Technische Universiteit Delft
Faculteit Elektrotechniek, Wiskunde en Informatica
Delft Institute of Applied Mathematics

When is a deck of cards well shuffled?

(Nederlandse titel: Wanneer is een pak kaarten goed geschud?)

Verslag ten behoeve van het
Delft Institute of Applied Mathematics
als onderdeel ter verkrijging

van de graad van

BACHELOR OF SCIENCE
in
TECHNISCHE WISKUNDE

door

RICARDO TEBBENS

Delft, Nederland
Augustus 2018

TU Delft Delft University of Technology

BSc verslag TECHNISCHE WISKUNDE

"When is a deck of cards well shuffled?"

(Nederlandse titel: "Wanneer is een pak kaarten goed geschud?)"

RICARDO TEBBENS

Technische Universiteit Delft

Begeleider

Dr. M.T. Joosten

Overige commissieleden

Dr. B. van den Dries

Prof. Dr.ir. M.C. Veraar

Augustus, 2018

Delft

Abstract

When is a deck of cards shuffled good enough? We have to perform seven Riffle Shuffles to randomize a deck of 52 cards. The mathematics used to calculate this, has some strong connections with permutations, rising sequences and the L_1 metric: the variation distance. If we combine these factors, we can get an expression of how good a way of shuffling is in randomizing a deck. We say a deck is randomized, when every possible order of the cards is equally likely. This gives us the cut-off result of seven shuffles. Furthermore, this gives us a window to look at other ways of shuffling, some even used in casinos. It turns out that some of these methods are not randomizing a deck enough. We can also use Markov chains in order to see how we randomize cards by "washing" them over a table.

Contents

1 Introduction

Imagine you are playing a card game with friends, family or a complete stranger. If you are on the winning side, you have nothing to worry about. However, if you are on the losing side, you always get the question if the cards have been shuffled well enough. But when is a deck of cards really shuffled enough? This question was answer by Dave Bayer and Persi Diaconis in 1992 [1]. They came up with the answer that shuffling seven times using a so called Riffle Shuffle should randomize a deck of cards. The result became well known after it even appeared in the *New York Times* [5]. But how did they come up with this result? And how can we use mathematics to solve something like card shuffling, which, by the looks of it, is far away from mathematics. In this report we will take a look at the analysis of how we can conclude that we need to shuffle a deck of cards seven times before it is randomized. To do this, we have to find a way to define card shuffling in a mathematical manner and find a way to express when a deck of cards is well randomized. Furthermore, we will look at some different methods of shuffling, some even used in places like casinos, and try to figure out if it is randomizing enough. We will do this using simulations, or in some cases using Markov chains.

2 Why do we need to shuffle seven times?

In this part, we will make a reconstruction of a paper written by Brad Mann [2]. It focuses mostly on the question how many times we should shuffle a deck of cards and why the answer turns out to be seven.

2.1 Do we really need to shuffle more than once?

Before we start with the actual mathematical modelling and analysis of this problem, it is good to get an intuition that we really have to shuffle multiple times. Even though shuffling a deck of cards only once will not fully randomize it, we would expect at least it should be difficult to find clear patterns. To check this feeling, we set up a little experiment. We get a deck of cards and put it in new deck order. This is the order the cards are in when you open the pack for the very first time. This is shown in Figure 1. We shuffle this deck one time, using a Riffle Shuffle. This means we cut the deck in two parts and riffle these together. Further explanation will follow in Section 2.3.

Figure 1: Our deck in new deck order.

The results we get from the shuffle is probably not what we expected or hoped. In Figure 2, we see that the red cards are separated from the black cards. This shows that our deck is not well shuffled at all, so it should boost the intuition that we need to shuffle multiple times.

Figure 2: Our deck after one shuffle.

2.2 What is a shuffle?

In this subsection, we will see how to define a deck of cards and rearranging cards, in a mathematical way. Furthermore, we will see some simple examples of shuffling and how we define them mathematically. Before we do this, we need a change in notation. First, we will talk about a deck with n cards. This is a more general approach. Further, it is difficult to keep talking about specific cards, like ace of clubs, seven of diamonds ect. From this point on we will label each card of the deck with the integers from 1 to n. So before a deck is shuffled, its order always is $123\cdots n$. This is what we will call the natural order. If a deck is reversed completely, it would look like $n\cdots 321$.

2.2.1 Permutations

Now we will have to describe the shuffling in a mathematical manor, and our new notation for a deck will help us. If we want to change $123\cdots n$ to $n\cdots 321$, we are actually mapping the set of integers, between 1 and n, to itself. This means we can see a shuffle as a *permutation*. A permutation is a bijection from one set to itself. This means that a permutation sends a single element from a set to another element from that same set. This can also be the same element, in this case we call the permutation the *identity*. The set we will use is S_n, which stands for all the possible orderings of n numbers. We write a single permutation as π. Let us look at an example. We take $n = 5$ and let the permutation π_1 be the reverse of the deck. The result will look like

i	1	2	3	4	5
$\pi_1(i)$	5	4	3	2	1

Even though giving a table like is clear and easy to visualize, it is also a lot of work and takes quite some space. Therefor it is easier to write just the second line of the table, assuming the first line is the natural order. To show the difference between a permutation and the order of a deck, we use brackets in for the permutations. So we would write $\pi_1 = [54321]$.

Before we proceed, it is important to point out the distinction between the ordering of a deck and a permutation. When we talk about the ordering of a deck, we mean the specific order the cards are in. A permutation only specifies a rearrangement. Let us take for example our permutation $\pi_1 = [54321]$. This changes the ordering 12345 to 54321, but also 25431 to 13452.

We now know shuffling can be seen as a permutation π on a deck of cards. But since we are trying to find out what happens when you shuffle multiple times, the question arises what happens if we follow it by some other permutation τ. If we take a deck ordering i, that would mean i changes to $\pi(i)$, followed by applying τ, changing it to

$\tau(\pi(i))$. We will call this a composition and write it as $\tau \circ \pi$, which is pronounced as τ *after* π. Note that we write the τ first, even though we apply π before. For example, we can take π_1 defined as before and we define $\pi_2 = [23451]$, so we basically move the first card to the back. If we apply π_2 first, followed by π_1, we can compute $\pi_1 \circ \pi_2$ as follows.

i	1	2	3	4	5
$\pi_2(i)$	2	3	4	5	1
$\pi_1 \circ \pi_2(i)$	1	5	4	3	2

2.2.2 Shuffles

We now know how to define a permutation and what it does to the ordering of a deck. But what if we are not sure which permutation we are using? This sounds odd, but this in the essence what a method of shuffling is. We can define it as a probability density on S_n, where each permutation is considered. In other words, we give each permutation a certain fixed probability of happening. It is trivial that all these probabilities have to sum up to one. Let us look at a very simple example: the top-in shuffle. We take the top card off the deck and reinsert it in the deck at a totally random position. This means it is even possible to put it back on top or on the bottom. The probability is uniform over all the possibilities, which implies that every option has a chance of $1/n$ of occurring. Note that this only counts for the permutations that can be perform. Let us take an example with $n = 3$. We can see that for example the permutation [321] is not possible with only one top-in shuffle, so this will have probability 0. The other probabilities are displayed as follows

permutations	[123]	[213]	[231]	[132]	[312]	[321]
probability	1/3	1/3	1/3	0	0	0

.

Note that with this definition of a shuffle, shuffling multiple times leads to a random walk on S_n. Let us say we look at a way of shuffling Q, this means each permutation π is given a probability $Q(\pi)$ of happening. Our starting point will be the identity of S_n, the trivial permutation that leaves every card at the same position. Now we take a step in the random walk, meaning that we choose a permutation π_1, according to the probabilities given by Q. Now we change arrange the deck as directed by the permutation π_1. For step two, we choose a permutation π_2 according to the probabilities given by Q, and we arrange the deck directed by it. In Section 2.2.1, we saw that this tells us that we used the permutation $\pi_2 \circ \pi_1$ on the deck we started with.

Now we are interested what density $\pi_2 \circ \pi_1$ has. We call this density $Q^{(2)}$. Note that the choices π_2 and π_1 are independent from each other, meaning that the probability of

first choosing π_1, followed by π_2, is equal to the product $Q(\pi_1) \cdot Q(\pi_2)$. We can say this for any particular permutation π, so $Q^{(2)}(\pi)$ is given by the sum of $Q(\pi_1) \cdot Q(\pi_2)$ over all pairs π_1, π_2 such that $\pi = \pi_2 \circ \pi_1$. We have to use this sum, because in general there are multiple ways of choosing π_1 and π_2 to get the same $\pi = \pi_1 \circ \pi_2$. This way of combining Q with itself, we call *convolution* and we write $Q * Q$:

$$Q^{(2)}(\pi) = Q * Q(\pi) = \sum_{\pi_1 \circ \pi_2 = \pi} Q(\pi_1)Q(\pi_2) = \sum_{\pi_1} Q(\pi_1)Q(\pi_1^{-1} \circ \pi). \tag{1}$$

The last equality follows form the fact that the equality $\pi_1 \circ \pi_2 = \pi$ is equivalent to $\pi_2 = \pi_1^{-1} \circ \pi$, and we substitute π_2 with this expression. With the term π_1^{-1} we mean the inverse of π_1, meaning that if you first use π_1, followed by the inverse, we end up doing nothing to the deck. So $\pi_1 \circ \pi_1^{-1}$ and $\pi_1^{-1} \circ \pi_1$ are equal to the identity permutation. As an example, let us take the permutation [536241]. If we want to find an inverse for this, we have to find a permutation that can change the ordering 536241 to the natural order 123456. Finding this permutation, is fairly easy. The ith number in the inverse should be the position of i in [536241]. Take for example $i = 1$. This is in position 6 in the permutation, so the inverse should start with 6. If we do this for all the numbers, we get that the inverse is [642513].

Now we have an expression for the probability density on S_n after two steps in the random walk. In this case, in both steps we use the same way of shuffling Q. If we look at it more generally, we can take two possibly different ways of shuffling Q_1 and Q_2. From (1), we can now see that the density is given by:

$$Q_1 * Q_2(\pi) = \sum_{\pi_1 \circ \pi_2 = \pi} Q_1(\pi_1)Q_2(\pi_2) = \sum_{\pi_1} Q_1(\pi_1)Q_2(\pi_1^{-1} \circ \pi).$$

We have looked at the case of shuffling twice, and we can expand this to an arbitrary number k, with for each step i we have a density on S_n given by Q_i. If we combine all of these, we end up with the convolution $Q_1 * Q_2 * \cdots * Q_k$. This means that first shuffling with Q_1, then Q_2 up through Q_k, is equal to shuffling with the shuffle specified by $Q_1 * Q_2 * \cdots * Q_k$. From this we can conclude that shuffling multiple times is the same as shuffling only once after convoluting the densities. The convolution of these k densities is very important for us, since it gives us the density after shuffling k times. Once we have this density, we can see when it gets close to the uniform density. However, this way with the convolutions is very complicated, especially when we start shuffling more than twice. Take for example $k = 3$. We get

$$Q_1 * Q_2 * Q_3(\pi) = (Q_1 * Q_2) * Q_3(\pi) = \sum_{\pi_1 \circ \pi_2 = \pi} Q_1(\pi_1)Q_2(\pi_2)Q_3(\pi_2^{-1} \circ \pi_1^{-1} \circ \pi).$$

As we can see, this will get very complicated once we raise the value of k. However, in Section 2.6.4, we will we see that there is an easier way to compute the density.

