Internal Report 2012–9 August 2012

Universiteit Leiden

Opleiding Informatica

Counting Classes of Klondike Solitaire Configurations

Johan de Ruiter

MASTER'S THESIS

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Counting Classes of Klondike Solitaire Configurations

Johan de Ruiter, johan.de.ruiter@gmail.com

August 23, 2012

Abstract

Klondike Solitaire – also known as Patience – is a well-known single player card game. We studied several classes of Klondike Solitaire game configurations. We present a dynamic programming solution for counting the number of "unplayable" games. This method is extended for a subset of games which allow exactly one move. With an algorithm based on the inclusion-exclusion principle, symmetry elimination and a trade-off between lookup tables and dynamic programming we count the number of games that cannot be won due to a specific type of conflict. The size of a larger class of conflicting configurations is approximated with a Monte Carlo simulation. We investigate how much gameplay is limited by the stock. We give a recursion and show that Pfaff-Fuss-Catalan is a lower bound. We consider trivial games and report on two remarkable patterns we discovered.

Preface

The research presented in this thesis was done in pursuit of a Master of Science degree in Computer Science from Leiden University in the Netherlands. The topic *Klondike Solitaire* was suggested to me by my supervisor, dr. Walter Kosters.

Originally I planned to focus my efforts on empirically determining bounds on the fraction of solvable Klondike Solitaire games for several different rule sets, but I was ultimately much more interested in the enumerative combinatorial aspects of the game. And so it came to pass, as other questions came to mind, and branched, and branched, that the end result turned out quite different.

I have been into algorithms ever since I learned of the Dutch Olympiad in Informatics, back in high school. For my 17th birthday my parents gave me a copy of Steven S. Skiena's *The Algorithm Design Manual* [1]. Especially Skiena's *War Stories* were a delight to read. I was fascinated by how seemingly too computationally intensive problems could be broken down with clever algorithms, datastructures and optimizations. Now, it would almost be sacreligious to compare my own work with that of Skiena, but I hope to have at least at times captured the spirit of the War Stories. This should become most apparent in the chapter *Counting Blocked Klondike Solitaire Deals*, where we pile a variety of techniques on top of one another in order to compute a 66 digit number within a reasonable amount of time.

A second influential factor, largely by proxy, has been Richard Bellman's *Dynamic Programming* [2]. In training for the Benelux Algorithm Programming Contest (3rd place 2006, winner 2007 and 2008, 4th place 2009) and the North-western European Regional Contest of the ACM ICPC (3rd place 2008), I completed near to 1,500 programming exercises. These tasks have deeply familiarized me with dynamic programming and its wide range of applications. In the chapter *Counting Unplayable Klondike Solitaire Deals* we present a very fast dynamic programming solution where previously only a much slower algorithm had been applied. The technique also appears, in more of a supporting role, elsewhere in the text.

During the later stages of my research I became increasingly interested in Generating Functions (a great resource is Wilf's *Generatingfunctionology* [3]) and their possible applications to the chapter *Counting No-braider Klondike Solitaire Deals*. Unfortunately, I have not yet been able to successfully apply them.

Acknowledgements

This work was done under the supervision of dr. Walter Kosters of LIACS, Leiden University. I want to thank Walter for our effective brainstorm sessions and for all the fun we had. Let me also express my gratitude to my second supervisor, dr. Hendrik Jan Hoogeboom and to dr. Jeannette de Graaf, both of LIACS, Leiden University. I thank Maarten Löffler for proofreading.

Next, I would like to thank my parents, Hans and Heleen, for their love, for their support, for their patience. I want to thank Shani for being the best possible influence. Thank you, Misha and Thomas, for sharing my passion for algorithm contests and problem solving, and for your great efforts to make us excel, both as individuals and as a team. Thank you, Martin, Harm, Arjan, Mirjan, Maarten, Daniel, for always sticking around.

Thank you, everyone with whom I had the pleasure of exchanging ideas throughout the years. Our diversions have been most valuable to me.

Contents

1 Introduction to Klondike Solitaire

Klondike Solitaire, sometimes referred to as *Patience* or simply *Solitaire* is such a well-known game that it hardly requires an introduction. For one thing, the game has been shipped with every version of Windows since Windows 3.0 [4], and of Windows 7 alone, over 525 million licenses have been sold [5].

Interestingly enough though, it has been said that "it is one of the embarrassments of applied mathematics that we cannot determine the odds of winning the common game of solitaire" [6]. It has been empirically determined that at least 82% and at most 91.44% of Klondike Solitaire games have winning solutions [7].

1.1 Rules and Terminology

When we speak of the *standard* rules of Klondike Solitaire, we are referring to the default rules in Solitaire for Windows [8]. The terminology used differs slightly between different articles in the literature, so we will specify the definitions used throughout this text.

For reference, Figure 1 shows a starting configuration of a game of Solitaire in Windows 7.

