# Forays into Van der Waerden Games

Albert Cheu, Lucianna Kiffer

November 1, 2015

We define the Van der Waerden games and present results concerning perfect play. Using experimental results, we conjecture that board sizes satisfying a property of random play obey a recurrence. Additionally, we define a circular variant of the games and perform a similar analysis.

## 1 Introduction

### 1.1 Definitions

Van der Waerden's theorem states that $\forall_{c,k}\exists_W$ such that any $c$-coloring of the numbers $[1, W]$ has a monochromatic $k$-length arithmetic progression (or "mono k-AP").

The original VDW $game(n, k)$ is played on the integers $[1, n]$ between two players; by convention, the first player is Red and the second is Blue. Each player colors integers in their color. If a player creates a mono k-AP in their color, they win the game. If the "board" completely colored without having a mono k-AP, the game is a draw.

Perfect games are the sequence of decisions made by players with optimal strategy. Because many perfect games lead to draws, let $G(k)$ be the least number such that someone will win perfect $game(G, k)$.

Random games are those that emerge when each player colors a random spot on the board on their turn. Let $H(k)$ be the least number such that Red has a first move that is likely to lead to a win for $game(H, k)$.

The initial goal of this research was to find or approximate as many $G(k)$ as possible. $H(k)$ numbers were found as well.

A useful metric to have is $N(n, k, i)$, the number of k-APs contained in the interval [1,n] that use integer $i$. For instance, $N(5, 3, 1) = 2$ because [1,2,3] and [1,3,5].

Additionally, we define $M(n, k) = max(N(n, k, i))$, the largest number of k-APs in $[1, n]$ that include any one integer in the range.

Our notation is summarized in the following table:

| | |
|---|---|
| $W(c,k)$ | Van der Waerden number of $c$ colors and mono k-AP |
| mono k-AP | monochromatic $k$-length arithmetic progression |
| $n$-size board | the integers $[1,n]$ |
| $game(n,k)$ | VDW game of board size $n$ and goal length $= k$ |
| Red and Blue | First and second player, respectively |
| $G(k)$ | Smallest $n$ s.t. $game(n,k)$ is a win |
| $H(k)$ | Smallest $n$ s.t. $game(n,k)$ is a "likely-win" |
| $N(n,k,i)$ | Number of k-APs contained in $[1,n]$ using $i$ |
| $M(n,k)$ | Maximum of $N(n,k,i)$ over all $i \in [1,n]$ |

The circular VDW $game_c(n,k)$ is played on a ring with $n$ spots, where the distance covered by mono k-APs cannot exceed $n$ (the progression does not "end after it begins").

A counterpart table can be constructed for the circular version

### 1.2 Proof: $G(k) \leq W(2,k)$ and $H(k) \leq W(2,k)$

If two players play $game(W(2,k),k)$ until the board is completely colored, the board will have a mono k-AP, by definition of $W(2,k)$. Otherwise, the game ends before the board is completely colored, which means a player has won. In either case, there is a winner.

### 1.3 Fact: If a perfect player can win, it is Red

If it is Blue, Blue has some strategy to attain a mono k-AP. But because Red goes first, Red could use that strategy.

### 1.4 Fact: Every first move in the circular variant is optimal

This directly follows from the fact that every first move is identical.

### 1.5 Continuity Conjectures

The notion "Red keeps on winning once they start" can be expressed as

$$C : \forall_k \forall_{n>G(k)} game(n,k) \text{ is a win.}$$

$C$ is in fact a corollary of a more complex idea:

$$C_{t_k} : \forall_{n,k} \exists_{T_k,t_k} (t_k(n) \geq T_k) \leftrightarrow game(n,k) \text{ is a win}$$

$C_{t_k}$ expresses the belief that some function of the board size, $t_k$, crosses a threshold, $T_k$, at $G(k)$. Note that for fixed $k$, $t_k$ must be a monotonically increasing function of variable $n$ or else continuity does not necessarily hold.