2.3 The Riffle Shuffle

To answer our main question how many times we should shuffle a deck of cards, we need to know what method of shuffling we want to use. The method we will use, will be the Riffle Shuffle, sometimes called the GSR shuffle, since it was developed by Gilbert and Shannon [6], and separately by Reeds [7]. We use the Riffle Shuffle, because it one of the most popular way of shuffling and it has some nice mathematical, what will talk about soon. Before we do that, let us see how the shuffle goes. We begin by cutting the deck into two packets, one containing k cards and the other containing $n - k$ cards. Once this is done, we interleave the cards from the two packets together. Here it is important that the cards from both packets stay in their relative order, meaning that the cards on positions $1, 2, \ldots k$ will stay in the order if we eliminate the cards form the other packet, from the shuffled deck. This can be seen in Figure 3. Once we interleave the two packets, in the third row, we can see that the packets keep their relative order.

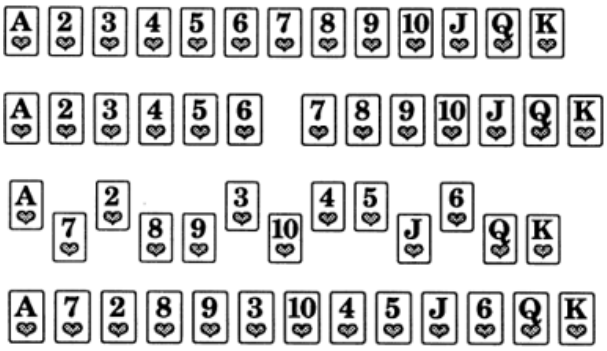

Figure 3: Schematic showing of the Riffle Shuffle, used in [1]. It shows that start with a deck of cards and cut them into two packets. Then we interleave them, maintaining their relative order.

Now we can take a deeper look on the mathematics behind this shuffle, starting with the cut. It would be ideal to cut the deck perfectly in half, but assuming this will give us two problems. First of all, what happens when n is odd? When this is the situation, it is very hard to decide how to cut the deck perfectly in half. The biggest problem of assuming this, is that this is not realistic. It is extremely difficult, especially when n is a large number, to cut a deck perfectly in half. That is why we cut the deck according to the binomial density, with parameter $\frac{1}{2}$. This will give us the probability of the cut occurring after exactly k cards, is equal to $\binom{n}{k}/2^n$. For the interleaving, we can assume that every interleaving is equally likely of happening. So we only have to determine how many possible interleavings are possible. To do that, it is smart only to look at one packet. If we choose the positions of one of the packets, the other packet will just fill in the empty spots. Then it follows that there are $\binom{n}{k}$ possible interleavings, since we only need to choose k of the n spots for the first packet. Note that $\binom{n}{k} = \binom{n}{n-k}$, so it does

not matter which of the two packets we look at. There follows that the probability for a particular interleaving occurring, is equal to $1/\binom{n}{k}$. If we now combine the results, we find the probability of a particular cut, followed by a particular interleaving, is equal to

$$\frac{\binom{n}{k}}{2^n} \cdot \frac{1}{\binom{n}{k}} = \frac{1}{2^n}.$$

Note here that this does not depend on k, meaning that every pair of a cut and a possible interleaving is equally likely.

Now that we have defined a density for the probability, we can denote it as a density R. We can find that the probability of a certain arrangement of the cards after a Riffle Shuffle is the number of ways of riffling which give the arrangement.

Let us look at an example now. Take the case $n = 3$, with a deck starting in the natural order.

k	cut deck	probability	possible interleavings
0	|123	1/8	123
1	1|23	3/8	123, 213, 231
2	12|3	3/8	123, 132, 312
3	123|	1/8	123

Table 1

If we now take a look at the results in Table 1, we can compute a probability for every possible interleaving. Note that the ordering 321 does not appear in the table, meaning that the chance of occurring after one riffle shuffle is 0. Now let us take a look at for example 132. It only occurs when we cut after 2 cards, which has a 3/8 chance. After this cut, there are three possible interleavings, each with the same conditional probability of 1/3. So the probability for 132 is $\frac{3}{8} \cdot \frac{1}{3} = \frac{1}{8}$. These same kind of arguments also hold for 213, 231 and 312, so they all have a probability of $\frac{1}{8}$. The interleaving 123 is more common; it can appear after every cut. The probability of occurring is $\frac{1}{8} \cdot 1 + \frac{3}{8} \cdot \frac{1}{3} + \frac{3}{8} \cdot \frac{1}{3} + \frac{1}{8} \cdot 1 = \frac{1}{2}$. Now we can write down the whole density for the Riffle Shuffle with $n = 3$, shown in Table 2.

permutations π	[123]	[213]	[231]	[132]	[312]	[321]
probability $R(\pi)$	1/2	1/8	1/8	1/8	1/8	0

Table 2

It is also possible to say something about the density for the Riffle Shuffle for general n. We get the following result.

Theorem 2.1. *Suppose we have a deck with n cards, $n \in \mathbb{N}$. Then we get the following probabilities.*

1. *The identity has probability* $\frac{n+1}{2^n}$.

2. *Permutations that are not possible, have probability 0.*

3. *All other permutations have probability* $\frac{1}{2^n}$.

Proof. The proof for number 2 is trivial. For number 3, we need an assumption that will be further explained in Section 2.5. The assumption is: *Every permutation with probability greater than 0, except the identity, is only possible after one particular cut.* Knowing this, it is easy to see that the probability should be equal to $\frac{1}{2^n}$. This follows directly from the fact that we need a particular cut, with a particular interleaving, and we know this has probability $\frac{1}{2^n}$.

Number 1, the identity, is a special case. We know that the identity is possible after every cut. Again for every particular cut, we have that the probability for the identity is $\frac{1}{2^n}$. Since there are $n+1$ different cuts, we have to add up the probabilities for all these different cuts. We get

$$\frac{1}{2^n} + \cdots + \frac{1}{2^n} = \frac{n+1}{2^n}.$$

Hence our result is true. □

If we look at the procedure of the shuffle again, there is still one vague aspect. We say that after cutting the deck into two packets, we interleave them in any way. There is an equivalent way of saying this, resulting in the next theorem.

Theorem 2.2. *Interleaving two packets in any way is equivalent to dropping cards onto one pile from the packets with probability proportional to the packet size.*

Proof. First we have to explain when the ways of interleaving cards are equivalent. We say that they are, when they have the same density. Since interleaving the two packets in any way has probability $1/\binom{n}{k}$, we have to prove that this is the same for dropping cards with probability proportional to the packet sizes. Now let us see what we mean by dropping cards from packets with proportional probability. We have two packets, one with size k and the other with size $n-k$. For the first card this means that we have a chance of k/n dropping from the first packet, and $(n-k)/n$ for dropping from the second. Let us say the first card drops from the first packet. The probability for the second card falling form the first packet is $(k-1)/(n-1)$ and $(n-k)/(n-1)$ for the second packet. And so on.

Let us say we have a way of dropping cards, starting with the first packets, then the second, first, first, second, first, and so on. This has probability of occurring given by

$$\frac{k}{n} \cdot \frac{n-k}{n-1} \cdot \frac{k-1}{n-2} \cdot \frac{k-2}{n-3} \cdot \frac{n-k-1}{n-4} \cdot \frac{k-3}{n-5} \cdots .$$

If we look at the product of the denominators, we see that this will end up at $n!$, since it is just a product of total cards left on the packets. The numerator is not that much harder. We can see two decreasing sequences of numbers, that of k and of $n-k$, both taking the product of every integer between itself and 1. So this will result to $k!(n-k)!$. This will give us a probability of

$$\frac{k!(n-k)!}{n!} = \frac{1}{\binom{n}{k}}.$$

Hence our result is true.

□

Now let $R^{(k)}$ stand for convoluting R with itself k times, meaning that we do the Riffle Shuffle k times in a row. The question now is how big k has to be such that $R^{(k)}$ gives a randomized deck.

2.4 Variation distance

When is a deck of cards truly randomized? If we look at this question really strictly, a deck of cards is fully randomized when all the different arrangements of cards are equally likely to occur. For a deck with n cards, we have a total of $n!$ possible arrangements, each having a chance of $1/n!$ of occurring. It turns out that for any fixed number of cards, it is impossible having such a uniform density, since we would need to shuffle an infinite number of times. This means we have to change our mindset. We were looking how we can randomize a deck of cards, but instead we have to look how close we can get to randomness. To measure how close densities are, we need a distance-measure, called a *metric*. For our problem, we will use a metric called the variation distance, which is essentially the L^1 metric on the space of densities. Let us say we have two probability densities Q_1 and Q_2. The variation distance between the two is defined as

$$\|Q_1 - Q_2\| = \frac{1}{2} \sum_{\pi \in S_n} |Q_1(\pi) - Q_2(\pi)|. \tag{2}$$

The $\frac{1}{2}$-term is only there to normalize the result, such that the distances are between 0 and 1.

Let us now look at the example $n = 3$ again. We take $Q_1 = R$ and take the density as we calculated it in Table 2. For Q_2 we take the complete reversal. This gives probability 1 to [321] and 0 to the rest. If we use (2) on these densities, we get the results from Table 3.

π	$Q_1(\pi)$	$Q_2(\pi)$	$\lvert Q_1(\pi) - Q_2(\pi)\rvert$
[123]	1/2	0	1/2
[213]	1/8	0	1/8
[312]	1/8	0	1/8
[132]	1/8	0	1/8
[231]	1/8	0	1/8
[321]	0	1	1
Total			2

Table 3

We get that $\|Q_1 - Q_2\| = \frac{1}{2}\sum_{\pi \in S_n} |Q_1(\pi) - Q_2(\pi)| = \frac{1}{2} \cdot 2 = 1$, meaning that these densities are as far apart as possible.

Now we have to implement the densities we need: $R^{(k)}$ and U being the uniform density. That means we can answer the question how many times we need to shuffle by plotting $\left\|R^k - U\right\|$ versus k.

2.5 Rising sequences

Before we can determine the density of $R^{(k)}$, we have to look at a concept that will be very important for us later. We are talking about rising sequences. A rising sequence of a permutation is a maximal consecutively increasing subsequence, meaning that we are looking for sequences of consecutive numbers, even though some different numbers can be in between. But what does this mean for cards? Let us take a deck in any sort of ordering. Start reading from the left till we find the card labeled by 1. Select that card and move along to number 2. If we find 2, we select it and move along to 3. Repeat the process, till we are at the end of the cards. We remove all the k selected cards and start from the front, looking for card number $k+1$. When there are no more cards left in the deck, we count the number of times we removed selected cards from the deck. This is the number of rising sequences of the deck.