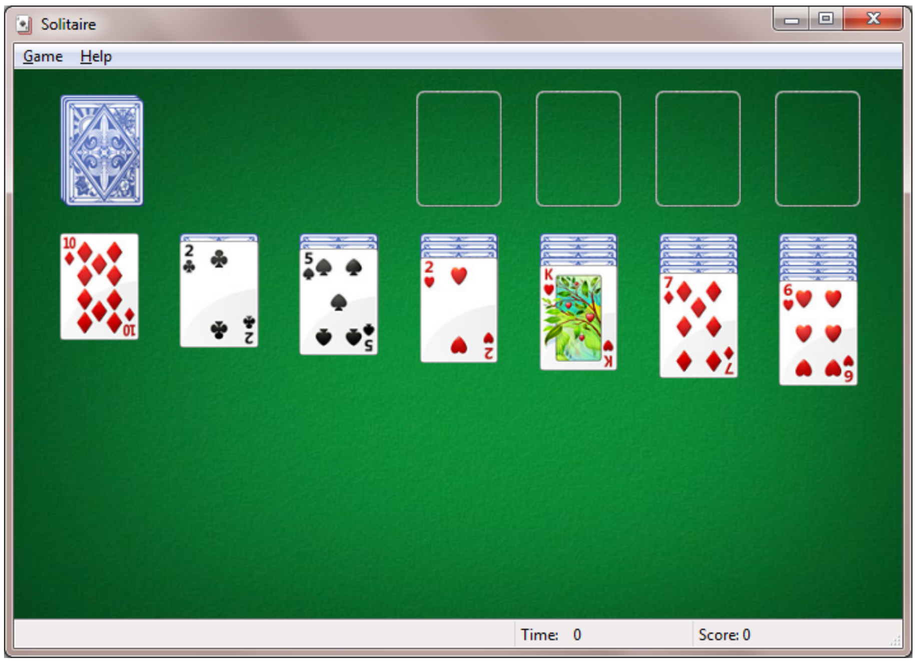

Figure 1: A screenshot of Solitaire in Windows 7 [8].

Klondike Solitaire is played with a regular deck of 52 playing cards. Such a deck contains cards of four *suits*: the *red* suits *hearts* ($\heartsuit$) and *diamonds* ($\diamondsuit$), and the *black* suits *clubs* ($\clubsuit$) and *spades* ($\spadesuit$). There are thirteen cards of each suit, commonly named (or *ranked*) *ace* (A), 2, 3, 4, 5, 6, 7, 8, 9, 10, *jack* (J), *queen* (Q) and *king* (K). In the context of Solitaire they might as well be regarded as numbers from 1 through 13.

The *foundation* – the area in the top right – is where cards can be stacked upon cards of the same suit, strictly increasing by one each time and starting with the ace. An ace can be put only at an empty location. The goal in Klondike Solitaire is to move all cards to the foundation.

The lower region of the screen is called the *tableau*. At the start of the game there are seven *piles* of cards, which have heights 1 through 7. For each pile the top card (i.e., the card dealt last) is dealt face up, while the other cards are dealt face down. Whenever the topmost face down card of a pile is exposed, it is turned over. Cards that face down are never moved around. A card may be moved on top of a face up card on the tableau, provided that the colors or their suits are different and that the rank of the moving card is one lower than the rank of the other card. One can also move groups around on the tableau, provided that the bottom most card (i.e., the one that will be placed directly on top of another pile) is of a different suit color and has rank one lower than the card it is moved upon. A king might be moved to an empty pile, as long as afterwards there are at most seven piles.

At this point different rule sets start to diverge. Bjarnason, Tadepalli and Fern [7] hold true to the rules of Solitaire in Windows 7, by allowing the player to move any number of topmost face up cards from a single pile at once, while Yan, Diaconis, Rusmevichientong and Van Roy [6] expressly state that when moving multiple cards at once, all face up cards from the *source* pile must be moved together. Within this text this distinction is irrelevant.

In the upper left part of the screen we find the *stock*. At the start of the game it consists of 24 cards facing down. The player can flip the cards of the stock over, three at a time, onto a new pile; the *waste*. The topmost card of the waste can be played to the foundation or to the tableau if the other rules do not prevent this. Once the stock is empty, the waste may be turned upside down at the stock location and reused. Most commonly this action may be performed an unlimited number of times (Bjarnason et al. [7]), although some rule sets limit the player to cycle through the stock only once or three times.

Solitaire for Windows 7 has an option to flip cards per one rather than per three. Where relevant we will indicate which variation we consider.

2 Counting Unplayable Klondike Solitaire Deals

It was reported by Latif [9] that *unplayable* games in Klondike Solitaire, i.e., games in which not a single move can be made, amount to approximately 0.25% of all possible games. Donkersteeg and Kosters reported [10] that they managed to compute the exact ratio of unplayable Klondike Solitaire games in about one minute on a 3 GHz processor with a brute force approach and using smart optimizations (≈ 200 lines of C++ code). We confirm the value reported in [10] with a dynamic programming solution that runs in mere milliseconds and consists of only a couple of lines of C++ code.

