Analyzing the game of Pan via a game-theoretic and computational approach

by Aron Samkoff

April 20, 2015

An essay in partial fulfillment of the requirements for the degree of Master of Science

Written under the direction of

Doron Zeilberger (chair),

János Komlós, Michael Saks, and Robert Wilson.

1 Introduction

Pan (lit. “Sir”) is a Polish card game typically played with a deck of 24 cards. From a standard 52-card deck, the ranks 2 through 8 are discarded. The ranks of the remaining cards are considered to be ordered from 9 to ace. These remaining cards are shuffled and dealt between the players (usually 2-4). The player with the 9 of hearts typically starts first by placing this card face up in the center to form the pile, and play then proceeds in a clockwise direction. On a given player’s turn, he or she may make a legal move, consisting of one of the following:

1. A card may be placed on the pile (“discarded”) if and only if that card has rank at least that of the topmost card on the pile.

2. The three topmost cards of the pile may be taken, or as many as can be taken without taking the 9 of hearts.

3. If the player has all three remaining nines (remember that the 9 of hearts always stays on the bottom of the pile) or all four cards of any other rank, the cards of this rank may be placed on the top of the pile, as long as this move agrees with rule 1.

The game is won by the player who first discards all of their cards. For the purposes of this paper, the analysis will be limited to a game of Pan with two players, Player A and Player B. Also note that in the rules 1-3, the only mention of the suit of any cards is distinguishing the 9 of hearts. This card is only in play for the first move. The suits of all the other cards are immaterial and therefore do not need to be recorded for a given position[1]

In this essay, I will analyze the game of Pan using basic combinatorics and game theory. I will also use ideas from artificial intelligence and implement these ideas with Maple. The accompanying Maple code `pan.txt` includes procedures that will be referenced throughout this essay.

1.1 An example

Before we proceed with the analysis, let us examine a game of Pan.

[1]In this paper, the term “position” will refer to a configuration of the cards. This should not be confused with “hand” which will refer to the cards that a player has.

Suppose Player A is initially dealt the hand {9h, 10, 10, 10, J, J, J, J, K, K, K, A} and Player B has {9, 9, 9, 10, Q, Q, Q, Q, K, A, A, A}. Since A has the 9 of hearts, he must place this card on the pile, so that the pile is just {9h}. By rule 3, since Player B has the remaining Nines, he may place all of these on the pile. Suppose Player A now places a 10 on top of the 9, which he can do by rule 1. B can now discard his last 10. After these moves, the players have arrived at the following position:

$$\begin{aligned} A :& \{10, 10, J, J, J, J, K, K, K, A\} \\ B :& \{Q, Q, Q, Q, K, A, A, A\} \\ Pile :& \{9, 9, 9, 9, 10, 10\} \end{aligned}$$

Player A now discards one of his Tens on the pile, after which Player B happily puts down his four Queens on the pile (rule 3). The position now looks like this:

$$\begin{aligned} A :& \{10, J, J, J, J, K, K, K, A\} \\ B :& \{K, A, A, A\} \\ Pile :& \{9, 9, 9, 9, 10, 10, 10, Q, Q, Q, Q\} \end{aligned}$$

A realizes things do not look good. He still has a 10 and four Jacks in his hand, but is unable to play these due to rule 1. If he places a King to the pile, B will place his last King on the pile and will have a hand full of three aces, at which point nothing can stop B from playing an ace on each subsequent turn, winning the game. A could take the top three Queens, but B would reply by placing a King and again, winning in six moves [2]. So A tries the only other move he has, and places an A on the pile. Alas, B is unfazed, and places another Ace on the pile:

$$\begin{aligned} A :& \{10, J, J, J, J, K, K, K\} \\ B :& \{K, A, A\} \\ Pile :& \{9, 9, 9, 9, 10, 10, 10, Q, Q, Q, Q, A, A\} \end{aligned}$$

[2]Unless stated otherwise, assume a “move” means a “half-turn”.

A is now forced to take the top three cards (two Aces and a Queen) into his hand, since he may not discard by rules 1 and 3. B places his King on the pile:

$$\begin{aligned} A :& \{10, J, J, J, J, Q, K, K, K, A, A\} \\ B :& \{A, A\} \\ Pile :& \{9, 9, 9, 9, 10, 10, 10, Q, Q, Q, K\} \end{aligned}$$

A is now doomed, for no matter what cards he plays in the next two turns, B will follow with an Ace.

1.2 Overview of essay and guiding questions

There are three basic questions I will address in this paper.

1. Given that a player moves first and both players play optimally, what are the probabilities that the player wins, loses, and draws?
2. For each k, what is the approximate average branching factor[3], and the approximate number of positions for k-Pan?
3. What proportion of the time should a player draw cards from the pile, and when should the player instead place a card on the pile?

In Section 2, I will develop some basic tools that will help analyze these questions and address the second question. In Section 3, I will apply techniques from combinatorial game theory to help answer the first question. In Section 4, I will develop a heuristic function for use in the minimax algorithm, and show that it leads to reasonably good play using a modest depth. In section 5, I present a more efficient version of the minimax algorithm, and develop a procedure that can be used to play against an AI opponent. I also use the modified minimax algorithm to find patterns in positions where taking the top three cards from the pile is the most favored move. Finally, I discuss how the rules of Pan can be relaxed or modified to create other interesting games.

[3]The average branching factor will be defined in Section 2.1.2.

1.3 Notation and Terminology

In the two player game of Pan, we will assume that after the cards are dealt, both players have complete information: Each player knows which cards are in Player A's hand, Player B's hand, and on the pile. We will usually consider the first move, that of placing the 9 of hearts, to already be made. Hence, after the cards are shuffled and dealt and the player with the 9 of hearts begins, there is no chance aspect to the game: both players have complete information, and there are no other random processes in the game.

We will see that the standard game of Pan can be computationally intensive to analyze. For this reason, I define the class of games *k-Pan*, for $k \geq 1$ which uses k ranks in play, and can be solved completely for small k. $k = 4$ will be used often in this paper, being the largest k for which k-Pan could be solved using the hardware available to me. The game of 6-Pan is the standard game of Pan, and for $k \leq 6$, we consider the ranks in play to be 9, 10, J, and so on. These ranks will also sometimes be referred to by natural numbers, with rank 1 being a '9', rank 2 being a '10', rank 3 being a Jack, and so on. I denote a player's hand as a vector in square brackets with k components, and with each component representing the number of cards of a given rank in the hand. For instance, Player B's starting hand in the previous example would be denoted $[0, 0, 0, 4, 1, 3]$, representing a hand in 6-Pan (standard Pan) and consisting of 4 Queens, 1 King, and 3 Aces. A given *position* will be denoted by the pair (A, B) of the hands for Players A and B. Since no cards ever go out of play, the pile, P (also a k-component vector), can be determined by subtracting $A + B$ from the vector $F = [4, 4, ..., 4]$: $P = F - (A + B)$. Recall that $P_1 \geq 1$ always, since the pile is assumed to have at least one 9 (the 9 of hearts). We may also explicitly provide the vector P in parentheses. The pile is said to be *trivial* when it consists only of the 9 of hearts, i.e., $P = [1, 0, 0..., 0]$.

Unless stated otherwise, assume that for the position (A, B), A represents Player A's hand and B represents Player B's hand, and it is Player A's turn to move first. In this paper, I generally draw no distinction between the names of the players. In other words, the position (X, Y) with Player A to move is considered as the same position as (Y, X) with Player B to move.

