Theoretical Computer Science 110 (1993) 197–214
Elsevier

Mathematical Games

Geography

A.S. Fraenkel

Department of Applied Mathematics and Computer Science, The Weizmann Institute of Science, Rehovot 76100, Israel

S. Simonson*

Department of Mathematics and Computer Science, Stonehill College, North Easton, MA 02357, USA

Communicated by C. Berge
Received July 1991
Revised June 1992

Abstract

Fraenkel, A.S., Geography, Theoretical Computer Science 110 (1993) 197–214.

Generalized Geography is an impartial two-person game played on a digraph $G=(V, A)$. In impartial Arc (Vertex) Geography, a token is initially placed on a special start vertex, and the players alternately move the token along unused arcs (vertices) of G. The player first unable to move loses and his opponent wins. The question of who wins these games IAG and IVG is known to be PSPACE-complete.

Both impartial versions with *two* tokens on special start vertices are proved PSPACE-complete even for DAGs but polynomial for directed trees. The partizan variations, PAG and PVG, with one token per player are PSPACE-complete even for bipartite degree-restricted digraphs. They are NP-hard for DAGs, but polynomial for directed trees.

1. Introduction

Geography is a childrens' game where two players alternately choose a country, each name beginning with the same letter that ends the previous country's name. The first player who is unable to choose a new country loses; his opponent wins.

Generalized Geography is a two-person game played on a directed graph (digraph) $G=(V, A)$ with a specified vertex $v_0 \in V$. Players alternate choosing a *new* arc from A.

Correspondence to: A.S. Fraenkel, Department of Applied Mathematics and Computer Science, The Weizmann Institute of Science, Rehovot 76100, Israel.

*Supported in part by NSF Grant CCR-8710730.

The first arc chosen must have its tail at v_0, and each subsequently chosen arc must have its tail at the vertex that was the head of the previous arc. The first player unable to choose such a new arc loses, and his opponent wins [4].

Generalized Geography is PSPACE-complete [12], even if the digraph is bipartite, planar, and has no in/out-degree exceeding 2 and no degree exceeding 3. These results remain true for the variation where the players choose vertices rather than arcs. (In this variation, each *new* vertex must be a follower of the last vertex chosen, and the player who first cannot choose such a vertex loses.) In fact, these restrictions were first proved PSPACE-complete for Vertex Geography [7].

We consider a variation of this game in which there are two specified vertices v_1 and v_2. This variation was first proposed in [13]. The players alternately choose arcs. We refer to the player who moves first as I and the player to move second as II. The first arc chosen by the two players, I and II, must have its tail, respectively, at v_1 and v_2; and each subsequently chosen arc must have its tail at the vertex that was the head of the arc previously chosen by that player. That is, rather than taking turns moving a single token, which is equivalent to the two Geography games described above, the players take turns each moving his own token. We call this variation Partizan Geography; henceforth, we shall refer to the original version as Impartial Geography.

The computational complexity of partizan games was investigated by Morris [8]. He considered the situation where a number of trivial games are given and each player at his turn may make a move in any one of these games. The new game is the disjunctive sum of the trivial games. An example of this is Nim (with n heaps), which is the sum of n trivial games, each played with a single heap.

Morris [8] proved that the decision problem of whether player I can win a disjunctive sum of trivial games is PSPACE-complete. Other work has shown that if repetitions of positions is prohibited, then this decision problem remains PSPACE-complete [9] or becomes ExpSPACE-complete [11].

Our work extends the results of Morris and others by showing that PSPACE-completeness is preserved even for various restricted Partizan Geography games. Our main results were completed in 1988 and presented at the International Conference of Algebraic Graph Theory in 1989 [3] and at Argonne Labs Symposium for Undergraduates in Science, Mathematics and Engineering in 1990 [13]. Work on Partizan Geography appears also in [1, 2], where PVG, under the name TRON, is proved to be PSPACE-complete.

Both Partizan and Impartial Geography can be played on arcs or vertices, giving us 4 games which we will abbreviate as PAG, PVG, IAG and IVG. The vertex and arc versions of Geography generally have the same complexity. This is in contrast to games like the Shannon Switching Game [5], which is solvable in polynomial time, while its vertex version, Generalized Hex, is PSPACE-complete [10]. There are, however, differences between the partizan and impartial versions of Geography. We will show that, in general, the partizan versions are more difficult to solve.

We also consider variants of IAG and IVG where 2 tokens are on the digraph, but each player may move either one on his turn. We call these IAG2 and IVG2. Their complexity resembles that of PAG and PVG more than that of IAG and IVG.

The arc versions of these 3 games are listed below as formal decision problems. The 3 corresponding vertex versions can be defined similarly except for how the game ends. In the vertex versions, a player loses when he is unable to move his token along an arc whose head has never contained a token (i.e. all his arc choices have had or currently have a token on their heads).

Impartial Arc Geography (IAG)

This is equivalent to the original Generalized Geography problem defined in [4].

Input: A digraph $G=(V, A)$ and a specified vertex v_0 in V.

Question: Does player I have a forced win in the following game played on G?

Players alternate moving a single token along arcs in A. The first move must be along an arc whose tail is at v_0 and each subsequent move must be along an as yet unused arc whose tail is at the vertex that contains the token. The player first unable to move the token along a new arc loses.

Partizan Arc Geography (PAG)

Input: A digraph $G=(V, A)$ and 2 specified vertices v_1 and v_2 in V.

Question: Does player I have a forced win in the following game played on G?

Players alternate moving tokens along arcs in A. Each player has one token and can move only that one. The first token moved by player I/II must be along an arc whose tail is at v_1/v_2, and each subsequently moved token must be along an as yet unused arc whose tail holds that player's token. The player first unable to move his token along a new arc loses.

Impartial Arc Geography with 2 Tokens (IAG2)

Input: A digraph $G=(V, A)$ and 2 specified vertices u and v in V.

Question: Does player I have a forced win in the following game played on G?

Players alternate moving either one of two tokens along arcs in A. The first token moved must be along an arc that has its tail at either u or v, and each subsequently moved token must be along an as yet unused arc whose tail holds a token. The player first unable to move a token along a new arc loses.