Intuitively, Red has an easier time winning when there are more possible k-APs to acquire in a given $game(n,k)$; as such, we posit

$$C_{M=t_k} : \forall_{n,k} \exists_{T_k} (M(n,k) \geq T_k) \leftrightarrow game(n,k) \text{ is a win.}$$

For clarity,

$$C_{M=t_k} \rightarrow C_{t_k} \rightarrow C$$

Note that these conjectures can describe random play in addition to perfect play; once it is likely that Red will win, it makes intuitive sense that larger board sizes will have the same result.

### 1.6 Proof: $G(1) = 1 = G_c(1)$

If a player only needs a 1-AP to win, Red only needs to claim one spot on the board (in this degenerate case, spacing is irrelevant).

### 1.7 Proof: $G(2) = 3 = G_c(2)$

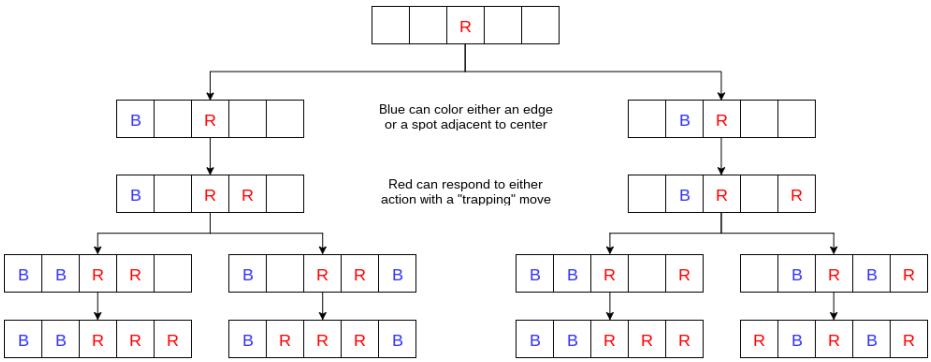
If a player only needs a 2-AP, Red only needs to color two numbers (in this degenerate case, any spacing between them is "equally spaced"). Thus, $n = 3$ allows Red to make two moves.

### 1.8 Proof: $G(3) = 5 = G_c(3)$

First consider the standard linear version.

Red cannot claim three evenly spaced numbers in $game(4, 3)$ because Red cannot claim three numbers at all; only two turns take place.

Red can win $game(5, 3)$ using the following strategy tree (for brevity of the case analysis, we eliminate symmetries):




Every response Blue has to Red's first move is met with a counter; Blue's only choice is to decide which 3-AP Red will obtain.

The same reasoning applies to the circular variant ("join the ends" of the board states in the above tree to obtain circular counterparts)

## 2 Game tree search, on the standard game

Case analysis becomes unfeasible past $k = 3$, so game-playing programs were written to play perfect games. If $game(n, k)$ is determined to be a draw, then $game(n + 1, k)$ is played; the search stops when a game is a win.

### 2.1 Minimax

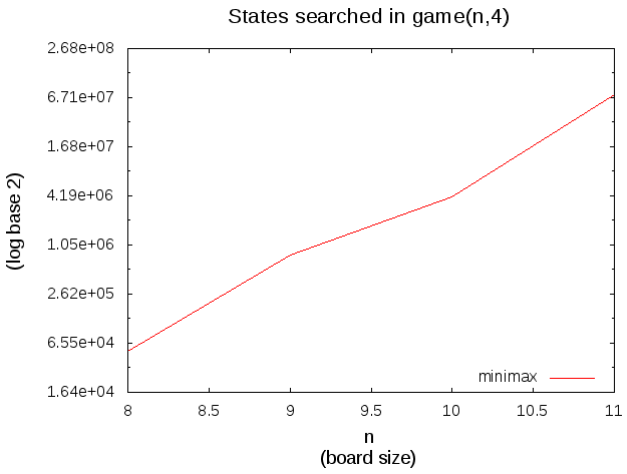
We implemented minimax, which explores all possible board configurations for $game(n, k)$.

