Bachelor thesis
Computer Science

Radboud University

Axis-Parallel Five-in-a-Row is PSPACE-Complete

Author
Laurens Kuiper
s4467299
University of Nijmegen
LKuiper@science.ru.nl

Supervisor/assessor:
Prof. dr. H. Zantema
Insitute for Computing and
Information Sciences,
University of Nijmegen
H.Zantema@tue.nl

Second supervisor/assessor:
Dr. J.C. Rot
Faculty of Science,
University of Nijmegen
J.Rot@cs.ru.nl

June 26, 2017

Abstract

Many games, even those for children, often turn out to have a high computational complexity. In this thesis, we will prove that the decision problem *"Does the white player have a winning strategy?"*, for the two-player board game *Axis-Parallel Five-in-a-Row*, is *PSPACE-complete*. We will use a reduction proof similar to the one found in a paper on the PSPACE-completeness of the game *Gobang* by *Stefan Reisch* (1980). Our proof has been verified by a backtracking algorithm, which can test certain claims by letting our heuristic play against a brute-force player.

Contents

1 Introduction

For many board games, the decision problem of whether a player has a winning strategy in a given situation has been proven to have a certain computational complexity. When we look at two-player board games, that decision problem is often *PSPACE-complete*. *PSPACE* is the complexity class of decision problems L for which a deterministic Turing machine M exists for which:

- M halts for every input;
- if M runs with input $x \in \{0,1\}^*$ on the initial tape, then it will end in an accepting state if and only if $x \in L$;
- the size of the part of the tape that is used by M while running with input x is polynomial in the length of the input, $|x|$.

To prove that a decision problem L_0 is PSPACE-complete, one must prove that $L_0 \in$ PSPACE, which is done by proving that an algorithm exists that can solve L_0 using an amount of memory polynomial in the length of L_0. Then, one must also show that L_0 is *PSPACE-hard*. Proving L_0's PSPACE-hardness can be done by taking an existing problem L_P, which is already proven to be PSPACE-hard, and proving that $L_P \leq_P L_0$. That is, proving that L_P can be reduced to L_0 in polynomial time. This means that solving L_P is not computationally harder than solving L_0, which shows that problem L_0 is at least as hard as problem L_P, which is PSPACE-hard. Because all PSPACE-hard problems can be reduced to any other problem in PSPACE, this shows that L_0 is at least as hard as every other PSPACE-hard problem. What also follows is that a solution to any PSPACE-hard problem is also a solution for any other problem in PSPACE.

"Gobang ist PSPACE-vollständig"[4] shows us that the decision problem *"Does the white player, currently taking his turn, have a winning strategy?"* for the game *Gobang*, when played on a $n \times n$ board is PSPACE-complete. In Gobang, traditionally played on a 19×19 board, both players try to create a row of five stones of their colour first, either diagonally or straight. The players do this by alternately taking turns placing stones of their own colour (either black or white) on the board. It is essential that the players prevent their opponent from creating a row of five, while attempting to create a row of their own.

In his paper, Reisch tells us that proving that the decision problem for Gobang is in PSPACE is easy. Then, he elaborately shows that a very specific variant of a game called *Generalized 'Geography'*, which he calls *'Bipartite Geography'*, can be reduced to Gobang. The fact that the decision problem for 'Bipartite Geography' is PSPACE-hard, and the transitivity of the reduction relation imply that Gobang is also PSPACE-hard, concluding the proof on the PSPACE-completeness of Gobang.

In this thesis we will take a look at how Reisch made his proof, and how we can apply his strategy to prove that a similar game, which we will call Axis-Parallel Five-in-a-Row (abbreviated as APR-5), is also PSPACE-complete. APR-5 is also a two-player game, played on an $n \times n$ board, in which both player try to create an axis-parallel row of five stones of their own colour (no diagonals allowed).

2 The ‘Geography’-game

The Generalized ‘Geography’-game is played on a directed graph. The player going first places his stone on a specific starting vertex s. Then, the players alternate taking turns, by placing their stone on a non-occupied vertex adjacent to the vertex where their opponent last played their stone. One player will eventually not be able to place his stone because all reachable vertices are occupied, causing him to lose. Note that an unoccupied adjacent vertex might not be reachable because the edge that connects it is directed outgoing from that vertex. This game is PSPACE-complete according to Reisch’ sources [5].

2.1 Reisch’ ‘Bipartite Geography’

Directed graphs can take on a lot of different shapes, and many of them cannot be directly translated to a two-dimensional playing field. Reisch requires the graph to have the following properties to make this translation possible, while maintaining PSPACE-completeness:

(2.1.I) The graph $G = (V,E)$ must be planar and bipartite, which means that it can be drawn in a 2-D space in such a way that no edges cross each other, and that the vertices of the graph can be divided into two disjoint sets V_1 and V_2 such that every edge $e \in E$ connects either a vertex $v \in V_1$ to a vertex $w \in V_2$, or a vertex $w \in V_2$ to a vertex $v \in V_1$.

(2.1.II) The starting vertex s has an indegree equal to 0, and an outdegree equal to 1. To all vertices $v \in V \setminus \{s\}$, the following applies: $\text{degree}(v) \leq 3$, $\text{indegree}(v) \neq 0$, $\text{outdegree}(v) \neq 0$.

(2.1.III) $s \in V_1$, so that the next rule follows: the player going first only places his stones on vertices in V_1, and the second player only on vertices in V_2.

If the graph complies with requirements (2.1.I), (2.1.II) and (2.1.III) it is called ‘Bipartite Geography’. Reisch does not prove that his requirements maintain PSPACE-completeness, but notes that it is proven implicitly in the paper *“Go is PSPACE-hard”*[2], by using results from *“Word problems requiring exponential time”*[3].

2.2 Transforming ‘Bipartite Geography’ to grid-format

Next, Reisch shows us what must be done to transform a given planar and bipartite ‘Geography’-graph to an equivalent graph that can be placed onto a two-dimensional grid. Graphs often cannot be placed onto a two-dimensional grid because they are not planar, which means they cannot be drawn in 2D without their edges overlapping. Another reason is that the vertices in these graphs can have high degrees, which requires a vertex to have many in- and outgoing directions, but there are really only four possible directions in a 2D-grid.

To solve this problem, a new graph $G' = (V', E')$ is to be constructed for the given ‘Geography’-graph $G = (V, E)$, which must satisfy the following conditions:

(2.2.I) G' has properties (2.1.I), (2.1.II) and (2.1.III).

(2.2.II) G' can be embedded in the two-dimensional space $\mathbb{R}^2$ by following these steps:

- (a) Each edge stands either horizontally or vertically.
  $[((x_1, y_1), (x_2, y_2)) \in E' \Rightarrow (x_1 = x_2) \vee (y_1 = y_2)]$.
- (b) Each edge is either of length 1 or length 2.
  $[((x_1, y_1), (x_2, y_2)) \in E' \Rightarrow |x_1 - x_2| + |y_1 - y_2| \in \{1, 2\}]$.
- (c) Edges going out from the same vertex make an angle of 180° with each other.
  $[((x_1, y_1), (x_2, y_2)), ((x_1, y_1), (x_3, y_3)) \subset E' \Rightarrow (x_1 = x_2 = x_3) \vee (y_1 = y_2 = y_3)]$.

(2.2.III) The starting player in a ’Geography’-game played on G' has a winning strategy if and only if he also has a winning strategy in the game on G.

The new graph G' can be obtained by replacing the edges of length 1 in the graph G with composite edges consisting of an odd number of edges where necessary. This means some edges will be very long, which allows us to space the graph out on the two-dimensional board nicely.

The construction of this modified graph is not a goal in itself, but rather a means to have a PSPACE-complete problem that lends itself well for translation to a two-dimensional board game.