In Section 2, we describe PSPACE-completeness and NP-hardness results. PAG and PVG are PSPACE-complete for bipartite digraphs with in/outdegree not exceeding 2 and degree not exceeding 3. They are NP-hard when G is acyclic or planar. The games IAG2 and IVG2 are PSPACE-complete even when G is acyclic.

In Section 3, we describe polynomial-time solutions for acyclic digraphs (DAGs) and trees. The games IAG and IVG are solvable in $O(n^2)$ time when G is acyclic, and $O(n)$ time when G is a directed tree. The games PAG and PVG are solvable in $O(n^2)$ time when G is a directed tree, and IAG2 and IVG2 are solvable in $O(n^4)$ time when G is a directed tree.

A summary of our main results appears in Table 1.

Table 1
Summary of our main results

	PAG (PVG)	IAG2 (IVG2)	IAG (IVG)
Degree-restricted bipartite digraphs	PSPACE-complete	PSPACE-complete	PSPACE-complete
Directed acyclic graphs	NP-hard	PSPACE-complete	Polynomial
Trees	Polynomial	Polynomial	Polynomial

2. PSPACE-completeness and NP-hardness results

It is known that IAG is PSPACE-complete for planar bipartite digraphs with in/outdegree not more than 2 and degree not more than 3 (henceforth, digraphs with this degree restriction will be called *degree-restricted*) [12]. It is not difficult to see that the same results holds for IVG.

In this section we describe PSPACE-completeness and NP-hardness results for our variations IAG and IVG. We first show that PAG and PVG are PSPACE-complete for bipartite degree-restricted digraphs. The reduction is from 3QBF, a version of quantified Boolean formula with exactly 3 literals per clause [4]. We illustrate our reduction proofs from 3QBF by means of the following example:

$$F=(\exists x_1)(\forall x_2)(\exists x_3)((x_1+x_2+\bar{x}_3)(\bar{x}_1+\bar{x}_2+\bar{x}_2)(x_2+x_3+x_3)).$$

The proofs, of course, are described in full generality, with the example provided as an aid for the reader.

Theorem 2.1. PVG *is* PSPACE*-complete for bipartite degree-restricted digraphs.*

Proof. First note that PVG is in PSPACE, since the number of moves in any play of the game is bounded by the number of vertices.

We show that 3QBF reduces to PVG. By adjoining a clause of the form $(x_n+x_n+\bar{x}_n)$, we may assume, without loss of generality, that the number of variables in the 3QBF formula is odd. Given an instance F of 3QBF, the construction and the two specified vertices are shown in Fig. 1a.

For each variable in F, we have a 4-vertex *diamond*, with the left and right vertices of the diamond representing the variable and its negation, respectively. For each clause in F, we have a *cluster* of 11 vertices. A clause cluster is shown in Fig. 1b. (For the sake of clarity, in Fig. 1a, we drew a clause cluster as a black box, with just the connections drawn in.) The existential variable diamonds are connected so that the bottoms and the tops of successive diamonds coincide. The very bottom vertex of this chain of existential diamonds connects to the 3 *existential connectors* in each clause cluster. The universal variable diamonds are similarly connected so that the bottoms and the tops of successive diamonds coincide. There is one extra vertex connected to the end of the universal diamond chain which connects to a *universal connector* in each clause

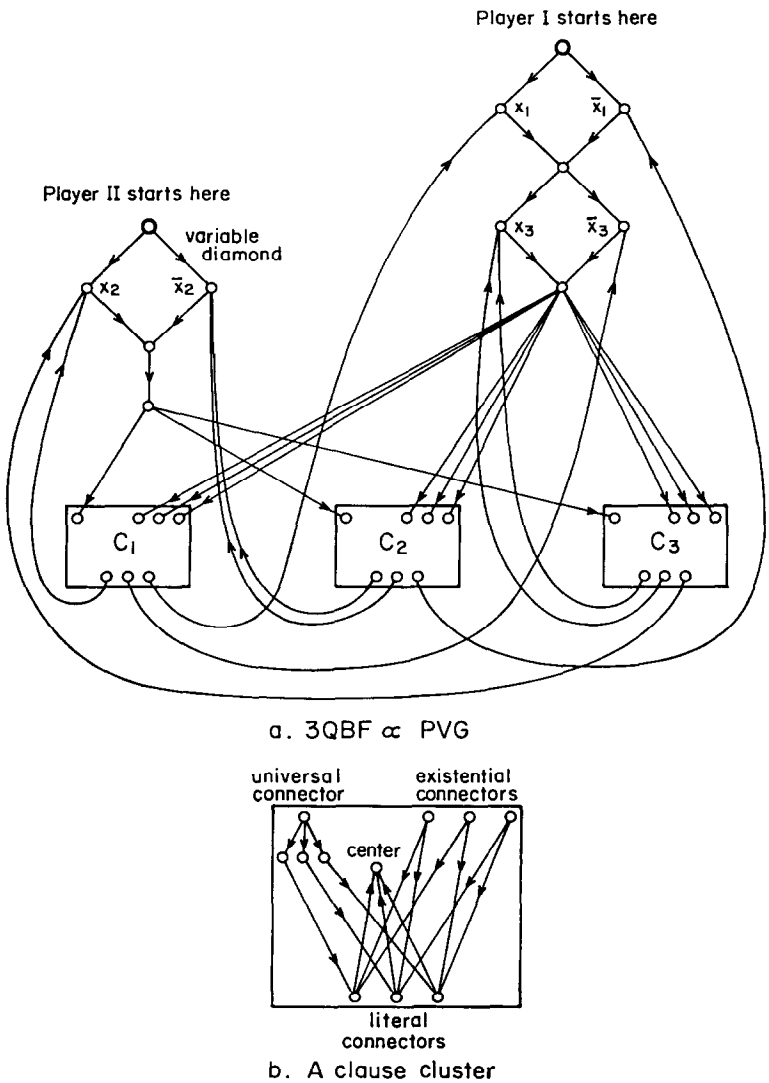


Fig. 1. Illustration of the proof of Theorem 2.1.

cluster. Finally, the 3 *literal connectors* in each clause cluster are connected to the appropriate diamond vertices representing the variables in that clause. The first player to move, I, starts at the top of the existential diamond chain, and the second player, II, starts at the top of the universal diamond chain.