Our algorithm scales linearly in both time and space for larger instances of the problem (more cards per suit, more piles on the table and more available cards in the stock), while it is expected that the approach followed by Donkersteeg and Kosters will quickly cease to be feasible as the problem size increases.

2.1 Conditions and Campaign Plan

As laid out in [10], a necessary and sufficient set of conditions for a game to be unplayable is:

- There are no aces among the accessible stock cards.
- There are no aces among the accessible tableau cards.
- There are no two accessible cards on the tableau of suits with a different color and with ranks differing by one.
- There is no accessible card in the stock which has a rank that is one less than an accessible tableau card of a suit of the opposite color.

In the standard version of Klondike Solitaire the number of accessible cards on the tableau at the time of dealing is seven and the number of accessible cards in the stock is eight. The number of ranks per suit in a regular deck of playing cards is thirteen. The number of accessible cards in the stock follows from the fact that there are 24 cards in the stock, but only every third card can be accessed. Under the circumstances it is irrelevant whether one is allowed to go through the stock a limited or an unlimited number of times.

When there are $piles$ cards accessible on the tableau and $stock$ cards accessible in the stock, the ratio of unplayable games equals the number of ways one can choose $piles$ cards to be accessible on the tableau and $stock$ cards to be accessible in the stock, while satisfying the above conditions, divided by the total number of ways one can choose $piles$ cards to be accessible on the table and $stock$ cards to be accessible on the stock. It is the numerator $N(ranks, piles, stock)$ that we will compute with a dynamic programming approach. With a deck size of $4 \times ranks$, the denumerator equals:

$$\frac{(4 \times ranks)!}{piles! \times stock! \times (4 \times ranks - piles - stock)!}$$

2.2 A Dynamic Programming Solution

We define $M(ranks, piles, stock, r, b)$ to be number of ways one can choose $piles$ cards to be accessible on the tableau and $stock$ cards to be accessible in the stock, using a deck with size $4 \times ranks$, with r red cards of rank $ranks$ and b black cards of rank $ranks$, while satisfying the conditions stated in the previous section. From the fact that there are two cards for each combination of rank and suit color, it follows that:

$$N(ranks, piles, stock) = \sum_{r=0}^{2} \sum_{b=0}^{2} M(ranks, piles, stock, r, b)$$

The function M allows for the following recursive definition:

$$M(n, p, s, r, b) = \sum_{r_p=0}^{r} \sum_{b_p=0}^{b} \sum_{r'=0}^{2} \sum_{b'=0}^{2} 2^f c M(n-1, p - p_r - p_b, s - s_r - s_b, r', b')$$

for $n, p, s \geq 2$ and $0 \leq r, b \leq 2$, and 0 otherwise, except $M(0, 0, 0, 0, 0) = 1$ and $M(1, 0, 0, 0, 0) = 1$. We define $s_r = r - p_r$, $s_b = b - p_b$ and $f = f_1 + f_2$, with:

$$f_1 = \begin{cases} 1 & \text{if } p_r = 1 \vee s_r = 1 \\ 0 & \text{otherwise} \end{cases}$$

$$f_2 = \begin{cases} 1 & \text{if } p_b = 1 \vee s_b = 1 \\ 0 & \text{otherwise} \end{cases}$$

Also:

$$c = \begin{cases} 1 & \text{if } (r' = 0 \vee p_b = 0) \wedge (b' = 0 \vee p_r = 0) \\ 0 & \text{otherwise} \end{cases}$$

We will discuss the rationale behind the recursive definition of M. Any possible choice of p and s cards from those cards with rank at most n (except the aces, and satisfying the previously specified conditions), containing r red cards of rank n and b black cards of rank n, can be built from valid choices of cards with rank at most $n-1$. We should distinguish between red cards of rank n that have been selected for the stock and those that have been selected for the table, say $r = p_r + s_r$. Likewise, $b = p_b + s_b$. Logically, this leaves $p - p_b - p_r$ cards to be picked for the piles and $s - s_b - s_r$ cards to be picked for in the stock. If we pick no red cards of rank n, or if we select two red cards of rank n to be in the same category (piles or stock), we do not have to choose between them, but if in one or both of the categories we pick one red card of rank n, we *do* need to make a choice, and this would involve a multiplication with a factor 2. This is reflected in the formula by 2^f, which has been broken down into f_1 and f_2, pertaining to the red and the black cards of rank n respectively. An important condition for the validity is that we do not choose any black cards of rank n if we have selected red cards of rank $n-1$ and no red cards of rank n if we have selected black cards of rank $n-1$. This is why we iterate over r' and b' from 0 to 2 and this is where the conditional factor c comes into play. It is 1 if and only if there is no such conflict, and 0 otherwise.

To avoid choosing aces, we set $M(1,0,0,0,0)$ to 1 and start iterating over n at $n = 2$.

Both the space and time complexity of this algorithm are $O(rps)$, meaning it scales only linearly in each of the dimensions of the problem. As a consequence, we can compute $N(13,7,8)$ exactly in a matter of miliseconds, which is a tremendous improvement over the algorithms previously in existence.

