Hence (I1) holds. (I2) also holds.

Assume now that (I1) and (I2) hold after the move $k-1$. First we prove that (I1) holds after the move k. Let x be a node such that $\text{size}(x) \leqslant 2^k$. Then each nonleaf node of T_x has a son of size not greater than 2^{k-1}. This implies that every nonleaf node after the move $k-1$ has one of its sons pebbled. Hence (I2) implies that after the move $k-1$

$$\text{size}(x/\text{cond}(x)) \geqslant 2^{k-1}$$

or $\text{cond}(x)$ is a leaf. In both cases $\text{cond}(x)$ was pebbled after the move $k-1$, since if it is not a leaf, then

$$\text{size}(\text{cond}(x)) \leqslant 2^{k-1},$$

because $\text{size}(x/\text{cond}(x)) \geqslant 2^{k-1}$. In the move k the operation pebble places a pebble on x. This proves (I1).

Next we prove that (I2) holds after the k-th move. Let $\text{cond}(x) = y$ after the move $k-1$. We know from the inductive hypothesis that y is a leaf (which ends the proof), or both sons of y are not pebbled or $\text{size}(x/y) \geqslant 2^{k-1}$. We consider the last two alternatives.

Case 1. $\text{size}(x/y) \geqslant 2^{k-1}$.

Let $z = \text{cond}(y)$ after the move $k-1$. If $\text{size}(y/z) \geqslant 2^{k-1}$, then after the operation square

$$\text{size}(x/\text{cond}(x)) \geqslant 2^k.$$

If not, then both sons of z were not pebbled after the move $k-1$ or z was a leaf. It is enough to consider only the first possibility. Let z_1 and z_2 be sons of z. We need to consider only the case where one of them, say z_1, is pebbled in the k-th move.

$\text{size}(z_1) > 2^{k-1}$, because z_1 was not pebbled in the previous move. Now the operation activate sets $\text{cond}(z)$ to z_2; hence

$$\text{size}(z/\text{cond}(z)) = \text{size}(z_1)$$

at this moment, and after two operations square $\text{cond}(x) = z_3$, where z_3 is a descendant of z_2,

$$\text{size}(x/\text{cond}(x)) \geqslant \text{size}(x/y) + \text{size}(z_1) \geqslant 2^k.$$

Case 2. y is not a leaf and both sons of y are not pebbled in the move $k-1$.

Let z_1 and z_2 be the sons of y. We consider only the case where one of z_1, z_2, say z_1, is pebbled in the k-th move. The operation activate sets $\mathrm{cond}(y)$ to z_2. We have

$$\mathrm{size}(y/z_2) = \mathrm{size}(z_1) > 2^{k-1},$$

because z_1 was not pebbled in the move $k-1$. After the first operation square we have

$$\mathrm{cond}(x) = z_2 \quad \text{and} \quad \mathrm{size}(x/\mathrm{cond}(x)) > 2^{k-1}.$$

We have now one more operation square. Let v be the value of $\mathrm{cond}(z_2)$ after the move $k-1$. If $\mathrm{size}(v/\mathrm{cond}(v)) \geqslant 2^{k-1}$, then now after the next operation square $\mathrm{cond}(x)$ is set to a descendant of $\mathrm{cond}(v)$ and $\mathrm{size}(x/\mathrm{cond}(x)) \geqslant 2^k$. Otherwise, we consider the subcase where the sons v_1 and v_2 of v are not pebbled in the move $k-1$ and one of them is pebbled in the move k. Assume that v_1 is pebbled. It can be proved analogously (as for the node z before) that after the operation activate (in the move k) $\mathrm{cond}(v) = v_2$ and $\mathrm{size}(v/v_2) \geqslant 2^{k-1}$. In the first operation square, $\mathrm{cond}(z_2)$ is set to v_2. At this moment we have $\mathrm{cond}(x) = z_2$, $\mathrm{cond}(z_2) = v_2$. In the second operation square, $\mathrm{cond}(x)$ is set to v_2 and

$$\mathrm{size}(x/\mathrm{cond}(x)) \geqslant \mathrm{size}(x/z_2) + \mathrm{size}(v/v_2) \geqslant 2^k.$$

The invariant (I2) is preserved.

If none of the sons of v is pebbled in the move k or v is a leaf, then the invariant is also preserved; after the move k we have $\mathrm{cond}(x) = v$ and it will be a leaf or both its sons will not be pebbled. This completes the proof of the Claim and of the theorem.

