Auto Conjecturing and Konane

Ian Shehadeh IRSHEHADEH@SMCM.EDU

Division of Mathematics
St. Mary's College of Maryland
St. Mary's City, Maryland, USA

Abstract

Konane is an ancient Hawaiian game that plays similar to checkers. We present *Konjecture* a suite of tools for computional analysis of Konane. Using these tools combined with the program *Conjecturing*, we investigate the outcome class of stair-shaped patterns on Konane boards.

Keywords:

1 Kōnane

Kōnane is an ancient Hawaiian game played on a rectangular grid. The size of this grid varies. There are two players, one playing with black pieces, and the other white pieces.

1.1 Kōnane Starting Phase

A game begins with black removing one of their corner pieces or one of their center pieces (Thompson (2005)). For example, in Figure 1 the game begins with black choosing one of the pieces outlined in red to remove. Once black removes a piece, white then chooses an adjacent piece to remove. For example, if black begins by removing $(3, 3)$, moving the position in Figure 2, white can now remove any of the highlighted pieces.

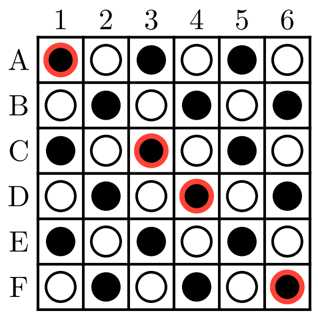


Figure 1: Starting Position

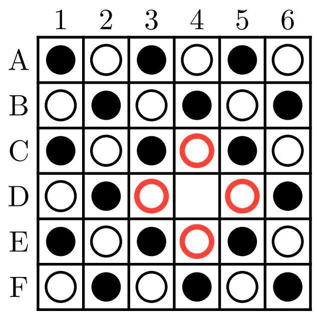


Figure 2: Turn 1

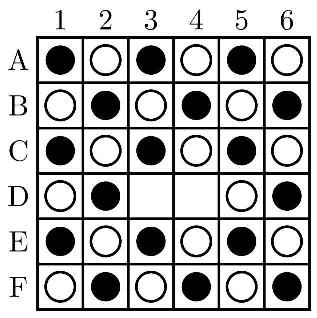


Figure 3: Turn 2

,

Once white removes a piece, normal play begins.

1.2 Kōnane Normal Play

On each player's turn, they must capture at least one of their opponent's pieces. The first player who is unable to capture any pieces loses.

A player may capture a piece of the opposing color is captured by *jumping* over it using one of their pieces into an empty space. Figure 4 shows white capturing a black piece by moving from $(0, 2)$ to $(0, 0)$.

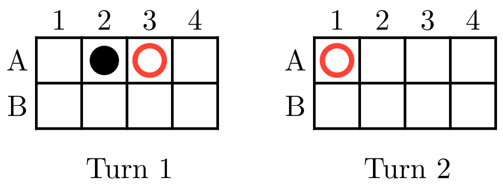


Figure 4: Capture

Multiple captures can be made in a single direction. For example, beginning from *Start* in Figure 5, if black plays first they can move to *Option 1*, *Option 2*, or *Option 3*.