As a final note of caution, in this paper I may wish to refer to the position (Y, X) with Player B to move first, as above. By contrast, the Maple code accompanying this paper, the player whose hand is given in the first coordinate is *always* assumed to move first. For instance, the position in the Maple code `[[1,1],[0,1]]`specifies that the player who has the hand $[1,1]$ is next to move. This position could be referred to in the paper as $([1,1],[0,1])$ with A to move, or as $([0,1],[1,1])$ with Player B to move.

Each player's objective is to obtain the zero vector through a sequence of moves, after which the game ends.

2 Some basic results

For games such as Pan, it can be very useful to have a way to generate a random position. This is usually done by choosing locations for each of the pieces or cards in play. The rules of the game generally cause this method to generate some positions that are impossible to achieve, but the hope is that there are relatively few of these positions. To this end, in this section, we will investigate a way to generate all of the positions in Pan by over-counting. We will also give an argument that can be used to compute a lower bound for the number of legal positions, and I will show that the lower bound and the upper bound are very good approximations to the number of Pan positions.

Let M_k represent the number of k-Pan positions that are actually attainable from a starting position. M_k can be computed directly for small k. The Maple procedure `generatemoves(k)` generates all the possible positions starting from the set of starting positions for k-Pan. By finding the number of operands (`nops` in Maple) of `generatemoves(k)` for $k = 1, 2, 3, 4$, we may compute M_k, as shown in Table 1. However, for larger k, computing M_k in this way becomes prohibitively time-consuming.

k	`nops(generatemoves(k))`
1	5
2	115
3	2134
4	33232

Table 1: The number of attainable positions in k-Pan for small values of k.

2.1 An upper bound for M_k

We can obtain an upper bound for the number of attainable positions in k-Pan by "over-counting". In chess, this could be done by placing the pieces arbitrarily on the board and counting these positions[4]. This would include positions which are not attainable from the initial starting position; for instance, positions with pawns on the first or last row, or with both kings in check are not attainable via legal moves.

For the game of Pan, every triple (A, B, P) representing an attainable position must have $P_1 = 4 - (A_1 + B_1) \geq 1$, and $P_i = 4 - (A_i + B_i) \geq 0$ for $i = 2, 3, ..., k$. Therefore,

$$\begin{aligned} &A_1 + B_1 \leq 3, \text{ and} \\ &A_i + B_i \leq 4 \text{ for } i = 2, 3, ..., k. \end{aligned} \tag{1}$$

To obtain an upper bound to the number of attainable Pan positions, we seek nonnegative integer solutions to the set of linear inequalities (1). To count the number of solutions to $x_1 + ... + x_m \leq n$, we define x_{m+1} to be $n - (x_1 + ... + x_m)$, so that x_{m+1} is also nonnegative. Now for each solution to the original inequality, there is a corresponding solution to the equation $x_1 + ... + x_m + x_{m+1} = n$. The number of nonnegative integer solutions to the equality $x_1 + ... + x_{m+1} = n$ is $\binom{n+m}{m}$ (I give a combinatorial argument for this fact in Section 2.1.1.) Hence there are $\binom{3+2}{2} = 10$ solutions to the first inequality, and $\binom{4+2}{2} = 15$ solutions to the second inequality.

Since we choose nonnegative solutions for each linear inequality independently, we have that there are at most $10 \cdot 15^{k-1}$ hands in k-Pan.

A Maple procedure was created to construct lists of potential positions corresponding to each solution to equation (1). This is done in the procedure `allpos(k)`, which recursively generates a vector of all potential positions for a game of k-Pan. When $k = 1$, we seek the set of nonnegative solutions to $A_1 + B_1 = n$ for $0 \leq n \leq 3$, which can be done directly by choosing $0 \leq A_1 \leq 3$, and for each such choice, choosing B_1 from 0 to $n - A_1$. For $k \geq 2$, start with the set of solutions to

$$\begin{aligned} &A_1 + B_1 \leq 3, \text{ and} \\ &A_i + B_i \leq 4 \text{ for } i = 2, 3, ..., k-1. \end{aligned} \tag{2}$$

[4]Let us exclude positions including more than the standard number of major pieces, as would be obtained after promoting pawns.

For each solution in this set, we may choose a solution to $A_k + B_k = m$ for $0 \leq m \leq 4$. Similarly to above, this can be done by letting A_k vary from 0 to m and B_k vary from 0 to $m - A_k$. All solutions to (1) are generated in this way.

Note that in Pan, there are positions which are generated by `allpos` but which are not attainable. For instance, the position $[3, 4, 4, 4], [0, 0, 0, 0]$, where Player A is to move, satisfies the conditions (1) but is not attainable. This is because Player B has just won on the previous move, but the pile is trivial, implying that Player B just placed a 9 of hearts on the pile. This is impossible since the hand $\{9\text{h}\}$ is not a starting hand. Other examples include all positions in which Player A has no cards, since the game is assumed to end immediately after a player runs out of cards.

2.1.1 Integer solutions to $x_1 + ... + x_m = n$ and the `randpos` procedure

Above, I asserted that the number of nonnegative integer solutions to

$$x_1 + ... + x_m = n \tag{3}$$

where $m > 0$, is $\binom{n+m-1}{m-1}$. This can be seen by a straightforward combinatorial argument. Start by lining up $n + m - 1$ dots in a row. Choose any $m - 1$ of these dots, and imagine that they represent the symbol '+'. We end up having $n + m - 1 - (m - 1) = n$ dots and $(m - 1)$ '+' signs, with some two '+' signs possibly adjacent. From this, we can read off a solution to the original equation: Each x_i is interpreted to be the number of the i-th (possibly empty) consecutive run of dots in between '+' signs. Furthermore, every solution of nonnegative integers to the equation corresponds to a row of dots and plus signs, and this correspondence is 1-1.

We could have used this argument instead of the one described above for the `allpos` procedure. In fact, this argument is used in the procedure `randpos(k)`. This procedure returns a random component of `allpos(k)`, i.e., a random potential Pan position. A direct way of creating such a procedure would be to simply randomly select a component from `allpos(k)`. However, `allpos(k)` is very slow for large k, so this is not a good approach. We wish to pick a random solution of nonnegative integers to

$$\begin{aligned} A_1 + B_1 + P_1 &= 4, & & \\ A_i + B_i + P_i &= 4, & & i = 2, 3, ..., k \\ P_1 &\geq 1 & & \end{aligned}$$

Notice the condition $P_1 \geq 1$ holds because the pile is always assumed to have the 9 of hearts. We can account for this condition by defining $P_1' = P_1 + 1$ and looking for nonnegative integer solutions to

$$A_1 + B_1 + P_1' = 3, \qquad (4)$$
$$A_i + B_i + P_i = 4, \qquad i = 2, 3, ..., k \qquad (5)$$

Using the combinatorial argument above, we can solve equation (4) by considering a set of $3 + 2 - 1 = 5$ objects, (`randpos` uses the numbers 1 through 5) and randomly choosing two distinct ones, say s and t, to represent the '+' signs. Let A_1 be the number of objects before the first '+' sign ($A_1 = min(s, t) - 1$ in `randpos`) and B_1 be the number of objects between the first two '+' signs ($B_1 = max(s, t) - A_1 - 2$ in `randpos`). P_1 is then determined as $P_1 = 4 - (A_1 + B_1)$. Equation (5) is simlar, except we use $4 + 2 - 1 = 6$ objects, again choosing two of them to represent '+' signs. In this way, `randpos` can generate random potential Pan positions much faster than selecting a random component of `allpos`.