3 Gobang is PSPACE-complete

As the definition in chapter one states: proving the PSPACE-completeness of a decision problem requires showing that the problem is in PSPACE, and showing that it is PSPACE-hard. In his paper, Reisch proves the PSPACE-completeness of Gobang, and starts by formally defining his decision problem as follows.

Instances of an $n \times n$ Gobang-situation are encoded in alphabet A. The instances for which the decision problem is true are given by A^*, a subset of A:

$$a \in A^* = \{\text{The encoding of a game situation in an } n \times n \text{ Gobang-game, where the player `white' that is currently taking his turn has a winning strategy.}\}$$

After this definition Reisch quickly states that Gobang must be in PSPACE, because it is a two-player game where both players have at most n^2 options for their moves. He does not prove this explicitly, but he does mention that the proof is simple, and would follow the exact same argumentation as the proof for some combinatorial games found in *"A combinatorial problem which is complete in polynomial space"*[1] by S. Even and R.E. Tarjan (1975) or *"On the complexity of some two-person-perfect-information games"*[5] by T.J. Schaefer (1978). We will show that Reisch was right about this in section 5.2.

Now, the only thing left to prove is the PSPACE-hardness of the decision problem for Gobang, which is substantially harder than the first part of PSPACE-completeness proof. Reisch has shown us how to create a grid-suitable version of any 'Bipartite Geography'-game by complying with requirements (a - c) from property (2.2.II), where the starting player has a winning strategy if and only if he also has one in the non grid-suitable version of that game (2.2.IV). Reisch reduces this grid-suitable version of 'Bipartite Geography' to Gobang by making small patterns of Gobang-situations which contain chains of forced moves. These patterns correspond with situations that occur within a 'Bipartite Geography'-game. All possible situations must have a corresponding pattern, and there must be a way to chain these patterns together, so that any instance of a 'Bipartite Geography'-game can effectively be translated to an equivalent Gobang-situation. We will take a look at how Reisch did this before applying his strategy to our own game.

3.1 Re-creating aspects of the 'Geography'-game in Gobang

Before explaining how each part of the grid-suitable 'Bipartite Geography'-graph is made on a $n \times n$ playing field, Reisch explains how some important details of the game play out on the Gobang-board by means of the example in Figure 3.1.

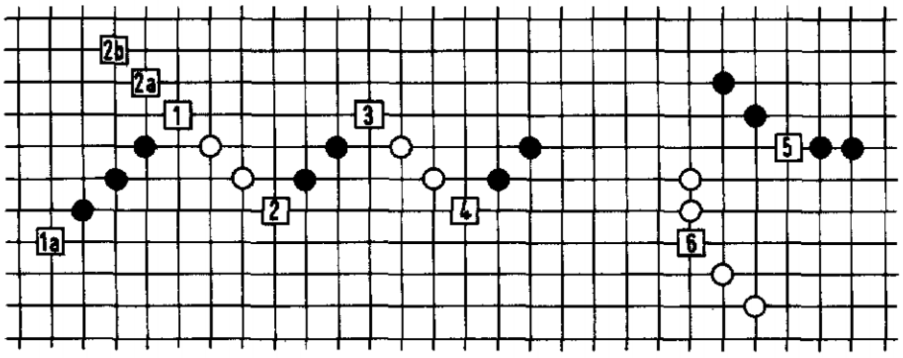


Figure 3.1: Reisch' Gobang-situation

In the situation depicted in the figure it is the white player's turn to make a move. The angular constructs to the right of the figure pose a constant threat. Both players can win the game with just two more moves after they are allowed to freely place their stone on $\boxed{5}$ or $\boxed{6}$. Therefore, every move that a player makes must block the opposing player's current threat, and create a significant new threat.

If white does not place his stone on $\boxed{1}$ or $\boxed{1a}$, black will win the game in two moves, by inevitably creating a diagonal row of 5 stones. However, if white chooses to place his stone on $\boxed{1a}$, black is not pressured by white, and black will win the game in three moves, by placing his stone on $\boxed{5}$. Therefore, white is forced to place his stone on $\boxed{1}$. The starting position s that was discussed in chapter 2 will be enforced in the reduced game similarly. So, if white chooses to place his stone on $\boxed{1}$, he will stop black's diagonal row, and threaten to win in two turns if he is not stopped immediately. Black must then place his stone on $\boxed{2}$ because he does not create a threat of his own by placing his stone on $\boxed{2a}$. Following the same logic, we can see that white must place his stone on $\boxed{3}$ next, to which black must respond by placing his stone on $\boxed{4}$.

Now, white has to block black's diagonal row through $\boxed{4}$, but cannot create his own threat in doing so. This allows black to place his stone on $\boxed{5}$ the next turn, which will then guarantee victory for black in the next two turns. What happens here resembles what happens in the 'Geography' game when a player runs out of unoccupied adjacent vertices to place his stone on: the other player wins.

3.2 Translating instances of the 'Geography'-game to Gobang

Because of the properties that Reisch has set to the graph of his grid-suitable 'Bipartite Geography'-game, only a handful of different edges and vertices occur within the game. All of these situations must be translated to an equivalent pattern of forced moves in a specific Gobang-situation. We won't go over all of these patterns in detail, because we want to create patterns for our own game later on in chapter 5. However, it is important for our own proof that we do know what patterns we need to cover all possible vertices and edges. We will only discuss three of Reisch' patterns in detail, to gain an understanding of how they work.

We already saw that Reisch made a chain of forced moves in Figure 3.1, moves $\boxed{\text{1a}}$ to $\boxed{4}$. He uses very similar constructions to re-create the edges of the 'Bipartite Geography'-graph. As stated in (2.2.II)(b), edges of both length 1 and length 2 are needed. In Reisch' proof, an edge of length 1 uses 15 spaces on the Gobang-board, as shown in Figure 3.2. The edge of length 1 connects two vertices with each other. The white player places his stone on the leftmost double-square (depicting a vertex), after which both players follow their forced moves on the single-squares, until the black player must place his stone on the rightmost double-square. We can invert the colours of the stones in this edge to create a new edge where the black player places his stone first. We can also rotate it by a multiple of 90° or mirror it horizontally/vertically to change the direction and/or orientation of the edge. This allows Reisch to cover multiple cases with a single pattern, and can be done on both his edge and vertex patterns.

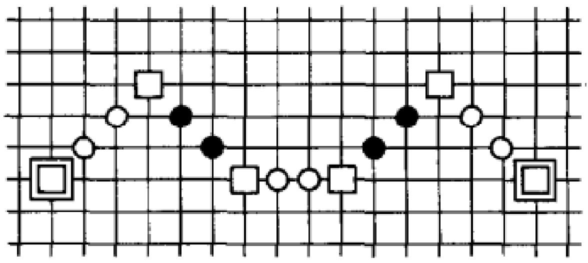


Figure 3.2: Reisch' edge of length one

In these figures the double-squares depict vertices, and the single-squares depict the intended forced moves. If the white player places his stone on the left double-square, there will be a chain of forced moves, just as in Figure 3.1, until the black player places his stone on the right double-square.

The next pattern, an edge of length 2, must naturally use $2 \times 15 = 30$ spaces. However, we cannot simply string two edges of length 1 together to make it. This is due to the fact that edges must connect two vertices that are to be occupied by differently coloured stones, because the graph is bipartite (2.1.I). Thus, concatenating two edges into one would result in the adjacent vertices being the same colour. Reisch' solves this by introducing a 'kink' in the edge, as shown in Figure 3.3. We can see that the perpendicular angles of the kink allow him to tweak which colour forced move ends up where.