Figure 2 shows a `C++` implementation of the proposed algorithm.

Note that in fact we do not have to iterate over the the number of red cards (respectively black cards) of rank $n-1$. Instead, we could iterate over *none* and *some*. We would still iterate over the number of red cards (respectively black cards) of rank n, but we would aggregate the results for one and two cards as *some*. This would reduce the amount of memory required by a factor $\frac{9}{4}$ and the amount of time spent by a factor $\frac{3}{2}$.

2.2.1 Results

The value we have computed for $N(13,7,8)$ is 72,099,595,172,416, which confirms the result found by Donkersteeg and Kosters [10] – approximately 0.2500186% of all games is unplayable (about one in 400).

The version of Solitaire in Windows Vista and Windows 7 has an option to flip the cards in the stock one at a time, meaning all cards in the stock are accessible. Using $N(13,7,24) = 89{,}367{,}495{,}137{,}280$, we determine that in this case the fraction of unplayable games is $\frac{89{,}367{,}495{,}137{,}280}{52!/(7!\times 24!\times 21!)}$, or approximately 0.0000177% (about one in 5,649,223).

```c
long long N(int ranks, int piles, int stock) {
  long long X=0, M[ranks+1][piles+1][stock+1][3][3];
  memset(M, 0, sizeof(M));
 M[0][0][0][0][0]=M[1][0][0][0][0]=1;

  for(int n=2;n<=ranks;n++)
    for(int p=0;p<=piles;p++)
      for(int s=0;s<=stock;s++)
        for(int r=0;r<=2;r++)
          for(int b=0;b<=2;b++)
            for(int sr=0,pr=r;sr<=r;sr++,pr--)
              for(int sb=0,pb=b;sb<=b;sb++,pb--)
                if(sr+sb<=s&&pr+pb<=p)
                  for(int _r=0;_r<=2;_r++) if(!(_r&&pb))
                    for(int _b=0;_b<=2;_b++) if(!(_b&&pr))
                      M[n][p][s][r][b]+=
                        M[n-1][p-pr-pb][s-sr-sb][_r][_b]<<
                          ((pr==1||sr==1)+(pb==1||sb==1));

  for(int i=0;i<9;i++)
    X+=M[ranks][piles][stock][i/3][i%3];
  return X;
}
```

Figure 2: An implementation of a dynamic programming algorithm to compute $N(ranks, piles, stock)$.

2.3 Single Move Klondike Solitaire Games

By extension, we can distinguish several classes of Klondike Solitaire games in which no more move is possible after any opening move – assuming moves cannot or will not be undone. The classes we will examine more closely are those in which exactly one ace is available – on the tableau or in the stock (*category A* and *category B* respectively) – which can be moved to the foundation, after which no other move can be performed, not even when taking into account the possibly newly available card on the tableau in case the ace was moved from a pile of height 2 or higher, and the potentially newly available cards in the stock in case the ace originated there.

2.3.1 Category A

Let us first explore the scenario in which there is exactly one ace among the seven cards accessible on the table and none elsewhere. There are six possible positions for which playing this ace will expose an extra card.

Only when the ace is located in the pile of height 1, this will not be the case. Furthermore, if the ace is located in the pile of height 1 there should not be a king among the accessible cards, since this would allow for moving a king to the empty pile after the ace has been moved to the foundation.

The ace can be of any of the four suits and the situation is completely symmetrical in this respect. Whichever ace we are dealing with, neither of the two 2's of the same color should be among the remaining cards.

If we define $N(n, p, s, P, S, Q)$ to be the number of ways one can select p cards to be accessible on the table and s cards to be accessible in the stock from a deck with ranks from 1 through n, while choosing the cards from the set P among those on the table, choosing the cards from the set S among those in the stock, and not choosing any of the cards from the set Q, the number of configurations in category A equals:

$$\begin{aligned} C_A &= 4 \times 6 \times N(13, 8, 8, \{\heartsuit A\}, \emptyset, \{\diamondsuit A, \clubsuit A, \spadesuit A\}) \times 7! \times 8! \times 36! \\ &+ 4 \times N(12, 7, 8, \{\heartsuit A\}, \emptyset, \{\diamondsuit A, \clubsuit A, \spadesuit A\}) \times 6! \times 8! \times 37! \end{aligned}$$

In a similar vein, we express the number of unplayable games as:

$$C_0 = N(13, 7, 8, \emptyset, \emptyset, \{\heartsuit A, \diamondsuit A, \clubsuit A, \spadesuit A\}) \times 7! \times 8! \times 37!$$

The factorials have been carefully chosen to accomodate the permutations of the accessible cards in the stock, the accessible cards on the tableau (except for the ace) and all inaccessible cards. The factor 4 stems from the symmetry among the aces.

2.3.2 Category B

In the scenario of category B there is exactly one ace present among the eight cards accessible in the stock and none elsewhere. The number of cards accessible in the stock after playing this ace to the foundation depends on the position of the ace in the stock.