2.1.2 Approximating the average branching factor

The game of Pan may be represented as a *game tree*. The game tree is a directed graph where nodes represent all the different positions of the game, and a directed edge is drawn from A to B if and only if there is a legal move to obtain B from A. When there is such a directed edge from A to B, B is referred to as a *child* of A. The procedures `allpos` and `randpos` can be used to estimate the *average branching factor* for the game of Pan. For any game tree, the branching factor of a node is the number of that node's children. By average branching factor, I mean the average taken over all potential positions, i.e., over all entries of `allpos`. In Maple, this can be done with the use of the procedure `legalmoves(A,B)`. This procedure returns a list of the children of the position (A, B) with Player A to move. The number of distinct children for a given position can therefore be found by `nops(legalmoves(A,B))`. The exact values of the average branching factor are given in Table 2 for $k \leq 4$. For larger values of k, a random sample of 10,000 is taken from `allpos`, without replacement, to estimate the average branching factor. The average branching factor appears to approach approximately 2.1 as k increases.

k	average branching factor
1	.4
2	1.4
3	1.858
4	2.019
5	2.057
6	2.082
7	2.107
8	2.105
10	2.085
100	2.109

Table 2: The estimated average branching factor for k-Pan (a random sample of 10,000 positions used for $k > 4$.

It is worth noting that the average branching factor is different from the average number moves a player may be able to legally make in the course of a game. In fact, the average taken over all positions assumes all positions are equally likely, and therefore that there is no difference between the players hands and the the cards on the pile. In actual play, it is likely that the ranks on the pile will be distributed very differently from those in the hands. For instance, it is rare that the pile will contain cards of many different ranks. Instead, one player typically will try to persuade the other to take low cards by placing a high card on the pile.

A second item to notice is that the average branching factor for Pan is quite small. The average branching factor of chess, for instance, is estimated to be about 35. However, the small average branching factor for Pan does not mean the game is simple to play. In fact, games can be very long. I observed expert games (where both players were rated 1750 and above) on the online game site kurnik.pl. Of the 11 games observed, two lasted beyond 100 plies (50 complete turns), with one game taking 195 plies (97 complete turns). Even with a modest average branching factor, play can be quite nuanced and complex. Also, the assumption that the cards in the players' hands are distributed the same as the cards on the pile would actually cause the average branching factor to be underestimated: when there are cards of many different ranks on the pile, there are very few available moves, since the top card will have a high rank.

In the remainder of Section 2, I will show that the number of potential positions is close to the number of actual Pan positions for small k. This suggests that the numbers in Table 2 are good approximations to the average number of children per position, where the average is taken over all actual Pan positions.

2.2 Towards a lower bound for the number of attainable positions

In this section, I describe how to compute a lower bound for M_k, the number of actual k-Pan positions. The main tool is a claim in this section which shows that certain positions are always attainable from a starting position. We will need the following easy but useful observation– that under mild conditions, the two players can agree to switch turns via a sequence of legal moves:

The Turn-Switching Sequence. *If it's Player A's turn, he has at least 3 cards (labeled X, Y, and Z) and the pile is trivial, the following sequence of moves switches the turns but otherwise does not change the position. Assume, without loss of generality, that X has rank at least as high as Y.*

1. A discards X.
2. B takes X.
3. A discards Y
4. B discards X.
5. A takes X and Y.

A and B now have the same hands as when they started but it is now Player B's turn. Note that the third card Z was useful in step 3, to guarantee that Player A did not prematurely end the game. The power of the turn-switching sequence is that it may always be performed as long as Player A has any three cards. Before we prove the main claim in this section, we will need the following Lemma:

Lemma. *Let B be a k-component vector with nonnegative integer entries, and $\sum_{i=1}^{k} B_i > a \geq 0$. Define F by $F_i = \min(B_i, a)$ for $i = 1, 2, ..., k$ and suppose $\sum_{i=1}^{k} F_i \leq a$. Then there exists t such that*

- *$B_t > a$,*
- *$F_t = a$, and*
- *$B_i = F_i = 0$ for $i \neq t$.*

Proof. Suppose for contradiction that for all i, $B_i \leq a$, so $F_i = \min(B_i, a) = B_i$. We have $a < \sum_i B_i = \sum_i F_i \leq a$, a contradiction. Therefore, there exists t such that $B_t > a$, so that $F_t = a$. If $k = 1$, we are done. Otherwise, suppose that $B_s > 0$ for some $s \neq t$. Then $\sum_i F_i \geq F_t + F_s = a + \min(B_s, a) > a$, a contradiction. Hence $s = t$. We have shown that only B_t is nonzero. Furthermore, for $i \neq t$, we have $F_i = \min(B_i, a) = 0$.

□

Claim. *Assume $k \geq 2$ and let (A, B) be any position with a trivial pile and Player A to move, such that Player A has at least one card. Then there is a sequence of legal moves from a starting position to (A, B) if in the position (A, B), Player B has at least three cards.*

The basic idea behind the proof is that from a starting position, one player 'gives' the other one all the cards of which there are too many in his hand compared to the target position. Then the turn-switching sequence is performed, and the other player does the same to arrive at the target position. The only item to watch out for is that Player B has three cards, so that he may perform the turn-switching sequence. The case where this does not happen is handled by the Lemma in Case 2 of the proof.

Proof. Let (C, D) be the position with $C = [1, 2, ..., 2], D = [2, 2, ..., 2]$ and Player B to move. Find the smallest i_1 such that $B_{i_1} < D_{i_1}$, if one exists. Player B places a card of rank i_1 on the pile, and Player A subsequently takes this card, arriving at a new position (C', D') with Player B to move. Again, find the smallest i_2 such that $B_{i_2} < D_{i_2}$, if one exists. Player B places a card of rank i_2 on the pile, and Player A takes the card, arriving at (C'', D'')

with Player B to move. This process is repeated until we arrive at a position (E, F), where $B_j \geq F_j$ for all j and with Player B to move. Note that for any j, if $B_j \leq 2$ then $F_j = B_j$, and if $B_j > 2$, then $F_j = 2$. In other words, $F_j = \min(B_j, 2)$ for all j.

We now proceed by cases according to the number of cards remaining in Player B's hand.

Case 1: Player B has at least 3 cards in his hand.

In this case, since Player A has at least one card, Player B may perform the turn-switching sequence. Thus, we have arrived at the position (E, F) with Player A to move. Now, since $B_j \geq F_j$ for all j, find a u_1 such that $B_{u_i} > F_{u_1}$ (if none exists, for all j, $B_j = F_j$, so $A_j = E_j$ and so we are done). Now, similar to before, place a card with rank u_1 on the pile, and Player B takes, arriving at a position (E', F') with Player A to move. This process is repeated, but must terminate when we arrive at a position (G, H) with Player A to move, where there does not exist u such that $B_u > H_u$. Hence, $B = H$, $A = G$, as desired.

Case 2: Player B has 2 or fewer cards remaining in his hand.

In this case, $\sum_{j=1}^{k} F_j \leq 2$, so applying the Lemma with $a = 2$ implies that there is a t such that $F_t = 2$, $B_t \geq 3$ and $B_j = F_j = 0$ for $j \neq t$. Since the pile is trivial, $E_t = 1$ if $t = 1$ and $E_t = 2$ if $t \geq 2$. Since $F_t \geq 1$, Player B may place a card of rank t on the pile. Since $E_t \geq 1$, Player A may now place a card of rank t on the pile, and Player B takes both of these cards. It is now Player A's turn and Player B has exactly 3 cards of rank t and no other cards, so if $B_t = 3$, we are done. If $B_t = 4$, then $E_t = 2$ so Player A has one more card of rank t to place on the pile, which Player B takes. Again, we arrive at the desired position (A, B).