Provided with $n$ and $k$, the algorithm first constructs a blank board and assumes the role of Red. It picks an uncolored spot to claim, then assumes the role of Blue. The depth-first exploration continues until there is a mono k-AP or the board is filled without one (a draw).

When all of a node's children are explored, the algorithm—still assuming the role of a player—reports the score of the "best" child to its parent. If the node belongs to Red, it will give priority to wins for Red; the opposite is true for Blue.

$game(n, k)$ is a win if the root node (the blank game state) reports a win for Red.

This approach is fully exhaustive, as the log-scale graph below attests:

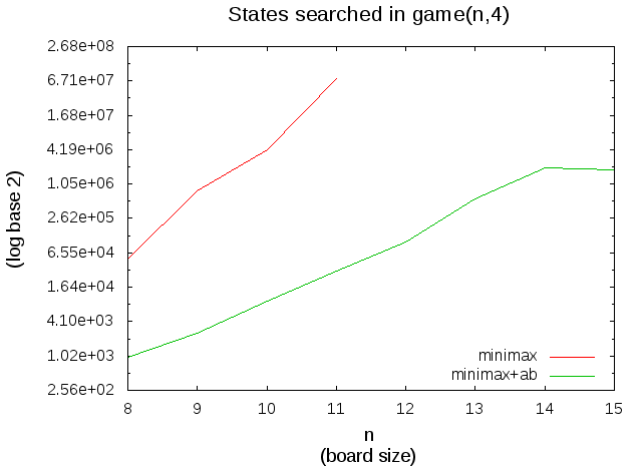



Using linear estimation, the approximate growth rate for $game(n, 4)$ is $10.4^n$.

$game(11, 4)$ was found to be a draw but execution of $game(12, 4)$ was intolerably slow; this approach was not enough to reach $G(4)$.

### 2.2 Alpha-beta pruning

We then employed alpha-beta pruning to avoid searching game states that a player would never choose.



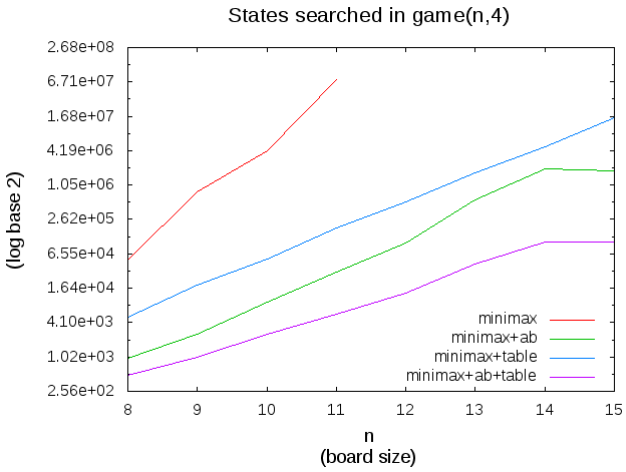

This approach gave a search rate of about $3.12^n$, and let us to find $G(4) = 15$.

Note that the number of nodes visited in $game(15, 4)$ does not conform to the expected exponential, in fact "deflecting downward"; the pruning appears to be particularly effective for $game(G(k), k)$.