Figure 3 shows a schematic representation of the accessible cards in the stock with one ace present. Initially every third card can be accessed. Besides the ace these are the cards displayed in blue. Directly after the ace, which is the twelfth card from the top (1-based), has been moved to the foundation, the eleventh card becomes accessible. All blue cards remain available after the ace has been moved. Now if we decide to cycle through the entire stock and start turning groups of three cards over from the beginning, things have changed. Every card that was at a position $p \equiv 1$

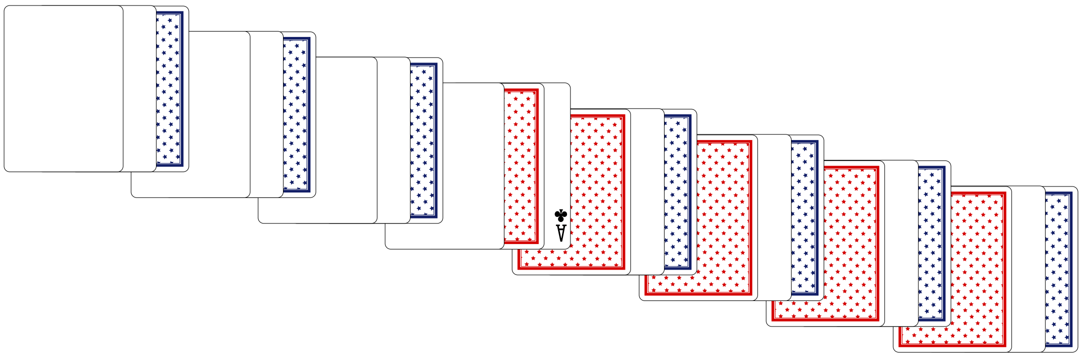

Figure 3: A schematic representation of the accessible cards in the stock when playing an ace from the stock to the foundation. The topmost card in the stock is displayed in the upper left. The cards with a blue back represent the cards accessible before the ace has been played, while the cards with a red back represent the cards newly accessible after the ace has been moved to the foundation. All playing card images used in the figures in this thesis were created by Aguilar [11].

$\pmod 3$ with $p > 12$ before, has been shifted into a position that is a multiple of three. These are the red cards positioned to the right of the ace in Figure 3.

More quantitatively speaking, the ace can be located at any position $3 \times i$ for $i \in [1, 8]$, and for such a position $3 \times i$ there will be $16 - i$ cards accessible in the stock after the ace has been played to the foundation. The number of games involved in this scenario can therefore be expressed as:

$$C_B = 4 \times \sum_{i=1}^{8} N(13, 7, 16-i, \emptyset, \{\heartsuit A\}, \{\diamondsuit A, \clubsuit A, \spadesuit A\}) \times 7! \times (16-i)! \times (28+i)!$$

2.3.3 Results

To compute N for any sets P, S and Q, the program from Figure 2 would need to be modified to take into account cards not to process and multiplication factors f_1, f_2 not to be applied. However, in our case we are solely working with special cases involving aces. Therefore, it will suffice to modify the values to be initialized for $n = 1$ and to start iterating over n at $n = 2$. For category A we initialize $M(1, 1, 0, 1, 0)$ as 1, while for category B we initialize $M(1, 0, 1, 1, 0)$ as 1.

Table 1 lists the queries of the function N that are used to compute the

class	query	result
C_0	$N(13, 7, 8, \emptyset, \emptyset, \{\heartsuit A, \diamondsuit A, \clubsuit A, \spadesuit A\})$	72,099,595,172,416
C_A	$N(12, 7, 8, \{\heartsuit A\}, \emptyset, \{\diamondsuit A, \clubsuit A, \spadesuit A\})$	3,845,001,461,416
	$N(13, 8, 8, \{\heartsuit A\}, \emptyset, \{\diamondsuit A, \clubsuit A, \spadesuit A\})$	40,275,958,345,024
C_B	$N(13, 7, 8, \emptyset, \{\heartsuit A\}, \{\diamondsuit A, \clubsuit A, \spadesuit A\})$	14,069,684,839,712
	$N(13, 7, 9, \emptyset, \{\heartsuit A\}, \{\diamondsuit A, \clubsuit A, \spadesuit A\})$	40,275,958,345,024
	$N(13, 7, 10, \emptyset, \{\heartsuit A\}, \{\diamondsuit A, \clubsuit A, \spadesuit A\})$	98,528,491,384,320
	$N(13, 7, 11, \emptyset, \{\heartsuit A\}, \{\diamondsuit A, \clubsuit A, \spadesuit A\})$	208,273,411,645,728
	$N(13, 7, 12, \emptyset, \{\heartsuit A\}, \{\diamondsuit A, \clubsuit A, \spadesuit A\})$	383,699,702,412,000
	$N(13, 7, 13, \emptyset, \{\heartsuit A\}, \{\diamondsuit A, \clubsuit A, \spadesuit A\})$	620,236,030,645,920
	$N(13, 7, 14, \emptyset, \{\heartsuit A\}, \{\diamondsuit A, \clubsuit A, \spadesuit A\})$	884,347,045,037,280
	$N(13, 7, 15, \emptyset, \{\heartsuit A\}, \{\diamondsuit A, \clubsuit A, \spadesuit A\})$	1,116,793,930,397,280