□

Now, we can easily extend this claim to work for positions with nontrivial piles, as long as there are "enough cards" in the players' hands.

Corollary. *Let (A, B) be any position with Player A to move and $k \geq 2$. Let a be the number of cards in Player A's hand, b be the number of cards in Player B's hand, and n be the number of cards on the pile (other than the 9 of hearts). Then (A, B) can be attained from a starting position if $\frac{n}{2} + b \geq 3$ and $a \geq 1$ if n is even, and $\frac{n-1}{2} + a \geq 3$ if n is odd.*

Proof. Let n be the number of cards on the pile of (A, B) (other than the 9 of hearts). Enumerate the pile of (A, B) as $x_1, x_2, ..., x_n$ with x_1 being the top card, and x_n as the bottom card (other than the 9 of hearts). Let D be the hand consisting of B together with the cards $x_1, x_3, x_5, ..., x_r$, and C be the hand consisting of A together with $x_2, x_4, x_6, ..., x_s$, with $r = n-1$, $s = n$ if n is even, and $r = n$, $s = n-1$ if n is odd. Let (C, D) be the position with Player B to move next if n is odd, and Player A to move if n is even. Then we may apply the claim to (C, D): If n is even, D has $\frac{n}{2}+b \geq 3$ cards, and C has at least $a \geq 1$ cards. If n is odd, C has exactly $\frac{n-1}{2}+a \geq 3$ cards, and D has at least $\frac{n+1}{2} \geq 1$ cards. Therefore, (C, D) is attainable from some starting position. It is readily checked that (A, B) can be obtained from (C, D) by having players alternately discard the cards $x_n, x_{n-1}, ...x_1$ in order. $\square$

The process described in the proofs above is constructive and can be adapted into an algorithm for constructing a sequence from a starting position to a position satisfying the conditions of the corollary. In the Maple code, the procedure `printpath(A,B)` takes hands for Player A and Player B satisfying the hypotheses of the corollary, and prints the sequence from the starting hand $[1, 2, ..., 2], [2, 2, ..., 2]$ with Player B to move to (A, B) with Player A to move, as described in the proofs above. The procedure returns `'ERROR'` if the hypotheses for the corollary or the claim were not satisfied by the target position (A, B). The reader may check that every move follows from the previous by a legal move, and that no illegal move is reached in the sequence (i.e., if an entry in some vector is negative or that the player to move next has no cards).

Recall that M_k represents the number of k-Pan hands that are attainable from a starting position. The corollary allows us to calculate lower bounds for M_k for $k \geq 2$. This is implemented in the Maple procedure `attainable(V)`. This procedure accepts a vector of positions V as an argument, and returns the vector whose components consist of only those positions attainable from some start position as outlined in the above proofs. This is done by merely checking that the conditions of the corollary hold for each position. The upper and lower bounds for M_k are summarized in Table 3.

Notice that in Table 3 the ratio of the lower bound to the upper bound appears to approach 1 monotonically as k increases, and therefore, M_k appears to be better approximated by these bounds as k increases. The lower bound in particular appears to approximate M_k especially well for small k.

k	upper bound for M_k	M_k	lower bound for M_k	lower bound as a fraction of upper bound
1	10	5	-	-
2	150	115	95	0.633
3	2250	2134	2067	0.919
4	33750	33232	33193	0.983

Table 3: Values of M_k and bound estimates for $k \leq 4$.

For $k = 2, 3, 4$, the full sets of potential positions were used; that is, `nops(attainable(allpos(k)))` was computed. For larger k, `allpos(k)` becomes too large to work with in Maple. The procedure `randsamp(k,n)` generates a random sample of n positions from `allpos(k)`, and was used to estimate the ratio of the lower bound to the upper bound of M_k for $k = 5$ and $k = 6$, where samples consisted of 20,000 positions from `allpos`. This approach led to the ratios of $19928/20000 \approx .9964$ for $k = 5$ and $19976/20000 \approx .9988$ for $k = 6$. Since the exact values for the upper bound are given by the formula $10 \cdot 15^{k-1}$, derived earlier in this section, we can approximate the lower bounds for M_k, and therefore, M_k itself. Using this approach, the number of 5-Pan positions is no more than $10 \cdot 15^4 = 506,250$ and equals approximately $.9964 \cdot 506250 \approx 504,400$. Similarly, for $k = 6$, the random sample approach gives an approximation of $19976/20000 \approx .9986$ for the ratio of the lower bound to the upper bound of M_k. The number of 6-Pan positions is no more than $10 \cdot 15^5 = 7593750$ and equals approximately $.9986 \cdot 7593750 \approx 7,584,000$.

A consequence of these calculations is that potential positions from the `allpos` procedure approximate the behavior of the attainable positions fairly well, especially when $k \geq 3$. This allows us to use these full sets of potential positions instead of the extremely slow `generatemoves` procedure to perform analyses.

3 Finding Winning Positions

3.1 Combinatorial game theory

Pan is a turn-based game with perfect information, and therefore, we may apply the techniques of combinatorial game theory. In this section, we will apply these techniques to arrive at each player's probability of winning the game. It should be noted that the rules of Pan imply that there are no "tied" or "stalemate" positions– that is, when a player cannot make a move, that player has won the game[5]. To see this, suppose it is Player A's turn. Player A must have at least one card, or else the game will have already ended. If the pile is trivial, he may, by Rule 1, place a card on the pile. Otherwise, if the pile is nontrivial, by Rule 2, he may take the topmost cards from the pile. Thus there is at least one legal move at all times.

To summarize, for fixed k, k-Pan is a game with a finite state space and no stalemate positions. We may apply backward induction to find winning and losing positions. Let S be the set of all positions, $|S| = n$, and G_0 be the graph with a node for each position in S and a directed edge between two positions $\mathbf{X}$ and $\mathbf{Y}$ if there is a legal move from $\mathbf{X}$ to $\mathbf{Y}$. We may examine each position and determine if, in that position, either player has won. Label the position W if Player A has won, and L if Player B has won. Call the resulting labeled graph G_1. (Note that most nodes likely remain unlabeled in G_1). Now, repeat the process, except for each node, label the position L if all of its children are labeled W, and label the position W if at least one of its children is labeled L. Call the resulting labeled graph G_2. This process can be iterated until it yields no more new labelings.

The idea behind the labeling technique is that if each player is playing "optimally," he will choose a winning position every time there is such a move available. Thus, the only way a player can lose is if every move he can choose is a losing position.[6]

A natural question to ask is when the iterative process described above terminates. If there are n positions in the state space, there can be no labeled position from which there is a sequence of optimal moves longer than n moves. Hence the process needs at most n steps to terminate. Furthermore, suppose

[5]This does not mean the game cannot end in a draw–in fact, I will show in Section 3.2 that there can be an endless cycle of moves in optimal play when $k \geq 3$.

[6]Note the assumption of optimal play assumes that each player is able to search through to the end of the game tree.

there exists some k for which no new nodes were labeled on the $(k+1)$-th iteration. Then no new nodes will ever be labeled on iterations $k+1$, $k+2$, ... , n. To see this, suppose for contradiction there is some position $\mathbf{X}$ which is labeled on the m-th iteration, $m > k+1$, where m is as small as possible. There exists a child $\mathbf{Y}$ of $\mathbf{X}$ which was labeled on the $(m-1)$-th iteration, and $m-1 > k$. But no nodes were labeled on the $(k+1)$-th iteration, so $m-1 \neq k+1$. Hence, $m > m-1 > k+1$ and there is a position $\mathbf{Y}$ which is labeled on the $(m-1)$-th iteration, a contradiction.