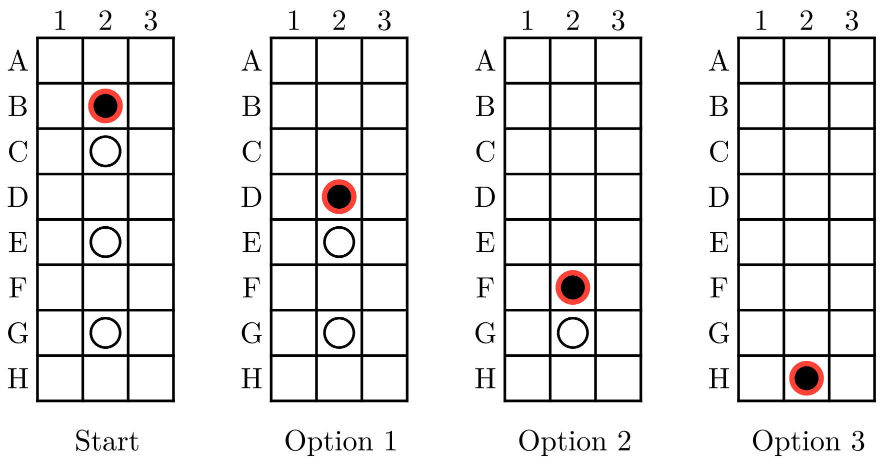


Figure 5: Capture

2 Conjecturing

Conjecturing (Larson and Van Cleemput (2016)) is a project that generates a series of true inequalities based on a table of objects with associated *invariants*. In our case, an object is a single Kōnane position, and invariants include things like *number of moves*, *number of black pieces*, *number of white pieces*, etc.

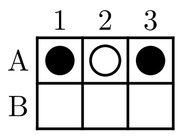
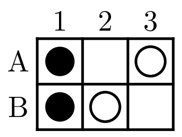
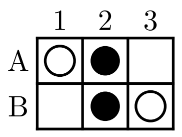
Games	#Black (I_1)	#White (I_2)	#Moves (I_3)
	2	1	0
	2	2	1
	2	2	2

Table 1: Conjecturing Inputs

We choose a single invariant to place on the left hand side of the inequality, and which operators to use. For example, given the data in Table 1, we could instruct *Conjecturing* to use I_2 on the left hand side of $\leq$, and create expressions using the operators $A+B, A*B, A+1, \sqrt{A}, \max(A,B)$. A few of the possible results are shown in Figure 6.

$$\begin{aligned} I_2 &\leq \max(I_1, I_2) \\ I_2 &\leq I_1 + I_2 \\ I_2 &\leq I_1 + 1 \\ I_2 &\leq I_1 * I_2 + 1 + 1 \end{aligned} \tag{1}$$

Figure 6: Conjecturing Output

3 Computation Analysis Kōnane

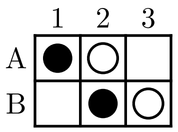
We were unable to find any open-source programs with a well-optimized implementation of Kōnane. But, computer-friendly grid game representations are well studied for games like Go, Chess and Checkers (Schaeffer et al. (2007)).

3.1 Data Structure

Definition 3.1.1 *Bit Field*: A bit field is a string of binary digits with a constant length. We write a bit fields right-to-left. Each bit has an associated index, beginning at 0.

Example: Given the bit field $B = 01101$, $B_0 = 1$, $B_1 = 0$, $B_2 = 1$, $B_3 = 1$, $B_4 = 0$,

We use two bit fields to represent a game of Kōnane. Given a board of width W and height H, we use bit fields with $W \times H$ elements. The first bit field (hereafter *BLACK*) has a 1 at index $y * W + x$ if and only if the game has a black piece in cell (x, y) (the top left cell is $(0, 0)$). Similarly, the bit field representing white (hereafter *WHITE*) has a 1 at $y * W + x$ if and only if the game has a white piece in the cell (x, y).



is represented as

$$\begin{aligned}\text{BLACK} &= 100010\\ \text{WHITE} &= 010001\end{aligned} \tag{2}$$

Figure 7: Bit Field Representation

This format is compact, and it allows us to use only a few CPU instructions to calculate each player's moves.

3.2 Solving Games

We use the *cgt* library for Rust [Maciosowski (2024)] to solve games. This library's algorithm is recursive: for a given position, it calculates the normal form of each of its children (moves from that particular board state). If it has no children then the normal form is 0. Otherwise it eliminates the reversible and dominated moves from the child positions, then attempts to coerce the game into a sum in the form $N + \uparrow(U) + *(S)$, where $N, U \in \mathbb{Z}$ and $S \in \mathbb{Z}^+$, failing that, the library keeps the game as a list of child positions.

The two most important optimizations is computing solutions in parallel, which the library achieves with *rayon* (2024), and storing solved positions in a cache. The cache allows calculations to be re-used when a particular game appears in multiple places in the tree.