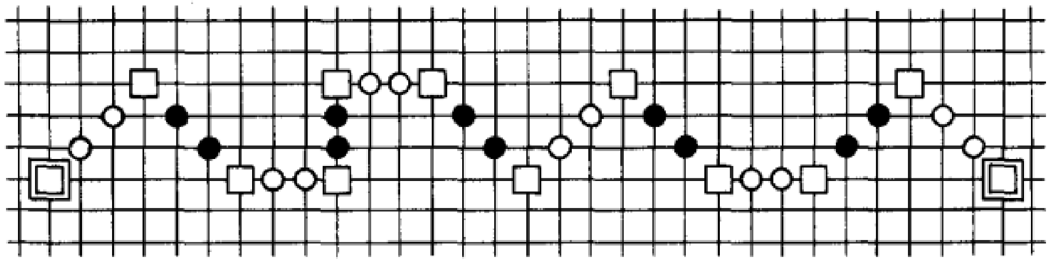


Figure 3.3: Reisch' edge of length two

Again, if the white player places his stone on the left double-square, black will eventually have to place his stone on the right double-square.

The patterns become more complex when we consider vertices. Vertices can have a degree of up to 3, due to property (2.1.II), and each vertex that is not starting vertex s has indegree ≥ 1. Edges must make an angle of 90° with eachother (2.2.II)(a). A vertex with outdegree = 2 must have it's outgoing edges make an angle of 180° due to requirement (2.2.II)(c). These requirements leave only one pattern for a vertex with outdegree = 2. Having two outgoing edges is interesting because it leaves one of the players to pick the direction of the game on the board. Thus, the intersection vertex must have a situation where that player can choose between one of two forced moves. Thankfully, rows have two sides, so Reisch was able to solve this quite easily. As we see in Figure 3.4, after the black player places his stone on $\boxed{1}$, and white responds with $\boxed{2}$, the black player can now make two moves that both stop white from creating a row of four with two open ends, while creating his own row of three with two open ends. Black does this by placing his stone on either $\boxed{3}$ or $\boxed{4}$, which makes the game continue towards either the left or the right, respectively. We can simply rotate/invert/mirror the vertex to create all other possible configurations, just like the edges.

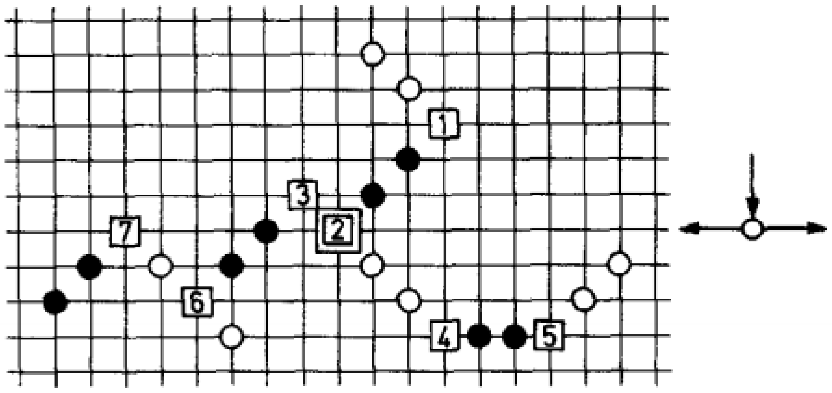


Figure 3.4: Reisch' vertex with outdegree 2

Thusfar we have seen the two edges of different length, and the one vertex with outdegree = 2. We now know how Reisch handled certain concrete situations when translating his 'Bipartite Geography'-game to Gobang. Due to the requirements that Reisch set, as discussed in chapter 2, there is only a small amount of patterns that have to be made.

Let us consider the vertices with outdegree = 1. The indegree of these vertices will be either equal to 1 or equal to 2. The vertices with indegree = 1 must have their two in- and outgoing edges make either a straight- or perpendicular angle with each other (recall that edges must stand either vertically or horizontally due to property (2.2.II)(b)).

There are also only two vertices with indegree = 2. With these, the *ingoing* edges can only make either a straight angle or a perpendicular angle with eachother, giving us two more patterns.

Reisch made two patterns for edges (length 1 and length 2), one pattern for vertices with outdegree = 2 and $2 + 2 = 4$ patterns for vertices with outdegree = 1, for a total of only seven patterns.

3.3 Concluding Reisch' proof

After showing his patterns, Reisch begins to complete his proof. First off, he shows that all of his seven patterns for vertices and edges can be effectively concatenated into a graph. Then, he proves that the white player, currently taking his turn, has a winning strategy in Gobang, if and only if the starting player also has a winning strategy in the associated 'Bipartite Geography-graph. He does this step-by-step, showing that every aspect of the 'Geography'-game plays out the same way in the translated Gobang-version. This shows that any given 'Bipartite Geography'-graph can effectively be translated into a Gobang-situation, for which the outcome of the decision problem is the same.

He then argues that even though a player can choose to deviate from the proposed forced moves, that player will not be able to gain an advantage, and his options to further deviate will be exhausted within a small number of moves. The player is then forced to go back to following the intended forced moves, if, by then, he has not lost by deviating. He says the proof that these moves are exhausted quickly is relatively easy, but rather tedious, and chooses not to elaborate further. This will be proved for our own game by a backtracking algorithm.

This concludes Reisch' proof that Gobang is PSPACE-hard, which implies that it is also PSPACE-complete, because he already said that the decision problem is in PSPACE. And, even though Reisch has skipped some details in his proof, he has very convincing evidence that the decision problem for Gobang is PSPACE-complete.

After his proof is complete, he shows that k-Gobang, which is the decision problem for Gobang but with rows of length k (for $k \geq 2$) instead of just $k = 5$, is also PSPACE-complete. He says this can be proved by replacing the rows with open ends of length 2 in his patterns by rows of length $(k - 3)$.

4 Thoughts on Reisch' proof

The quick proof at the end of Reisch' paper leaves the reader with some questions. Why does his proof also work for lower values? Does it mean that Reisch' strategy is applicable for similar games?

4.1 *k*-Gobang is only PSPACE-complete for *k* greater or equal to 5

How exactly does the proof for k-Gobang play out, when k takes on it's lowest possible value, $k = 2$? Reisch says the rows in his patterns should be replaced by rows of length $(k - 3)$, which is equal to -1 for $k = 2$. However, it is not possible for a row to have a length lower than 1. This is most likely an oversight by Reisch.

This does cause the next question to arise: *"What is the lowest k for which Reisch' proof holds?"*. Because a row cannot have a length lower than 1, and the rows in his patterns must have length $k - 3$, we are inclined to believe that the lowest k for which the proof holds should be $1 + 3 = 4$. But, this causes problems in some of the patterns that Reisch made. For example, let's look at his vertex with indegree = outdegree = 1, where the edges make a perpendicular angle, in Figure 4.1.

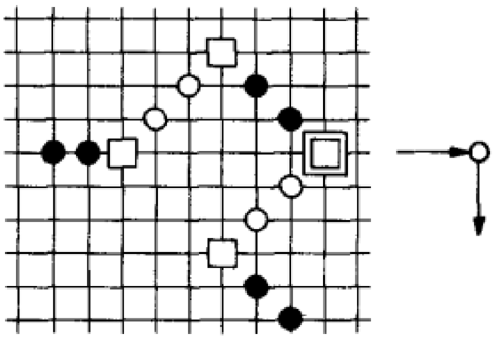


Figure 4.1: Reisch' vertex with degree = 2, perpendicular angle

If we replace the rows of length two with rows of length one, the cornerpiece contracts and we get a pattern where deviating from the intended strategy actually does reward greatly, and can be shown easily.