Table 1: The queries of the function N that are used to compute the number of unplayable games and the number of games of category A and category B.

number of unplayable games and the number of games of category A and category B. When we insert the values listed in the table into the formulas derived earlier, we find that $\frac{C_A}{52!} = 0.00098212\ldots$ and $\frac{C_B}{52!} = 0.00016328\ldots$. Since $C_0 + C_A + C_B$ constitutes a lower bound on the number of games that allow at most one move, we can conclude that at least $0.3645\ldots\%$ of all games allows at most one move.

3 Counting Blocked Klondike Solitaire Deals

Some compositions of the tableau in Klondike Solitaire prohibit successful completion of a game even when playing with very relaxed sets of rules and with complete information about the locations of all the cards. We count the number of games which can not be won due to the presence of a specific type of conflict.

3.1 Type I Blocked Klondike Solitaire Deals

Consider a situation immediately after dealing where a 7 of hearts ($\heartsuit 7$) blocks access to the 8 of spades ($\spadesuit 8$), the 8 of clubs ($\clubsuit 8$) and the 3 of hearts ($\heartsuit 3$) by having been dealt on top of them. The $\heartsuit 7$ cannot be moved to the foundation before the $\heartsuit 3$ has been moved to the foundation, but the $\heartsuit 3$ cannot be moved to the foundation before the $\heartsuit 7$ has been moved elsewhere. The $\heartsuit 7$ can not be moved to an empty position, because it is not a king and it cannot be moved on top of another pile because both of the black 8's are inaccessible as long as the $\heartsuit 7$ has not been moved.

In short, the $\heartsuit 7$ has nowhere to go and the game cannot be won — it is *blocked*. In this chapter we will describe a reasonably fast algorithm to compute the exact fraction of games that cannot be won precisely because this type of conflict is present immediately after the cards have been dealt.

More generally spoken, when within a single pile we have a card with value x of a suit s that blocks access to the two cards with value $x + 1$ of the color that is not the color of the suit s and to a card with value y of the suit s with $y < x$, the game is blocked. Any such (unordered) set of four cards we will call a *primitive blocking set*. The card with value x is said to act as the *lock*. The primitive blocking set mentioned in the example above is illustrated in Figure 4. Here the $\heartsuit 7$ is the lock.

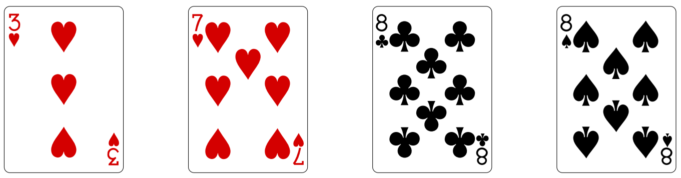

Figure 4: A primitive blocking set with $\heartsuit 7$ as its lock. As the word *set* suggests, the order of the cards does not matter. However, note that $\heartsuit 7$ should be on top of the other cards in the primitive blocking set for the set to block the game.

A regular deck of playing cards has thirteen cards of each of four suits and therefore contains

$$4\sum_{x=2}^{12}(x-1) = 264$$

primitive blocking sets.

In a pile of height h there are $\binom{h}{4}$ subsets of positions that could potentially be occupied by a primitive blocking set. For a given subset of four positions in a pile, there are $3!$ ways the cards of a primitive blocking set can be assigned to these positions such that the game is blocked, because the lock card must be dealt after the other three cards, but otherwise the order is free. It follows that there are

$$3!\binom{h}{4} \times 4\sum_{x=2}^{12}(x-1) = 1{,}584\binom{h}{4}$$

ways to assign a primitive blocking set to positions in a pile of height h.

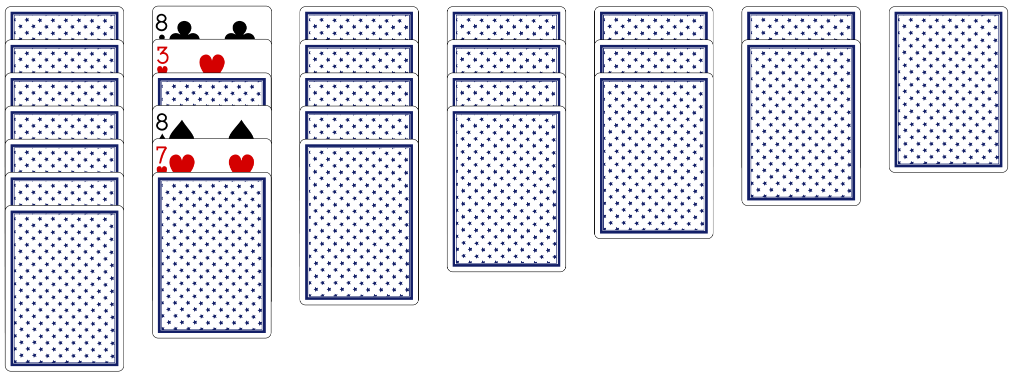

Figure 5: A primitive blocking pattern, as depicted here, is the assignment of the cards of a primitive blocking set to locations in the tableau, such that we can tell the deck is blocking without knowing any of the other cards. All cards in this figure and the figures to follow within this chapter are facing down, even when displayed facing up.