Let us now look at an example. We take $n = 8$ and the ordering is 14256738. Start at the left and look for a 1, then a 2 and so on. after going through the deck once, we have $\overline{1}4\overline{2}56\overline{7}\overline{3}8$, where the overlined numbers are selected. If we remove those, we get 45678, which is also a rising sequence, so we remove those and we are done. This means we removed cards from the deck twice, meaning that the deck has two rising sequences. It is also nice to notice that this ordering could be the result of a Riffle Shuffle with a cut after 3 cards. In fact, it is easy to see that any ordering with exactly two rising sequences, or even only one, is a possible result of a Riffle Shuffle. This is because the cards in the two

packets maintain their relative order.

In the proof of Theorem 2.1, we made the assumption that every permutation with a probability greater than 0, except the identity, is only possible after one particular cut. Now we know about rising sequences, we have the knowledge to justify this assumption. Let us say we use the Riffle Shuffling once, resulting in us performing a permutation π. We know that, no matter the value of n, that this permutation has either one or two rising sequences. If it has only one, the permutation must be the same as the identity, so it does not count for the assumption. This means we only have a permutation with two rising sequences. If we look at both of these sequences individually, we get the cut used in the Riffle Shuffle. This follows from the fact that after interleaving the two packets, they maintain their relative order. So by looking at the rising sequences, we find the cut that was used. This means that every permutation, except for the identity, can only be done with one particular cut.

2.6 The a-shuffle

It is now clear that when a deck in natural order undergoes a single Riffle Shuffle, it ends up with either one or two rising sequences. But what can we say when shuffle multiple times? To answer this question, it is better to look at a more general case of the Riffle Shuffle: the a-shuffle.

2.6.1 Defining the a-shuffle

An a-shuffle is another probability density on S_n. The procedure is similar to that of the Riffle Shuffle. We start with a being a certain positive integer. We cut the deck into a different packets. Note that some of these packets can be empty. Now we interleave the cards from the packets together in any way, as long as the relative order of packet are maintained. Note that the Riffle Shuffle is a specific form of the a-shuffle with $a = 2$.

Now let us look at the mathematics behind this shuffle. For the cut, we make a packets, with non-negative sizes $p_1, p_2, \ldots, p_a$. Since we used the binomial density for the Riffle Shuffle, a 2-shuffle, we have to use a more general form here, namely the multinomial density, with parameter $1/a$. This will look like $\binom{n}{p_1,p_2,\ldots,p_a}$. The probability of a particular cut will then be

$$\frac{\binom{n}{p_1,p_2,\ldots,p_a}}{a^n}.$$

Note that we must have $p_1 + \cdots + p_a = n$. For the interleaving, we can again use the same arguments as with the Riffle Shuffle. Every interleaving is equally likely, meaning that it is only necessary to find the amount of possible interleavings. We need to know how to how many different ways of choosing, among n positions in the deck, p_1 for the

first packet, p_2 for the second packet, and so on. We end up with a multinomial coefficient $\binom{n}{p_1,p_2,\ldots,p_a}$, so the probability for one particular interleaving is $1/\binom{n}{p_1,p_2,\ldots,p_a}$. If we combine the cut and the interleaving, we find that the probability of a particular pair of a cut and interleaving, is equal to

$$\frac{\binom{n}{p_1,p_2,\ldots,p_a}}{a^n} \cdot \frac{1}{\binom{n}{p_1,p_2,\ldots,p_a}} = \frac{1}{a^n}.$$

Again we see, that we fill in $a = 2$, we get the same probability as with the Riffle Shuffle. We will denote the density of an a-shuffle by R_a, with $R = R_2$.

Interleaving the packets in any way, is not the only way to describe the riffling the a different packets together. We have two equivalent descriptions. First, we have the general form of Theorem 2.2, meaning that we drop cards on one pile form a packet with probability proportional to its size. The proof of this is the same as the case $a = 2$. But there is also a different description.

Theorem 2.3. *An equivalent description of the a-shuffle is given by cutting multinomially into $p_1, p_2 \ldots, p_a$ and riffling p_1 and p_2 together, then riffling the resulting pile with p_3, and so on.*

Proof. Before we begin the proof, we have to define what we mean by an equivalent description. It means that the probability the probability for any cut with interleaving is the same for both descriptions. Since we both cut with a multinomial density in both cases, it is enough to proof that riffling the packets one by one has the same probability as interleaving the packets in any way. in mathematical terms: we have to proof that the chance of any interleaving is $1/\binom{n}{p_1,p_2,\ldots,p_a}$ when shuffling one by one. We will do this proof with induction, for $a \geq 2$.

Step 1

Take $a = 2$. This means $p_1 = p$ and $p_2 = n - p$. From the Riffle Shuffle we know that there are $\binom{n}{p}$ possible interleavings. Hence there follows

$$\binom{n}{p} = \frac{n!}{p!(n-k)!} = \frac{n!}{p_1!p_2!} = \binom{n}{p_1, p_2}.$$

This means that the probability for any interleaving is $1/\binom{n}{p_1,p_2}$. So it holds for $a = 2$.

Step 2

Assume for some a the probability for any interleaving is $1/\binom{n}{p_1,p_2,\ldots,p_a} = 1/\binom{\sum_{i=1}^{a} p_i}{p_1,p_2,\ldots,p_a}$. We need to prove that for $a + 1$, the chance for a particular interleaving is equal to $1/\binom{\sum_{i=1}^{a+1} p_i}{p_1,p_2,\ldots,p_{a+1}}$.

Let us say that we first shuffle the first a packets together. By assumption, we know that

we get a certain interleaving with $1/\binom{\sum_{i=1}^{a} p_i}{p_1,p_2,\ldots,p_a}$ chance. Than we shuffle packet $a+1$ with the rest. If we want to know the probability for a certain interleaving after this shuffle, we need to multiply $1/\binom{\sum_{i=1}^{a} p_i}{p_1,p_2,\ldots,p_a}$ with the chance of any interleaving of shuffling the deck with packet $a+1$, which we know is equal to $1/\binom{\sum_{i=1}^{a+1} p_i}{p_{a+1}}$. Let us leave the '1 divided' out for a second, for readability purposes. Then this gives

$$\begin{aligned}\binom{\sum_{i=1}^{a} p_i}{p_1, p_2, \ldots, p_a} \cdot \binom{\sum_{i=1}^{a+1} p_i}{p_{a+1}} &= \frac{(\sum_{i=1}^{a} p_i)!}{p_1!p_2!\cdots p_a!} \cdot \frac{(\sum_{i=1}^{a+1} p_i)!}{p_{a+1}!(\sum_{i=1}^{a} p_i)!} \\ &= \frac{(\sum_{i=1}^{a+1} p_i)!}{p_1!p_2!\cdots p_{a+1}!} \\ &= \binom{\sum_{i=1}^{a+1} p_i}{p_1, p_2, \ldots, p_{a+1}}.\end{aligned}$$

So the probability for any interleaving is equal to $1/\binom{\sum_{i=1}^{a+1} p_i}{p_1,p_2,\ldots,p_{a+1}}$. Hence it holds for $a+1$.
From step 1 and 2 and the principle of induction, we obtain the asked result. □

This description will be easy for simulating the shuffles.

2.6.2 Relation to rising sequences

You might ask why we should even consider a-shuffles, while we only need the answer to a specific case of the a-shuffle, namely the Riffle Shuffle being a 2-shuffle. One of the main reasons is its relation to rising sequences. For the a-shuffle, we have the following nice result.

Theorem 2.4. *The probability of achieving a permutation π when doing an a-shuffle is given by $\binom{n+a-r}{n}/a^n$, where r is the number of rising sequences in π.*

Proof. First note that if we fix where all the $a-1$ cuts occur, then every permutation that can obtained after interleaving, can only be obtained in one way: just drop the cards in the same order the permutation tells us to. Thus we find that the probability of obtaining any permutation is the number of ways we can cut the deck such that we can still obtain that permutation, divided by the total number of ways of making cuts and interleaving for an a-shuffle.

The denominator in the expression is easy to prove. We know this should be the possible ways to cut and interleave the deck, and we know this should be equal to a^n, since there are this many n digit base a numbers. The relation to n digit base a numbers will become clear in Section 2.6.3.

The numerator needs a bit more work and some reasoning from taking a closer look at the cards. Let us take a deck of cards in the natural order and we apply the permutation π to it. If there are r rising sequences in the deck, we know where $r-1$ cuts must

have been. This follows from the fact that they must have occurred between pairs of consecutive cards in the original deck, such that the first card ends a rising sequence and the second begins another. Hence we know that there $(a-1)-(r-1) = a-r$ cuts that are unspecified, meaning that it does not matter where they are. Now we have to count the number of ways we can make $a-r$ cuts in n cards. We can do this by a simple principle. We make $(a-r)+n$ spaces, where we can put either a cut or a card. If we look from the perspective of the cards, there are $\binom{(a-r)+n}{n}$ ways to do this. Hence we find that our result is true. □

2.6.3 Specifying a particular a-shuffle

There is a nice and easy way to specify how a particular a-shuffle is done, which will seem useful in the next section. Let A be a n digit number, only containing base a numbers. We count the number of 0's in A, which will be the amount of cards in the first packet of the a-shuffle, p_1. Then p_2 will be the amount of 1's in A, and so on. When we have done this, we cut the deck at the specified places $p_1, \ldots, p_a$. From the first packet we start placing to cards on the locations of the 0's in A, while maintaining the relative order. The same goes for the second packet with the 1's and so on. This will give the particular cut and interleaving corresponding to A. By reflection we can see that this code is a bijective correspondence between n digit base a numbers and the set of all possible cuts and interleaving a deck with n cards, according to the a-shuffle. If we choose the n digit base a numbers according to the uniform density, we get the correct uniform density for cutting and interleaving in an a-shuffle, meaning that we get the right probabilities for an a-shuffle.

Let us look at an example. We take $n = 9$ and $a = 4$, and let $A = 101321302$. From counting the specific numbers, we find $p_1 = 2, p_2 = 3, p_3 = 2$ and $p_4 = 2$, since there are two 0's, three 1's, two 2's and two 3's. Thus we cut the deck as 12|345|67|89. If we now place the cards their relative positions in A, we get a shuffled deck of 314865927.

With this way of specifying an a-shuffle, we also get a nice result when we try to shuffle twice in a row, using different ways of shuffling. Let us say we have A is an n digit number with base a and B an n digit number with base b. If we first apply A, and then follow it by B, we get the same result as when we apply $A^B \& B$ once. John Finn figured out this formula, which we will not give a formal proof of in this report. The formula still needs some explaining. First we take a look at A^B. We basically apply B to A, meaning that rearrange A according to the permutation B. The & is a bit more work, but still rather straight forward. For $A^B \& B$, we change every ith digit in A^B into $A^B_i \cdot b + B_i$, with base ab.

