Acta Informatica 16, 465–475 (1981)

Acta Informatica

Black-White Pebbles and Graph Separation

Thomas Lengauer*

Bell Laboratories, Murray Hill, N.J. 07974, USA

Summary. We exhibit a close relationship between two topics in computational complexity. One topic is the analysis of storage requirements for nondeterministic computations. The corresponding mathematical model is a well known black-white pebble game on directed acyclic graphs. The other topic is the search for small separators of undirected graphs. We model a dynamic version of the concept of a separator with a vertex separator game. This game is closely related to graph layout and searching problems. We show that instances of the black-white pebble game and the vertex separator game can easily be transformed into each other. As an application of this result both games are shown to be NP-complete.

1. Introduction

This paper relates two different areas in complexity theory, namely *black-white pebbling* of directed acyclic graphs (dags) and *separators* of undirected graphs.

Pebble games on directed acyclic graphs that model register allocation for straight-line programs have been widely discussed. For an overview see [13]. Most of the the games studied model deterministic straight-line programs, using pebbles of only one color (black). For these games tight or asymptotically tight bounds for the space complexity and time-space tradeoffs for many classes of dags have been found.

Recently interest in nondeterministic straight-line programs has arisen. The motivation is to find out whether nondeterminism is powerful enough to save large amounts of space (i.e., registers). Cook and Sethi introduced in [1] a generalization of the standard black pebble game that incorporates nondeterminism. In addition to using black pebbles to represent registers containing deterministically computed values, they use white pebbles to represent registers that contain nondeterministic guesses to be verified later.

* Current address: Fachbereich 10, Universität des Saarlandes, D-6600 Saarbrücken, Federal Republic of Germany

0001-5903/81/0016/0465/$02.20

We introduce their black-white pebble game (BWP) here as follows. The game is played on a dag G according to the following rules.[1]

(i) All vertices start out pebble-free.
(ii) All vertices end up pebble-free.
(iii) Each vertex receives and loses a pebble at least once.
(iv) A white pebble can be placed on a pebble-free vertex at any time.
(v) A black pebble can be removed from a vertex at any time.
(vi) A white pebble on a vertex v may be turned black if all of the immediate predecessors of v are pebbled.

An *instance* (G, K) of BWP (where G is a dag and K is a positive integer) is called *positive* if G can be pebbled using at most K pebbles. Otherwise, (G, K) is called a *negative* instance of BWP.

The dag G represents the straight-line program, its vertices being intermediate results and its edges connecting the inputs of an operation with its output. Each pebble can be either black or white at different times, and represents a register. If the pebble lies on vertex v, the corresponding register contains the intermediate result represented by v. If the pebble is white, the contents of the register are a nondeterministic guess that has yet to be verified. If the pebble is black, the contents of the register are a verified value. Computation time is modeled by the length of the pebbling strategy, i.e., the number of applications of the rules (iv), (v) and (vi). Storage space is modeled by the maximum number of pebbles used at any instant in the game. In this paper we are only concerned with the latter complexity measure.

The space complexity of BWP has been studied for the class of all dags by [12], for pyramid graphs by [1] and for directed rooted trees by [12, 11] and [9]. The results in these papers imply that there are graphs on which BWP needs only half of the pebbles used in the black pebble game, and on trees this is the best savings possible. It is an open question, whether this statement holds for all dags. The best lower bound on the number of pebbles necessary to pebble a general dag in BWP is proved in [12] and shows that using black-white pebbles cannot save more than a square root factor of the number of black pebbles used.

The above observations are essentially the only results known about the power of black-white pebbles to save space. [7] shows that in general the space complexity and time-space tradeoff in BWP are the same as in the standard black pebble game (where every white pebble has to be turned black immediately). Since the bounds studied in [7] are only tight up to a constant factor and may not be achieved on the same graphs for the two games, [7] does not answer the above open question.