We call the assignment of the cards in a primitive blocking set to positions in a certain pile on the tableau a *primitive blocking pattern*. An example is shown in Figure 5. As primitive blocking patterns need to have their cards assigned within a single pile with height at least 4, the total number of distinct primitive blocking patterns is

$$1{,}584\sum_{h=4}^{7}\binom{h}{4} = 88{,}704.$$

3.2 Inclusion-Exclusion

We are interested in counting the number of deals in which at least one primitive blocking pattern is present. To count this number we use an approach based on the *inclusion-exclusion principle*. The inclusion-exclusion principle states that the number of elements in the union of two finite sets equals the sum of the number of elements for each of the sets minus the number of elements in their intersection, that is, $|A \cup B| = |A| + |B| - |A \cap B|$ for sets A and B.

This statement can be generalized to

$$\begin{aligned}\left|\bigcup_{i=1}^{n} A_i\right| &= \sum_{i=1}^{n} |A_i| - \sum_{i,j:1\leq i<j\leq n} |A_i \cap A_j| \\ &+ \sum_{i,j,k:1\leq i<j<k\leq n} |A_i \cap A_j \cap A_k| - \ldots + (-1)^{n-1}|A_1 \cap \ldots \cap A_n|\end{aligned}$$

where $A_1, A_2, \ldots, A_n$ are sets [12].

The sets we concern ourselves with in this counting problem are $88{,}704$ subsets of the set of all $52!$ possible deals. Each of these $88{,}704$ subsets is associated with a single primitive blocking pattern: it contains these deals that have the four cards of the associated primitive blocking pattern assigned to the designated locations. We will call such a subset a *simple restricted set*. The term *restricted* derives from the fact that all the assigned cards are assigned to locations within the same pile. Primitive blocking patterns or simple restricted sets which have their cards assigned to the same pile, we will call *co-restricted*. Each of the simple restricted sets A_i with $1 \leq i \leq 88{,}704$ has $(52-4)! = 48!$ elements, because four cards are fixed and all others can be freely permuted.

To bluntly evaluate all $2^{88{,}704}$ intersections of subsets of the $88{,}704$ simple restricted sets, however, would clearly be infeasible.

We observe that for many tuples of simple restricted sets the intersection has to be empty. This is always due to one or both of the following reasons:

1. Two or more of the sets in the tuple are associated with primitive blocking patterns which have assigned the same card to a different location.

2. Two or more of the sets in the tuple are associated with primitive blocking patterns which have assigned different cards to the same location.

We can write every intersection clause in the inclusion-exclusion formula as the intersection of four possibly empty sets, each being the intersection of simple restricted sets in the clause that are co-restricted, because there are only four interesting piles. Such a composition by means of intersection of one or more simple restricted sets associated with the same pile, we call a *composite restricted set*.

At the same time, every quadruple of composite restricted sets (here we consider two composite restricted sets distinct if they were formed from a different set of simple restricted sets, even if they contain exactly the same elements) associated with different piles corresponds with exactly one clause in the inclusion-exclusion formula. The sign of a clause corresponding to a quadruple of composite restricted sets can be determined from annotations, provided that we annotate each composite restricted set with an indication of whether it was constructed from an odd or an even number of simple restricted sets.

Any blocking pattern arising from combining primitive blocking patterns that have no conflicts (i.e., they do not assign the same card to a different location on the tableau and they do not assign different cards to the same location on the tableau), regardless if they have their cards located within the same pile or not, we call a *composite blocking pattern*. In particular, composite restricted sets are associated with composite blocking patterns that combine the card-to-location assignments of the individual primitive blocking patterns.

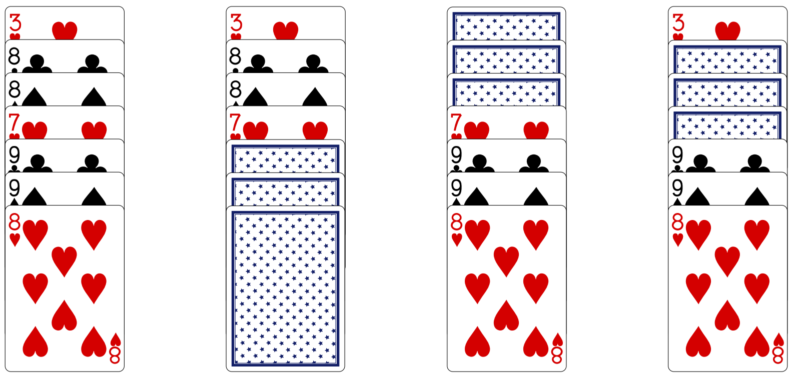
Figure 6: Four piles of height 7 which were extracted from tableaus in which no other cards were assigned to locations. Hence, the leftmost pile can be said to represent a composite blocking pattern, whereas the other piles represent primitive blocking patterns.