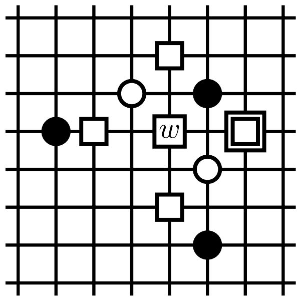


Figure 4.2: Same as Figure 4.1, but with rows of length 1

On the pattern shown in Figure 4.2, the players must make a row of length 4. The white player can choose to place his stone on $\boxed{w}$ at any point in the game where he is not under immediate pressure of losing the game. This move creates a row of length 3 with two open ends, which will surely deliver him victory on the next turn. This is not an intended move, and breaks Reisch' proof for $k = 4$. 4-Gobang might still be PSPACE-complete, but that cannot be proved in this manner. We already noticed that Reisch' proof does not hold for $k < 4$, so we can conclude that his proof only holds for $k \geq 5$.

5 Axis-Parallel Five-in-a-Row

Despite Reisch' oversight at the end of his paper, his proof strategy is very interesting. He modified the Generalized 'Geography'-game in a way that made it possible to reduce to a Gobang-situation. Then, he created seven Gobang-patterns that, when stringed together, can simulate every possible situation in the original game.

We are interested if this strategy is applicable to other, similar games. Luckily, the rules of Gobang are easily adjusted. Reisch' proof already holds for a modified version of Gobang where the row length is k for $k \geq 5$. If we disallow diagonal row victories, we have already created a new game. We will call that game *'Axis-Parallel k-in-a-Row'*. Can we use Reisch' strategy to prove that this game is also PSPACE-complete? If so, then what is the lowest value of k for which it holds? If we think about this new game, it is not interesting for $k \in \{1, 2, 3\}$, because the player going first will win on his first, second or third turn respectively. For $k = 4$ it already becomes much more interesting, because the game can go on for a long time. For the most part, the time on this thesis was spent trying to prove that Axis-Parallel Four-in-a-Row is PSPACE-complete. However, it is very difficult to construct patterns for this game, because rows of four are so easily made, that many deviations from the intended strategy gave an advantage. These advantages were found using a backtracking algorithm, which is explained in the next chapter.

For $k = 5$ the game can also go on for a very long time, and gives us a lot of flexibility in making patterns. We suspect that Reisch' strategy will work for Axis-Parallel Five-in-a-Row (abbreviated as APR-5), and that we can prove that it has the same complexity as Gobang. Is 5 the lowest value of k for which our game is PSPACE-complete?

5.1 Definition

We will try to follow Reisch' strategy as much as possible to prove APR-5's PSPACE-completeness, but this game is different and brings out some new challenges specific to this game. We must first define our decision problem.
Instances of an $n \times n$ APR-5-situation encoded are encoded in alphabet A. The instances for which the decision problem is true are given by A^*, a subset of A:

$$a \in A^* = \{\text{The encoding of a game situation in an } n \times n \text{ APR-5-game, where the player 'white' that is currently taking his turn has a winning strategy.}\}$$

We will prove the following theorem about this decision problem:

Theorem 5.1. APR-5 is PSPACE-complete.

This requires us to also prove two lemma's. First, that APR-5 $\in$ PSPACE and second, that APR-5 is PSPACE-hard.

5.2 APR-5 in PSPACE

To prove the first lemma, we must show that there is a polynomial-space bounded algorithm for determining the outcome of our decision problem. We shall do this by creating a game tree representation of our game, in which our algorithm can determine who will win. Then, we shall prove that this algorithm uses polynomial-space bounded memory during run-time.

Reisch did not prove this lemma for his decision problem for Gobang, but he did cite a few sources in which proofs for other games could be found, and said that the proof for Gobang would be similar. Our proof is based on one of these sources: *"A Combinatorial Problem Which Is Complete in Polynomial Space"*[1]. As we will see, the proof for APR-5 also holds for Gobang, because both games are played in the same fashion, and on the same $n \times n$ board. This means that Reisch' assumption about Gobang being in PSPACE was right.

Lemma 5.2. APR-5 $\in$ PSPACE.

Proof. On an $n \times n$ board only n^2 moves can be made in any game before the board is full. Suppose we construct a directed, rooted game tree T for our game, where each vertex denotes a certain situation in that game. We will use the term *situation* to refer to a certain configuration of an APR-5-game. The root of T then denotes the initial board state. The sons of any vertex $v \in T$ denote the situations that are reachable from situation v, by letting the player whose turn it is in v make one move. We will use $v \rightarrow w$ to express that w is a son of $v \in T$. A leaf of T corresponds to a final game situation. We shall call a vertex of T a white vertex if it's white's turn to move in the situation denoted by v, and a black vertex if it's black's turn to move. Now that we have defined a game tree representation of the problem, we can describe the algorithm.

T has a depth of at most n^2 (the maximum number of moves that can be made) and contains at most $(n^2)!/(n^2-i)!$ vertices at depth i, for a total of at most $\sum_{i=0}^{n^2}(n^2)!/(n^2-i)!$ vertices in the entire graph T. A lot of these vertices correspond to the same situation, but with different sequences of moves. For any vertex $v \in T$, let $W(v) = 1$ if the white player has a forced win from situation v, let $W(v) = 0$ otherwise. $W(v)$ is easy to calculate if v is a leaf of T, because the winner is the player who made the last move in that situation (he won by either finishing a row of five or filling the last spot on the board). The colour of the player who made the last move is the colour of w, with $w \rightarrow v$, and v a leaf of T. We can determine $W(v)$ for all non-leaf vertices with the following recursive formula:

If v is a white vertex, $W(v) = 1$ if there exists $v \rightarrow w$ such that $W(w) = 1$;
$W(v) = 0$ otherwise.

If v is a black vertex, $W(v) = 1$ if for all $v \rightarrow w$ such that $W(w) = 1$;
$W(v) = 0$ otherwise.

We want to know which player will win before the game starts. Therefore, we must calculate the value of $W(r)$, where r is the root of T. We can easily calculate $W(r)$ by using our recursive formula to explore T in a depth-first fashion. Alongside storing the graph, we need a stack to store the moves made in reaching the current position. The amount of storage required for this stack is

$O(n^2 \log n^2)$ bits (each move can be encoded in $\log n^2$ bits, with a maximum of n^2 moves). We will need no more than $O(n^4)$ storage for storing game situations plus any additional work area (one board state of $O(n^2)$ for each depth level of the tree, up to depth n^2). Thus, the total amount of storage required for our algorithm to determine the winner is polynomial-bounded, which completes the proof for Lemma 5.2. □

The conclusion of this proof, and the fact that there is no obvious way to determine the winner of this game in polynomial time makes it even more reasonable to suspect that this decision problem is PSPACE-complete.

5.3 APR-5 is PSPACE-hard

In this section, we shall prove the second lemma that we need to prove PSPACE-completeness.

Lemma 5.3. APR-5 is PSPACE-hard.

We already have the PSPACE-complete, grid-suitable 'Bipartite Geography'-game. We need to reduce the decision problem for that game to the decision problem for APR-5. We must create patterns of situations in the APR-5-game that correspond with all the edges and vertices that can occur within 'Bipartite Geography'. Those patterns need to be able to be concatenated to re-create a graph, so that every 'Bipartite Geography'-situation can be simulated in our APR-5-game.