The iterative process described above is applied to Pan in the Maple procedure `winorlose(k,m)`. This procedure returns a table of those k-Pan positions which are labeled after m iterations of the above process. The integers 1 and -1 represent W and L, respectively.

By experimenting with different values of `m`, we can quickly discover values for which `winorlose(k,m)` = `winorlose(k,m+1)` using Maple. The number of iterations until no new labels are discovered is reported in column 2 of Table 4 for $k \leq 4$.

k	longest sequence with optimal play	M_k	no. of positions winning for either player	ratio of no. drawing positions to M_k
1	3	5	5	0
2	15	115	115	0
3	38	2134	2038	.045
4	84	33232	28667	.137

Table 4: The number of winning positions in k-Pan.

These values can be used to determine the number of winning positions in a game of k-Pan as follows. For instance, running the command `[seq(winorlose(4,84)[generatemoves(4)[i]], i = 1..33232)]` would return a vector of 1s, -1s and unassigned variables corresponding to those positions which are winning, losing, and drawing, respectively.[7] The number of occurrences of 1's and -1's may be counted, for instance, with

[7]Note that this code is for illustrative purposes only, and would be very slow unless one had previously computed and assigned a variable to `generatemoves(4)`. Also note the values of 84 and 33232 coming from row 4 of Table 4.

`ListTools[Occurrences]`, and these numbers are reported in the fourth column of Table 4.

Notice that in Table 4, the maximum number of labeled positions is equal to M_k, the number of k-Pan positions, for $k = 1, 2$, but that there are unlabeled positions for $k = 3, 4$. This implies that there are positions that are not winning for either player, as we discuss in the next section.

3.2 Drawing positions

Even when a game does not have “stalemate” positions (unlike chess or tic-tac-toe, for instance) there may still exist positions where neither player can win if they both play optimally. When the state space is finite, the only way this can happen is if in optimal play, the same sequence occurs twice, i.e. a cycle in the state space is traversed in the course of optimal play. This can be seen by considering an arbitrary unlabeled position $\mathbf{X_1}$ of the fully labeled state space. If, by playing optimally, the players arrive at the position $\mathbf{Y}$ which is labeled either L or W, then one player will lose. Therefore, the players must avoid labeled positions in order to draw. Let S be the (finite) subset of unlabeled positions, and assume it has n elements. Since there are no stalemate positions, there is a sequence $\mathbf{X_1}, \mathbf{X_2} ... \mathbf{X_{n+1}}$ of legal moves obtained by optimal play. By the pigeonhole principle, at least two of those positions are identical, and we have traversed a cycle of positions in the state space.

Table 4 above shows that there are drawing positions for $k = 3, 4$. In fact, the proportion of positions which are drawn appears to grow as k increases. As shown in the next section, there even exist starting positions which are drawing for $k = 4$.

3.3 Characterizing starting positions

One of the guiding questions in this essay was whether we can determine the probability that a player can win a randomly dealt starting position if both players play optimally. Recall that a starting position in Pan is created by shuffling the $4k$ cards in play, and dealing $2k$ cards to each player. Immediately after this, the player with the 9 of hearts must discard this card to form the pile. Therefore, the position (A, B) with Player A to move is a starting position if and only if the pile is trivial and $\sum_{i=1}^{k} A_i = 2k$, where

$0 \leq A_1 \leq 3$ and $0 \leq A_i \leq 4$ for $i > 1$.

The Maple procedure `startpos(k)` returns the vector whose components consist of all starting positions for a game of k-Pan by checking each component of `allpos` for the above condition. The number of components in `startpos` is given in the second column of Table 5.

The number of winning, losing, and drawing starting positions are found using `winorlose`, as done in the previous subsection. The difference is that `startpos` is used whereever `generatemoves` was used in the previous subsection. For example, when $k = 2$, there are 4 possible starting positions: $S_1 = ([0,4],[3,0]), S_2 = ([1,3],[2,1]), S_3 = ([2,2],[1,2])$, and $S_4 = ([3,1],[0,3])$. S_1 is clearly winning for Player A, as on the first move, he can place all four tens on the pile, and wins. One would suspect S_2 to also be winning, and indeed, `winorlose(2,15)[[[1,3],[2,1]]]` returns `1`. It can similarly be checked that S_3 is losing and S_4 is winning. Thus exactly three of the four starting positions are winning.

k	total # of starting positions	# winning positions	# losing positions	# drawing positions
1	1	0	0	1
2	4	3	0	1
3	16	11	0	5
4	70	34	13	23

Table 5: The number of winning, losing, and drawing starting positions.

Let us now calculate the probability that Player A can win in optimal play given that he was dealt the 9 of hearts. Note we cannot merely perform arithmetic on the columns of 5, since the starting hands are not equally likely to occur. For instance, in the previous example, there is only one way the cards can be dealt to result in a starting hand of S_1, since Player A holds all four Tens, and Player B holds every 9 but the 9 of hearts. By contrast, there are $\binom{3}{1} \cdot \binom{4}{3} = 12$ ways to choose the hand S_2: there are $\binom{3}{1}$ ways to choose a 9 that is not a 9 of hearts for Player A, and $\binom{4}{3}$ ways to choose 3 tens for Player A. Player B's hand is then determined by these choices. Table 6 summarizes the number of ways to deal each of the 2-Pan starting hands.

The total number of ways to deal the starting hands for $k = 2$ could be

Position	Result for Player A	# ways to deal
[0,4],[3,0]	win	$\binom{3}{0}\binom{4}{4} = 1$
[1,3],[3,1]	win	$\binom{3}{1}\binom{4}{3} = 12$
[2,2],[1,2]	lose	$\binom{3}{2}\binom{4}{2} = 18$
[3,1],[0,3]	win	$\binom{3}{3}\binom{4}{1} = 4$

Table 6: Determining the number of winning and losing deals with $k = 2$.

found by computing

$$\sum_{i=0}^{3}\binom{3}{i}\binom{4}{4-i}$$
$$=\sum_{i=0}^{3}\binom{3}{i}\binom{4}{i}.$$

However, this must also be the number of ways to select 4 cards from 7 (all cards in play except the 9 of hearts), so this equals $\binom{7}{4} = 35$. Therefore, the probability of being dealt a winning hand given one was dealt the 9 of hearts is $\frac{18}{35} \approx .486$. The probability a player was dealt a losing starting hand given he moved first is approximately $1 - .486 = .514$.

In general, given that Player A has a starting hand $[A_1, A_2, ..., A_k]$, the number of ways this hand could be dealt is

$$\binom{3}{A_1}\binom{4}{4-A_2}\cdots\binom{4}{4-A_k}$$
$$=\binom{3}{A_1}\binom{4}{A_2}\cdots\binom{4}{A_k}.$$

The total number of ways to deal a starting hand is $\binom{4k-1}{2k}$. Therefore, we may calculate the probability that a random starting position is winning. This is implemented in the maple procedure `probofwinning(k,Ahas9h)`. This procedure takes k as an argument, and returns the probability that Player A wins. If `Ahas9h` = `1`, the procedure assumes that Player A was dealt the 9 of hearts, and therefore was the first player to move. Otherwise, the procedure assumes Player B was dealt the 9 of hearts and that Player A was the second player to move. From these results, the probability of drawing can be calculated as $1 - \mathrm{P}(\text{A wins}) - \mathrm{P}(\text{A loses})$. The results are summarized for $k \leq 4$ in Table 7.

k	Prob. of winning given that player had 9h	Prob. of losing given that player had 9h	Prob. of drawing given player had 9h
1	1	0	0
2	.514	.486	0
3	.405	.595	0
4	.216	.432	.352

Table 7: The probabilities of being dealt winning, losing, and drawing hands given a random k-Pan starting position.