As said, we will not prove this formula formally here. But that this formula holds, and how it works, can best be shown in an example. Suppose we have $A = 210102$ with

base 3 and $B = 203100$ with base 4. If we apply these to a deck in the natural order, we get Table 4.

i	1	2	3	4	5	6
$\pi_A(i)$	5	3	1	4	2	6
$\pi_B \circ \pi_A(i)$	2	5	6	4	3	1

Table 4

Now we try to compute $A^B\&B$ and see if get the same result. This computation is shown in table 5.

A	2	1	0	1	0	2
B	2	0	3	1	0	0
A^B	0	2	2	1	1	0
B	2	0	3	1	0	0
$A^B\&B$	2	8	11	5	4	0

Table 5

If we apply this to a deck with the natural order, we get 256431. So this gives the same result as doing one shuffle after the other.

2.6.4 The multiplication theorem

We are close to constructing a graph for the variation distance. There is only one fundamental problem we have not faced yet: How can we describe a Riffle Shuffle when we apply it multiple times, without having to convolute the densities? The answer lies within the next theorem.

Theorem 2.5. *An a-shuffle followed by a b-shuffle is equivalent to a single ab-shuffle.*

Proof. To prove this, we have to find a formula that is actually a one-to-one correspondence between a pair of one n digit base a number and one n digit base b number, and the set of n digit base ab numbers. In Section 2.6.3, we already made such a formula. The $A^B\&B$ formula that we introduced, satisfies this criterion. This follows from the fact that we combine an n digit base a number and an n digit base b number, and make it into an n digit base ab number. Since the probability densities for a, b and ab-shuffles are induced by the uniform densities on the sets of n digit base a, b or ab numbers, it is implied by the properties of the one-to-one correspondence, that the densities on S_n of an a-shuffle followed by a b-shuffle is the same as an ab-shuffle. Hence our result holds. □

2.7 Combining all the results

We now have all the ingredients to answer the questions how many times we have to do a Riffle Shuffle before a deck is randomized well enough. If we do a Riffle Shuffle k times, we know that we are basically doing k times a 2-shuffle. By Theorem 2.5, this is equivalent to a single $2 \cdot 2 \cdots 2 = 2^k$-shuffle. If we fill this in the formula we found in Theorem 2.4, we find that in the $R^{(k)}$ density there is a chance of $\binom{2^k+n-r}{n}/2^{nk}$ of getting a permutation π with r rising sequences. Since in the uniform density U every permutation has the same probability $1/n!$, this gives us

$$|R^{(k)}(\pi) - U(\pi)| = \left| \binom{2^k + n - r}{n} / 2^{nk} - \frac{1}{n!} \right|. \tag{3}$$

We want to find the variation distance between $R^{(k)}$ and U, given by

$$\left\| R^{(k)} - U \right\| = \frac{1}{2} \sum_{\pi \in S_n} |R^{(k)}(\pi) - U(\pi)|.$$

To compute this, we need to sum over all the possible permutations, but we have a quicker way to do this. Note that if we fix n, (3) only depends the amount of rising sequences. This means we only have to sum over the amount of possible rising sequences, which is from 1 to n. There is only one problem left: Theorem 2.4 gives the probability for specific permutation with r rising sequences, but we need the probability of any permutation with r rising sequences. We can solve this by multiplying with the amount of possible permutations with r rising sequences. These are exactly the *Eulerian numbers*. We will specify these numbers by $A_{n,r}$, with n being the amount of cards in the deck and r the amount of rising sequences. There are multiple formulas for these numbers, one of them is

$$A_{n,r} = \sum_{k=0}^{r} (-1)^k \binom{n+1}{k} (r + 1 - k)^n.$$

If we now take these Eurelian numbers into account, we get the variation distance we want:

$$\left\| R^k - U \right\| = \frac{1}{2} \sum_{r=1}^{n} A_{n,r} \left| \binom{2^k + n - r}{n} / 2^{nk} - \frac{1}{n!} \right|.$$

Even though this formula still is difficult to calculate by hand, but it is now easy and quick for computer program *Python* to compute its graph. We take here a standard deck of cards with $n = 52$.

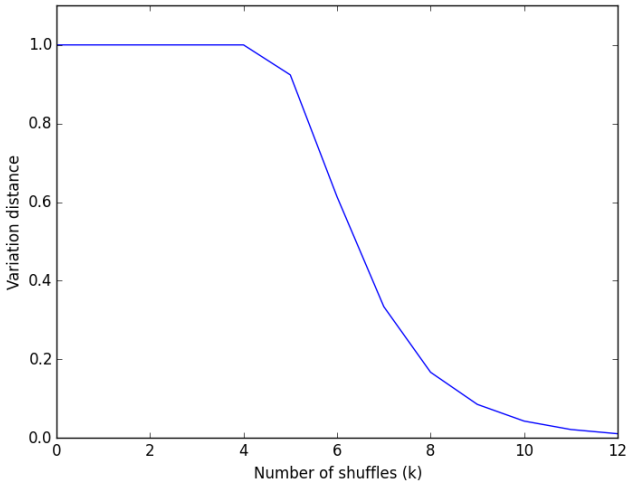

Figure 4: Graph of the variation distance versus k.

As can be seen in Figure 4, the variation distance starts to drop fast at $k = 5$, becomes more horizontal again at $k = 7$ and almost reaches 0 at $k = 12$. Since $k = 7$ seems like a good middle point for the cutoff, we can conclude that seven shuffle is said to be enough for a deck of 52 cards.

We can also do this for different values of n. In Table 6, some different values of n are shown for shuffling from 1 up to 10 times. One can choose for him or herself when the variation distance is close enough to 0 for a saying that a deck is almost is fully randomized. For now, we put the crossline at 0.5. Hence we need respectively 5, 6, 7, 8 and 8 shuffles to randomize deck of 25, 35, 52, 78 and 100 cards.

n	1	2	3	4	5	6	7	8	9	10
25	1.000	1.000	0.999	0.775	0.437	0.231	0.114	0.056	0.028	0.014
32	1.000	1.000	1.000	0.929	0.597	0.322	0.164	0.084	0.042	0.021
52	1.000	1.000	1.000	1.000	0.924	0.614	0.334	0.167	0.085	0.043
78	1.000	1.000	1.000	1.000	1.000	0.893	0.571	0.307	0.153	0.078
100	1.000	1.000	1.000	1.000	1.000	0.982	0.747	0.429	0.224	0.112

Table 6

3 Discussing the analysis

In Section 2, we computed the answer for the question how many times we need to shuffle a deck of cards. In this section we will look at some possible discussion points about this analysis. These points are also mentioned in [2].

3.1 Is seven really enough?

Even though we concluded that shuffling a deck seven times should randomize a deck, there is still room for some discussion about this number. There are examples where seven times is not good enough. As presented in [2], Peter Doyle invented a game of solitaire which shows, that we did not shuffle good enough.

It goes as follows. We start with a deck of 52 cards, turned face-down. We label the cards from top to bottom with $123\ldots(25)(26)(52)(51)\ldots(28)(27)$. Now perform the Riffle Shuffle seven times, which should normally randomize the deck. We make three piles A, B and C. Now take the top card. If it is number 1, we place it face up on pile A. If it is 27, place it face up on pile B. If it is a different card, put it face up on C. If the next card is the immediate successor of the top card from either A or B, place it on the respective pile. Otherwise we place it on C. Once we placed all the cards from the original deck on one of the three piles, We pick up C, turn it upside down and start placing those cards. The game ends when either A or B is full, meaning that they contain 26 cards. The pile that is full first, wins.

Since shuffling seven times should randomize the deck, we would expect that both piles win for half of the time. However when we compute this game into *Python*, we find that A wins about 83% of the time. This has to do with rising sequences. We expect them to come from both the first and second of the original deck in roughly the same numbers and length. The only problem here is that these will be in a forward direction, while we collect cards for B in a backward direction. This means that a rising sequence for A can be picked up in one sweep through the deck, while for B we can only pick up one card at a time.

The key here is that this game will turn out to be almost as far away from being a fair game as possible. This is a consequence of an equivalent definition of the variation distance,

$$\|Q_1 - Q_2\| = \max_{S \subset S_n} |Q_1(S) - Q_2(S)|$$

where $Q_1(S)$ is defined as $\sum_{\pi \in S} Q_1(\pi)$. This actually means that the variation distance is an upper bound for the difference between the probabilities of an event given by two

densities. This counts for this game. If we look at the percentage we expected A to win, and how much it actually won, we see that this is really close to the variation distance of seven Riffle Shuffles and the uniform density. The difference is probability is $|0.83 - 0.50| = 0.33$, while we know from Section 2.7 that the variation distance is equal to 0.334. This shows us that this game is a worst case scenario.

3.2 Should we use the variation distance?

We introduced the variation distance as measure for the distance between two densities. It looks reasonable to do, since we take the difference of all the probabilities and add them up in the end. But is the variation distance not too strict? The next example shows why this might be the case.

Let us say we have deck containing n cards, face down. We know the deck has been perfectly randomized, meaning that you have the uniform density $U(\pi) = 1/n!$ for all $\pi \in S_n$. Now suppose we take of the top card and we see which one it is. You now put it back in the deck in the top half of the deck, at random. Even though you changed one position of only one card, the variation distance will be affected greatly by this. This is because we cut the amount of possible orderings in half, since we know that one particular card is in the top half of the deck and not in the bottom half. We call the new density $\bar{U}$, which is $2/n!$ in the top half of the deck and 0 in the bottom. This gives a variation distance of

$$\left\|U - \bar{U}\right\| = \frac{1}{2}\left(\frac{n!}{2}\left|\frac{2}{n!} - \frac{1}{n!}\right| - \frac{n!}{2}\left|0 - \frac{1}{n!}\right|\right) = \frac{1}{2}.$$

Since we set the line for ourselves at 0.5, that would mean that $\bar{U}$ is barely random enough, while we know only the position of one card, and not even its exact position.

3.3 Another way of computing variation distances

In Section 2, we saw a way to calculate the variation distance. However, in the original paper written by Bayer and Diaconis [1], a different approach is used. In this section we will look at how they calculated the variation distances. The theorem we will introduce in this section, will not be proven here. Furthermore, all the theorems and knowledge we obtained in Section 2, are used.

The following theorem gives a different expression for the variation distance.

Theorem 3.1. *Let Q^k be the density after performing a Riffle Shuffle k times. Let U be the uniform density. For $k = \log_2(n^{3/2}c)$, with fixed $0 < c < \infty$, as n tends to ∞,*

$$\left\|Q^k - U\right\| = 1 - 2\Phi\left(\frac{-1}{4c\sqrt{3}}\right) + O_c\left(\frac{1}{n^{1/4}}\right)$$

with

$$\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-t^2/2} dt.$$

Using this theorem, we can calculate the variation distance first calculating c, and then filling in k and c. When we do this, we get the same results as we get in Table 6 in Section 2.7. There some advantages and disadvantages to using this formula instead of the one we used in Section 2.7. A big advantage is that we can compute the variation distance for larger values of n. In the method we used, you have to multiply and divide some very large number with each other, for example $A_{n,r}, 2^{nk}$ and $1/n!$. This can give problems once the value of n is higher than 100. On the other hand, in the formula used in Theorem 3.1 we assume that n tends to infinity. This might cause some inaccuracy for smaller values of n, like 32 or 52.