THEOREM 2. *For each n there is a tree with n leaves which requires $\lceil \log n \rceil$ moves to pebble the root.*

Proof. It is enough to consider only the numbers n which are powers of 2. In this case the regular full binary tree with n leaves satisfies the assertion.

One can ask if analogous results hold for any binary dag. The parallel pebble game works in the same manner for binary dags as for binary trees.

THEOREM 3. *For $n \geqslant 3$ there is a binary dag which requires $\lfloor (n-2)/2 \rfloor$ moves to pebble the root, where n is the number of nodes.*

Proof. We construct the required graphs G_n recursively. G_3 has nodes 1, 2 and 3. The nodes 1 and 2 are sons of the node 3. The graph G_{n+1} is constructed from G_n by adding the node $n+1$, which is the root, and whose sons are n and $n-1$. After the first moves, nodes 3 and 4 are pebbled. After the second, nodes 5 and 6 are pebbled, and so on. The root is pebbled after $\lfloor (n-2)/2 \rfloor$ moves. This completes the proof.

It can be proved that for binary dags G the number of moves is bounded by $\lceil \log(\text{tree-size}(G)) \rceil$, where tree-size$(G)$ is the number of paths from the root to a leaf. Tree-size of the graphs G_n is exponential.

We show two applications of the parallel pebble game to compute certain functions on trees.

(1) Let val(x) be a value associated with each node of the tree, and let © be an associative binary operation computable in constant time. We compute for each node x the value

$$\text{result}(x) = \text{val}(y_1) \copyright \text{val}(y_2) \copyright \ldots \copyright \text{val}(y_k),$$

where $y_1, y_2, \ldots, y_k$ are all nodes of the subtree T_x. Typical examples of the operation © are $\min(v_1, v_2)$ or usual addition of numbers. Our algorithm to compute result(x) is pebble-driven. We use the auxiliary table res and we maintain the following invariant during the parallel pebbling:

If node x is pebbled, then

$$\text{res}(x) = \text{result}(x)$$

else if $\text{cond}(x) = y$, then

$$\text{res}(x) = \text{val}(y_1) \copyright \text{val}(y_2) \copyright \ldots \copyright \text{val}(y_r),$$

where $y_1, \ldots, y_r$ are all nodes which are in the subtree T_x and which are not in the subtree T_y. Whenever we are executing the operation square or pebble, then we execute simultaneously

$$\text{res}(x) := \text{res}(x) \copyright \text{res}(\text{cond}(x)).$$

Whenever we execute the operation activate and we set $\text{cond}(x) := z_1$ because the other son z_2 is pebbled, then we execute simultaneously

$$\text{res}(x) := \text{val}(x) \copyright \text{res}(z_2).$$

In this way, after pebbling the root, the value of res(x) equals the required value result(x). Hence we can compute such a function on a tree in $\log n$ parallel time using a linear number of processors on a P-RAM. If $\text{val}(x) = 1$ for each node x, then the computed function is the number of descendants of each node.

(2) Define height(x) to be the length (number of edges) of the longest path from x to a leaf. We use auxiliary tables h, $h1$. We maintain, during the pebble game, the following invariant:

If x is pebbled then $h(x) = \text{height}(x)$ else $h(x)$ is the height of x in the subtree of T_x with cond(x) treated as a leaf, $h1(x)$ is the length from x to cond(x).

Whenever we perform the operation square or pebble, then we execute simultaneously:

$$h(x) := \max(h(x), h1(x) + h(\text{cond}(x))), \qquad h1(x) := h1(x) + h1(\text{cond}(x)).$$

When we set $\mathrm{cond}(x) := z_2$ in the operation activate because the other son z_1 of x is pebbled, then we execute

$$hl(x) := 1, \quad h(x) := h(z_1) + 1.$$

In this way we compute height(x) in $O(\log n)$ parallel time on a P-RAM. Analogously we can compute the length of the shortest path from x to a leaf.

We introduce now another very simple game, which we call the *top-down game*. The operations square and pebble are as before. Introduce the operation activate1:

for each node x and its sons z_1, z_2 **parallel do** $\mathrm{cond}(z_1) := x$; $\mathrm{cond}(z_2) := x$.

The following fact can be easily proved:

FACT. *After performing the sequence of operations*

$$\text{activate1; (square)}^k\text{; pebble,}$$

each node is pebbled if initially the root is pebbled and $k = \lceil \log n \rceil$.