## 2.3 Transposition tables

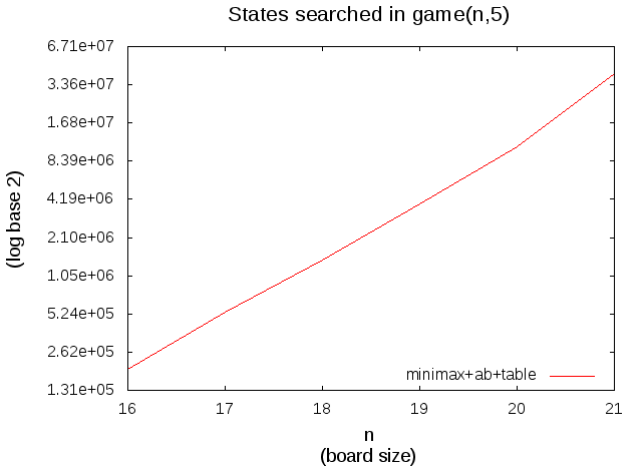
Since there are multiple ways to reach a game state, we employed the use of transposition tables. These data structures cache information about states that have been explored, saving time in the future, at the cost of increased memory usage.

The graph below shows transposition tables applied to pure minimax, and applied to both minimax and alpha-beta pruning:




As can be easily seen, transposition tables do not reduce the search space as much as alpha-beta pruning ($3.65^n$ as compared to $3.12^n$). Still, the two techniques work well together and yield the best search space, $2.47^n$.

This reduction in the search space was insufficient for determining $G(5)$:




The algorithm determined that $game(21, 5)$ is a draw but evaluating $game(22, 5)$ was intolerably slow, running for hours without end. This arose because transposition table had entries for all nodes visited so far; the operating system was forced to use virtual memory.

## 2.4 Move ordering

In all the tree search algorithms described so far, child nodes were explored in a "middle-out" fashion; that is, positions closer to $\frac{n}{2}$ were colored before those further away. This order results in fewer explored board states than coloring in a "left-right" fashion.

Although we learned that coloring middle-out was better than coloring sequentially, it remained to be seen whether there were even better tricks. To this end, we tried the killer heuristic and "greedy" move ordering.

If playing on a position yields an alpha-beta cutoff, that position is searched first in the next loop The killer heuristic gives precedence to positions that led to alpha-beta cutoffs in other paths.

However, the heuristic made the search more laborious.

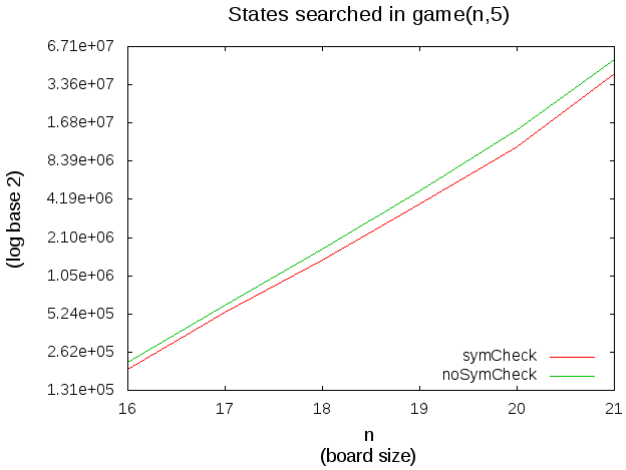
With greedy move ordering, positions that appear in many available k-APs are granted precedence over those that appear in few k-APs. (Note that playing on a position "takes away" available k-APs from the opponent, so multiple data structures were made to keep track of them).

Again, this move ordering prolonged the tree search.

These negative results compelled us to use middle-out for the remainder of the research.

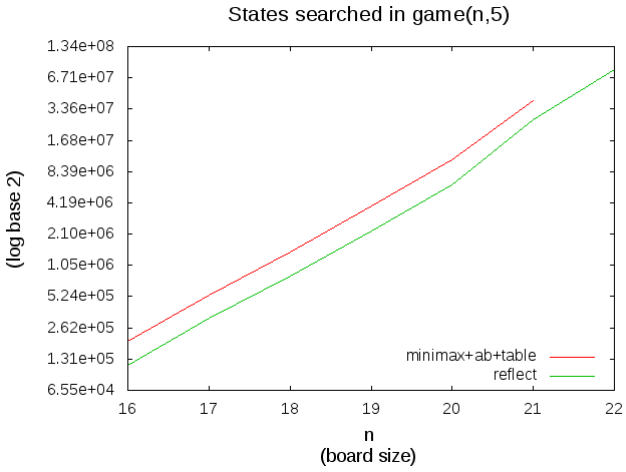
## 2.5 Symmetry and reflection exploitation

A form of symmetry exploitation was already implemented: whenever the current board state is symmetric, only half of the empty spots need to be explored.