Note that in Theorem 3.1, we have an expression for k, namely $k = \log_2(n^{3/2}c)$. Here counts that $c = 2^j$, where j stands for the number of shuffles performed after $\frac{3}{2}\log_2(n)$ shuffles. If we set $j = 0$, we get an estimate of how many shuffle we need to perform to mix up n cards, namely $\frac{3}{2}\log_2(n)$. This estimates for some values of n are shown in Table 7. Note that these values are all higher than the results we obtained in Section 2.7.

n	25	32	52	78	100
$\frac{3}{2}\log_2(n)$	6.97	7.50	8.55	9.43	9.97

Table 7

4 Different ways of shuffling

Now that we have found the solution to the problem how many times we should perform a Riffle Shuffle till a deck of cards is randomized enough, we can ask ourselves what the case is if we shuffle using a different method. A place where shuffling cards is of great importance, is in a casino. In this chapter we will look at some different ways of shuffling, including some used in casinos.

4.1 Simulations

To do this, it is not always possible to describe the shuffle in a mathematical attractive way, like we did with the Riffle Shuffle. To still come up with a solution, we can simulate a deck and the ways of shuffling, using *Python*. The code of these simulations is shown in Appendix A. The simulations for this section will mostly consist of two tests to see if a deck is well shuffled. We can only say that a method of shuffling is randomizing enough when it passes both of the tests.

4.1.1 Top card test

The first little test is what we will call the *top card test*. As the name suggests, we will look where the original top card of a deck goes after we shuffle it. Let us say we have a deck containing n cards. If a deck is well randomized, one would expect that the probability of the top card being in a specific position, is equal for every position, so $1/n$. In this test we will count the amount of times the top card is still at the top after performing the shuffle, which we will call t. We will do this 52.000 times, meaning that, since in our case $n = 52$, we expect to find t to be around 1000.

4.1.2 Estimating the variation distance

In Section 2, we tried to estimate how random a deck is by calculating the variation distance. This was possible because we there is a nice connection between the a-shuffle and rising sequences. With most other shuffling methods, this connection is not there. This means we have to try and simulate this connection. In Theorem 2.4 we got the probability that we get a permutation π with r rising sequences. By simulation, we can try to find this probability for other ways of shuffling. We do that as follows. We simulate a deck of cards and perform a method of shuffling on it. Once it has been shuffled, we count the amount of rising sequences and call this b_r. We do this 5000 times. After, we have a list of the amount of times we have r rising sequences. If we divide those amounts by 5000, we get a probability of any permutation with r rising sequences. This means we have to divide these probabilities by their respective Eulerian number, giving us the probability of a specific permutation with r rising sequences. We can fill these in the

formula we got for the variation distance, which will look like

$$\|Q - U\| = \frac{1}{2} \sum_{r=1}^{n} A_{n,r} \left| \frac{b_r}{5000 \cdot A_{n,r}} - \frac{1}{n!} \right|. \tag{4}$$

This gives us an estimate for the variation distance between a random density Q and the uniform density. The exact results of the simulations, meaning the r rising sequences and how many times it occurred, can be found in Appendix B.

To show that this works, it might be good to look at an example. Let us look at the cases $Q = R^{(6)}, Q = R^{(7)}$ and $Q = R^{(8)}$, so we perform the Riffle Shuffle six, seven and eight times. From Table 6, we know that the variation distances are respectively 0.614, 0.334 and 0.167. If we now simulate these shuffles and fill in the results in (4), we get the estimate variation distances 0.622, 0.322 and 0.162. These are really close to the actual values. If we look at the differences and use a 95 % confidence interval, we can conclude that we have an estimated standard deviation of 0.028.

4.2 Reverse riffling

The first problem we will look at, is not really a different way of shuffling. It is presented as an open problem in [3] and it tells us about a shuffling machine that is supposed to perform a Riffle Shuffle. From Theorem 2.2, we know that we can describe riffling two packets together as dropping the cards form a packet with probability proportional to the packet size. This machine is not working properly though, since during the riffling, it drops the cards from a packet with probability "opposite" proportional to the packet size. In mathematical terms, this means the if we have packets 1 and 2 with sizes k and $n - k$, the probability that we drop a card from packet 1 is equal $(n - k)/n$, while it should be k/n. Once all of the cards from one packet are dropped, the rest from the other packet will be placed on top.

Now let us look the simulations and see what variation distance comes out. The results are shown in Table 8.

We can see that the variation distance drops a lot slower then when we perform a regular Riffle Shuffle. Also note that if we compare these variation distances with the ones we found in Table 6 in Section 2.7, we almost need twice as many shuffles here for the same result. This means that once the probability for the riffling is "reversed", we need 12 shuffles, instead of seven.

k	est. variation distance (± 0.028)
6	0.987
7	0.953
8	0.882
9	0.784
10	0.675
11	0.555
12	0.440
13	0.343
14	0.257
15	0.208
16	0.150
17	0.121
18	0.090
19	0.067
20	0.052
21	0.031

Table 8

4.3 Casino hand-shuffle

Most big casinos shuffle using a machine. A lot of the smaller casinos on the other hand still let the dealers shuffle the cards by hand. They do this in a fixed way, going as follows: first perform a Riffle Shuffle three times, then a Strip Shuffle, followed by one more Riffle Shuffle and ended by a full cut. There are two methods of shuffling that need a deeper explanation. First being the Strip Shuffle. This is a special version of the 5-shuffle. We cut the deck into five packets, using the multinomial density. Instead of interleaving the packets, we take the packet that was first on top and put it on the bottom. Then we we take the packet second from the top and place it on top of the other, and repeat this for the other packets. We can see this schematically in Figure 5.

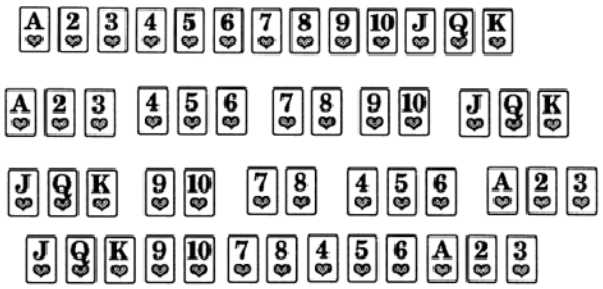

Figure 5: Schematic showing of the Strip shuffle.

Now we look at the full cut. This a simpler version of the Strip Shuffle. It is the same method, but now only using two packets. So we cut the deck into two packets with the binomial density, followed by putting the bottom packet on top of the other. In Figure 6, we can see how this looks schematically.

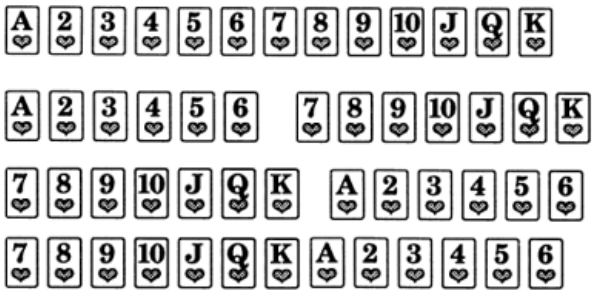

Figure 6: Schematic showing of the full cut.

Let us perform our two tests on this shuffle, beginning with the Top card test. Simulations show that the top card gets back on top 1079 of the times, where we expect around 1000. This means that this is close to what we, meaning that we get the feeling that the way of shuffling in casino really randomizes the deck. Now let us look at the simulating of the variation distance. If we fill in the data from the simulating, we get a value for the variation distance equal to 0.942 (± 0.028), which is almost equal to perform a Riffle Shuffle five times. This means that the shuffle used in casinos is not randomizing enough, which is a surprising result.

4.4 Casino shelf-shuffle

As said, most big casinos use shuffling machines. One of this machines is presented in [4]: the shelf shuffling machine. The shelf shuffler works in a very interesting way. Let us say we have m shelves. Once we put the deck of cards in the machine, it starts dividing the cards over the shelves. Every card can be dropped on a certain shelf with the uniform density, meaning that the probability that a card goes to self i, is equal to $1/m$. Once in the machine, it can be placed in two ways: on top or on the bottom of the packet on that shelf, both with probability 1/2. Once every card is placed on one of the shelves, the machine drops it back to one pile. Every shelf had probability $1/m$ to be dropped first. For the second, every remaining shelf has probability $1/(m-1)$ to be dropped on top of the first one, and so on. Let us look at a schematic example with $m = 5$ in Figure 7.

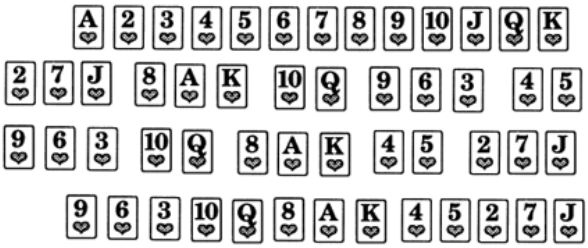

Figure 7: Schematic showing of the shelf shuffle with $m = 5$.

In the case represented in [4], we get $m = 10$. We will look at it a bit more general, with multiple values of m, namely 5, 10, 50, 100 and 200. It is now time to look at the results from the simulations, so that we can determine for which values of m, the shelf shuffle is a randomizing shuffle. The results of the top card test are shown in Table 9.

m	t
5	57
10	397
50	1030
100	936
200	965

Table 9

We can already see that t in the cases $m = 5$ and $m = 10$ is by far not close enough to the expected value of 1000, while the cases 50, 100 and 200 are close enough. Meaning that we can already conclude that the shelf shuffle with 5 and 10 shelves is not randomizing enough. Let us now look at the estimations of the variation distance, displayed in Table 10.

m	est. variation distance (± 0.028)
5	0.040
10	0.029
50	0.013
100	0.016
200	0.024

Table 10

Even though we know the cases with 5 and 10 shelves are not randomizing enough, we can see that their variation distance is very low. This seems weird, but can be explained rather easily. The variation distance as we calculate and estimate it, is purely based on the number of rising sequences. However this does not tell us anything about

the locations. Take for example the ordering $[(51)(52)(49)(50)\ldots 3412]$. This ordering has 26 rising sequences, meaning that, according to the variation distance, it should be a well randomized deck. However, we can see that this is not the case at all. So we can conclude that the cases with 5 and 10 shelves are no randomizing shuffles, even though they have a low variation distance.

The cases with 50, 100 and 200 shelves have both a low variation distance, and the values from the top-card test all lie near the expected 1000. This means that we can conclude that these are randomizing shuffles.