To further optimize the Konjuecturing, we modified *cgt*'s algorithms for removing dominated and reversible positions. The most significant change was to have the functions update their operands in-place, instead of allocating new arrays for the move lists, this change gave roughly a 50% speed-up, which was noticeable when solving large collections of Kōnane games.

3.2.1 Testing

To ensure accurate results, this implementation must be carefully tested.

The underlying bit-field representation is tested using a method called property-based testing. These types of tests are concerned with a specific *property*, some check that theoretically always holds given a certain class of inputs, then we generate random input values and check the property to verify that it holds in practice.

The game's implementation has several specific tests, which come from simple cases (do we get the correct set of moves for a 2×2 game, etc.), and property tests based on established facts about specific Kōnane patterns. We've also begun to write tests based on well-known Kōnane positions, primarily from *Playing Kōnane Mathematically* Ernst (1995). The first of these tests builds a series of alternating black and white pieces in a line, then puts it in normal form and compares the result with the expected value, which is outlined in *Playing Kōnane Mathematically* Ernst (1995). Note although we only test these cases, the implementation is general enough to handle arbitrary games of Kōnane.

The test suite has expanded to the point we are reasonably confident about the program's accuracy. But, we'll continue to add unique test cases as they arise.

3.3 Invariants

Most invariants are arbitrary, they are measures that "seem interesting". But facts about particular positions from prior research inform the ideas. For example, in *Kōnane has Infinite Nim Dimensions*, Santos and Silva (2008), construction of new positions relies on a "focal point," a piece that can be captured in a few ways, and opens both players up to new moves. It suggests that the number of pieces that can be captured, and the number of moves for a given player may be interesting attributes. Currently, we track the following invariants:

- Average distance from each piece to the nearest border.
- Total possible moves.
- Number of unique pieces that can be captured.
- Counts of each tile state.
- Distance between the highest and lowest piece.
- Distance between the furthest left and furthest right piece.
- The nim-value of a game
- The number-value of a game

All of these can (except for the game values), can be calculated for a single player, or both.

4 Known Facts

This section is a list of known facts about the game that informed our analysis.

4.1 Space Complexity

Kōnane is PSPACE-Complete Hearn (2009), meaning it takes a polynomial amount of space relative to its input size, and any other problem in PSPACE can be solved in polynomial time if Kōnane can be solved in polynomial time.

4.2 Existence of Arbitrary Nim Dimension

If there is a $*n$ position on an $l_n \times c_n$ board, then there is a $*m$ position on an $l_m \times c_m$ board such that:

$$\begin{aligned} l_m &= l_n + c_n + 2 \\ c_m &= l_n + 3 \end{aligned} \tag{3}$$

The base case is $l_2 = 7$, $c_2 = 6$. Santos and Silva (2008)

4.3 Solid Linear Pattern

A solid linear pattern is series of N alternating white and black pieces, with jumps allowed on either side.

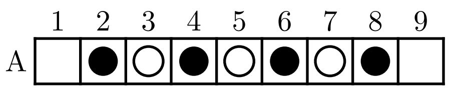


Figure 8: $N = 7$

Theorem 4.3.1 *(Outcome of Solid Linear Pattern)* For a solid linear pattern with N stones, the outcome class can be found by the function:

$$\mathrm{SLP}(N) = \begin{cases} -j & \text{if } N = 2j+1 \\ 0 & \text{if } N = 4j \\ * & \text{if } N = 4j+2 \end{cases} \tag{4}$$

Source: Ernst, *Playing Kōnane Mathematically* Ernst (1995)

4.4 Solid Linear Pattern With Tail

We say a solid linear pattern has a tail of length M, if there are M alternating pieces below the left-most piece in the solid linear pattern.