We will start by showing how important aspects of the 'Geography'-game play out in our APR-5 in Figure 5.1, analogous to Figure 3.1. In this situation

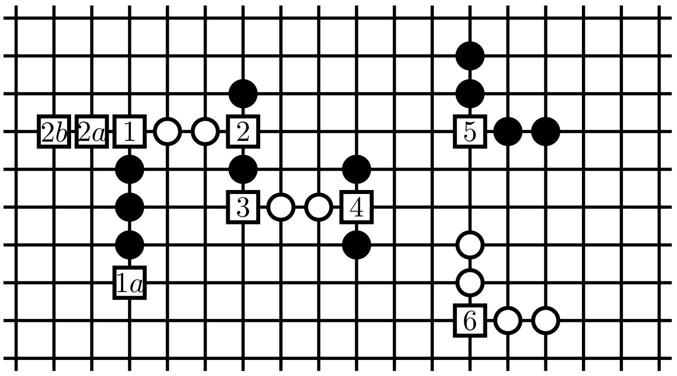


Figure 5.1: An example of a APR-5 situation

it is white's turn to make a move. Black already has a threat: a row of three with two open ends. If black manages to create a row with of four with two open ends, he will surely win on his next turn. Thus, white is left with two choices; if he chooses to place his stone on $\boxed{1a}$, then black will respond with placing his stone on $\boxed{5}$, in the angular construct to the right. Black now has a guaranteed win within the next two turns, because he has two ways to create a row of four with 2 open ends, which cannot be stopped. So it seems white must place his stone on $\boxed{1}$; indeed, black is now threatened, and he does not have the freedom to place his stone in the construct to the right. He must stop white's threat by placing his stone on either side of white's row of three. If he places his stone on $\boxed{\text{2a}}$, he does not threaten white, allowing white to place his stone on $\boxed{6}$, which will guarantee white victory. Because of the constructs, both players must continually pressure each other. Thus, black must place his stone on $\boxed{2}$, to which white must respond with $\boxed{3}$, and black must respond with $\boxed{4}$. Now, white cannot stop black's threat with his own threat, and can only delay the inevitable. He can place hist stone on $\boxed{\text{2a}}$ to create a row of four with one open end, to which black must immediately respond with placing his stone on $\boxed{\text{2b}}$. Now, white has run out of threats. He cannot place his stone on $\boxed{6}$, because he is still threatened by black's open-ended row of three through $\boxed{4}$. He can stop black's threat, but black is simply able to respond with $\boxed{5}$, granting him the win in the next two turns.

Because we have already looked at Figure 3.1, it should be clear that $\boxed{1}$ is a way of forcing a starting vertex s, and steps $\boxed{\text{1 - 4}}$ are a chain of forced moves that form the basis for our edges. The angular constructs to the right with $\boxed{5}$ and $\boxed{6}$ ensure that the player that runs out of threats will lose, just as the player that runs out of moves in the 'Bipartite Geography'-game also loses.

5.3.1 Patterns

If we take a close look at the chain of forced moves in situation in Figure 5.1, we see a recurring phenomenon. The black player starts with a vertical row, which white must block by extending his own horizontal row. Black must then block white's threat with a vertical row, which white must then block with a new horizontal row, and so forth. With this basic pattern, the white- and black player cannot change the orientation of their rows. This leaves us with two choices: we can either leave the orientations as they are, or find a way to swap them around.

Leaving the orientations as they are requires us to create two patterns for each pattern in Gobang, because inverting the colours would also change the orientations of the coloured rows in our game. Gobang does not have this problem, because it allows for diagonal rows. He could simply invert the colours of his patterns to create the same situation for both white and black vertices.

Finding a way to swap the orientation of the coloured rows allows us to re-use vertex patterns by inverting their colours, but will certainly increase the length of our edges. The latter option seems like a better choice, because we will have to make less patterns.

Each of our patterns has been tested with a backtracking algorithm, which verifies that the chains of moves in the patterns are indeed forced, by showing that deviating from the chain will not give an advantage, and might even lead to a loss. We will elaborate on the verification of patterns in section 5.3.2, and show the algorithm in chapter 6.

Edge piece patterns

We cannot simply show what a whole edge looks like, because edges will look different depending on which vertices they connect, and depending on the orientation of the white and black coloured rows (horizontal/vertical). Rather, we will describe edges pieces that can be concatenated into a whole edge.

There are two simple edge pieces of different lengths that can be concatenated. Their lengths, 5 and 6, allow us to create whole edges of a certain length that space the vertices on the grid out perfectly. The short edge pieces of length 5 have to be concatenated by alternating the two patterns shown in Figure 5.2. Not alternating leads to unwanted behaviour caused by stone $\boxed{x}$.

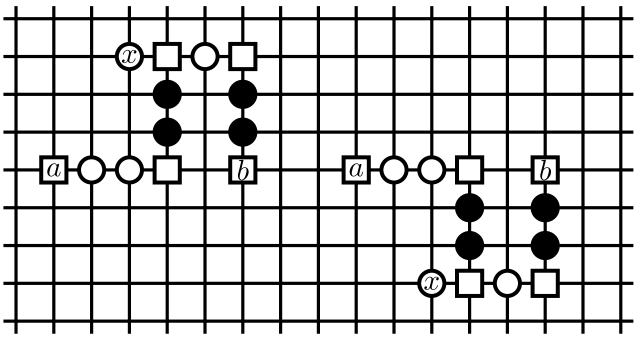


Figure 5.2: Short edge piece, length 5

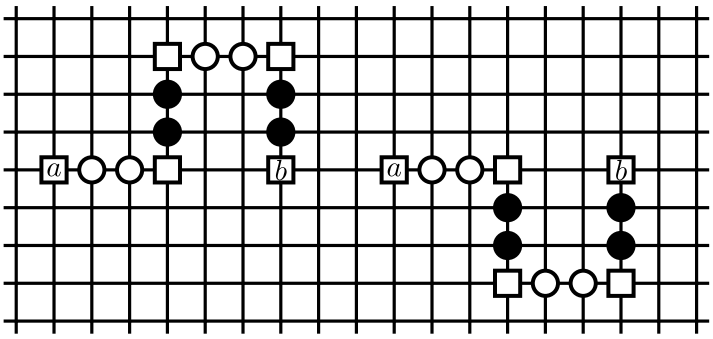


Figure 5.3: Regular edge piece, length 6

The regular edge pieces of length 6, shown in Figure 5.3, can also be concatenated by alternating, but do not have to.
These two patterns have the following property: if there is a white stone on $\boxed{a}$, there will be a chain of forced moves until white places his stone on $\boxed{b}$.

The last edge piece is a bit more complex, as it is the one that allows us to swap the orientation of the coloured rows. It has a length of 10, shown in Figure 5.4. This edge piece can be horizontally mirrored as well.

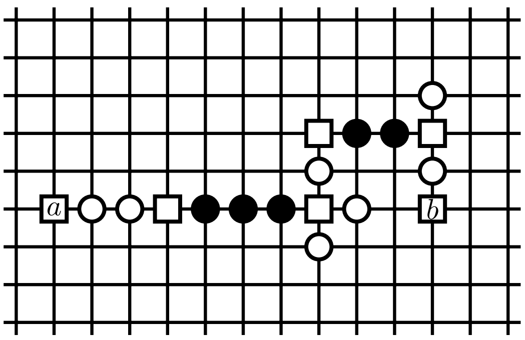


Figure 5.4: Orientation swap edge piece, length 10

If there is a white stone on $\boxed{a}$ in Figure 5.4, there will be a chain of forced moves until black places his stone on $\boxed{b}$.

As an example, the short, orientation swap and regular edge pieces can be concatenated like shown in Figure 5.5. Only one version of the short edge piece is seen here, but the other version can been seen in action in Figure 5.12.

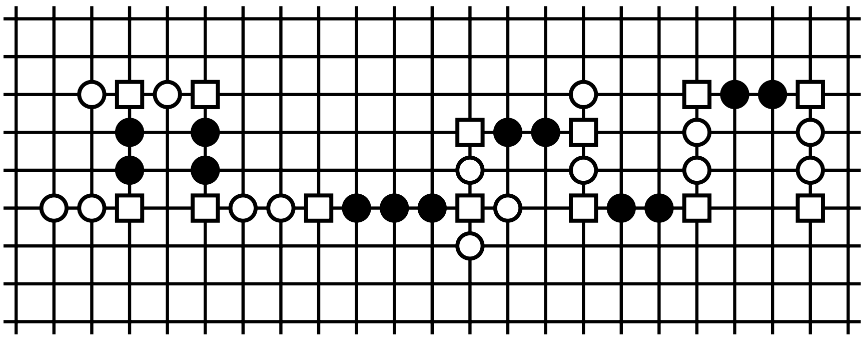


Figure 5.5: Concatenation of edge pieces

We will discuss the set length of whole edges consiting of these edge pieces in the '*Determining edge length*' section.