That successful exploit highlighted the simplicity of the board, one that is further utilized in the following trick.

Observe that, because $[1, R, R, 4, B]$ is a win, the reflection $[B, 2, R, R, 5]$ is also a win. Generally, if the reflection $r$ of a state $s$ is known, all information about $s$ can be deduced from that of $r$. This effectively halves the search (and storage) space:




$game(22, 5)$ is a draw. $game(23, 5)$ did not terminate after seven hours.

## 2.6 Principal variation search

PVS assumes that the first move chosen by minimax on a given game state is part of perfect play; it adjusts the alpha-beta values accordingly. If this assumption is correct, then the number of states searched will be reduced.

However, the effect on our search space was negligible.

### 2.7 Conjecture: The "win-window"

Based on observation of perfect $game(n \geq 15, 4)$ as well as the effectiveness of the middle-out move ordering, we believe that there is a contiguous subsequence of $[1, n]$ centered at $\frac{n}{2}$ such that one of Red's winning strategies includes playing first in that "win-window."

## 3 Game tree search on the circular variant

The same techniques above were applied to the circular variant. An additional trick was found.

Consider the following state: [1,R,B,4,5]. Rotation preserves the relative order of the coloring: [R,B,3,4,5], [B,2,3,4,R], [1,2,3,R,B], [1,2,R,B,5]. This reduces the size of the transposition table.

It was found that $G_c(4) = 10$ and $G_c(5) = 21$.

## 4 Monte Carlo

Because tree search became unfeasible for $k = 5$ (standard version) and $k_c = 6$ (circular variant), we employed the Monte Carlo method.

### 4.1 Algorithm overview

As with tree search, the inputs of the Monte Carlo program are $n$ and $k$.

Unlike tree search, the output is a sequence of $\frac{n}{2}$ numbers each ranging from 0 to 1. The $i$th output is the program's confidence that playing on board position $i$ as the very first move will lead to a win. (Because the initial game state is symmetric, only half of the board needs to be examined.)

At initialization, the program represents an upper portion of the game tree: provided with $n$ and a limit to the number of stored nodes, the program calculates the greatest depth to store.

Each game state's node contains the following information:

```
-Node-
Constant Integer:
    position (the number to be colored)
    depth
Constant List: children
Constant Boolean: redPlayer (which player's turn)
Counter Integer:
    numTrials (how many trials used this node)
    redWins (how many trials Red won using this node)
    blueWins (likewise for Blue)
```

The root node start is at depth 0 and start.redPlayer=true.

The program plays random trial games until the user exits.

For every trial, the program traverses the tree from start to a node of maximum depth (described in the next subsection).

For every node in that path, the program colors the board at integer node.position with the color of the parent (i.e. if the program moves from start to a node with position=3, 3 would be colored red because start.redPlayer=true).

When a node of maximum depth is reached, the remainder of the board is colored in a random game pattern; that is, Red and Blue take turns coloring randomly.

Once a player wins (or the game draws), the program works its way back up to the root. Each node on the path has its counters updated accordingly: if Blue won, node.blueWins increments but node.redWins remains the same, and vice versa. node.numTrials always increments.

Every ten-thousand trials, the program produces its output by iterating through the nodes at depth 1 (the first move) and calculating the ratio of node.redWins to node.numTrials.

Every millionth trial, the program prompts the user to continue or exit; the user stops execution when the values have converged to his or her satisfaction.

### 4.2 Tree traversal pattern

As stated previously, the tree is traversed in a depth-first pattern millions of times. When at a given node, the algorithm decides which child to recurse upon.