We claim that player I can win PVG on G if and only if F is true. If F is true, then I chooses his path so that he goes through the existential literals which he wants to be true. That is, blocking a literal-labeled vertex on a diamond corresponds to choosing a true value for that literal.

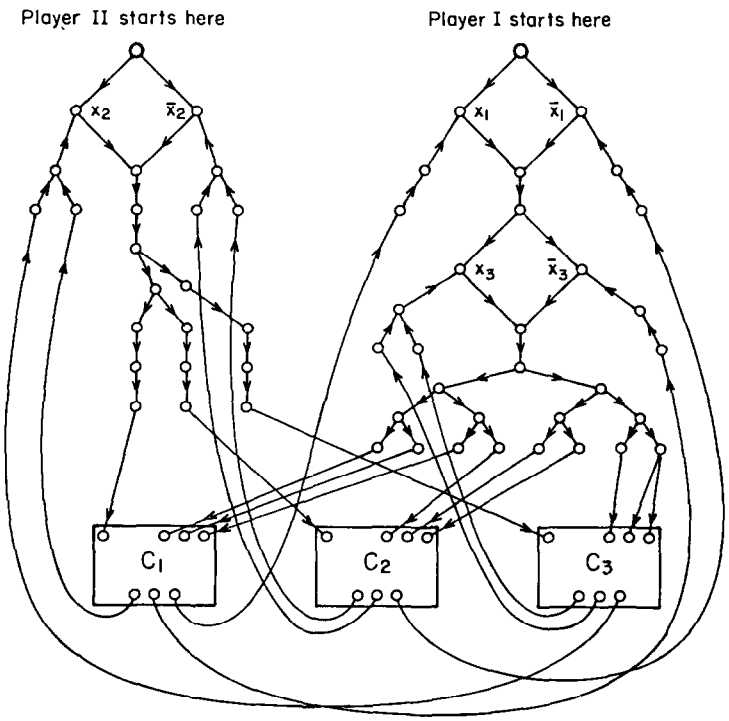
Meanwhile, II is choosing a path (T/F values) for all the universal variables. After this phase, II needs to choose a clause, hoping to find a clause which has all false literals. Player I then chooses an existential connector in the same clause. Note that, in each clause cluster, every existential connector is adjacent to exactly two literal connectors. Player I chooses this vertex such that the third literal connector in the clause, i.e. the one he is not adjacent to, is itself adjacent to a true literal. That is, the path from the third literal connector to the diamond literal vertex is blocked. We know that player I can do this, since we assume that F is true, and every clause must, therefore, have at least one true-valued literal.

At this point, II has two options. He chooses to move either towards this third vertex, or else towards one of the other two vertices. If he chooses the latter, then player I can block him off immediately and win. If he chooses to move to the third vertex, then I moves to the *center* in the clause cluster, via a literal connector. This blocks II from proceeding to the center, and, since II is at a vertex adjacent to a true-valued literal, he is blocked from the other side as well, and player I wins.

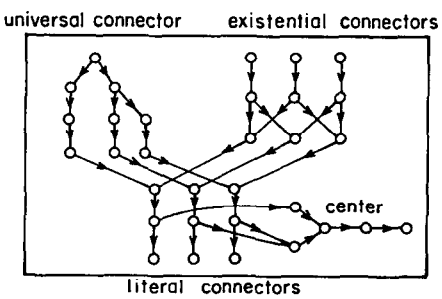
If F is false, then II can win as follows. He chooses a path through the universal variables that includes exactly the vertices which he wants to be true, trying to ensure that at least one clause will have all false-valued literals. Meanwhile, I chooses his own path (T/F values) through the existential literals. At the end of this phase, II chooses a clause cluster whose literal connectors are each adjacent to false-valued literals. Player I then chooses a pair of these 3 literal connectors, or he moves to a different clause. Player II simply moves towards the third vertex (the one not chosen by I). Now, since the vertex is adjacent to a false-valued literal, II can move to it if I moves to the center. If I moves to a literal, then II moves to the center; so, II makes the last move in any case. Note that this strategy also works if two literal connectors are connected to the same literal, and I is on one literal connector and II on the other.

The reduction can be modified to have the degree restrictions we claim. Fig. 2 shows the result of modifying the digraph in Fig. 1. The idea is to replace the arcs of a vertex with high degree with a directed binary tree. For every vertex of in/outdegree k, a directed binary tree with k leaves is introduced. Consider the edges from the literal connectors to the literals. Let k_j ($\bar{k}_j$) be the number of occurrences of literal x_j ($\bar{x}_j$) in F. In order to ensure that all paths from the literal connectors to the literals are the same length, and to achieve the low degree, we let $h = \max_{1 \leqslant j \leqslant n}(\lceil \log_2 k_j \rceil, \lceil \log_2 \bar{k}_j \rceil)$. (In Fig. 2, $h = 1$.) We construct a binary tree T of height h in between each literal connector and its corresponding literal. If a particular $\log k_i$ is smaller than h, then the tree for x_i still keeps its required height h, but is pruned so that the number of leaves is exactly k_i. The new clause clusters have $30 + h + 1$ vertices. The $h + 1$ comes from a chain of vertices p of length $h + 1$ connected to the center, and is needed in order to keep the path length from the clause cluster through the center equal to the path lengths to the literals.

The other high-degree vertices that cause problems are those with edges between the last diamond literal and the clause clusters. The trees in this case all have height



a. 3QBF $\propto$ PVG with degree restrictions



b. A degree restricted clause cluster

Fig. 2. Degree restriction applied to Fig. 1.

$h' = \log_2(3m)$, where m is the number of clauses ($h' = 4$ in Fig. 2). To make it easier to see the idea, we again draw the clusters as black boxes.