Vertex patterns

In chapter 3 we learned that only five vertex patterns have to be made to translate a graph of the 'Bipartite Geography'-game to a grid, due to the requirements set in chapter 2.

We shall consider the vertices with indegree = outdegree = 1 first. There are two variants: the in- and outgoing edges can make either a straight or perpendicular angle with each other (Figure 5.6 and 5.7).

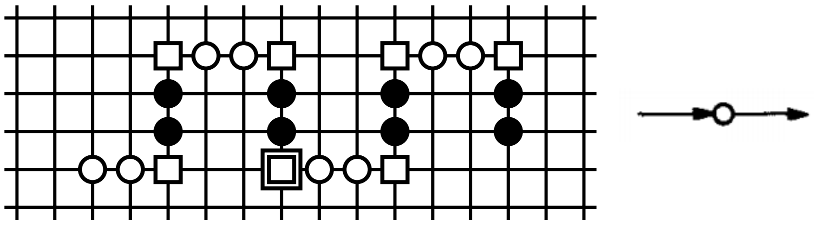


Figure 5.6: Vertex with indegree = outdegree = 1, straight

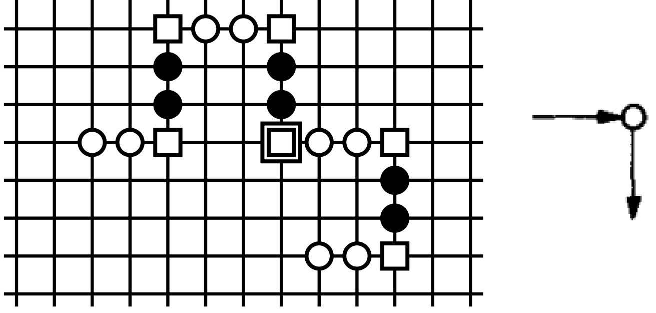


Figure 5.7: Vertex with indegree = outdegree = 1, perpendicular

Just as in Reisch' patterns, the single-squares show where the forced moves must be placed, with the double-squares being the precise position of the vertex. In these two figures the white player must place his stone on the double-square. As we can see, these two patterns are made by fitting regular edge piece patterns together. Therefore, the chain of forced moves is much the same as in the edge piece patterns. The other vertex patterns require a different construction, due to them having a higher degree: 3.

There are two patterns with indegree = 2 and outdegree = 1, because the ingoing edges can now make either a straight or perpedicular angle with each other. They both use the same construction to make two ingoing edges converge into one outgoing edge (Figure 5.8). The ingoing edges can be extended in the desired direction with edge piece patterns, to create both required patterns, shown in Figure 5.9 and 5.10.

Only the construction in Figure 5.8 has been tested by the backtracking algorithm (explained in the next chapter), because extending the edges lead to an extremely long run-time. However, we know that the extended patterns must also be correct, because the extensions are of the same form as the regular and short edge pieces, whose interactions have all been tested.

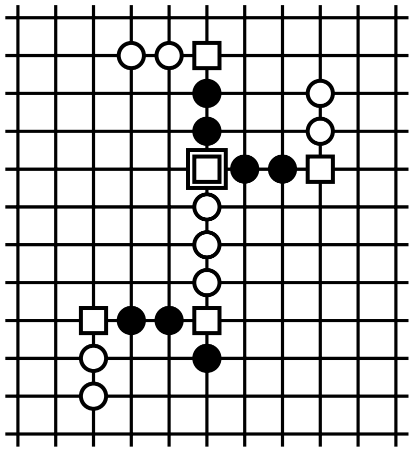
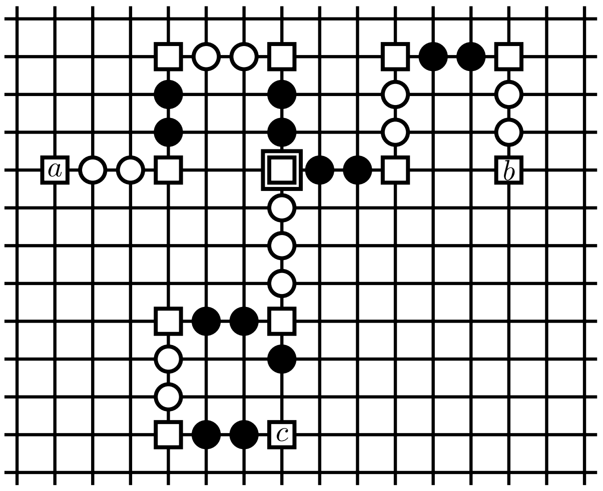


Figure 5.8: Converging two incoming edges into one outgoing edge



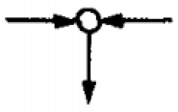
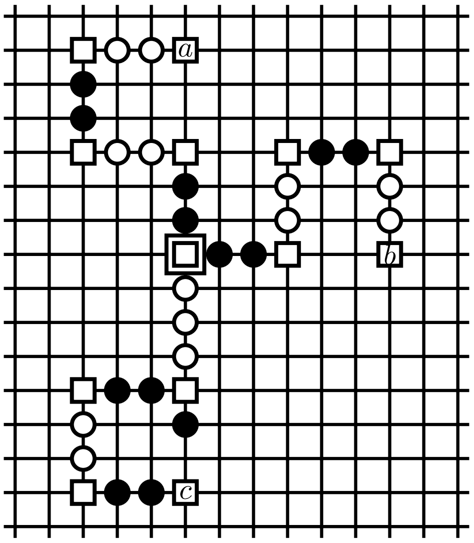


Figure 5.9: Vertex with indegree = 2, outdegree = 1, straight



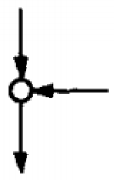


Figure 5.10: Vertex with indegree = 2, outdegree = 1, perpendicular

In Figure 5.9 and Figure 5.10, if there is either a white stone on $\boxed{a}$ or $\boxed{b}$, there will be a chain of forced moves, forcing white to eventually place his stone on $\boxed{c}$. Both are white vertices. Note that the white, vertical row of length 3 with two open ends can be made into a row of length 4 with one open end, to which black then must respond immediately. This does not lead to an advantage for white, as tested by our algorithm.

Finally, we must have a vertex with indegree = 1, outdegree = 2. It is shown in Figure 5.11. Recall that there is only one vertex with outdegree = 2 due to requirement (2.2.II)(d). We already saw the Gobang version of this pattern in Figure 3.4.
If a black stone is placed on $\boxed{a}$, then a black stone must be placed on $\boxed{b}$ or $\boxed{c}$, after the chain of forced moves. White shall place his stone on the vertex.