It is interesting that the player without the 9 of hearts appears to have a greater probability of winning for $k = 3, 4$. In other words, player with the advantage in k-Pan is with the player who moves *second*, at least for these k.

For $k = 4$, the odds are stacked against the player who starts with the 9 of hearts: the probability that the player with the 9 of hearts wins is less than half of the probability the other player wins. Computing power prevented this analysis from checking whether this advantage holds for larger values of k. At first, the advantage uncovered here is surprising, especially given that the goal of the game is to discard all of one's cards. It would seem that if anything, the player that is first to move would have a 'head start' in discarding cards.

However, on closer examination, there are reasons why the player to move first might actually be at a disadvantage for certain $k \geq 2$. For any starting position in k-Pan, consider only the 9s. Also, let's assume that Player A has the 9 of hearts and has not yet placed it, so that there are four 9s among the two players. There are only four possibilities: Player A has four Nines and Player B none, Player A has three Nines and Player B one, etc. Note we cannot have Player A having no Nines and Player B four, since Player A was assumed to have at least the 9 of hearts. When Player A has four 9s, he is in a tough position indeed: after placing the 9 of hearts, Player B will place a 10 (or higher) and Player A is already in trouble since he is unable to place his three remaining 9s.

Moreover, the positions in which one player has exactly three 9s play very differently. If Player B has three 9s, he may place them all on the pile at once immediately. If Player A has three 9s, however, once again, he will not have the chance to play them all immediately: Player B can place his 9 on the pile, Player A can place another, and Player B will now place a 10 or higher on the pile, leaving Player A with a 9.

By these considerations, the distribution of the Nines between the two players appears to favor the player without the 9 of hearts, since the 9 of hearts has special significance. Note also that this argument does not take into account the likelihood of being dealt the different hands. In fact, this factor is the why the odds of receiving a winning hand for $k = 2$ are almost even, despite the fact that three of the four distinct starting hands are winning for the player without the 9 of hearts. For all other k, it appears that this advantage is real.

4 Minimax and heuristic functions

The backwards induction method described in Section 3 can be conceptualized in a different way. Let a position that is winning for Player A always be labeled with +1 and a position winning for Player B labeled with -1. Start by labeling those states which are known to be winning for Player A or B, as before. On each of the proceeding iterations, given an unlabeled position whose children are all labeled, assign that position a +1 if it is Player As turn and there exists a child labeled as +1. If it is Player As turn and all children are labeled -1, label it -1. If it is Player Bs turn, label it a -1 if there exists a child labeled -1, and otherwise label it +1. There are two points to make. First, this method is equivalent to the approach taken in the combinatorial game theory approach described in Section 3. Second, at each step, Player A is taking the *maximum* value of the winning positions, and Player B is taking the *minimum* value of the winning positions.

This second perspective makes some intuitive sense- each player is choosing the 'best' possible position from a selection of choices, and can be extended as follows. Assign $+\infty$ and $-\infty$ to winning positions for Players A and B, and for all other positions, evaluate them using a formula or algorithm that tries to approximate how much that position favors Player A. Such a function is called the *heuristic function.* A well-known example of a simple heuristic function occurs in chess, whereby pieces on the board are assigned values, (e.g., a queen is worth 8, a rook is worth 5, a bishop is worth 3, etc.). Every piece that Player A has adds points to the heuristic score, and every piece that Player B has deducts points from the heuristic score.

An advantage of using this 'minimax' algorithm with heuristic functions is that it need not be carried out until the very end of the game tree, i.e., until a value of $\pm\infty$ is reached. Rather, we can specify through how many levels, or 'plies' we would like to carry out the algorithm and work *forwards.* The greater depth we choose, the greater analytical power we obtain for a given position.

Another way to increase analytical power is to increase the accuracy of the heuristic function, with the drawback that more computing time is devoted to evaluating the heuristic function. In practice, fast heuristic functions are favored over extremely accurate ones, as this allows the algorithm to search deeply in the game tree. For the game of Pan, I will introduce a heuristic function, and show that it is quite fast and accurate.

The heuristic function for Pan used in this essay is implemented in the procedure `evalscore(A,B)`. This procedure evaluates the position (A, B), with Player A to move first, and returns a heuristic value. The formula used to compute this value is based on the following two principles, given a Pan position:

1. All else being equal, the player with more high-ranking cards and fewer low-ranking cards generally has an advantage.

2. All else being equal, the player with fewer *groups* of cards generally has an advantage. Each card in the hand counts as its own group if there are 2 or fewer of that card in the hand, or if there are 3 of fewer of that card and it is not a 9. Each set of three Nines (not the 9 of hearts) and four of any other rank also counts as a single group.

To illustrate Principle 1, consider a game of 6-Pan. Aces are the highest ranking cards in this game, and the player with many or all of the aces generally has an advantage. Supposing that Player A has all the Aces, he may 'bully' B into taking cards on the pile. For instance, if the pile is $[4, 0, 0, 0, 0, 0]$, Player A may discard an Ace, and B is forced to take the Ace along with two Nines. On the next move, A can discard a card greater than a 9, thus forcing B to 'dig out' in order to place his two Nines. We can also see from this example that having too many Nines makes things difficult, and requires high-ranking cards to help the player's position. For instance, in order for B to 'dig out' to place a 9, he must cause A to take some cards and expose a 9 as the top card on the pile. He may try to do this by discarding an Ace. Now A has a choice of whether to take the Ace and a 9 (and the card he just discarded) or discard another Ace. If he decides to discard another Ace, B will likely take so as to have two Aces, and so on.

The second principle ensures that the algorithm also places value on the number of turns it would take to discard all the cards in the hand. This includes the idea that the 3-Pan hand $[1,4,1]$ is better than $[1,3,1]$. This takes advantage of the third rule of Pan, allowing complete sets of a given rank to be discarded in one move. This principle also helps "smooth" the behavior of heuristic function for positions just outside of the given depth. Without this principle, the algorithm would favor hands that "hoard" high-ranking cards. For instance, we would want the algorithm to prefer the hand $[0,0,1]$ over $[0,0,3]$.

In `evalscore`, I implement each of these principles by computing a separate score for each of them. The final score is the sum of these two numbers. For Principle 1, I compute a rank-weighted average of all the cards in the player's hand minus the cards in the opponent's hand. That is, given the position (A, B) with A to move, define

$$Score_1(A,B) = \sum_{i=1}^{k}(A_i - B_i)(i - \frac{k+1}{2}).$$

Note the multipliers $(i - \frac{k+1}{2})$ which are just the numbers $1, 2, 3, ...k$ translated to be centered around zero.

For the component of the score addressing Principle 2, let N_A represent the total number of groups of cards Player A has, and N_B be the total number of groups of cards Player B has. Define

$$Score_2(A,B) = \frac{1}{2}(N_B - N_A).$$

Finally, define the heuristic function

$$f(A,B) = Score_1(A,B) + Score_2(A,B).$$

The Maple procedure `evalscore(A,B)` returns $f(A, B)$. Note the scalar $\frac{1}{2}$ in $Score_2$, indicating that of the two scores, $Score_1$ has more influence over the final heuristic value.

Note that both $Score_1$ and $Score_2$ are *anti-symmetric* in the sense that

$$Score_i(B, A) = -Score_i(A, B)$$

and therefore

$$\begin{aligned} f(B, A) &= Score_1(B, A) + Score_2(B, A) \\ &= -Score_1(A, B) - Score_2(A, B) \\ &= -(Score_1(A, B) + Score_2(A, B)) \\ &= -f(A, B) \end{aligned}$$

so f is also anti-symmetric.