Finally, the reader should note that, although the graph in our reduction shown in Fig. 1 is bipartite, the degree modified version in Fig. 2 is not if $h + h'$ is odd. It is, however, an easy matter to modify it and make it bipartite. An extra edge is adjoined, directed from each root of T to its literal (at the diamond). To preserve timing, one extra edge is also adjoined to each path p. This final modification preserves the degree restriction, and provides a bipartite graph. This completes the proof. □

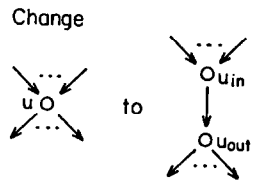


Fig. 3. PVG $\propto$ PAG.

This result for PVG remains true for PAG because we can reduce PVG to PAG without affecting the bipartite or degree restrictions.

Theorem 2.2. PAG *is* PSPACE*-complete for bipartite degree-restricted digraphs.*

Proof. First, note that PAG is in PSPACE since the number of moves is bounded by the number of edges. We reduce PVG to PAG. Given a digraph G for PVG, we do a local substitution for each vertex, as shown in Fig. 3. Each vertex u in G is replaced by an arc (u_{in}, u_{out}). The arcs that go into u, if any, will now go into u_{in}, and the arcs that go out from u, if any, will now go out from u_{out}. If the starting vertices in G were u and v, then the new starting vertices will be u_{in} and v_{in}. Finally, for every vertex u in G with outdegree zero, we add an arc (u_{out}, u_{extra}). Call this new graph G'. We claim that player I has a forced win for PVG on G if and only if he has a forced win for PAG on G'. Every sequence of moves for PVG on G corresponds to a sequence of moves for PAG on G', and vice versa. The thing to note is that in PAG the odd numbered moves of each player are *dummy* moves where the player is forced to choose an arc (u_{in}, u_{out}), and the even-numbered moves correspond to *true* moves between vertices in G. Further note that a player in PAG will never get stuck on an even-numbered move, only on an odd-numbered move. Therefore, a player in PVG will lose on his kth move iff he will lose in PAG on his $(2k-1)$th move. Note that the u_{extra} vertices are added to vertices of outdegree zero to guarantee that this is always true. Otherwise, a player in PAG might be stuck on an even-numbered move, i.e. the move coming out of some u_{out} vertex.

Finally, note that the construction preserves the bipartiteness and the degree restrictions. □

Theorem 2.3. PVG *and* PAG *are* NP*-hard for planar digraphs.*

Proof. There is a simple reduction from Directed Planar Hamiltonian Path with specified endpoints [4] to PVG. Given a digraph G with n vertices and specified start and end vertices, construct a digraph H consisting of a chain of n vertices and let player I start at the vertex with indegree 0. Let player II start at the start vertex of G. It is easy to see that II will win PVG if and only if G has a Hamiltonian path from the start vertex to the end vertex. This, along with Theorem 2.2, implies that PAG is also NP-hard for planar digraphs. □

In another direction we show that PVG and PAG are NP-hard even when G is acyclic. Here the reduction is from Vertex Cover [4]. PSPACE-complete results would be better here, but we leave this as an open question.

Theorem 2.4. PVG *and* PAG *are* NP-*hard for DAGs.*

Proof. We reduce Vertex Cover to PVG. The result for PAG follows from Theorem 2.2. We describe the reduction, without loss of generality, by the example in Fig. 4. Given a graph H and a positive integer k, we construct a digraph G with two specified vertices. We claim that the original graph H has a vertex cover of size k if and only if II can win PVG on the digraph G. Before we describe the details of the reduction, note the *cigars* in Fig. 4. The cigar notation is a shorthand to make the diagram easier to

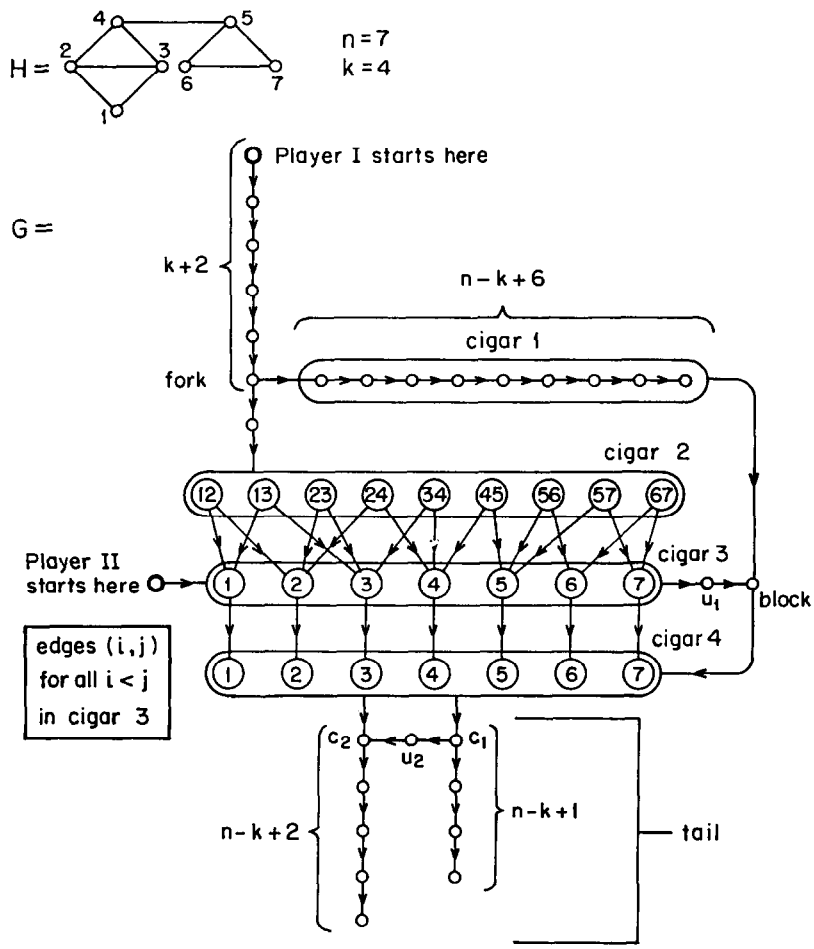


Fig. 4. Vertex Cover $\propto$ Acyclic PVG.

draw. If an arc from a vertex v ends at the border of a cigar, it means that there is an arc from v to each of the vertices encircled by that cigar; and if an arc to v begins at the border of a cigar, it means that there is an arc from each one of the vertices encircled by that cigar to v. Note that if an arc goes right through the wall of a cigar, then it is just a normal arc between two vertices.