---

[1] We use the following definitions from graph theory: A *directed graph* $G=(V, E)$ has a set V of vertices and a set E of directed pairs of vertices called *edges*. The edge (v, w) is directed from v and to w. A sequence $(v_0, v_1), (v_1, v_2), \ldots, (v_{k-1}, v_k)$ of edges is called a *path*. The vertex v_0 is called a *predecessor* of the vertex v_k, v_k is called a *successor* of v_0. If $k=1$ then v_0 is called an *immediate* predecessor of v_k. The immediate successor is defined analogously. If $v_0=v_k$ then the path is called a *directed cycle*. A *directed acyclic graph* is a graph without directed cycles. In an *undirected* graph G the edges have no direction, i.e., they are sets rather than ordered pairs. Undirected paths and undirected cycles are defined analogously to the definitions given above

Many other questions about BWP seem to be hard to answer. For instance, it is not known whether a result analogous to the one proved in [5] on the black pebble game holds for BWP. Motivated differently, Sethi obtained a partial solution to the problem solved in [5] by considering a "progressive" version of the black pebble game: Each vertex could be pebbled at most once, i.e., no recomputation was possible in the straight-line program. Sethi showed that the progressive black pebble game is NP-complete. He then modified his proof to show that the unrestricted black pebble game is NP-hard.

We will similarly restrict BWP. By exchanging rule (iii) with the following rule:

(iii') Each vertex receives and loses a pebble *exactly* once,

we define a progressive black-white pebble game (PBWP) that effectively disallows recomputation. It models the nondeterministic storage requirement of a straight-line program using a minimum amount of time. (Note, that for directed trees recomputation is of no advantage. Thus BWP and PBWP have the same complexity on trees. This is particularly interesting, since most of the known results on BWP deal with trees.) We will show that PBWP is closely related to a certain "dynamic" notion of separation of undirected graphs.

Separation of undirected graphs is another important area in complexity theory. It has applications in a wide variety of disciplines, ranging from numerical analysis to the complexity of VLSI layouts. A *separator* of an undirected graph is a set of vertices (in the case of a vertex separator) or edges (in the case of an edge separator) whose removal separates the graph into two components of approximately equal size. Asymptotic upper and lower bounds on the size of separators have been found for several classes of graphs. (For an overview of separator theorems see [6]. Nontrivial separator theorems for planar graphs and graphs of fixed genus are given in [10] and [14].) The above definition of a separator is a "static" one. We will here define a corresponding "dynamic" notion of a separator. The definition will be in terms of a pebble game on undirected graphs called the *vertex separator game* (VSG).

The VSG is played on an undirected graph G according to the following rules.

(i) All vertices start out pebble-free.
(ii) All vertices end up pebbled.
(iii) A move consists of placing a pebble on a pebble-free vertex.

The *vertex cut* after the i-th move consists of all vertices that are pebble-free and adjacent to a pebbled vertex after the i-th move. Let S be a strategy for VSG on G, i.e., a permutation of the vertices of G that represents a sequence of moves (applications of rule (iii)) in VSG. $\bar{S}$ denotes the *reversal* of S, i.e., the strategy that makes the moves of S in reversed order. VSG(S) we denotes the maximum size of a vertex cut after any move in S. Note that in general $\mathrm{VSG}(S)=\mathrm{VSG}(\bar{S})$ does not hold. A pair (G, K) with G being an undirected graph and K being a positive integer is a *positive* instance of VSG if $\mathrm{VSG}(S)\leqq K$. Otherwise, (G, K) is a *negative* instance of VSG. Let VSG(G) be the minimum K such that (G, K) is a positive instance of VSG.

The VSG varies in spirit somewhat from other pebble games studied in the

literature. Usually the cost measure in a pebble game is the number of pebbles used in the game. A number of nontrivial move rules incorporate the "semantic" characteristics of the game. Here the situation is reversed. The rules are trivial and the semantics of the game is contained in the cost measure. We could have defined the game differently, such that it complies better with the established tradition of pebble games. We chose not to do this, because in the given version the game is more intuitive and easier to work with.

The concept of a dynamic separator is not a new one. Indeed, if we use edge separators instead of vertex separators in the above definition, we define a well known problem on undirected graphs, namely, the *min-cut linear arrangement problem* ([2, 3, 8]). The min-cut linear arrangement problem formalizes the cost measure for certain approaches to VLSI layout. It is discussed in more detail in [8].

Clearly VSG is easier to work with than PBWP. Section 2 shows that PBWP and VSG are polynomially reducible to each other. More importantly the corresponding reductions are straightforward and only consist of a certain encoding of directed into undirected neighborhoods and vice versa. Thus results on VSG can easily be applied to PBWP.

Section 3 gives an example of how the correspondence between the PBWP and VSG can be exploited. It contains a straightforward NP-completeness proof for VSG that by virtue of the results in Sect. 2 can be carried over to PBWP.