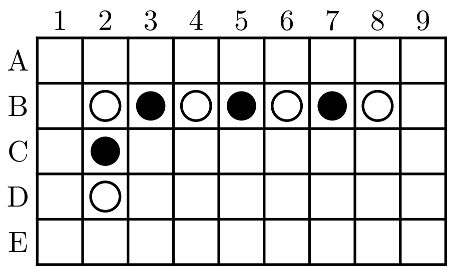


Figure 9: Linear with Tail $N = 7, M = 2$

Given a solid linear pattern of length N with a tail of length 1, the normal form is

$$\begin{array}{rll} * & \text{if } N \in \{2,3\} & \\ \{* \mid \downarrow\} & \text{if } N = 4 & \\ \{* \mid -2j+2\} & \text{if } N = 4j, & \text{where } j > 1 \\ \{2j-1 \mid *\} & \text{if } N = 4j+1, & \text{where } j > 0 \\ \{* \mid -2j+1\} & \text{if } N = 4j+2, & \text{where } j > 0 \\ \{2j \mid 0\} & \text{if } N = 4j+3, & \text{where } j > 0 \end{array} \tag{5}$$

Source: Ernst, *Playing Kōnane Mathematically* Ernst (1995)

4.5 Solid Linear Pattern with Offset Tail

A solid linear pattern of length N with *offset* tail of length M is identical to a solid linear pattern with tail, but with the top left piece removed.

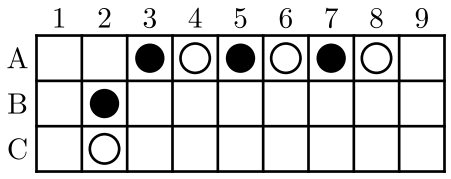


Figure 10: Linear with Offset Tail $N = 7, M = 2$

$$\begin{array}{rl} \downarrow & \text{if } N = 3 \\ j-1 & \text{if } N = 2j, \quad \text{where } j > 0 \\ * & \text{if } N = 4j+1, \text{ where } j > 0 \\ 0 & \text{if } N = 4j+3, \text{ where } j > 0 \end{array} \tag{6}$$

Source: Ernst, *Playing Kōnane Mathematically* Ernst (1995)

4.6 Somewhat Solid Linear Patterns

Let $S(a, b, c, ...)$ be a series of solid linear patterns beginning with a white tile, each separated by a space.

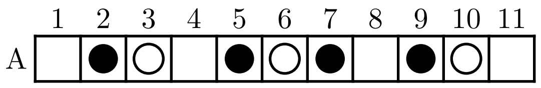


Figure 11: $S(2, 3, 2)$

Quick Facts (2010):

- $S(2a + 1, 2b + 1, 2c + 1) = a + b + c$
- $$S(2a+1, 2b+1, 2c) = S(2a+1, 2(b+1)) + S(2k-2) = \begin{cases} \star\,(i - (b+1) + (c-1)) & \text{if } a > b+1 \\ \star\,(2^{a+(b+1)-1} + (k-1)) & \text{if } a \leq b+1 \end{cases} \tag{7}$$

4.7 Penetration

The penetration of a particular Kōnane configuration is how far its members are able to move. Rectangular configurations have a maximum penetration of 4 Ernst (1995).

4.8 Patterns in Real Play

This is a living list of patterns observed in real play, and general facts about the game:

- When the game was first documented by European settlers the most common board size was 14×17, although anything from 8×8, to 13×20 was also used. Modern games are often played on an 18×18 board (2010).
- "In real Kōnane games, play frequently proceeds to the corners after center of the board has been largely cleared" Ernst (1995)

5 Linear with Offset Tail

In this section, we'll analyze the outcome class of $\text{LOT}_1(n)$. Games in this pattern have a black stone in position C3, and $n - 1$ alternating black and white pieces in row B, beginning at cell B3 and moving right.