This property of f makes intuitive sense: if Players A and B switch hands, the magnitude of the advantage one player has over the other should stay the same, while the sign should change. Furthermore, if Players A and B both have the same hand with a large number of cards, we would expect that the relative advantage one player has over the other would be negligible. In other words, we would expect $f(A, A)$ to be close to zero. In fact, as with any anti-symmetric function,

$$\begin{aligned} 0 &= f(A, A) - f(A, A) \\ &= f(A, A) + f(A, A) \\ &= 2f(A, A) \end{aligned}$$

so that $f(A, A) = 0$.

Note there are certainly positions where both players have the same hand but the position is winning for Player A, for instance, the 7-Pan position $[0, 0, 0, 0, 0, 0, 2], [0, 0, 0, 0, 0, 0, 2]$. However, the idea is that the heuristic function approximate a player's advantage for a great number of different positions to avoid searching deeply in the game tree. There will always be cases in which the heuristic function is 'wrong', but we wish to obtain a fast heuristic function that is reasonably accurate for most positions.

As a check for the reasonableness of the heuristic function, I performed an analysis using a random sample of 2000 4-Pan hands. For each position, when the product of the value of the heuristic function and the label from `winorlose` was negative, this was coded as a *miss*. For instance, when the heuristic function returned a negative value and `winorlose` labeled the position as ‘1’, this was coded as a miss. When the product of the result of the heuristic function and the label from `winorlose` was positive, this was coded as a *hit*. The result of the analysis is reported in Table 8.

# hits	# misses	% accuracy
1362	361	79.0%

Table 8: The accuracy of the heuristic function f on a random sample of 2000 4-Pan hands.

The result is that the heuristic function made the ‘correct’ determination 79.0% of the time. This provides confidence that the heuristic function is accurate enough to be used in the minimax algorithm. Furthermore, the heuristic function is very fast, since it uses only a few basic arithmetic operations. For instance, a random 5000-Pan position, consisting of two 5000-component vectors, takes 0.016 seconds to evaluate, according to Maple.

The recursive procedure `minmaxfinal(A,B,depth, maximizingPlayer)` implements the minimax algorithm for Pan in Maple, using the heuristic function f discussed above. When `maximizingPlayer` is set to `true`, Player A is assumed to be the maximizing player, and Player B the minimizing player. The result is a vector with two components. The first component is minimax score for the position, and the second component is the child of a position which achieves that score.

5 Refining and applying minimax to Pan

5.1 Alpha-beta pruning

Although the `minmaxfinal` procedure does work for small k, it is quite slow. For instance, the following is a typical computation in Maple, evaluating a 5-Pan position to a depth of 10:

```
settime := time():
minmaxfinal(x, 10, true);
time()-settime;
```

```
              [3/2, [[1, 2, 2, 3, 1], [1, 2, 2, 1, 3]]]
                             128.982
```

The fact that this computation took over two minutes makes the procedure impractical for use with higher values of k; there is almost no benefit over the combinatorial game theory approach. Fortunately, there are various ways of making the minimax algorithm more efficient. One such way used in this paper is *alpha-beta pruning.* In this method, branches of the game tree which are logically impossible to influence the result of the minimax algorithm are discarded, thereby increasing the efficiency of the algorithm.

The recursive Maple procedure `alphabeta(A,B,alpha,beta,maximizingPlayer)` implements the alpha-beta pruning algorithm for the Pan position (A, B). Initial values for alpha and beta should be `-infinity` and `infinity`, respectively, and maximizingPlayer should be set to `true`.

The speed improvements gained by alpha-beta are dramatic. For instance, with the above 5-Pan position, the following computation takes just over 9 seconds:

```
settime := time():
alphabeta(x, 10, -infinity, infinity, true);
time()-settime;
```

```
              [3/2, [[1, 2, 2, 3, 1], [1, 2, 2, 1, 3]]]
                              9.173
```

In fact, the speed savings are highly dependent on the order in which the moves are examined. If promising moves are examined before worse moves, the alpha-beta cutoffs can be triggered early. In Pan, this can be done by first considering the move which *discards the card(s) with the smallest possible rank.* This includes placing a full group of 3 or 4 cards on the pile. Then, discarding progressively higher rank cards are considered. Lastly, taking at most 3 cards from the pile is considered. The procedure `legalmoves(A,B)` lists legal moves in this way. When this adjustment is made to the ordering, the above computation was done in 1.794 seconds!

The latest version of the procedure `alphabeta` already includes these modifications to the order of moves, and hence enjoys significant speed improvement to a random move ordering. The reader can check the running time of `minmaxfinal` versus `alphabetapan` for a random position using `randpos`.

I will now show that the procedure `alphabeta` using the `evalscore` procedure for the heuristic function produces reasonable results. In a similar way that the heuristic function was checked for reasonableness above, the result of `alphabeta` can be compared with the label of the position in `winorlose`. Whenever the sign of the two values agreed, this was coded as a *hit*, and when the sign disagreed, this was coded as a *miss*. This was done for a random sample of 2000 4-Pan positions, the results of which are summarized in Table 9.

depth	# hits	# misses	% accuracy
2	1513	210	87.8%
6	1555	168	90.2%
10	1599	124	92.8%
14	1623	100	94.2%

Table 9: The accuracy of the alpha-beta pruned minimax algorithm on a random sample of 2000 4-Pan hands.

Note that all of the percentages in Table 9 are above 79.0%. This makes sense since the minimax algorithm is expected to improve upon the heuristic function alone. In general, the accuracy of the algorithm improves steadily as the depth increases.

A final check for the alpha-beta pruned minimax procedure is presented in Table 10. In this check, 2000 random 4-Pan hands were used to analyze the suggested moves provided by the second component of the vector returned by `alphabeta`. Recall that this entry represented a position for which the minimax algorithm achieved the value returned by the minimax algorithm, and therefore represents a move that should in some sense be 'best'. The move suggested by the procedure was coded as a 'good' move whenever `winorlose` labeled the position (A, B) as winning for Player A and the `winorlose` also labeled the suggested move as winning for Player A. Similarly, a suggested move was coded as 'bad' whenever `winorlose` labeled the position (A, B) as winning for A but the labeled the suggested move as winning for B.

depth	# good move suggestions	# bad move suggestions	proportion of good moves
2	1016	10	.990
8	1037	3	.997
12	1042	0	1

Table 10: The proportion of 'good' move suggestions for the alpha-beta pruned minimax algorithm on a random sample of 2000 4-Pan hands.

The alpha-beta pruned minimax algorithm allows a human player to play k-Pan with an AI player for reasonably small k. This is implemented in the Maple procedure `alphabetapan(A,B,depth,[advisor])`. In this procedure, the user provides a position (A, B) representing the starting position for the game, with the user to move first. `depth` represents the depth used in the alpha-beta pruned minimax algorithm, and the optional `advisor` value may be set to 1 to get move suggestions from the game engine. Below is an example output once the procedure is run:

```
> alphabetapan([1, 3, 2], [2, 1, 2], 14);
                         "Pile:", [1, 0, 0]
"Choose a legal move:", [0, 3, 2], [1, 2, 2], [1, 3, 1], "undo"
```

The user can enter any hand from the legal moves provided in the output. Note that only a single hand may be entered in the dialog box. For instance, entering $[1, 2, 2]$ represents choosing the position $[1, 2, 2], [2, 1, 2]$ with Player B to move. Note also that at any time, the user may undo the last move chosen by entering "undo" (with the quotation marks). In practice, a depth of 14 typically results in perfect play for $k \leq 3$ and near-perfect play for $k = 4$ while keeping the run-time for the procedure at acceptable speeds for $k \leq 4$.