4.5 Washing cards

Not everyone is good at shuffling. Riffling two packets together, for example, can be a very difficult task. But there is one method of shuffling that can be used by everyone, even by little children. We are talking about washing cards. You spread the cards all over the table and you start moving them around. The cards gets mashed up rather quickly and it looks like there is no structure to be found in this way of shuffling. We will try to simplify and actually put some kind of structure to this problem. Before we do this, let us see how we will try to simulate this. Because of the lack of structure, it is very difficult to actually simulate washing cards and putting them to the test like we did with the examples before. In this case, it is better to look at a different, important aspect of mathematics: Markov chains.

4.5.1 Markov chains and the relation with card shuffling

Before we start with discussing our problem in more detail, it might be better to do fast introduction to Markov chains and show how we can use them in card shuffling. A *Markov chain* is a stochastic process, that moves between a finite set of states S. This can be described by a sequence of random variables $X_0, X_1, \ldots$, each taking values in S, where $X_t = i$ means that the process is in state $i \in S$ at time t. A Markov chain is only dependent on its current state. What happened in the past does not have any influence. That is why we call a Markov chain memoryless. The process goes as follows. We start in a certain state, let us say X_0. Then we can take a step to a different state, based on a density defined over these states. Once we are in the new state, called X_1, we can take a step again to again a different state, again based on a density. And so on. We can capture these density for the states in something that is called a *transition matrix* p. The elements of the matrix are called *transition probabilities* p_{ij}. p_{ij} is defined as $\mathrm{P}(X_t = j \mid X_{t-1} = i)$. This means the probability that after t steps we are in state j, given that we were in state i after $t-1$ steps. Let us look at the example from Section 2.2.2, the top-in shuffle. We know that this had the following density if we start with the natural order.

permutations	[123]	[213]	[231]	[132]	[312]	[321]
probability	1/3	1/3	1/3	0	0	0

But we can compute such a density for every beginning order. If we capture these densities in a matrix, we get the transition matrix for the top-in shuffle.

$$p = \begin{array}{c} \\ [123] \\ [213] \\ [231] \\ [132] \\ [312] \\ [321] \end{array} \begin{array}{c} \begin{array}{cccccc} [123] & [213] & [231] & [132] & [312] & [321] \end{array} \\ \begin{pmatrix} 1/3 & 1/3 & 1/3 & 0 & 0 & 0 \\ 1/3 & 1/3 & 0 & 1/3 & 0 & 0 \\ 0 & 0 & 1/3 & 0 & 1/3 & 1/3 \\ 0 & 0 & 0 & 1/3 & 1/3 & 1/3 \\ 1/3 & 0 & 0 & 1/3 & 1/3 & 0 \\ 0 & 1/3 & 1/3 & 0 & 0 & 1/3 \end{pmatrix} \end{array}$$

On the left, next to the matrix, we can see the order that we start in. Above the matrix are the possible orders of the deck after performing the shuffle. It is easy to check that for other starting orders, these densities are indeed correct.

So now we have the probability for $\mathrm{P}(X_t = j \mid X_{t-1} = i)$, but can we not do this for more than one step? Meaning that if we know X_0, can we say anything about X_t for general t? It turns out that we only need the tth power of p for that. For example, we take $t = 7$. We get

$$p^7 = \begin{array}{c} \\ [123] \\ [213] \\ [231] \\ [132] \\ [312] \\ [321] \end{array} \begin{array}{c} \begin{array}{cccccc} [123] & [213] & [231] & [132] & [312] & [321] \end{array} \\ \begin{pmatrix} 0.171 & 0.167 & 0.167 & 0.167 & 0.167 & 0.163 \\ 0.167 & 0.171 & 0.167 & 0.167 & 0.163 & 0.167 \\ 0.167 & 0.167 & 0.171 & 0.163 & 0.167 & 0.167 \\ 0.167 & 0.167 & 0.163 & 0.171 & 0.167 & 0.167 \\ 0.167 & 0.163 & 0.167 & 0.167 & 0.171 & 0.167 \\ 0.163 & 0.167 & 0.167 & 0.167 & 0.167 & 0.171 \end{pmatrix} \end{array}$$

So if we start with 123, the probability we are in 213 after seven shuffles is equal to 0.167. Note that the densities are close to the uniform density, meaning that shuffling a deck of three cards seven times with the top-in shuffle, is randomizing the deck.

4.5.2 Back to washing cards

Now we know how to use Markov chains for card shuffling problems, we can try to bring some structure to washing cards. Please note that this will be a strongly simplified version and further research can be done in this case. Let us begin by defining the setup. Normally when we wash cards, they are on a big pile, one laying over the other. We will

assume that all the cards are separate form each other, laying in a circle. This is shown schematically in Figure 8.

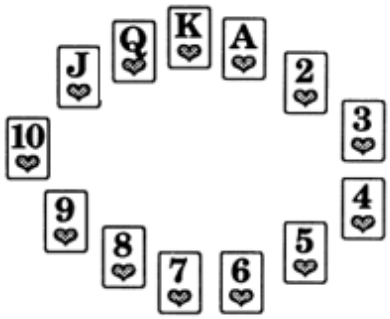

Figure 8: Schematic showing of our setup for washing cards.

The reason we do this, is that this makes it easy for us to number the positions of the cards, giving us an order of the deck. We also have to define one "shuffle". We say that moving the cards in half a circle is one shuffle. This means that we place each card somewhere in the next half of cards behind, and we do this for every card. If we look for example at the card in position 1, with one shuffle, it can only move move to places 1 up to 27. We do this by a binomial density, basically meaning that it is more likely for a card in position 1 to move to positions around 13 then moving to spots around 26 or staying near position 1.

Now we have the densities per position, we can make our transition matrix p. This will be a 52x52-matrix, which is too big to show on paper. To still give an idea of the procedure, we show it with six cards. Our p would look like

$$p = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} \begin{array}{c} \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\ \begin{pmatrix} 1/8 & 3/8 & 3/8 & 1/8 & 0 & 0 \\ 0 & 1/8 & 3/8 & 3/8 & 1/8 & 0 \\ 0 & 0 & 1/8 & 3/8 & 3/8 & 1/8 \\ 1/8 & 0 & 0 & 1/8 & 3/8 & 3/8 \\ 3/8 & 1/8 & 0 & 0 & 1/8 & 3/8 \\ 3/8 & 3/8 & 1/8 & 0 & 0 & 1/8 \end{pmatrix} \end{array}.$$

On the left are now the begin position of the cards, and on top are the possible positions after a shuffle. If all these densities come close to the uniform density, then the position of the cards is close to random, meaning that the deck is close to randomized. We say that a the position of card is close to random, if the difference of the probabilities with the uniform density is maximum 0.01. For this example, we get

$$p^9 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} \begin{array}{c} \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\ \begin{pmatrix} 0.167 & 0.173 & 0.173 & 0.167 & 0.161 & 0.161 \\ 0.161 & 0.167 & 0.173 & 0.173 & 0.167 & 0.161 \\ 0.161 & 0.161 & 0.167 & 0.173 & 0.173 & 0.167 \\ 0.167 & 0.161 & 0.161 & 0.167 & 0.173 & 0.173 \\ 0.173 & 0.167 & 0.161 & 0.161 & 0.167 & 0.173 \\ 0.173 & 0.173 & 0.167 & 0.161 & 0.161 & 0.167 \end{pmatrix} \end{array}.$$

Meaning that we need nine shuffles to randomize a deck with six cards. If we now do the same for 52 cards and we use the same criterion that the difference between the probabilities and the uniform density is maximum 0.01, we find that we need 29 shuffles before the deck is close to randomized.

References

[1] Dave Bayer and Persi Diaconis. *Trailing the Dovetail Shuffle to its Lair.* [*Annals of Applied Probability*] 2(2), 294-313, 1992

[2] Brad Mann. *How many times should you shuffle a deck of cards?* [*UMAP J.*] 15(4), 303-332, 1994

[3] Persi Diaconis *Mathematical developments from the analysis of riffle shuffling* [*Groups, Combinatorics and Geometry*] World Scientific, N.J., 73-97, 2003

[4] Persi Diaconis, Jason Fulman and Susan Holmes. *Analysis of casino shelf shuffling machines.* [*The Annals of Applied Probability*] Vol.23, No.4, 1692-1720, 2013

[5] Gina Kolata *In Shuffling Cards, Seven is Winning Number.* [*New York Times*] Jan. 9. 1990