Unfortunately we are not able to extend this correspondence to BWP. Rule (iii') seems to be essential for the similarity of PBWP to games on undirected graphs. Abandoning this rule deprives us of our main proof device.

2. The Relationship between PBWP and VSG

The purpose of this section is to give formal evidence for the similarity of PBWP to VSG. Towards this end we define two transformations on graphs, one from dags to undirected graphs and the other in the reverse direction.

Definition 1. a) Given a dag $G=(V, E)$ let $G_u=(V, E_u)$ where

$$E_u=\{\{v, w\} \mid (v, w)\in E\}\cup\{\{v, w\} \mid (v, u), (w, u)\in E \text{ for some } u\in V\}.$$

Thus G_u makes cliques out of the immediate predecessors of every vertex in G and then ignores edge directions in G. (See Fig. 1.)

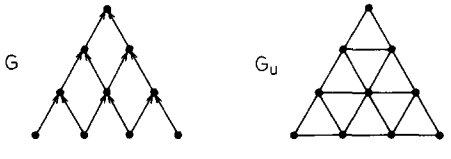

Fig. 1. A dag G and its corresponding undirected graph G_u

b) Given an undirected graph $G=(V, E)$ let $G_d=(V_d, E_d)$ be the following directed graph:

$$V_d = V \cup E$$
$$E_d = \{(v, \{v, w\}), (w, \{v, w\}) \mid \{v, w\} \in E\}.$$

Thus G_d erects a complete binary tree of height 1 over each edge in E, and then discards the edges in E. (See Fig. 2.)

Both transformations are quite straightforward. Transformation (b) never constructs graphs with paths whose length is greater than 1. Thus the graphs it yields can easily be pebbled in the unrestricted BWP using only three pebbles. Thus rule (iii') is a significant prerequisite for the following discussion.

By transforming the instance (G, K) of PBWP into the instance $(G_u, K-1)$ of VSG resp. transforming the instance (G, K) of VSG into the instance $(G_d, K+2)$ of PBWP, we now show that VSG and PBWP are polynomially equivalent. First we reduce PBWP to VSG.

Theorem 2. *(G, K) is a positive instance of* PBWP *if and only if $(G_u, K-1)$ is a positive instance of* VSG.

Proof. "⇒": Assume that S is a strategy for pebbling G in PBWP. We will transform S into a strategy S_u for pebbling G_u in VSG. In S_u we pebble a vertex v at time t if at time t the vertex v loses its pebble in S. A move in S that does not remove a pebble from a vertex in G, generates a null operation in S_u, i.e., an operation that does not do anything except take one time step.

We now show that at each time t, if $\{v, w\}$ is an edge in G_u such that w is pebbled and v is pebble-free in G_u at t (and thus on the cut), then v has a pebble at t in G. The proof is done by case analysis. (Note that for each vertex v in G there is exactly one time t_1 that v receives a pebble and exactly one time t_2 that v loses its pebble. We will say that v has already received its pebble at time t if $t > t_1$, and that v has already lost its pebble at time t if $t > t_2$. This means that a vertex that has no pebble on it may have received (and lost) its pebble already resp. may not have (received and) lost its pebble yet.)

Case 1. Assume that (v, w) is an edge in G.
At time t vertex w has already lost its pebble in G, whereas vertex v has not lost its pebble yet. Therefore v must have a pebble.

Case 2. Assume that (w, v) is an edge in G.
An argument analogous to Case 1 applies.

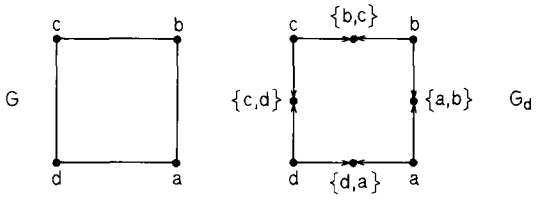

Fig. 2. An undirected graph G and its corresponding dag G_d

Case 3. Assume that v and w have a common immediate successor u in G. Since w lost its pebble already, u has to either have a black pebble on it or u has lost its pebble, too. In each case v, not having lost its pebble yet, has to have a pebble on it.

Therefore (G_u, K) is certainly a positive instance of VSG. We can sharpen this result by observing that in S the number of pebbles before a removal is one greater than the number of pebbles after the removal. Thus if at time t a pebble is placed in S_u, resulting in a cut-size of k, then at time $t-1$ there are $k+1$ pebbles on G in S. This proves the first half of the theorem.

