Analysis of General-Shuffle Trick Failure Rates

by consideration of contiguous group members

Tyson Jones

May 16, 2015

The Trick

A mathematician is armed with a sense of showmanship and an ordered deck of n cards, in any arbitary ordering that he can privately recall in the spotlight. Their audience has equipped only a rudimentary understanding of randomness and a naive faith in their ability to randomly riffle-shuffle; a shuffle where the whole deck is split into two piles which are then merged back together by sequential random selection between the bottom cards of each pile.

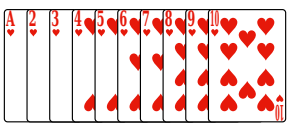
The mathematician forfeits the ordered deck to their audience who riffles the cards several times (three or less, if the mathematician seeks a high chance of success), then finally removes and memorises the top card and inserts it into a randomly chosen location. Convinced the misplacement of the top card was into an already arbitrarily ordered deck and is thus rendered undetectable, the foolish audience returns the deck to the mathematician.

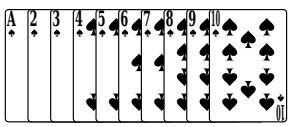
The mathematician then repeatedly removes and reveals the top card from the remaining deck and places it atop the pile which bares this card's previous in the original ordering, otherwise placing the card (face up) in a new pile. Upon the depletion of the deck into piles, should the mathematician possess the luck they've previously calculated is required, only a single pile with one card will remain among the piles of multiple cards, and this card will be that memorised by the awe struck audience. Curtains close.

The Logic

The mathematician succeeds in identifying the misplaced card because the ordered deck has not been riffle-shuffled enough times to nullify its original orderedness; the deck is not at any appropriate *proximity* to being *random*, and as explored, misplacing the top card detectably breaches a remaining but altared orderedness.

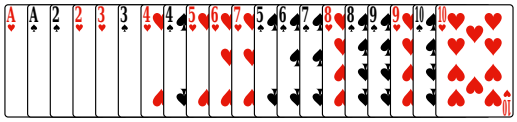
Starting with an ordered deck, splitting the deck into two piles creates two separate ordered, contiguous card groups. For example, we'll imagine the deck was *split* between the hearts ♥ and spades ♠ suits, separating them into our two piles.





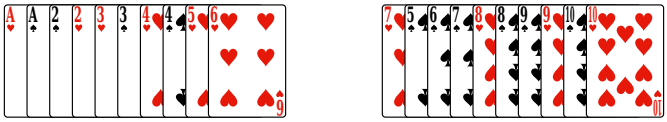
We next observe that the riffle operation preserves the relative order among cards in each pile, since the riffle is performed by sequentially, randomly selecting (by our clumsy thumbs), between the pile's bottom cards, the next card to join the growing, final deck.

One possible outcome of riffling the example piles together follows.

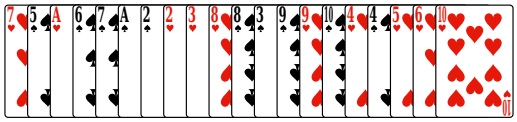


The merged deck is unlikely to conserve the original orderedness between all cards among the two piles, but is gauranteed to preserve the relative order among each pile.

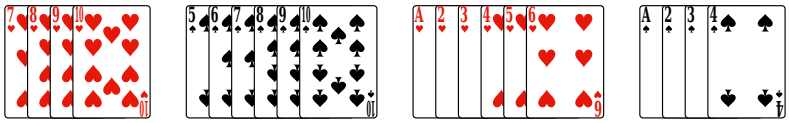
As we repeat the riffle shuffle, we further divide the original deck into progressively smaller collections of relative orderedness, which we'll dub *groups*. For example, after another split, we have two piles each containing two groups.



When we riffle these together, there is an unreliably low chance (intuitive even without quantification) that the relative order among all heart cards will be preserved, similarly for the spades. Instead, the riffle will only preserve the relative order of each pile, which each feature two intertwined groups of relative orderedness. This is best understood by example, where we've riffled our piles together again.



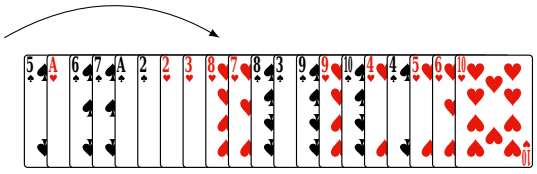
To our naive audience, this may appear randomised, but we can still clearly observe the relative order among the now four groups.



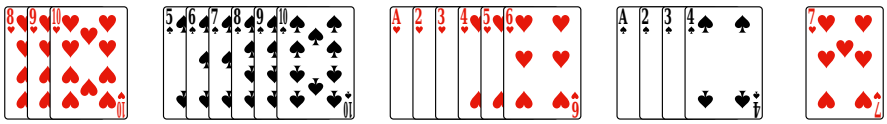
If we continued riffle shuffling, the average size of the groups would continue to decrease while the number of groups increased, until the size of each group is one; every card is in a unique group. We must keep in mind that the group size is a result of preservation of order by the riffle operation, and not coincidental orderedness by truly random operations. I.e. if we riffle-shuffled a million times to result in a perfect ascending deck, we would still consider each card to belong to a unique group, with some coincidental and irrelevant ordering between groups.