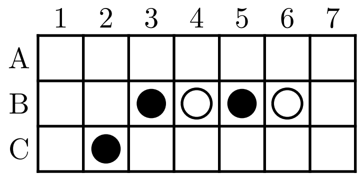


Figure 12: $\mathrm{LOT}_1(5)$

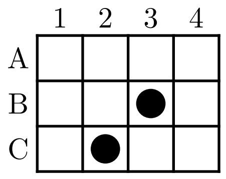


Figure 13: $\mathrm{LOT}_1(2)$

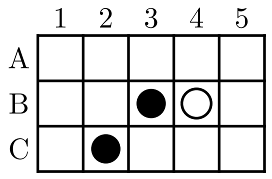
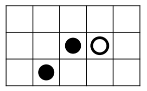
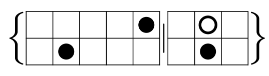
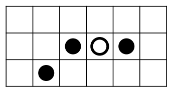
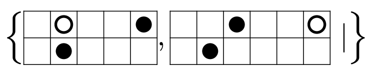


Figure 14: $\mathrm{LOT}_1(3)$

First, the early positions have no moves available, so $\mathrm{LOT}_1(1) = \mathrm{LOT}_1(2) = 0$.

$\mathrm{LOT}_1(3) =$  $=$  $= \{0|\,*\} = \uparrow$

$\mathrm{LOT}_1(4) =$  $=$  $= \{|\,*, 0\} = -1.$

To find the general case, we'll look at the even and odd cases individually.

5.1 $\mathrm{LOT}_1(n)$ When n is Even

Consider a game $\mathrm{LOT}_1(2i)$, because there are an even number of stones, we know the right-most stone is black. So, black has no moves available, and white has two: they can capture the left-most black stone in B3, moving the game to $\mathrm{SLP}(2i-3) + \mathrm{SLP}(2)$ (the white and black stones in column 2 will be out of reach for the solid linear pattern in columns 5+). Or, white can capture the right-most white stone (7B in Figure 15), moving the game to $\mathrm{LOT}_1(2(i-1))$.

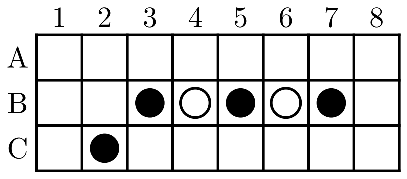


Figure 15: $\mathrm{LOT}_1(6)$

By Theorem 4.3.1 we know $\mathrm{SLP}(2i-3) + \mathrm{SLP}(2) = (-i+1) + *$. Now, we have $\mathrm{LOT}_1(6) = \{|-1+*, \mathrm{LOT}_1(4)\} = \{|-1+*, -1\}$. Because -1 dominates $-1+*$, $\mathrm{LOT}_1(6) = \{|-1\} = -2$. Assuming $\mathrm{LOT}_1(2i) = -(i-1)$, then $\mathrm{LOT}_1(2(i+1)) = \{|-(i-1)+*, -(i-1)\} = \{|-(i-1)\} = -i$. Because $\mathrm{LOT}_1(2(3)) = -2$, $\mathrm{LOT}_1(2(2)) = -1$, by induction it follows that $\mathrm{LOT}_1(2(i+1)) = -i$.

Therefore, in the general case we can say $\mathrm{LOT}_1(2i) = -(i-1)$, for $i \geq 1$.

5.2 $\mathrm{LOT}_1(n)$ When n is Odd

Consider a game $\mathrm{LOT}_1(2j+1)$, the right-most stone is white. So, black and white both have a single move available. White can capture the black stone in B3, moving the game to $\mathrm{SLP}(2) + \mathrm{SLP}(2(j-1))$, and black can capture the right-most white stone, moving the game to $\mathrm{LOT}_1(2j-1)$.