We describe an application. Let $\mathrm{val}(e)$ be a value associated with an edge e. Let

$$\mathrm{result}(x) = \mathrm{val}(e_1) \copyright \mathrm{val}(e_2) \copyright \ldots \copyright \mathrm{val}(e_k),$$

where $e_1, \ldots, e_k$ are edges on the path from x to the root and © is an associative binary operation, x is not the root. Introduce the table $\mathrm{res}(x)$ and maintain, during the top-down game, the invariant

$$\mathrm{res}(x) = \mathrm{val}(e_1) \copyright \ldots \copyright \mathrm{val}(e_r),$$

where $e_1, \ldots, e_r$ are edges from x to $\mathrm{cond}(x)$. Whenever we perform the operation square, we execute simultaneously

$$\mathrm{res}(x) := \mathrm{res}(x) \copyright \mathrm{res}(\mathrm{cond}(x))$$

if $\mathrm{cond}(x)$ is not the root. In this way we compute the function result in $\log n$ parallel time. When the node is pebbled, then $\mathrm{res}(x) = \mathrm{result}(x)$.

Let z_1 and z_2 be the left and the right sons of x, respectively. Define $\mathrm{val}(z_1, x) = 0$ and $\mathrm{val}(z_2, x) =$ number of descendants of z_1. Let © be the addition of numbers. Now we can number the nodes in postorder if for each node x we sum the value of $\mathrm{result}(x)$ and the number of descendants of x. In a similar manner we can compute the preorder numbering of the nodes of a given binary tree. The techniques can be generalized to trees with a larger number of sons. The parallel preorder and postorder numberings of the tree were computed in [4] using Euler tour technique; we have given an alternative method, based on pebble games.

CONCLUSION. Pebble games show how graph-theory and combinatorial analysis can be used to obtain important results in complexity theory. Pebble

games are also useful in the design of algorithms. We can say that the algorithms obtained are *pebble-driven*. The algorithmic techniques obtained are quite general.

References

[1] P. Dymond and M. Tompa, *Speedups of deterministic machines by synchronous parallel machines*, J. Comput. System Sci. 30 (1985), pp. 149–161.

[2] O. Gabber and Z. Galil, *Explicit constructions of linear-sized super-concentrators*, ibidem 22 (1981), pp. 407–420.

[3] J. Hopcroft, W. Paul and L. Valiant, *On time versus space*, J. Assoc. Comput. Mach. 24 (1977), pp. 332–337.

[4] J. Hopcroft and J. Ullman, *Introduction to Automata, Languages and Computations*, Addison-Wesley, 1979.

[5] M. Klawe, *A tight bound for black and white pebbles on the pyramid*, J. Assoc. Comput. Mach. 29 (1985), pp. 218–228.

[6] T. Lengauer and T. Tarjan, *Asymptotically tight bounds on time space tradeoffs in a pebble game*, ibidem (1982), pp. 1087–1130.

[7] F. Mayer and der Heide, *A comparison of two variations of a pebble game on trees*, Theoret. Comput. Sci. 13 (1981), pp. 315–322.

[8] G. Miller and J. Reif, *Parallel tree contraction and its applications*, FOCS (1985), pp. 478–489.

[9] W. Paul, N. Pippenger, E. Szemerdei and W. Trotter, *On determinism versus nondeterminism and related problems*, ibidem (1983), pp. 429–438.

[10] W. Paul, R. Tarjan and R. Celoni, *Space bounds for a game on graphs*, Math. Systems Theory 20 (1977), pp. 239–251.

[11] N. Pippenger, *Advances in pebbling*, ICALP (1982), pp. 407–417.

[12] W. Rytter, *Parallel time $O(\log n)$ recognition unambiguous cfl's*, pp. 380–389 in: Fund. of Comp. Theory, Lect. Notes in Comput. Sci. 199 (1985).

[13] – *The complexity of two way pushdown automata and recursive programs*, pp. 341–356, in: A. Apostolico and Z. Galil (eds.), *Proceedings of Combinatorial Algorithms on Words*, NATO Series F12, Springer-Verlag, 1985 (presented at the conference in June 1984).

[14] R. Tarjan and U. Vishkin, *Finding biconnected components and computing tree functions in logarithmic parallel time*, SIAM J. Comput. 14 (2) (1985), pp. 862–873.

[15] L. Valiant, *Graph theoretic properties in computational complexity*, J. Comput. System Sci. 13 (1976), pp. 278–285.

[16] R. Wilber, *White pebbles help*, STOC (1985), pp. 103–112.

INSTITUTE OF INFORMATICS
WARSAW UNIVERSITY
00-901 WARSZAWA