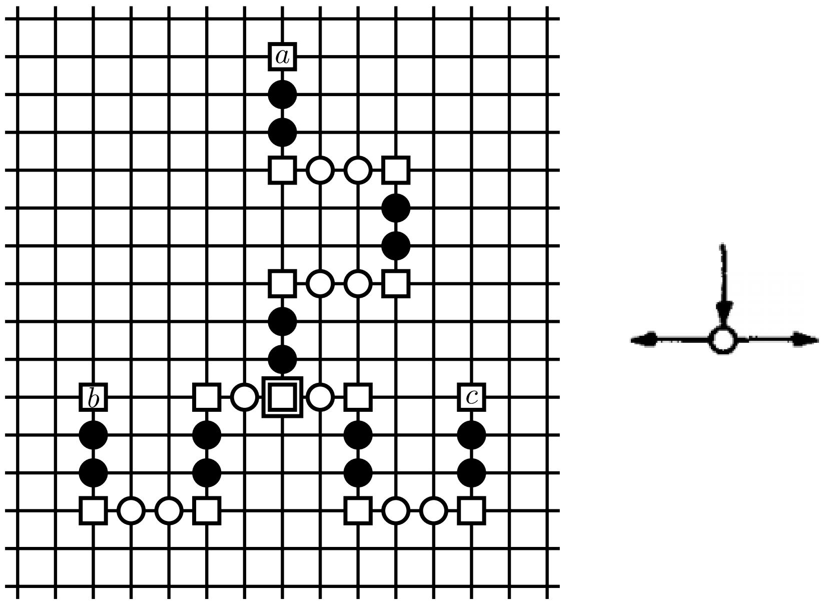


Figure 5.11: Vertex with indegree = 1, outdegree = 2

Determining edge length

The vertex patterns we described allow us to attach edge piece patterns so that, eventually, vertices can be concatenated together by whole edges. The length of these whole edges depends on how big the vertex patterns are, and how we can extend them with edge pieces them until they have a uniform length. There must also be enough space to swap the orientation of the coloured rows, which can be done with the orientation swap edge piece, length 10.

The in- and outgoing edges of the vertex patterns are of the same form as our edge pieces, and fit perfectly onto them, therefore we do not have to test the full concatenation of an edge with our algorithm. We must find a fixed length which can be reached by extending each vertex with multiples of 5 and 6, which are the lengths of our short and regular edge pieces. Here are the lengths of all the vertex patterns (length excludes the precise vertex point on the grid):

- Indegree = outdegree = 1, straight: in- and outgoing edges have length 6
- Indegree = outdegree = 1, perpendicular: ingoing edge has length 6, outgoing has length 3
- Indegree = 2, outdegree = 1, straight: ingoing edges have length 6, outgoing has length 7
- Indegree = 2, outdegree = 1, perpendicular: one of the ingoing edges has length 6, the other ingoing edge has length 9. Outgoing edge has length 7
- Indegree = 1, outdegree = 2: ingoing edge has length 9, outgoing edges have length 5

It seems that we have five different lengths: 3, 5, 6, 7 and 9. These lengths can all be extended to a fixed length of 29 as follows:

$(3) + 6 + 4 \times 5 = 29$
$(5) + 4 \times 6 = 29$
$(6) + 5 + 3 \times 6 = 29$
$(7) + 2 \times 5 + 2 \times 6 = 29$
$(9) + 4 \times 5 = 29$

If we now want to connect these extended vertex patterns, we get the length of both sides, 29, with the orientation swap edge piece of length 10 in between for a total whole edge length of $29 + 10 + 29 = 68$. If the swap piece is not needed, we can substitute it with two short edge piece patterns. To illustrate a whole edge, we will take a look at the connection of two vertices in Figure 5.12. We will connect our vertex with indegree = 1, outdegree = 2 with a vertex that has indegree = 2, outdegree = 1, straight.

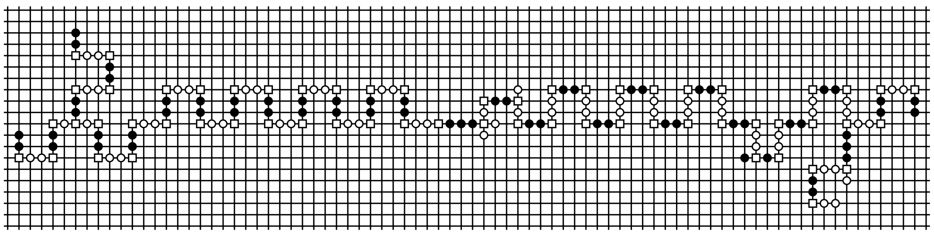


Figure 5.12: Connecting two vertex patterns

The grid is so big that it had to be downscaled, but we can see that the whole edge takes up 68 spaces.

We now have whole edges of length 1, but we must also create edges that are length 2. These edges must take up $2 \times 68 = 136$ spaces on our board. They must also contain an orientation swap edge piece of length 10, and be buildable from short- and regular edge pieces. We will not show them as it would have to be downscaled to an unreadable level, but we can calculate that this is entirely possible: $10 + 21 \times 6 = 136$.

The lowest possible whole edge length might not be 68, but that is not important for our proof. As long as we find a length which allows us to line everything up nicely with out edge pieces of length 5 and 6, the proof holds.

5.3.2 Verification

Our patterns have been verified by a backtracking algorithm. We will take a look at the correctness of the algorithm in chapter 6. For now, we shall assume the algorithm to be correct, so that we can explain how these patterns come about. We want to make sure that deviating from the intended chain of forced moves does not lead to an advantage, so that the players will have to follow the intended path. Therefore, we have devised two claims for each pattern:

1. The white player will win.
2. If the white player is disallowed to make a certain move, black will win.

The patterns have been carefully made so that these claims hold for each pattern. If the colours of the patterns were to be inverted, then the players in the claims would also be swapped. When testing the claims, the patterns include the angular constructs that appear on the right side of Figure 5.1, so that the players must keep pressuring each other with their moves.

When our patterns are concatenated, white should no longer have an advantage over black (apart from a small first-player advantage). The patterns are all adjusted in a way that we expect expect a certain player to win, so that we can test our claims. If the wrong player wins, we can adjust our pattern accordingly. There also should not be a situation in which the white player is disallowed to make a certain move, because this is never the case in APR-5, but this allows us to test both claims on the same pattern.

Let us take a look at a vertex pattern for which the both claims have been verified, the vertex with indegree = 1, outdegree = 2, Figure 5.13. In this figure

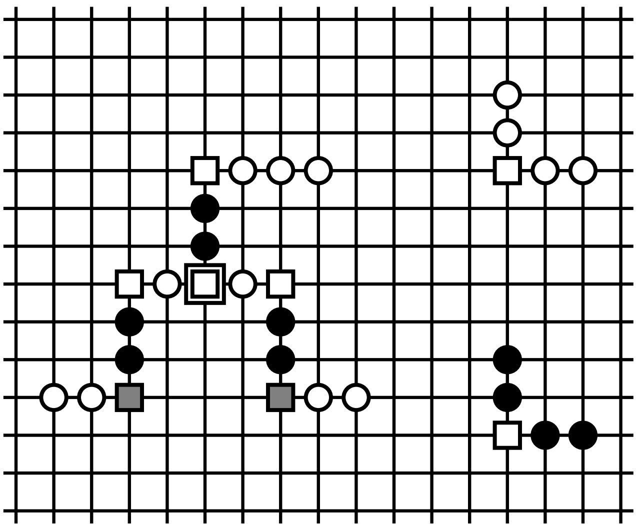


Figure 5.13: Example of a verified pattern

we have already made white's first move, and it is now black's turn. The grey squares are places that white may not make a move when we test the second claim.

For the first claim, white must win. Black must block white's threats and choose whether to go left or right after white places his stone on the vertex point. However, both the left- and right chain of forced moves end in a threat for the white player. When black blocks this threat, he does not create a threat of his own, allowing the white player to place his stone in his angular construct on the right, granting white the win in the next two turns. The algorithm verifies that white does indeed always win in this situation. We can then conclude that any deviation from the intended strategy made by black does not change the outcome of the game.

The second claim requires the black player to win, and disallows the white player from making a move on the grey squares. When we disallow white placing his stones on there, the chain of forced moves will end with a black threat, which the white player cannot block by making a threat of his own. Black then places his stone on the angular construct, guaranteeing him the win in the next two turns. The algorithm verifies that black always wins in this situation, which allows us to conclude that any deviation made by the white player does not gain him any advantage. The angular constructs and disallowed fields have been added to every pattern, after which all the patterns have been successfully tested.