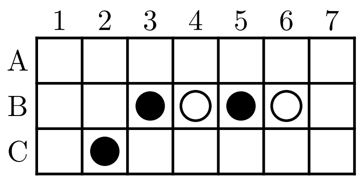


Figure 16: $\mathrm{LOT}_1(5)$

So, we have the game $\mathrm{LOT}_1(2j+1) = \{\mathrm{LOT}_1(2j-1)|\mathrm{SLP}(2) + \mathrm{SLP}(2(j-1))\}$. by Theorem 4.3.1, we know

$$\mathrm{SLP}(2j) = \begin{cases} 0 & \text{if j is even} \\ * & \text{if j is odd} \end{cases} \tag{8}$$

So let's split LOT_1 into even and odd cases for j:

$$\begin{aligned} \mathrm{LOT}_1(2j+1) &= \begin{cases} \{\mathrm{LOT}_1(2(j-1)+1) \mid * + *\} & \text{if j is even} \\ \{\mathrm{LOT}_1(2(j-1)+1) \mid * + 0\} & \text{if j is odd} \end{cases} \\ &= \begin{cases} \{\mathrm{LOT}_1(2(j-1)+1) \mid 0\} & \text{if j is even} \\ \{\mathrm{LOT}_1(2(j-1)+1) \mid *\} & \text{if j is odd} \end{cases} \end{aligned} \tag{9}$$

Recall $\mathrm{LOT}_1(3) = \uparrow$, so $\mathrm{LOT}_1(5) = \{\uparrow \mid 0\} = *$, and $\mathrm{LOT}_1(7) = \{* \mid *\} = 0$. By the above recursive definition of $\mathrm{LOT}_1(2j+1)$, we can see this pattern repeat, leading to the simplified definition:

$$\mathrm{LOT}_1(2j+1) = \begin{cases} * & \text{if j is even} \\ 0 & \text{if j is odd} \end{cases} \tag{10}$$

5.3 Putting it Together

Theorem 5.3.1 *(Outcome of* LOT_1*(n))*

$$\mathrm{LOT}_1(n) = \begin{cases} * & \text{if } n = 4j+1 \\ 0 & \text{if } n = 4j+3 \\ -(j-1) & \text{if } n = 2j \end{cases} \tag{11}$$

The first 2 cases are from Section 6.2, and the last is from section 6.1.

6 Conclusion

We did not find any significant results analyzing Kōnane with Conjecturing. However, the project had a few interesting side-effects. The most interesting of which was using Jupyter Note Books and Sage Math for combinatorial game theory research; Creating an efficient implementation of Kōnane, which integrates with some CGT tools; and small improvements to those tools.

6.1 Further Work

At the time of writing, there is no well-known Sage Math package for combinatorial game theory. Such a package could be useful, since it would allow people reasearching combinatorial game theory to leverage the existing tools that integrate with Sage more easily (for example Conjecturing).

References

Ernst, Michael. 1995. "Playing Konane Mathematically: A Combinatorial Game-Theoretic Analysis". *UMAP Journal* 16:95–121

Hearn, Robert A. 2009. "Amazons, Konane, And Cross Purposes Are PSPACE-complete". Edited by Michael H. Albert and Richard J. Nowakowski. *Games of No Chance 3*. Mathematical Sciences Research Institute Publications. Cambridge: Cambridge University Press. https://doi.org/10.1017/CBO9780511807251.015

Larson, C.E., and N. Van Cleemput. 2016. “Automated Conjecturing I: Fajtlowicz's Dalmatian Heuristic Revisited”. *Artificial Intelligence* 231 (February):17–38. https://doi.org/10.1016/j.artint.2015.10.002

Maciosowski, Tomasz. 2024. “Combinatorial Game Theory Library in Rust”

Santos, Carlos, and Jorge-Nuno Silva. 2008. “Konane Has Infinite Nim-Dimension”. *Integers [Electronic Only]* 1 (January)

Schaeffer, Jonathan, Yngvi Björnsson, Akihiro Kishimoto, Martin Müller, Robert Lake, Paul Lu, and Steve Sutphen. 2007. “Checkers Is Solved”. *Science* 317 (October):1518–22. https://doi.org/10.1126/science.1144079

Thompson, D. 2005. “Teaching a Neural Network to Play Konane”. In

“1xn Konane: A Summary of Results.” 2010. *Games of No Chance [2] More Games of No Chance*. Games of No Chance. Cambridge [u.a.]: Cambridge Univ. Press

“Rayon-Rs/rayon.” 2024