"$\Leftarrow$": Let us assume a strategy S which pebbles G_u in VSG with a maximum cut-size of K. We will construct a strategy S_d that pebbles G in PBWP with $K+1$ black-white pebbles. S_d simulates each placement of a pebble in G_u by several placements and one removal on G. In particular, if vertex v is the i-th vertex to be pebbled by S, this placement is simulated in S_d by the following sequence of moves.

1. Put a white pebble on all vertices that are adjacent to v in G_u, pebble-free in G_u, and pebble-free in G.
2. Put a black pebble on v unless it already has a black pebble.
3. Turn all pebbles on immediate successors of v in G black.
4. Remove the black pebble from v.

We will show the following statements by induction on i.

Induction Hypothesis. Let t_i be the time in S_d of the completion of the simulation of the i-th move in S. The strategy S_d as defined above is a valid strategy in PBWP up to time t_i. Furthermore at t_i the following holds:

(a) A pebble has been removed from vertex x in G at or before t_i exactly if x is pebbled in G_u after the i-th move in S.

(b) A vertex x has a pebble in G exactly if it is on the cut (and thus pebble-free) in G_u after the i-th move in S.

(c) A vertex x has a black pebble in G exactly if x is pebble-free in G_u and there is an immediate predecessor of x in G that has a pebble in G_u after the i-th move of S.

(b) The maximum number of pebbles on G used by S_d up to time t_i is at most one greater than the size of the maximum cut on G_u during the first i moves of S.

Proof of the Induction Hypothesis. Trivial for $i=0$: The cut in G_u is empty and no pebbles are on the graph G.

$i>0$: Assume that the induction hypothesis is true for $i-1$. Furthermore assume that in the i-th move of S vertex v in G_u is pebbled.

All placements in the simulation of the i-th move of S except the placement of the pebble on v are trivially valid, because they only involve white pebbles, and vertices in G that are pebble-free in G_u and thus by (a) have not been pebbled yet in G. The placement on v only happens, if at t_{i-1} vertex v does not have a black pebble in G. By (c) in this case all immediate predecessors of v in G are pebble-free in G_u after the $(i-1)$-st move of S. Therefore step 1 of the

simulation assures that all immediate predecessors of v in G are pebbled in G before the black pebble is placed on v.

We still have to show that after the placement on v in G the pebbles on the immediate successors of v in G can be turned black. So let w be an immediate successor of v in G whose pebble is turned black in step 3 of the simulation. Because of (c) all of w's inputs in G are pebble-free in G_u after the $(i-1)$-st move in S. Furthermore, by definition of G_u, they are all adjacent to v in G_u. Thus after step 1 of the simulation they are all pebbled in G. This shows that S_d is a valid strategy in BWP on G, up to the simulation of the i-th move in S.

Statement (a) follows trivially from step 4 of the simulation. This statement also implies with the definition of the simulation that S_d is indeed a valid strategy in PBWP.

We are left to prove statements (b)–(d). First consider statements (b) and (c). Consider them as statements in the variables x and j (substituted for i) that by the induction hypothesis hold for any vertex substituted for x and for any value $j<i$. We now show that they also hold for any vertex substituted for x if $j=i$. We distinguish several cases.

Case 1. Statements (b) and (c), for any vertex x that is not v and is not adjacent to v in G_u.
Such a vertex is neither touched by the simulation of the i-th step of S, nor does it move on or off the cut in G_u, in the i-th step of S.

Case 2. Statements (b) and (c), for $x=v$.
In this case x ends up without a pebble in G, and with a pebble and off the cut in G_u.

Case 3. Statement (b), for any vertex x that is adjacent to v in G_u.
In this case x is either pebbled in G_u or pebble-free and on the cut, after the i-th move of S. In the first case the simulation of the i-th move of S does not touch x. In the second case x ends up with a pebble in G.

Case 4. Statement (c), for any vertex x that is adjacent to v in G_u but not an immediate successor of v in G. The statement does not change during the simulation of the i-th move of S.

Case 5. Statement (c), for any vertex x that is an immediate successor of v in G.
In this case x either ends up with a black pebble in G or without a pebble in G. In the first case by (a) x is pebble-free in G_u, and furthermore v is one of x's immediate predecessors in G with a pebble in G_u after the i-th move in S. In the second case by (a) x is pebbled in G_u after the i-th move of S.