The reduction consists of 4 cigars connected together in a specific way, and a *tail*. Let n be the number of vertices in H and let e be the number of edges in H. Cigar 1 has $n-k+6$ vertices; cigar 2 has e vertices, each labeled by the H-edge it represents; and cigars 3 and 4 each have n vertices. Player I starts at the top of a chain of $k+3$ vertices. The $(k+2)$th vertex from the top is called the *fork* and connects to the leftmost vertex in cigar 1. The $(k+3)$th vertex from the top connects to each vertex in cigar 2. Cigar 2 connects to cigar 3 in the natural way shown in Fig. 4. That is, each vertex in cigar 2, which represents an edge i, j in H, is connected to vertices i and j in cigar 3. The starting vertex for II is connected to each vertex in cigar 3. Within cigar 3, there are arcs (i, j) for all pairs of vertices i, j, where $i < j$. These arcs are not drawn in Fig. 4 for the sake of clarity. Cigar 3 is connected to cigar 4 as shown, and every vertex in cigar 3 connects to a vertex labeled u_1. Every vertex in cigar 1 is connected to a vertex labeled *block*, and u_1 is also connected to the block. The block is connected to every vertex in cigar 4. Finally, there is a *tail* which consists of two chains: a shorter one with $n-k+1$ vertices and a longer one with $n-k+2$ vertices. The head of the shorter chain, c_1, is connected to a vertex u_2, which itself is connected to the head of the longer chain, c_2. Every vertex in cigar 4 is connected to both c_1 and c_2.

If H has a vertex cover of size k, then II can win. Player II's strategy is to choose the k vertices in the vertex cover by moving left to right through cigar 3. After II has chosen the kth vertex, I moves into the fork, and II moves out of cigar 3, to u_1. If I now moves to the right, he will lose. This is because he has only $n-k+6$ moves left, while II also has that many by moving via c_1 and c_2. Hence, player I must move down from the fork and subsequently choose some uv in cigar 2. Since II has chosen a vertex cover in cigar 3, at least one of u or v is blocked. Player II now chooses a vertex u or v in cigar 4, that was not previously chosen by him in cigar 3, i.e., that was not necessarily included in the vertex cover. (Note that if both vertices were chosen by him in the vertex cover, then he can pick either one now.) Player I is now forced to move into the vertex in cigar 3, which has an arc to the vertex in cigar 4, on which II is residing. Player II continues by moving in order to c_1, u_2 and c_2, thereby blocking off both chains in the tail from player I. Player I can try to sneak around through cigar 3 and down into cigar 4, but, when he is ready to enter the tail, II would have finished blocking off c_2. Furthermore, it does not help I to traverse cigar 3 exhaustively since he has at most $n-k$ moves and II has more.

If H has no vertex cover of size k, then player I can win. First note that II must choose at least k vertices in cigar 3. If he chooses less, and then moves out of cigar 3 to the right or down, he gets a path of total length at most $n+6$, while player I can always move along a path of total length $n+7$ by moving to the fork, and then through cigar 1.

Hence, we assume that II chooses k or more vertices from cigar 3. Player I can then win using the following strategy. He moves to the fork, and waits to see if II will choose more than k vertices or exactly k. That is, does II stay in cigar 3 or move out? We examine each case separately.

Case 1: Player II moves out (*i.e. chooses exactly k vertices*)

If II moves from cigar 3 into cigar 4, player I can win by moving into cigar 1 for a path of total length $n+9$, while player 2 gets total length at most $n+5$. Therefore, II must move out of cigar 3 to u_1. Then player I will move down from the fork and choose an edge which was not covered by any of the k vertices chosen by player 2. This edge must exist because we assume that H has no vertex cover of size k. Player I then enters cigar 4 just after II leaves cigar 4 into the tail. We claim that this ensures a win for player I since II cannot block both chains in the tail. If II tries by moving to c_1 and then to u_2, I blocks him by moving to c_2. If II does not move to u_2 then I wins by moving to c_2. If II moves into c_2 from cigar 4, then I moves into c_1 after II has left c_2, and since the short chain is just one shorter than the long chain, player I will be the winner.

Case 2: Player II stays in the cigar (*i.e. chooses more than k vertices*)

If II stays in cigar 3, then I moves into cigar 1, and stays in cigar 1 as long as II stays in cigar 3. If II subsequently moves down from cigar 3 into cigar 4, player I then moves to the block, and enters cigar 4 just after II leaves cigar 4 for the tail. As in case 1, this ensures a victory for player I. Finally, if II moves out of cigar 3 to u_1, then I wins immediately by moving out of cigar 1 to the block, blocking player II. □

To conclude this section, we show that both IAG2 and IVG2 are PSPACE-complete even for DAGs. IAG2/IVG2 are the arc and vertex variations of Geography where there are two tokens that the players can move, and either player can move either token at his turn.

Theorem 2.5. IAG2 *is* PSPACE-*complete for* DAGs.

Proof. Since IAG2 is clearly in PSPACE, it suffices to show that 3QBF reduces to IAG2. As in the proof of Theorem 2.1, we may assume without loss of generality that n, the number of variables of the given instance F of 3QBF, is odd. The reduction is illustrated in Fig. 5. In *regular play*, the $3n$ diamond arcs are traversed first, where player I makes the last move with token 1. Then player II selects a clause with token 2, and player I selects a literal. Player I wins if and only if formula F is true.

If I, at any time during the play, selects a clause, then II moves onto the 2-path. Since the total number of moves is now $3n+3=3(n+1)$ which is even, II wins. The same argument shows that if I embarks, at any time, on the 2-path, then II wins.