[6] Edgar Gilbert *Theory of Shuffling* [*Technical memorandum*] Bell Labs, 1955

[7] Jim Reeds *unpublished manuscript* 1981

A Python-code

```
                    #### Defining cards and shuffles ####

import random
from numpy import *

####Defining a card
class Card(object):

    suit_names = ['Clubs', 'Diamonds', 'Hearts', 'Spades']
    rank_names = [None, 'Ace', '2', '3', '4', '5', '6', '7', \
                  '8', '9', '10', 'Jack', 'Queen', 'King']


    def __init__(self, suit = 0, rank = 2):
        self.suit = suit
        self.rank = rank

    def __str__(self):
        return '%s of %s' % (Card.rank_names[self.rank], \
                             Card.suit_names[self.suit])

    def __cmp__(self, other):
        t1 = self.suit, self.rank
        t2 = other.suit, other.rank
        return cmp(t1, t2)

#### Defining a deck of cards
class Deck(object):

    def __init__(self):
        self.cards = []
        for suit in range(4):
            for rank in range(1, 14):
                card = Card(suit, rank)
                self.cards.append(card)

    def __str__(self):
        res = []
```

```
39          for card in self.cards:
40              res.append(str(card))
41          return '\n'.join(res)
42
43      def __repr__(self):
44          res = []
45          for card in self.cards:
46              res.append(str(card))
47          return '\n'.join(res)
48
49      ### Help functions
50
51      # Pick the top card
52      def pop_card(self):
53          return self.cards.pop()
54
55      # Add a card to the deck
56      def add_card(self, card):
57          self.cards.append(card)
58
59      # Sort the cards into original order
60      def sort(self):
61          self.cards.sort()
62
63      # Move cards from one deck to another
64      def move_cards(self, hand, num):
65          for i in range(num):
66              hand.add_card(self.pop_card())
67
68      # Counts the amount of cards in a deck
69      def count(self):
70          res = 0
71          trash = Deck()
72          trash.empty()
73          while True:
74              try:
75                  self.move_cards(trash, 1)
76                  res += 1
77              except:
78                  break
```

```
79          trash.move_cards(self, res)
80          return res
81
82      # Empties a deck
83      def empty(self):
84          waste = Hand('waste')
85          while True:
86              try:
87                  self.move_cards(waste, 1)
88              except:
89                  break
90
91      # Cuts a deck into two packets
92      def cut(self):
93          top = Deck()
94          top.empty()
95          res = self.count()
96          n = random.binomial(res, 0.5)
97          self.move_cards(top, n)
98          return top
99
100     # Riffles two packets together
101     def riffle(self, deck):
102         n = self.count()
103         hulp = Deck()
104         hulp.empty()
105         self.move_cards(hulp, n)
106         m = deck.count()
107         while self.count() < (n+m):
108             d = deck.count()
109             h = hulp.count()
110             r1 = random.randint(1 ,d+h+1)
111             if r1 <= h:
112                 r = 0
113             else:
114                 r = 1
115             if (r == 0) and (h > 0):
116                 hulp.move_cards(self, 1)
117             elif (r == 1) and (d > 0):
118                 deck.move_cards(self, 1)
```

```
119
120     ### Methods of shuffling
121
122     # Top-in Shuffle
123     def top_in(self, num = 1):
124         for i in range(0, num):
125             top = Deck()
126             top.empty()
127             pick = Hand('pick')
128             self.move_cards(pick, 1)
129             res = self.count()
130             n = random.randint(0,res)
131             self.move_cards(top, n)
132             pick.move_cards(self,1)
133             top.move_cards(self,n)
134
135     #Reversing a deck
136     def reverse(self):
137         hulp1 = Deck()
138         hulp2 = Deck()
139         hulp1.empty()
140         hulp2.empty()
141         n = self.count()
142         self.move_cards(hulp1, n)
143         hulp1.move_cards(hulp2, n)
144         hulp2.move_cards(self, n)
145
146     # The a-shuffle
147     def ashuffle(self, a = 2, num = 1):
148         for i in range(0, num):
149             lst = []
150             n = self.count()
151             for i in range(0, a-1):
152                 hulp = Deck()
153                 hulp.empty()
154                 b = random.binomial(n, 1/float(a))
155                 if self.count() < b:
156                     self.move_cards(hulp, self.count())
157                 else:
158                     self.move_cards(hulp, b)
```

```
159                 lst.append(hulp)
160             for i in range(0, a-1):
161                 self.riffle(lst[i])
162 
163     # Riffle Shuffle
164     def rshuffle(self, num = 1):
165         for i in range(0, num):
166             self.ashuffle()
167 
168     ## Reversed Riffle
169 
170     # Riffle with reversed probability
171     def riffleOp(self, deck):
172         n = self.count()
173         hulp = Deck()
174         hulp.empty()
175         self.move_cards(hulp, n)
176         m = deck.count()
177         while self.count() < (n+m):
178             d = deck.count()
179             h = hulp.count()
180             r1 = random.randint(1 ,d+h+1)
181             if r1 <= d:
182                 r = 0
183             else:
184                 r = 1
185             if ((r == 0) and (h > 0)) or d == 0:
186                 hulp.move_cards(self, 1)
187             elif ((r == 1) and (d > 0)) or h == 0:
188                 deck.move_cards(self, 1)
189 
190     # The a-shuffling using reversed riffling
191     def ashuffleOp(self, a = 2, num = 1):
192         for i in range(0, num):
193             lst = []
194             n = self.count()
195             for i in range(0, a-1):
196                 hulp = Deck()
197                 hulp.empty()
198                 b = random.binomial(n, 1/float(a))
```

```
                if self.count() < b:
                    self.move_cards(hulp, self.count())
                else:
                    self.move_cards(hulp, b)
                lst.append(hulp)
            for i in range(0, a-1):
                self.riffleOp(lst[i])

    # Riffle shuffle using reversed riffling
    def rshuffleOp(self, num = 1):
        for i in range(0, num):
            self.ashuffleOp()


    ## Casino shuffles

    # Strip shuffle
    def strip(self, num = 1):
        for i in range(0, num):
            n = self.count()
            hulp1 = Deck()
            hulp1.empty()
            self.move_cards(hulp1, n)
            hulp1.reverse()
            while self.count() < n:
                hulp2 = Deck()
                hulp2.empty()
                b = random.binomial(n, 1./5)
                if hulp1.count() < b:
                    hulp1.move_cards(hulp2, hulp1.count())
                    hulp2.move_cards(self, hulp2.count())
                else:
                    hulp1.move_cards(hulp2, b)
                    hulp2.move_cards(self, b)

    # Full cut
    def fullcut(self, num = 1):
        for i in range(0, num):
            top = self.cut()
            hulp = Deck()
```

```
            hulp.empty()
            n = self.count()
            self.move_cards(hulp, n)
            t = top.count()
            top.move_cards(self, t)
            hulp.move_cards(self, n)

    # Casino hand-shuffle
    def casino(self, num = 1):
        for i in range (0, num):
            self.rshuffle(3)
            self.strip()
            self.rshuffle()
            self.fullcut()

    # Casino shelf-shuffle
    def shelf(self, m = 10, num = 1):
        for t in range (0, num):
            n = self.count()
            lst = [None] * m
            for k in range (0, m):
                lst[k] = Deck()
                lst[k].empty()
            hulp1 = Deck()
            hulp1.empty()
            hulp2 = Deck()
            hulp2.empty()
            while self.count() > 0:
                self.move_cards(hulp1, 1)
                r1 = random.randint(0, m)
                for i in range (0, m):
                    if r1 == i:
                        r2 = random.randint(0,2)
                        if r2 == 0:
                            hulp1.move_cards(lst[r1], 1)
                        else:
                            b1 = lst[r1].count()
                            lst[r1].move_cards(hulp2, b1)
                            hulp1.move_cards(lst[r1], 1)
                            hulp2.move_cards(lst[r1], b1)
```

```
279             while self.count() < n:
280                 r3 = random.randint(0, m)
281                 b2 = lst[r3].count()
282                 if b2 > 0:
283                     lst[r3].move_cards(self, b2)
284 
285 #### Defining a hand
286 class Hand(Deck):
287 
288     def __init__(self, label=''):
289         self.cards = []
290         self.label = label
291 
292 
293     def __str__(self):
294         club_lst = ['Clubs']
295         dia_lst = ['Diamands']
296         hea_lst = ['Hearts']
297         spa_lst =['Spades']
298         tot_lst = []
299         for cards in self.cards:
300             if cards.rank == 1:
301                 ranks = 'A'
302             elif cards.rank == 11:
303                 ranks = 'J'
304             elif cards.rank == 12:
305                 ranks = 'Q'
306             elif cards.rank == 13:
307                 ranks = 'K'
308             else:
309                 ranks = cards.rank
310             if cards.suit == 0:
311                 club_lst.append(str(ranks))
312             elif cards.suit == 1:
313                 dia_lst.append(str(ranks))
314             elif cards.suit == 2:
315                 hea_lst.append(str(ranks))
316             elif cards.suit == 3:
317                 spa_lst.append(str(ranks))
318         for elt in (club_lst, dia_lst, hea_lst, spa_lst):
```

```
319             tot_lst.append(' '.join(elt))
320         return '\n'.join(tot_lst)
321
322
323
324
325
326
327
328
329
330
```

```
                    #### Calculating the variation distance ####

from Cards_mod import *
import operator as op
from math import *
import matplotlib.pyplot as plt

## Help functions

# Defining the binomial coefficient
def ncr(n, r):
    if n < r:
        return 0
    else:
        r = min(r, n-r)
        numer = reduce(op.mul, xrange(n, n-r, -1), 1)
        denom = reduce(op.mul, xrange(1, r+1), 1)
        return numer//denom

# Defining the Eulerian numbers
def A(n, m):
    res = 0
    for k in range (0, m+1):
        res += (-1)**k * (m-k)**n * ncr(n+1, k)
    return res

## Variation distances

# Variation distance for an a-shuffle
def var_dis(k, n = 52, a = 2):
    res = 0
    for r in range(1, n+1):
        res += A(n, r) * abs(ncr(n-r+(a**k), n)- float(a**(n*k))/factorial(n))
    return res/float(2*(a**(n*k)))

# Variation distance, using a list of rising sequences
def var_dis_seq(lst):
    res = 0
    n = len(lst)
    for i in range(0,n):
```

```
41          lst[i] = lst[i]/5000.
42      for r in range(1, n+1):
43          res += A(n, r) * abs(float(lst[r-1])/A(n,r) - 1./factorial(n))
44      return res/2.
45
```

```
                #### Doyle's solitaire game ####

from Cards_mod import *

# Function for numbering the cards
def Card_num(card):
    ranks = 13
    if card.suit == 0:
        return card.rank + 26
    elif card.suit == 1:
        return card.rank + 39
    elif card.suit == 2:
        return ranks - card.rank + 14
    elif card.suit == 3:
        return ranks - card.rank + 1

# The solitaire game
def Soliltaire(res = 1, x = 7):
    a_lst = []
    b_lst = []
    for i in range(0, res):
        deck = Deck()
        A = Deck()
        A.empty()
        B = Deck()
        B.empty()
        C = Deck()
        C.empty()
        deck.rshuffle(x)
        while (A.count() < 26) and (B.count() < 26):
            while deck.count() > 0:
                a = A.count()
                b = B.count()
                hand = deck.pop_card()
                num = Card_num(hand)
                if num == a + 1:
                    A.add_card(hand)
                elif num == b + 27:
                    B.add_card(hand)
                else:
```

```
                    C.add_card(hand)
            C.move_cards(deck, C.count())
        if A.count() == 26:
            a_lst.append('a')
        elif B.count() == 26:
            b_lst.append('b')
    a = len(a_lst)
    b = len(b_lst)
    print 'A: ', 100*float(a)/(a+b), '%, B: ', 100*float(b)/(a+b), '%'
```

```
                    #### The tests for shuffling ####

from math import *
from Cards_mod import *

# Function gives a number to a card
def Card_num(card):
    if card.suit == 0:
        return card.rank
    elif card.suit == 1:
        return card.rank + 13
    elif card.suit == 2:
        return card.rank + 26
    elif card.suit == 3:
        return card.rank + 39

# Counts the amount of rising sequences in a deck
def RiseSeq(deck):
    res = 0
    count = 0
    hulp1 = Deck()
    hulp1.empty()
    hulp2 = Deck()
    hulp2.empty()
    deck.reverse()
    while deck.count() > 0:
        n = deck.count()
        for i in range(0, n):
            card = deck.pop_card()
            num = Card_num(card)
            if num == count + 1:
                hulp1.add_card(card)
                count += 1
            else:
                hulp2.add_card(card)
        h = hulp2.count()
        hulp2.move_cards(deck, h)
        res += 1
    return res