To prove statement (d) note that the maximum number of pebbles on G during the simulation of the i-th move in S is one greater than the number of pebbles on G at time t_i. By (b) this number is the same as the cut-size in G_u after the i-th move in S.

Statement (d) applied to the entire strategy S_d proves the second part of the theorem. □

Now we reduce VSG to PBWP. Note that the proof of Theorem 3 makes use of Theorem 2.

Theorem 3. *The instance* (G, K) *of* VSG *is positive if and only if the instance* $(G_d, K+2)$ *of* PBWP *is positive.*

Proof. The proof follows from a combination of Theorem 2 with the following theorem. □

Let G_{du} be the graph obtained from G by applying to G the transformations 1.b) and 1.a) in this order. Thus $G_{du}=(V_{du}, E_{du})$ where

$$V_{du}=V\cup\{e' \mid e\in E\}$$

$$E_{du}=E\cup\{\{v, e'\}, \{w, e'\} \mid e=\{v, w\}\in E\}.$$

G_{du} makes out of each edge in G the complete graph on three vertices (see Fig. 3).

Theorem 4. *The instance* (G, K) *of* VSG *is positive if and only if the instance* $(G_{du}, K+1)$ *of* VSG *is positive.*

For the proof of Theorem 4 we need a normal form lemma for VSG strategies on G_{du}.

Lemma 5. *Let* S *be a strategy for* VSG *on* G_{du}. S *can be transformed into a strategy* S' *for* VSG *on* G_{du} *with the following properties:*

a) $VSG(S')\leqq VSG(S)$.

b) *Let* t_z *be the time in* S' *at which a vertex* $z\in V_{du}$ *is pebbled. Let* $\{v, w\}\in E$ *and* w.l.o.g. *let* v *be pebbled before* w *in* S'. *Then* $t_{\{v,w\}'}<t_v$, *and at all times* t *such that* $t_{\{v,w\}'}<t<t_v$ *a vertex* $\{v, u\}'$ *with* $u\in V$ *is pebbled in* S'.

Condition (b) essentially says that each vertex $\{v, w\}'$ is pebbled in S' directly before the first of v and w is pebbled in S'.

Proof. S is transformed into S' by an iterative process. Each step works on a vertex $\{v, w\}'$ for which condition (b) does not hold yet. The placement of a pebble on $\{v, w\}'$ is moved directly before the placement of a pebble on the first of v and w (say v). Certainly this transformation does not make condition (b) false for any vertex $\{x, y\}'$ for which it held before. We now show that it does not increase the cut either. The argument is by case analysis.

Case 1. $t_{\{v,w\}'}<t_v$. Delaying the placement on $\{v, w\}'$ could only increase the cut if $\{v, w\}$ is placed on it. But since at all times during $[t_{\{v,w\}'}, t_v)$ both neighbors of $\{v, w\}'$ are pebble-free this does not happen.

Case 2. $t_v<t_{\{v,w\}'}$. We can move the placement on $\{v, w\}'$ directly after the placement on v without increasing the cut, because the placement on $\{v, w\}'$, as

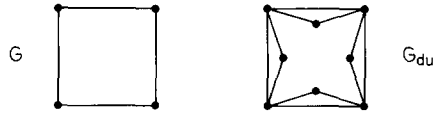

Fig. 3. An undirected graph G and its corresponding undirected graph G_{du}

long as it happens after t_v, moves $\{v, w\}'$ off the cut while not changing the cut otherwise. We then can switch the placements on v and $\{v, w\}'$. This does not increase the cut, because both before and after the transformation the first of the two placements increases the cut by at least 1 or 2, depending on whether v is on the cut before the placements. □

We now are ready to prove Theorem 4.

Proof of Theorem 4. "⇒": Let S be a strategy for VSG on G such that $VSG(S) \leqq K$. We transform S into a strategy S_{du} for VSG on G_{du} by simply inserting the placements on vertices $\{v, w\}'$ into S directly before the first of the placements on v or w. Thus S_{du} when restricted to vertices in V equals S. But before each placement in S_{du} on a vertex $v \in V$ all vertices $e' \in V_{du} - V$ that are adjacent to v and still pebble-free are pebbled. It is easy to see that the cut in G after $v \in V$ is pebbled is the same as the cut in G_{du} after v is pebbled. Furthermore a placement on a vertex $e' \in V_{du} - V$ in S_{du} increases the cut whereas a placement on a vertex $v \in V$ in S_{du} decreases the cut by exactly 1. Thus the maximum cut in S' has size at most $K+1$.