Of course, our sensible mathematician would not allow the audience to shuffle the deck the number of times necessary to divide all cards among unique groups, for this renders the trick impossible (or at least, extraordinarily unlikely to work). The mathematician can identify the top card when misplaced into the deck because it is likely to pass the proceeding card in its group, thereby violating the relative order in its group and thus forming its own single-card group.

For example, if we misplace our top card (7♥) anywhere past the proceeding group member (8♥)...



then upon distributing the deck among contiguous piles...



we discover our top card has been revealed in a unique single-card pile.

This example clearly presents another opportunity to fail the trick (beyond shuffling all groups to their minimum size); if the top card does not passing a proceeding card of its group in its misplacement (which is more probable when the card is placed toward the top of the deck) then it will not violate the relative order among its group and therefore cause no change to the final dealt piles. It will remain hidden in its penultimate group and therefore not be revealed in a single-card pile to the mathematician, as seen in our example if 7♥ is placed anywhere before the 8♥.

The Maths

To begin quantifying the probability of success and failure of this inherently stochastic trick, we need first to establish an appropriate stochastic model. In this revision, we'll consider the deck (at any time) to be a series of ...

The Simulation

(Is broken)

```
package tyson.jones.trickfailure;

public class FailureSimulation {

    /* total playing cards, > 2 on account of sampleSplit */
    static final int DECK_SIZE = 52;

    /* list of number-of-shuffles to simulate (serially) */
    static final int[] SHUFFLES = {1, 2, 3, 4, 5};

    /* number of times to simulate trick for each shuffle number */
    static final int SIM_TRIALS = 1000000;

    /* number of simulations after which to print progress
     * (for each shuffle number) */
    static final int PRINT_THRESHOLD = SIM_TRIALS / 10;
```

```
public static void main(String[] argv) {

    // perform simulation for each shuffle-number
    for (int shuffles: SHUFFLES) {
        int total_fails = 0;
        int print_counter = 0;

        // for each shuffle-number, perform the many simulations
        for (int sim=1; sim < 1 + SIM_TRIALS; sim++) {

            // print at designated progress points
            if (++print_counter >= PRINT_THRESHOLD) {
                print_counter = 0;
                System.out.print(100*sim/SIM_TRIALS + "% ");
            }

            // prepare the playing cards
            int[] cards = new int[DECK_SIZE];
            for (int i=0; i < DECK_SIZE; i++) {
                cards[i] = i + 1;
            }

            // perform the shuffles
            for (int shuffle = 0; shuffle < shuffles; shuffle++) {

                int[] new_cards = new int[DECK_SIZE];
                int split_index = sampleSplit();
                int left = 0;
                int right = split_index;

                // merge splits by sequential pile section
                for (int z=0; z < DECK_SIZE; z++) {
                    double prob_left = (split_index - left)/ (double)
                            (DECK_SIZE - right + split_index - left);
                    if (Math.random() <= prob_left) {
                        new_cards[z] = cards[left];
                        left += 1;
                    } else {
                        new_cards[z] = cards[right];
                        right += 1;
                    }
                }

                cards = new_cards;
            }

            // misplace top card into deck (binomially)
            int top_card = cards[0];
            int insert_index = sampleInsert();
            for (int y=0; y < insert_index; y++) {
                cards[y] = cards[y + 1];
            }
            cards[insert_index] = top_card;
```

```
                // form piles of contiguous cards dealt from top
                int[] piles = new int[DECK_SIZE];
                int[] sizes = new int[DECK_SIZE];
                for (int card: cards) {
                    for (int j=0; j < DECK_SIZE; j++) {
                        if (card == piles[j] + 1) {
                            piles[j] = card;
                            sizes[j] += 1;
                        }
                    }
                }

                // detect failure by multi single-cards, or no top-card pile
                boolean found = false;
                boolean failed = false;
                for (int k = 0; k < DECK_SIZE; k++) {
                    if (sizes[k] == 1) {
                        if (piles[k] == top_card)
                            found = true;
                        else
                            failed = true;
                    }
                }
                if (!found)
                    failed = false;

                // record failure
                if (failed)
                    total_fails++;
            }

            System.out.println(shuffles + " shuffle: p = " +
                            (total_fails/(double) SIM_TRIALS));
        }

    }

    /* samples a binomial distribution */
    static int getBinomial(int n, double p) {
        int x = 0;
        for(int i = 0; i < n; i++)
            if(Math.random() < p)
                x++;
        return x;
    }

    /* samples binomial for rand num in [2, 50] */
    static int sampleSplit() {
        return 2 + getBinomial(DECK_SIZE - 2, 0.5);
    }

    /* samples binomial for rand num in [1, 50] */
    static int sampleInsert() {
        return 1 + getBinomial(DECK_SIZE - 1, 0.5);
    }
```

```
}
```