```

```
# Top-card test for casino shelf-shuffle
def TopShelf(m, n = 1):
    res = 0
    for i in range(0, n):
        deck = Deck()
        deck.shelf(m)
        card = deck.pop_card()
        if Card_num(card) == 52:
            res += 1
        if i%1000 == 0:
            print i/1000
    return res

# Top-card test for casino hand-shuffle
def TopCas(n = 1):
    res = 0
    for i in range(0, n):
        deck = Deck()
        deck.casino()
        card = deck.pop_card()
        if Card_num(card) == 52:
            res += 1
        if i%1000 == 0:
            print i/1000
    return res
```

```
                    #### Computing transition matrices ####

from math import *
from numpy import *

# Transition matrix for n = 52
mat = [[1.49e-8, 3.87e-7, 4.84e-6, 3.87e-5, 2.23e-4, 9.8e-4, 3.43e-3, 9.8e-3, 2.33e-2
        [0, 1.49e-8, 3.87e-7, 4.84e-6, 3.87e-5, 2.23e-4, 9.8e-4, 3.43e-3, 9.8e-3, 2.33
        [0, 0, 1.49e-8, 3.87e-7, 4.84e-6, 3.87e-5, 2.23e-4, 9.8e-4, 3.43e-3, 9.8e-3, 2
        [0, 0, 0, 1.49e-8, 3.87e-7, 4.84e-6, 3.87e-5, 2.23e-4, 9.8e-4, 3.43e-3, 9.8e-3
        [0, 0, 0, 0, 1.49e-8, 3.87e-7, 4.84e-6, 3.87e-5, 2.23e-4, 9.8e-4, 3.43e-3, 9.8
        [0, 0, 0, 0, 0, 1.49e-8, 3.87e-7, 4.84e-6, 3.87e-5, 2.23e-4, 9.8e-4, 3.43e-3,
        [0, 0, 0, 0, 0, 0, 1.49e-8, 3.87e-7, 4.84e-6, 3.87e-5, 2.23e-4, 9.8e-4, 3.43e-
        [0, 0, 0, 0, 0, 0, 0, 1.49e-8, 3.87e-7, 4.84e-6, 3.87e-5, 2.23e-4, 9.8e-4, 3.4
        [0, 0, 0, 0, 0, 0, 0, 0, 1.49e-8, 3.87e-7, 4.84e-6, 3.87e-5, 2.23e-4, 9.8e-4,
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 1.49e-8, 3.87e-7, 4.84e-6, 3.87e-5, 2.23e-4, 9.8e-
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1.49e-8, 3.87e-7, 4.84e-6, 3.87e-5, 2.23e-4, 9.
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1.49e-8, 3.87e-7, 4.84e-6, 3.87e-5, 2.23e-4,
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1.49e-8, 3.87e-7, 4.84e-6, 3.87e-5, 2.23e
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1.49e-8, 3.87e-7, 4.84e-6, 3.87e-5, 2.
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1.49e-8, 3.87e-7, 4.84e-6, 3.87e-5,
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1.49e-8, 3.87e-7, 4.84e-6, 3.87e
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1.49e-8, 3.87e-7, 4.84e-6, 3.
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1.49e-8, 3.87e-7, 4.84e-6,
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1.49e-8, 3.87e-7, 4.84e
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1.49e-8, 3.87e-7, 4.
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1.49e-8, 3.87e-7,
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1.49e-8, 3.87e
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1.49e-8, 3.
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1.49e-8,
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1.49e
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1.
        [1.49e-8, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        [3.87e-7, 1.49e-8, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        [4.84e-6, 3.87e-7, 1.49e-8, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        [3.87e-5, 4.84e-6, 3.87e-7, 1.49e-8, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        [2.23e-4, 3.87e-5, 4.84e-6, 3.87e-7, 1.49e-8, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        [9.8e-4, 2.23e-4, 3.87e-5, 4.84e-6, 3.87e-7, 1.49e-8, 0, 0, 0, 0, 0, 0, 0, 0,
        [3.43e-3, 9.8e-4, 2.23e-4, 3.87e-5, 4.84e-6, 3.87e-7, 1.49e-8, 0, 0, 0, 0, 0,
        [9.8e-3, 3.43e-3, 9.8e-4, 2.23e-4, 3.87e-5, 4.84e-6, 3.87e-7, 1.49e-8, 0, 0, 0
```

```
        [2.33e-2, 9.8e-3, 3.43e-3, 9.8e-4, 2.23e-4, 3.87e-5, 4.84e-6, 3.87e-7, 1.49e-8
        [4.66e-2, 2.33e-2, 9.8e-3, 3.43e-3, 9.8e-4, 2.23e-4, 3.87e-5, 4.84e-6, 3.87e-7
        [7.92e-2, 4.66e-2, 2.33e-2, 9.8e-3, 3.43e-3, 9.8e-4, 2.23e-4, 3.87e-5, 4.84e-6
        [0.115, 7.92e-2, 4.66e-2, 2.33e-2, 9.8e-3, 3.43e-3, 9.8e-4, 2.23e-4, 3.87e-5,
        [0.144, 0.115, 7.92e-2, 4.66e-2, 2.33e-2, 9.8e-3, 3.43e-3, 9.8e-4, 2.23e-4, 3.
        [0.155, 0.144, 0.115, 7.92e-2, 4.66e-2, 2.33e-2, 9.8e-3, 3.43e-3, 9.8e-4, 2.23
        [0.144, 0.155, 0.144, 0.115, 7.92e-2, 4.66e-2, 2.33e-2, 9.8e-3, 3.43e-3, 9.8e-
        [0.115, 0.144, 0.155, 0.144, 0.115, 7.92e-2, 4.66e-2, 2.33e-2, 9.8e-3, 3.43e-3
        [7.92e-2, 0.115, 0.144, 0.155, 0.144, 0.115, 7.92e-2, 4.66e-2, 2.33e-2, 9.8e-3
        [4.66e-2, 7.92e-2, 0.115, 0.144, 0.155, 0.144, 0.115, 7.92e-2, 4.66e-2, 2.33e-
        [2.33e-2, 4.66e-2, 7.92e-2, 0.115, 0.144, 0.155, 0.144, 0.115, 7.92e-2, 4.66e-
        [9.8e-3, 2.33e-2, 4.66e-2, 7.92e-2, 0.115, 0.144, 0.155, 0.144, 0.115, 7.92e-2
        [3.43e-3, 9.8e-3, 2.33e-2, 4.66e-2, 7.92e-2, 0.115, 0.144, 0.155, 0.144, 0.115
        [9.8e-4, 3.43e-3, 9.8e-3, 2.33e-2, 4.66e-2, 7.92e-2, 0.115, 0.144, 0.155, 0.14
        [2.23e-4, 9.8e-4, 3.43e-3, 9.8e-3, 2.33e-2, 4.66e-2, 7.92e-2, 0.115, 0.144, 0.
        [3.87e-5, 2.23e-4, 9.8e-4, 3.43e-3, 9.8e-3, 2.33e-2, 4.66e-2, 7.92e-2, 0.115,
        [4.84e-6, 3.87e-5, 2.23e-4, 9.8e-4, 3.43e-3, 9.8e-3, 2.33e-2, 4.66e-2, 7.92e-2
        [3.87e-7, 4.84e-6, 3.87e-5, 2.23e-4, 9.8e-4, 3.43e-3, 9.8e-3, 2.33e-2, 4.66e-2

# Function to calculate distance between a percentage and the uniform density
def va_dis(x):
    return abs(x - 1./52)

# Transition matrix for the example with n = 6
matex = [[0.125 ,0.375 ,0.375 ,0.125 ,0 ,0],
         [0, 0.125, 0.375, 0.375, 0.125, 0],
         [0, 0, 0.125, 0.375, 0.375, 0.125],
         [0.125, 0, 0, 0.125, 0.375, 0.375],
         [0.375, 0.125, 0, 0, 0.125, 0.375],
         [0.375, 0.375, 0.125, 0, 0, 0.125]]
```

B Appendix

Riffle Shuffle $k = 6$															
r	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
b_r	4	12	38	130	363	648	924	1002	835	568	318	104	43	10	1

Riffle Shuffle $k = 7$																
r	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33
b_r	3	21	60	177	401	666	883	978	828	533	278	118	40	12	1	1

Riffle Shuffle $k = 8$														
r	19	20	21	22	23	24	25	26	27	28	29	30	31	32
b_r	7	23	78	188	431	696	912	974	814	481	237	105	36	18

Reversed riffle $k = 6$																		
r	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	25
b_r	1	13	44	123	218	453	669	797	824	658	529	341	171	85	49	18	6	1

Reversed riffle $k = 7$																		
r	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
b_r	2	10	43	93	236	369	595	734	744	727	552	402	259	136	58	26	11	3

Reversed riffle $k = 8$																		
r	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
b_r	4	16	34	93	221	380	564	674	781	723	637	406	251	117	65	23	5	6

Reversed riffle $k = 9$																		
r	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
b_r	6	6	26	54	137	274	434	619	738	765	712	522	350	202	102	33	15	5

Reversed riffle $k = 10$																		
r	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
b_r	3	3	15	44	93	213	393	537	728	786	781	623	378	238	99	46	15	5

Reversed riffle $k = 11$																	
r	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
b_r	16	39	79	191	336	593	745	823	797	626	387	206	109	28	17	6	2

Reversed riffle $k = 12$																		
r	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33
b_r	2	13	27	88	185	374	599	793	871	826	583	355	184	62	28	6	2	2

Reversed riffle $k = 13$																		
r	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33
b_r	1	6	18	43	140	276	473	670	836	832	751	496	267	137	40	10	3	1

Reversed riffle $k = 14$																	
r	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33
b_r	3	9	20	74	184	345	562	788	880	819	638	375	181	96	20	5	1

Reversed riffle $k = 15$																	
r	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33
b_r	3	3	18	39	111	260	518	752	918	856	723	438	231	96	26	6	2

Reversed riffle $k = 16$																		
r	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34
b_r	1	2	17	30	91	190	415	698	866	938	788	500	278	110	49	20	5	2

Reversed riffle $k = 17$																
r	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34
b_r	2	32	80	193	382	642	843	929	832	559	319	129	38	18	1	1

Reversed riffle $k = 18$															
r	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33
b_r	5	21	52	154	331	650	815	883	840	623	358	180	68	15	5

Reversed riffle $k = 19$																	
r	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35
b_r	5	14	58	145	299	536	816	962	848	665	375	174	70	23	8	1	1

Reversed riffle $k = 20$																
r	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34
b_r	1	10	48	126	288	524	832	873	930	654	387	216	79	23	7	2

Reversed riffle $k = 20$																	
r	18	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35
b_r	1	8	31	117	266	520	788	914	922	670	438	210	82	26	4	2	1

Casino hand-shuffle															
r	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
b_r	9	24	113	256	545	760	923	891	685	441	200	103	36	13	1

Casino shelf-shuffle $m = 5$																		
r	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35
b_r	1	2	10	40	127	275	458	704	815	918	686	530	283	95	42	6	3	2

Casino shelf-shuffle $m = 10$																	
r	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34
b_r	1	2	10	33	85	227	504	713	899	989	690	437	258	107	35	7	3

Casino shelf-shuffle $m = 50$																	
r	18	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35
b_r	1	2	34	91	241	482	705	933	930	748	465	230	106	26	4	1	1

Casino shelf-shuffle $m = 100$																
r	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34
b_r	2	7	27	70	232	455	709	928	935	751	494	254	96	30	8	2

Casino shelf-shuffle $m = 200$																	
r	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34
b_r	1	1	5	35	118	293	474	699	956	891	721	466	235	75	22	6	2