"⇐": Let S_{du} be a strategy for VSG on G_{du} such that $\mathrm{VSG}(S_{du}) \leqq K+1$. W.l.o.g. we can by Lemma 5 assume that S_{du} fulfills condition 5.b). Therefore, as in "⇒" above, deleting all placements on vertices $e' \in V_{du} - V$ yields a strategy S for VSG on G with $\mathrm{VSG}(S) \leqq K$. □

3. The VSG is NP-complete

The NP-completeness proof of VSG is done by reducing a modified version of the min-cut linear arrangement problem to it (see [3]). This modification minimizes the number of edges passing *over* each vertex rather than those passing *between* neighboring vertices in the arrangement.

Definition 6. An instance of the *Modified Mincut Linear Arrangement Problem* (MMCLA) is a pair (G, K) where $G=(V, E)$ is an undirected graph with vertex set $V=\{v_1, \ldots, v_n\}$ and K is a non-negative integer, called the *with*. The instance (G, K) is *positive* if there is a labeling $\lambda: V \to \{1, \ldots, n\}$ of the vertices such that

$$\max\{\mathrm{width}(v_i) \mid 1 \leqq i \leqq N\} \leqq K$$

where

$$\mathrm{width}(v_i) = |\{\{v_{j_1}, v_{j_2}\} \in E \mid \lambda(v_{j_1}) < i < \lambda(v_{j_2})\}|.$$

The MMCLA can be proved NP-complete by reducing SIMPLE MAXCUT to it (see [3]) in a similar fashion as done in [2] with the standard min-cut linear arrangement problem.

Theorem 7. *The* VSG *is NP-complete.*

Proof. Let $(G=(V, E), K)$ be an instance of the MMCLA. We construct an instance (G', K') of VSG as follows:

Let $V=\{v_1, \ldots, v_n\}$. Then we define $G':=(V', E')$ where

$$V'=\{v_i(j) \mid 1 \leqq i \leqq N, 0 \leqq j \leqq N\}$$

$$E'=\{\{v_i(j), v_i(k)\} \mid 1 \leqq i \leqq N, 0 \leqq j<k \leqq N\} \cup \{\{v_i(j), v_j(i)\} \mid \{v_i, v_j\} \in E\}$$

$$K'=K+N.$$

Thus each vertex in G becomes an $(N+1)$-clique in G'. Edges between vertices in G become edges between corresponding cliques in G', such that edges converge in different vertices. A vertex $v_i(j)$ in the $(N+1)$-clique $C_i=\{v_i(j) \mid 0<j<N\}$ representing vertex $v_i \in V$ has at most one edge leaving C_i. Furthermore C_i has at least one vertex $v_i(0)$ that has no edges leaving C_i.

We now show that (G, K) is a positive instance of the MMCLA if and only if (G', K') is a positive instance of VSG.

"$\Rightarrow$": Assume that there is a labeling of G such that $\max\{\text{width}(i) \mid 1 \leqq i \leqq N\} \leqq K$. We will pebble G' achieving a maximum vertex cut of size at most K'.

We pebble the cliques C_i in the order $i=1, \ldots, N$. The vertices in each clique are pebbled in the order $j=0, \ldots, N$.

It is easy to see that directly after the vertex $v_i(0)$ is pebbled the number of vertices in the cliques $C_{i+1}, \ldots, C_N$ that are on the cut is exactly width(i). In clique C_i vertices $v_i(1), \ldots, v_i(N)$ are on the cut. Those are all vertices on the cut. Thus after $v_i(0)$ is pebbled the cut has size width$(i)+N$. From then on until we finish pebbling C_i the size of the cut does not increase. This is, because each vertex $v_i(1), \ldots, v_i(N)$ is taken off the cut when it is pebbled. Furthermore pebbling it can only put vertices outside C_i on the cut. But there is only one edge leading outside C_i out of each vertex inside C_i. Thus the size of the cut never exceeds

$$\max\{\text{width}(i) \mid 1 \leqq i \leqq N\}+N \leqq K+N=K'.$$

"$\Leftarrow$": Conversely assume that we have a strategy for pebbling G' in the VSG with a maximum cutsize of at most $K'=K+N$. We label the vertices of G in the sequence in which their corresponding cliques in G' receive their first pebble.