5.3.3 Conclusion

We have covered every situation in the 'Bipartite Geography'-game with patterns. We have also shown that the vertex patterns can be connected by edge piece patterns to re-create any situation in the 'Bipartite Geography'-game. Deviating from the chains of forced moves in the patterns does not lead to an advantage for either player. Therefore, the player that has a winning strategy in the 'Bipartite Geography'-game must also have one in the reduced APR-5 version. We can now conclude that 'Bipartite Geography' can be effectively reduced to an equivalent APR-5 problem. The reduction can be done in polynomial time, because there is a fixed number of pieces for each part that has to be translated. This concludes the proof for Lemma 5.3. □

Having proved both lemma's required for Theorem 5.1, we can conclude that APR-5 is PSPACE-complete. □

6 The backtracking algorithm

In chapter five we argued that APR-5 was PSPACE-hard by using patterns that were assumed to satisfy two claims. In this chapter we will see how the correctness of those claims can be verified. Recall the claims that we discussed in section 5.3.2:

1. The white player will win.
2. If the white player is disallowed to make a certain move, black will win.

These claims can be verified by a recursive backtracking algorithm, in which the players alternate taking turns. Because we have to test two claims, we will need two algorithms. Aside from the differences they have in their accepting and rejecting states, and the different roles of the white and black player, they are almost identical.

6.1 Explanation in pseudocode

In our algorithms, a *situation* is the configuration of an APR-5-game. The variable *nextSituations* is an array of *situation*s that have a follow-up move from the current *situation*. In terms of the game tree that we discussed in section 5.2, these *nextSituations* are the children of the current *situation* in T.

The function heuristic() takes a *situation* and adds just one new *situation* to the *nextSituations* array, that being the *situation* with the move that heuristic() deems best. The bruteforce() function also takes a *situation*, but adds every *situation* reachable from the current *situation* to the *nextSituations* array. In terms of the game tree, the *nextSituations* array is then filled with ALL the children of *situation* in T.

Lastly, the algorithm is recursively called with one of the *situation*s in *nextSituations*. It is the other player's turn in this next situation, hence the negation of $playerColour$.

6.1.1 Testing the first claim

We will call the algorithm that decides whether the first claim is true for a given input *situation* TEST1. In this algorithm, the white player will follow a heuristic, while the black player tries every possible move in every single *situation*. TEST1's accepting state is victory for the white player, and it's rejecting state is victory for the black player.

Algorithm 1 Testing the first claim

```
1:  procedure TEST1($situation, playerColour$)
2:      if win($situation$, white) then return
3:      if win($situation$, black) then
4:          output($situation$)
5:          exit
6:      if $playerColour$ = white then
7:          $nextSituations \leftarrow$ heuristic($situation$)
8:      else
9:          $nextSituations \leftarrow$ bruteforce($situation$)
10:     for $x \in nextSituations$ do
11:         TEST1($x, \neg playerColour$)
```

6.1.2 Testing the second claim

The algorithm that decides whether the second claim is true for a given input situation is called TEST2. The roles of the black and white player are reversed in this second algorithm: the black player now follows the heuristic, while the white player brute-forces. Due to of the restriction in the second claim, the white player is not allowed to make a certain move in his brute-force efforts. The accepting and rejecting states of TEST2 are inverted compared to TEST1.

Algorithm 2 Testing the second claim

```
1:  procedure TEST2($situation, playerColour$)
2:      if win($situation$, black) then return
3:      if win($situation$, white) then
4:          output($situation$)
5:          exit
6:      if $playerColour$ = black then
7:          $nextSituations \leftarrow$ heuristic($situation$)
8:      else
9:          $nextSituations \leftarrow$ bruteforce($situation$)
10:     for $x \in nextSituations$ do
11:         TEST2($x, \neg playerColour$)
```

When testing our second claim, the roles are reversed compared to testing the first claim. The black player now follows the heuristic, while the white player tries every possible move. If our pattern already passed the first test, we know that the white player will win in this pattern. Therefore, we restrict the white player from making a certain move, for which we believe that white cannot win anymore. That restricted move is white's last move in the intended chain if forced moves (remember that white's intended forced moves are the white-filled squares in the figures). If we restrict white from making that move, he cannot answer black's most recent threat with a threat of his own, therefore allowing black to place a stone on his angular construct, guaranteeing black the win in the next two moves.

6.2 Correctness

As we can see, the tests for both claims are very similar. If the heuristic player wins in the current situation, the algorithm returns, stopping that branch of the recursion. If the bruteforce player wins in the current situation, the algorithm outputs that situation, revealing how it happened (this should not happen if the pattern is made correctly). If none of the players wins yet, an array is filled with all the moves that must be tried. That array will contain only one move if it is the heuristic player's turn, and up to $n \times n$ moves if it is the bruteforce player's turn. Then, we enter a loop which goes through all of the moves in the array. Each iteration of the loop makes one move of the array on a copy of the current situation, and recursively calls the algorithm on the newly created situation.

There are a finite number of possible moves in any game situation, with a maximum of $n \times n$. In some situations, the board can completely fill up deep into the recursion, which means that the moves array for that situation will be empty. This should not happen in our patterns, however, because the heuristic player should have a solid strategy with which he can win within a reasonable amount of turns. These observations allows us to conclude that every move in the moves array will be done, and that the algorithm must terminate either when the bruteforce player wins, or when the bruteforce player has lost with every possible strategy. Our patterns vary in size, but the algorithm computed anywhere between 2 and 20 million wins per pattern for the heuristic player before concluding that the bruteforce player could not win there.

It is important to note that the heuristic does not have to be perfect for our claims to be correct. If the heuristic is good enough and manages to win against the bruteforce player in every possible situation, we can still conclude that the pattern is correct, because the bruteforce player did exhaust every possible strategy, and did not manage to win once.

7 Axis-Parallel k-in-a-Row

In chapter 4 we asked the question if 5 is the lowest rowlength for which the decision problem for *'Axis-Parallel k-in-a-Row'* is PSPACE-complete. We will call the decision problem for k-in-a-row APR-k. It is defined as follows:

APR-k = $\{a \in A^*$ is the encoding of a game situation in a $n \times n$ APR-k-game, where the player 'white' that is currently taking his turn has a winning strategy.$\}$

Theorem 7.1. APR-k is PSPACE-complete for $k \geq 5$.

The proof for this theorem relies on the length of the rows in our patterns. The proof that APR-$k \in$ PSPACE still holds. As we saw in section 4.1, reducing the size of k leads to problems in Reisch proof, and the same goes for our proof. Increasing the size of k however, does not lead to any problems. Therefore, we can state that by increasing the rows in our patterns by x for $k = 5+x$ and $x \geq 0$, we can also effectively reduce APR-k to 'Bipartite Geography', for $k \geq 5$. □

Bibliography

[1] S. Even and R. E. Tarjan. A Combinatorial Problem Which Is Complete in Polynomial Space. *Journal of the ACM*, 23(4):710–719, October 1976. doi:`10.1145/321978.321989`.

[2] D. Lichtenstein and M. Sipser. Go is PSPACE-hard. *Annual Symposium on Foundations of Computer-Science*, 19th edition:48–54, 1978. doi:`10.1109/SFCS.1978.17`.

[3] A.R. Meyer and L.J. Stockmeyer. Word problems requiring exponential time. *Proceedings of the Annual ACM Symposium on Theory of Computing*, 5th edition:1–9, 1973. doi:`10.1145/800125.804029`.

[4] S. Reisch. Gobang ist PSPACE-vollständig. *Acta Informatica*, 13:59–66, 1980. doi:`10.1007/BF00288536`.

[5] T.J. Schaefer. On the complexity of some two-person-perfect-information games. *Journal of Computer and System Sciences*, 16(2):185–225, April 1978. doi:`10.1016/0022-0000(78)90045-4`.