So, assume that II selects a clause c_i before the diamond chain has been fully traversed. We may assume that II embarks on this experiment only if F *can* be made true by I, since, otherwise, II can win using regular play. Further note that once II has

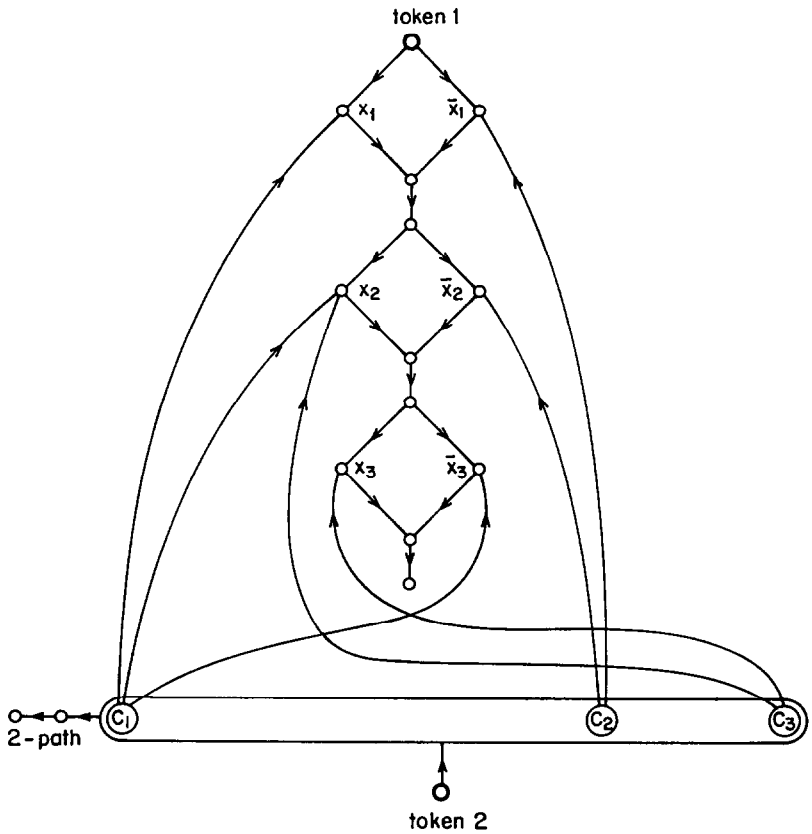


Fig. 5. 3QBF $\propto$ Acyclic IAG2.

selected a clause, I necessarily has to select a literal, since, otherwise, II can move onto the 2-path and win.

Suppose first that the clause c_i selected by II has already been made true during the partial diamond traversal. Then I moves to a literal previously traversed, which is the last move for this token. Since the total number of moves is now $3n+2$, which is odd, I wins.

So, assume that the clause c_i selected by II is not yet true. Then there is an odd-indexed variable x_k or $\bar{x}_k$ in c_i, say x_k, on the part of the diamond chain not yet traversed, since, otherwise, F could be made false by player II. Then I moves token 2 to x_k and follows the strategy of moving token 2 whenever II moves it. Note that, since k is odd, there is an even number of moves from x_k to the leaf. Thus, II will be the first one to move token 1 again. Hence, the choice of odd-indexed variables by I and even-indexed variables by II has been preserved; so eventually, player I will be able to move onto x_k, thereby winning. (In particular, note that, when II selected the clause c_i prematurely, token 1 could not have been on the top vertex of diamond k.) □

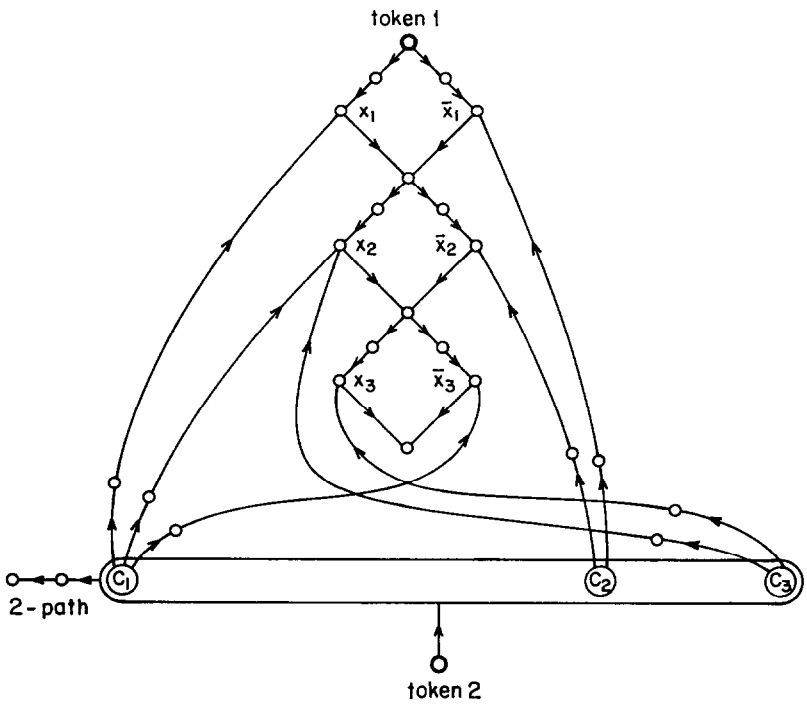


Fig. 6. 3QBF $\propto$ Acyclic IVG2.

Theorem 2.6. IVG2 *is* PSPACE-*complete for* DAGs.

Proof. The reduction from 3QBF to IVG2 is illustrated in Fig. 6. The arguments are similar to those used in the proof of Theorem 2.5 and, hence, are omitted. □

3. Polynomial results for DAGs and trees

We first observe that IAG and IVG (the problems are identical on DAGs) can be solved in $O(e)$, time where e is the number of arcs in G, and $G=(V, A)$ is a directed acyclic graph. This result is due to the lack of blocking when these games are played on DAGs. The games become Nim-like and yield to standard methods.

A more complicated idea works for PAG and PVG but only for directed trees. This is the best one can hope for in the light of Theorem 2.4. We define $V_T(u, v)$ to be the winner (with best play) of PVG played on the directed tree T with starting points u and v, where u moves first and $u \neq v$.

PAG and PVG, unlike IAG and IVG, are different games even when G is a directed tree. For example, the game shown in Fig. 7 is a win for u in PVG and a win for v in PAG if u moves first. Hence, we define $E_T(u, v)$ to be the winner (with best play) of

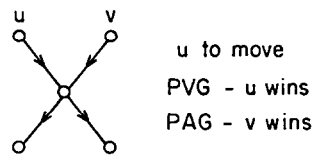


Fig. 7. PVG $\neq$ PAG on directed trees.