If a child has never been used in a trial before (that is, child.numTrials=0), it will be chosen immediately. If multiple children are untouched, the leftmost child is chosen.

Otherwise, the algorithm picks the child with maximum score:

```
score(Node child, Boolean parentIsRed, Integer totalNumTrials):
  Constant Integer numTrials := child.numTrials
  Constant Integer numWins
  if parentIsRed
    numWins := child.redWins
  else
    numWins := child.blueWins

  return avgSuccess(numTrials, numWins) + regret(numTrials, totalNumTrials)
```

Ideally, the program will run trials using nodes that have had high average success:

$$avgSuccess(numTrials, numWins) = \frac{numWins}{numTrials}$$

As seen in the pseudocode, "success" is relative to the parent (who chooses the child).

But the program should also visit nodes it would "regret" not visiting; those that have been in relatively few trials may still yield good results:

$$regret(numTrials, totalNumTrials) = \sqrt{2\frac{\log(totalNumTrials)}{numTrials}}$$

### 4.3 Remarks

Identical experiments were run on the circular game variant; as every first move is the same, only one percentage was output.

Although it was believed that the technique would give estimates for $G(k)$, the $H(k)$ values were the only ones gleamed from the random sampling. This was determined by running the algorithm with millions of game states stored and with only the top level; between them, there was no qualitative change in the data.

# 5 Results

The following table presents the game numbers found in this research:

| $k$ | $G(k)$ | $G_c(k)$ |
|---|---|---|
| 1 | 1 | 1 |
| 2 | 3 | 3 |
| 3 | 5 | 5 |
| 4 | 15 | 10 |
| 5 | $> 22$ | 21 |

## 5.1 Conjecture of H(k)

The application of the Monte Carlo algorithm gave the following estimates for $H(k)$:

| $k$ | $\sim H(k)$ | $\sim H_c(k)$ |
|---|---|---|
| 5 | 23 | 21 |
| 6 | 39 | 31 |
| 7 | 67 | 51 |
| 8 | 105 | 81 |

Consider the candidate sequence for $H(k) : [15, 25, 41, 67, 109, \ldots]$. It has the following elegant recurrence relation:

$$\forall_{k>5} H(k) = H(k-1) + H(k-2) + 1$$
$$H(5) = 25$$
$$H(4) = 15$$

Equivalently,

$$\forall_{k\geq 4} H(k) = 2F(k+2) - 1,$$

where $F(i)$ is the $i$th Fibonacci number.

As with the standard game, consider $H_c : [21, 31, 51, 81, 131, \ldots]$ which has the recurrence relation

$$\forall_{k>6} H_c(k) = H_c(k-1) + H_c(k-2) - 1$$
$$H_c(6) = 31$$
$$H_c(5) = 21$$

Equivalently,

$$\forall_{k\geq 5} H_c(k) = 10F(k-2) + 1,$$

Supporting evidence for the linear game conjecture is given by tests of the continuity conjecture (section 1.4). Recall the most verbose version:

$$C_{M=t_k} : \forall_{n,k} \exists_{T_k} (M(n,k) \geq T_k) \leftrightarrow game(n,k) \text{ is a win.}$$

Testing $C_{M=t_k}$ yields another recurrence relation, this time for $T_k$:

| k | H(k) | $M(H(k),k)$ |
|---|---|---|
| 5 | 25 | 20 |
| 6 | 41 | 30 |
| 7 | 67 | 50 |
| 8 | 109 | 80 |
| 9 | 176 | 130 |

It appears that $T_k = 10F(k-2)$.

Note that the table claims $H(9) = 176$ even though the conjectured $H(k)$ recurrence entails $H(9) = 177$. The recurrence could be an approximation; specifically, it may be that $H(k) = H(k-1) + H(k-2) + c(k)$ where the magnitude of $c(k)$ is small.