We now show that with this labeling width$(v_i)+N$ is at most as large as the cut-size in G directly after the clique C_i received its first pebble. In order to show this we prove that at this time an edge that contributes to width(v_i) in G has at least one of its end vertices on the cut in G'. Let $\{v_{j_1}, v_{j_2}\}$ be an edge contributing to width(v_i) in G. In G', $\{v_{j_1}(j_2), v_{j_2}, (j_1)\}$ is an edge between a clique C_{j_1} that has already received its first pebble, and a clique C_{j_2} that has not yet received a pebble. If $v_{j_1}(j_2)$ has a pebble, then $v_{j_2}(j_1)$ is on the cut, since this vertex, being in C_{j_2} is still pebble-free. If $v_{j_1}(j_2)$ is pebble-free, it itself is on the cut, since some vertex in C_{j_1} is already pebbled.

Thus each edge contributing to width(v_i) in G also contributes a vertex to the cut in G'. All these vertices are outside C_i. Furthermore, since C_i just received its first pebble, there are N vertices on the cut in C_i. Thus the cut has size at least width$(v_i)+N$. Maximizing over i completes the proof. □

4. Conclusions

We have established a link between the progressive black-white pebble game PBWP and a game VSG on undirected graphs that makes the concept of a vertex separator dynamic. The VSG is a natural game on undirected graphs, and is closely related to certain graph layout and searching problems. In fact, it is the vertex separator version of an edge separator game that is better known as the min-cut linear arrangement problem of undirected graphs, and is of importance to linear circuit layout. By linking PBWP to VSG, we exhibit a correlation between two important problem domains – the register allocation problem for nondeterministic straight-line programs, and the search for small separators of undirected graphs. It is hoped that this connection will help research in both areas.

Acknowledgements. The NP-completeness proof given in this paper has been inspired by a similar proof of M.R. Garey, D.S. Johnson and C. Papadimitriou of the NP-completeness of a different graph searching game (communicated in [4]). Many valuable suggestions by A.V. Aho helped improve this presentation.

References

1. Cook, S.A., Sethi, R.: Storage requirements for deterministic polynomial finite recognizable languages. JCSS **13**, 25–37 (1976)
2. Gavril, F.: Some NP-complete problems on graphs. Proc. 11th Conf. on Information Sciences and Systems, John Hopkins University, Baltimore, MD (1977), 91–95
3. Garey, M.R., Johnson, D.S.: Computers and Intractability, A Guide to the Theory of NP-Completeness, Freeman, San Francisco, CA (1979)
4. Garey, M.R., Johnson, D.S., Papadimitriou, C.: Private communication
5. Gilbert, J.R., Lengauer, T., Tarjan, R.E.: The pebbling problem is complete in polynomial space. SIAM J. Comp. **9**, 513–525 (1980)
6. Leiserson, C.E.: Area-efficient graph layouts (for VLSI). 21st Annual *IEEE-FOCS* (1980), 270–281
7. Lengauer, T.: Upper and lower bounds for time-space tradeoffs in a pebble game. Technical Report STAN-CS-79-745, Stanford University, Stanford, CA 94305 (1979)
8. Lengauer, T.: Upper and lower bounds on the complexity of the min-cut linear arrangement problem on trees. Technical Report, Bell Laboratories, Murray Hill, N.J. 07974 (1980). (To appear in SIAM Journal of Algebraic and Discrete Methods)
9. Lengauer, T., Tarjan, R.E.: The space complexity of pebble games on trees. Inf. Proc. Let. **10**, 184–188 (1980)
10. Lipton, R.J., Tarjan, R.E.: A planar separator theorem. SIAM J. Appl. Math. **36**, 177–189 (1977)
11. Loui, M.C.: The space complexity of two pebble games on trees. LCS Report No. 133, M.I.T., Cambridge, Mass (1979)
12. Meyer auf der Heide, F.: A comparison between two variations of a pebble game on graphs. University of Bielefeld, Bielefeld, West-Germany (1978)
13. Pippenger, N.: Pebbling. Proc. of the 5th IBM Symposium on Math. Foundations of Comp. Sci., Hakona, Japan (1980)
14. Philipp, R., Prauss, E.: Über Separatoren in planaren Graphen. ACTA Informatica **14**, 87–97 (1980)
15. Sethi, R.: Complete register allocation problems. SIAM J. Comp. **4**, 226–248 (1975)

Received November 21, 1980 / August 10, 1981