PAG played on a directed tree T with starting points u and v, where u moves first (here u and v can be equal). Note that, in general, $V_T(u, v) \neq V_T(v, u)$ and $E_T(u, v) \neq E_T(v, u)$.

The algorithms for computing V_T and E_T are very similar. So we will concentrate on V_T, explaining later on how to modify the ideas to work for E_T. We calculate V_T for all pairs $u, v \in V$, where $u \neq v$, by distinguishing 2 cases.

Case 1: *There is a directed path from v to u, or from u to v.*

This case can be decided by calculating the depths of two subtrees of T and comparing them. The point is that one player has already blocked the other one from the subtree rooted at the vertex of the first player. They are now effectively playing the game on two disjoint trees and the one with the longest path is the winner.

The actual calculation is done as follows. Assume that there is a path from u to v. Then we calculate the subtree of T rooted at v (call this T_v), and the subtree of T rooted at u with T_v deleted (call this $T_{u,v}$). If $\text{depth}(T_v) < \text{depth}(T_{u,v})$ then $V_T(u, v) = u$ else $V_T(u, v) = v$.

(If the path is from v to u, then if depth $(T_u) > \text{depth}(T_{v,u})$ then $V_T(u, v) = u$ else $V_T(u, v) = v$.)

Case 2: *There is no directed path between u and v.*

Here we use a recursive relationship (similar to the one which can be used to solve IVG):

$$V_T(u, v) = u \text{ iff } \exists (u, w) \in A \text{ such that } V_T(v, w) = w,$$

$$V_T(u, v) = v \text{ iff } \forall (u, w) \in A,\ V_T(v, w) = v.$$

The reason we separate two cases here is that the calculations for each case do not work for the other one. The reader should first note that the recursive relationship above does not work for case 1. The tree in Fig. 8 shows an example where $V_T(u, v) = u$ but the recursive relation yields v.

Furthermore, the calculation in case 1 does not work in general for case 2. When there is no directed path between the two vertices but the two vertices have a common descendant, it breaks down. In Fig. 9, $V_T(u, v) = u$, but the calculation of case 1 (regardless of which two subtrees you compare) yields v.

As before, we can calculate V_T for all pairs $(u, v) \in V \times V$ (where $u \neq v$) by dynamic programming. We will need a queue of vertices in reverse topological order called Q. We describe the algorithm below.

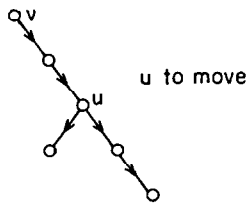


Fig. 8. A counterexample.

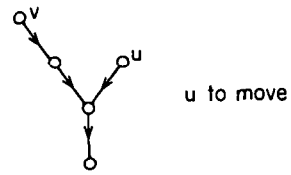


Fig. 9. Another counterexample.

Algorithm 1. Polynomial-time algorithm for PVG on directed trees

Initialize

for all vertices $u \in V$ with outdegree 0, $V_T(u, v) = v$ (for all vertices $v \neq u$).

for all vertices $u, v \in V$ where outdegree$(u) > 0$, and outdegree$(v) = 0$, $V_T(u, v) = u$.

Main Loop

while Q is not empty

begin

$x = \text{delete}(Q)$;

for all vertices y already removed from Q **do**

Compute $V_T(x, y)$; Compute $V_T(y, x)$;

end

Compute Routine

if there is a directed path from x to y, or from y to x,

then compute V_T according to case 1,

else compute V_T according to case 2.

Theorem 3.1. PVG *can be solved in* $O(n^2)$ *time on directed trees.*

Proof. We show that Algorithm 1 can be implemented in $O(n^2)$ time.

The initialization, including the reverse topological order in Q, takes $O(n)$ time. Deciding whether or not there is a path between 2 vertices can be done in $O(1)$ time by calculating all $O(n^2)$ answers in advance and looking them up when needed. The preprocessing time to do all these calculations takes $O(n^2)$ time.

Each pair of vertices in the main loop is considered exactly once by virtue of the reverse topological order processing. Hence, for each vertex u, Compute(u, x) and Compute(x, u) are each computed exactly once for each vertex $x \in V$. The work done by Compute on a given pair of vertices (u, x) is proportional to outdegree(u). Hence, the total work done by case 2, over all pairs of vertices, is at most $(n-1)\sum_{v \in V}$ (outdegree(v)), which is $O(n^2)$ for directed trees.

The total work done by case 1 is as follows. Each call to case 1 can be accomplished in $O(1)$ time if we do the appropriate preprocessing. To accomplish this, we need to calculate in advance the depths of all the subtrees that may be used by the calculations

in case 1. Once this is done, the work in case 1 reduces to one comparison and 2 lookups.

For every vertex u in V, we compute the depth of the subtree of T rooted at u. This can be done in $O(n)$ time. If a vertex has outdegree 0 then its depth is 0, else its depth equals the maximum of its childrens' depths plus 1. This recursive relationship can be computed with a reverse topological order scan of the tree in linear time.

Then, for every vertex v in V, we remove v and its descendents from T to create a new tree. For every vertex u in this new tree, we calculate the depth of the subtree rooted at u. As before, this calculation can be done in $O(n)$ time. Furthermore, since we need to do this calculation for each $v \in V$, there are n such calculations, giving a total preprocessing time of $O(n^2)$. □

All of the previous discussion is for PVG. For PAG, a few modifications are necessary. Case 2 works exactly as before but case 1 breaks down. The point is that the subtree of T rooted at v is not blocked completely from the player at u. In fact, u can move to v. He will be blocked by only one arc leaving v. This is the arc that the player at v chooses when he leaves v. Without loss of generality, we can assume that the player at v will choose the arc which gives the longest path.