5.2 Deciding when to take from the pile

The procedures developed in this paper can be used to address the third guiding question presented in the Introduction. That is, whether computational techniques can be of any guidance in deciding when a player should take the top three cards from the pile. To examine this, a random sample of 4000 4-Pan positions was selected, and of these, only the positions with the pile having more than three cards were selected. `alphabeta` was run on these positions with a depth of 8. Recall from section 5 that with a depth of 8, `alphabeta` suggested a 'good' move 99.7% of the time in 4-Pan. The suggested moves were then analyzed to see which ones the procedure had recommended taking three cards from the pile. The results of this analysis can be found in Tables 11 and 12 and indicate that of the 3291 positions with 4 cards or more on the pile, the `alphabeta` algorithm recommended the player take for 2344 of these positions, or about 71% of the time.

top 3 cards of pile	# positions occurring	# positions where take is best	ratio	rank sum
$[0, 0, 0, 3]$	759	724	.954	12
$[0, 0, 1, 2]$	475	419	.882	11
$[0, 1, 0, 2]$	169	145	.858	10
$[0, 0, 2, 1]$	438	353	.806	10
$[1, 0, 0, 2]$	55	44	.8	9
$[0, 1, 1, 1]$	192	148	.771	9
$[1, 1, 0, 1]$	54	39	.722	7
$[0, 2, 0, 1]$	152	100	.658	8
$[0, 0, 3, 0]$	239	153	.640	9
$[1, 0, 1, 1]$	52	33	.635	8

Table 11: The relative strengths of the top three pile cards in 4-Pan.

This figure is further broken down according to the top three cards of the pile. For instance, row 3 of Table 11 indicates that out of the 438 positions examined having two Jacks and a Queen on the top of the pile, taking these cards was the best move about 81% of the time, as judged by the algorithm. Thus, the top three cards can be ranked by relative strength. This can serve as a rough guide for when to take from the pile. In the fifth column, the sum of the ranks of the top the cards is reported. In the previous example, the rank sum is 3 + 3 + 4 = 10 (two Jacks and a Queen). Note the general decreasing trend in the rank sum of the top three cards as the ratio decreases. There are several exceptions, including $[0, 0, 3, 0]$ which was recommended to be taken a good deal less often than other piles with the same rank sum. The remaining possibilities for the top three cards of the pile are reported in Table 12. Note the large jump in the proportion for those positions where a take is not the best move the majority of the time to those where a take is the best move the majority of the time.

Tables 11 and 12 suggest that the decision about whether or not to take can often be made on the basis of the values of the top three cards of the pile alone. This is intuitive because taking two Nines and a Jack is going to seldom seem like a good idea no matter what the cards the player may have in the hand. The deflated values of some of top three cards can also be explained. For instance, taking three Jacks will result in the player having exactly three Jacks on the hand about half the time. By the second principle in Section 4, this is undesirable since it will take that player 3 extra turns to discard all of his cards, unless he manages to obtain the fourth Jack, which he is unlikely to receive 'for free'.

These results also suggest that the heuristic technique of evaluating the rank sums of the top three cards of the pile could be used for any k-Pan game. For a game of k-Pan, the rank sums vary from 3 to $3k$. The midpoint of the rank sums occurs at $\frac{3(k+1)}{2}$. Note that the break in the rank sums between Table 11 to Table 12 occurs at roughly around 7 or 8 and $\frac{3(4+1)}{2} = 7.5$.

top 3 cards of pile	# positions occurring	# positions where take is best	ratio	rank sum
$[0, 1, 2, 0]$	171	78	.456	8
$[2, 0, 0, 1]$	31	13	.419	6
$[0, 2, 1, 0]$	143	54	.378	5
$[1, 0, 2, 0]$	42	13	.310	7
$[1, 1, 1, 0]$	66	17	.258	6
$[0, 3, 0, 0]$	85	7	.082	6
$[1, 2, 0, 0]$	68	2	.029	5
$[2, 1, 0, 0]$	40	1	.025	4
$[2, 0, 1, 0]$	46	1	.022	5
$[3, 0, 0, 0]$	14	0	0	3

Table 12: The relative strengths of the top three pile cards in 4-Pan. (cont.)

Note that the relatively small depth used in the procedure `alphabeta` causes some limitations to the analysis. When one player has a large advantage, the procedure may falsely identify a move as 'best' even when it is not. For instance, for the position $[0, 3, 0, 4], [2, 1, 2, 0] (P = [2, 0, 2, 0])$, the correct move is to place a Queen. But with depth = 8, the computer evaluates taking to have the higher minimax score. The fact is that both moves stay winning for Player A, and so in this sense are both 'good', although one unnecessarily lengthens the game. In general, a higher depth would be desirable, as it may take many moves to evaluate 4-Pan positions. The trade-off, of course, is that the computing time can quickly become prohibitive in this type of analysis.

6 Other Pan-like games

It is difficult to ignore the seemingly arbitrary nature of some of the rules of Pan. For instance, why take at most three cards in rule 2? One might consider defining the more general (k, j)-Pan, played with k ranks, allowing to take j cards or the number of cards on the pile other than the 9 of hearts. However, there is some sense in which $j = 3$ is the most natural choice. When $j = 1$, the game is uninteresting since there is never any way to 'dig out' in order to place lower cards in the hand. Once a player is unable to place a low card, he will never be allowed the opportunity again. When $j = 2$, drawing is easy. Supposing a game with 6 ranks (i.e., a game of (6,2)-Pan) and Player A wishes to force B to take a 9 on the pile, he may play an Ace. Regardless of whether B has an Ace, he may simply take the Ace and the 9 and, after A places the card he wishes, discard the Ace on the pile. A and B may be forced to go back and forth like this, neither one wanting to give up another Ace, and the game ends in a draw. Thus, $j = 3$ is the smallest j for which the game is interesting.

Other rules, however, do seem to be more arbitrary. It might be interesting to disallow placing more than one card at a time (i.e., removing rule 3). Note that even if this is done, the claim and corollary in Section 2 would still apply since they do not rely on rule 3. Also, the game need not be played with 4 suits, but instead (with some imagination) n suits. Rule 3 could be modified to allow any full set of cards of any particular rank could be discarded at any time, if not removed entirely.

Finally, the 9 of hearts need not have a special status at all. There is no real reason why the 9 of hearts should not be taken in the hand. Though in practice, this card serves to decide who goes first, that could be decided by other means, for instance, who dealt the cards, a flip of a coin, etc. Alternatively, all luck could be removed from the game by using the same starting position every time, such as $[2, 2, 2, 2, 2, 2], [2, 2, 2, 2, 2, 2]$. These variations are outside the scope of this essay, but could serve as a basis for further study.

7 Acknowledgments

I would like to thank my wife, Aneta, for introducing me to the game of Pan and for supporting me emotionally throughout my studies. I would also like to thank my father-in-law, Krzysztof Biesiadecki, for playing Pan with me for hours and beating me repeatedly. Finally, I thank my advisor, Dr. Doron Zeilberger, as well as all the professors on my committee: Dr. János Komlós, Dr. Michael Saks, and Dr. Robert Wilson. I would also like to thank all of my math professors, who patiently taught me and encouraged me to learn more mathematics. Finally, a special thanks to my friends, Kellen Myers and Michael Solway, for their helpful feedback on this essay.