Hence, when there is path from u to v, we do the following. First, we compute the depth of the subtree of T rooted at v. This is exactly as before. Then we need to find among the children of v, a child x, such that the depth of the subtree of T rooted at x is maximal. We remove x and all its descendents from T, and calculate the depth of the subtree of T rooted at u. Finally, we compare these two depths to determine which player wins. (If the path is from v to u, the computation is similar).

As before, a preprocessing step is necessary to implement these calculations in $O(1)$ time. This preprocessing can be done as before in $O(n^2)$ time. This gives us the following result.

Theorem 3.2. PAG *can be solved in* $O(n^2)$ *time on directed trees.*

The situation for undirected trees is even easier than for directed trees. This is true for both PVG and PAG. Here there is always a path between the two vertices u and v. The player to move must decide whether he wishes to move along the path between him and the other player, or to leave this path. If he leaves the path, then the two players are blocked from one another and a depth calculation similar to case 1 on directed trees determines the winner. Hence, the player to move can determine by a simple calculation whether he can win by moving off the path to the other player. If this calculation says that he cannot win by doing so, then he stays on the path to the other player and hopes his luck will change. The reader can check that the corresponding decision problems can be done in linear time. This result is summarized below.

Theorem 3.3. PVG *and* PAG *can be solved in* $O(n)$ *time on undirected trees.*

These techniques can also be used to exhibit polynomial-time algorithms for IAG2 and IVG2 on directed trees. The resulting algorithms, however, are not as fast as for PVG and PAG.

We outline the ideas for IVG2. Let $V2_T(u, v)$ be *first* when the player to move first wins IVG2 on T with starting vertices u and v, else *second*. (Note that here, in contrast to PAG and PVG, $V2_T(u, v) = V2_T(v, u)$.)

We distinguish 2 cases just as we did for PVG. When there is no directed path from u to v from v to u, then we use the following recursive relationship:

$$V2_T(u, v) = \textit{first} \textbf{ iff}$$

$$\exists (u, w) \in A \mid V2_T(w, v) = \textit{second}, \text{ or } \exists (v, z) \in A \mid V2_T(u, z) = \textit{second}.$$

When there is a directed path from u to v or v to u, the players, like in PVG, are essentially playing the game on two disjoint trees. For PVG, the decision of who wins in this situation depends on which tree was deeper. This depth calculation was done in $O(1)$ time, by doing an $O(n^2)$ preprocessing step to calculate the depth of every subtree. For IVG2, this situation of disjoint trees is not as easy to solve. For a given pair of trees, one needs to set up another recursive relationship. A straightforward method to do this will take $O(nm)$ time, where m and n are the number of vertices in the two trees. Furthermore, each pair of vertices (u, v), where there is a path between u and v, may create a different pair of disjoint trees. This implies an upper bound of $O(n^4)$ for the whole computation. The ideas carry over to IAG2 giving us the following result.

Theorem 3.4. IVG2 *and* IAG2 *can be solved in* $O(n^4)$ *time on directed trees.*

With a careful implementation and analysis, we conjecture that this can be cut down to $O(n^2)$.

4. Conclusions and future work

We have analyzed a number of variations of the game Geography. These included Impartial Geography, Partizan Geography, Impartial Geography with 2 tokens, and the vertex and arc versions of each. Impartial Geography, also known as Generalized Geography, was previously known to be PSPACE-complete for many restricted classes of digraphs. We showed that it was polynomial-time-solvable for DAGs (directed acyclic graphs).

We showed that all the other variations are PSPACE-complete or NP-hard even for DAGs. We described polynomial-time algorithms for these problems on directed trees.

It is easy to analyze these games for certain special digraphs. For example, one can look at complete DAGs and characterize who wins for any of the games above. But larger classes of digraphs are difficult to analyze.

There are a number of open questions that remain:

(1) Can one show that PAG and PVG are actually PSPACE-complete for DAGs instead of just NP-hard?

(2) Can one improve the $O(n^4)$ bound on the algorithm for IAG2 and IVG2 on directed trees?

(3) Can one describe heuristics for any of the difficult versions of these games and prove that they do reasonably well most of the time?

References

[1] H.L. Bodlaender, Complexity of path-forming games, *Theoret. Comput. Sci.* **110** (1993) 215–245.

[2] H.L. Bodlaender and T. Kloks, Fast algorithms for the tron game on trees, Tech. Report RUU-CS-90-11, Department of Computer Science, Utrecht University 1990.

[3] A. Fraenkel, On some combinatorial games, Abstracts of Int. Conf. on Algebraic Graph Theory (1989) Leibnitz, Austria, p. 11, Publ. of Institut für Mathematik und Angewandte Geometrie, Montanuniversität Leoben, A-8700 Leoben, Austria, June 1989.

[4] M. Garey and D. Johnson, *Computers and Intractability* (Freeman, San Francisco 1979).

[5] A. Lehman, A solution of the Shannon switching game, *J. Soc. Indust. Appl. Math.* **12** (1964) 687–725.

[6] D. Lichtenstein, Planar formulae and their uses, *SIAM J. Comput.* **11** (1982) 329–343.

[7] D. Lichtenstein and M. Sipser, GO is Pspace-hard, *J. ACM* **27** (1980) 393–401.

[8] F.L. Morris, Playing disjunctive sums is polynomial space complete, *Internat. J. Game Theory* **10** (3/4) (1981) 195–205.

[9] A. Pultr and F.L. Morris, Prohibiting repetitions makes playing games substantially harder, *Internat. J. Game Theory* **13** (1) (1984) 27–40.

[10] S. Reisch, Hex ist Pspace-vollständig, *Acta Inform.* **15** (1981) 167–191.

[11] J.M. Robson, Combinatorial games with exponential space complete decision problems, in: *Proc. 11th Symp. on Mathematical Foundations of Computer Science*, Lecture Notes in Computer Science, Vol. 176 (Springer, Berlin, 1984) 498–506.

[12] T. Schaefer, Complexity of some two-person perfect-information games, *J. Comput. System Sci.* **16** (1978) 185–225.

[13] C. Simonson, *SIGART Newsletter* **96** (1986) 3.

[14] S. Simonson and J. Villimek, Efficient algorithms for Partizan Geography on trees, Argonne Labs Symp. for Undergraduates in Science, Mathematics and Engineering, Argonne, IL (1990).