Because some composite blocking patterns can be the result of different combinations of primitive blocking patterns (in particular, some composite blocking patterns can be the result both of combining an odd and of combining an even number of primitive blocking patterns), it is best to maintain counts of occurrences of tuples containing a composite blocking pattern and the parity of the number of primitive blocking patterns used to construct it.

As an example, consider Figure 6. Each of the four piles in Figure 6 was extracted from a tableau in which no other cards were assigned to locations. The leftmost pile represents a composite blocking pattern that can be obtained from combining the other piles, which all represent primitive blocking patterns, in three ways: The second pile could be combined with third pile, the second pile could be combined with the fourth pile, or the second pile could be combined with the third *and* the fourth pile.

3.3 Composite Restricted Blocking Patterns

In line with other definitions, with the term *composite restricted blocking pattern* we refer to a composite blocking pattern composed of co-restricted primitive blocking patterns.

We observe that a composite restricted blocking pattern restricted to a pile of height 7 can contain at most four distinct primitive blocking patterns. A pile of height 6 can contain at most three primitive blocking patterns, while in a pile of height 5 there can be only two and in a pile of height 4 there can be only one. This suggests a recursive approach with a recursion depth of at most ten, with an upper bound of evaluating

$$\prod_{h=4}^{7}\prod_{k=0}^{h-3}\binom{1{,}584\binom{h}{4}}{k} \approx 5.6 \times 10^{84}$$

compositions of primitive blocking patterns. Although this is an enormous improvement over having to evaluate $2^{88{,}704}$, it would still be completely infeasible for any modern computer unless we could do a significant amount of backtracking.

We investigate the number of composite restricted blocking patterns to gain more insight into the feasibility. Table 2 shows in how many ways different numbers of primitive blocking patterns can be combined without conflicts within piles of height 4 through 7 as computed by an elaborate enumeration by hand and confirmed by an exhaustive search by a computer.

By multiplying the entries from the row with totals in Table 2 we get a much more reasonable value for the number of composite blocking patterns to evaluate, provided that we precompute all valid composite re-

#patterns \ height	4	5	6	7
0	1	1	1	1
1	1,584	7,920	23,760	55,440
2		21,120	187,440	2,452,560
3			237,600	4,316,400
4				2,280,960
total	1,585	29,041	448,801	9,105,361

Table 2: The number of valid compositions of distinct primitive blocking patterns classified according to the number of primitive blocking patterns used and the height of the pile in which they are used. Empty cells denote zeros.

stricted blocking patterns. The resulting number, 1.9×10^{20}, is somewhat reassuring, although it is still too large.

A very important observation at this point is that once we have combined primitive blocking patterns restricted to the same pile for every pile, we do not actually have to distinguish between all different composite restricted blocking patterns. Rather, we can aggregate those co-restricted composite restricted blocking patterns associated with the same sets of cards (*composite blocking sets*) and built from the same number of primitive blocking patterns, because in combining composite restricted blocking patterns which are not co-restricted, we will never assign different cards to the same location! We will get back to this in Section 3.4.

It would actually suffice to consider only the parity of the number of primitive blocking patterns instead of the number itself. We chose to use the number of primitive blocking patterns in our table for clarity. Note that it is in fact impossible to have three primitive blocking patterns combine into one or to have a composite blocking pattern that can be made both from combining two and from combining four distinct primitive blocking patterns, so it really does not matter what we choose to do.

We will systematically derive the values in Table 2. The patterns discovered can be used to generate the composite blocking sets much faster than by brute force. The row concerning zero primitive blocking patterns is included for completeness. Once we start combining the composite blocking sets, we should also take into account the scenarios in which one or more piles have no primitive blocking patterns in them.

3.3.1 A single primitive blocking pattern

In Table 2, the row concerning a single primitive blocking pattern is computed as described in Section 3.1. Table 3 serves to introduce the notation used in the remainder of this section and visualizes the structure of compositions of primitive blocking sets and the constraints placed upon them. We will call this a *configuration diagram*. The letters $\mathbb{A}$ represent the cards of a single primitive blocking set. The *-symbol indicates which card is the lock of a this primitive blocking set. We will refer to $\mathbb{A}^*$ as the $\mathbb{A}$-*lock*. In the event of multiple letters occupying the same cell, this should be interpreted as a set of roles a card plays in different primitive blocking sets.

	$\cdots$	i	$\cdots$	j	$j+1$	$\cdots$
$\heartsuit$		$\mathbb{A}$		$\mathbb{A}^*$		
$\diamondsuit$						
$\clubsuit$					$\mathbb{A}$	
$\spadesuit$					$\mathbb{A}$	

Table 3: A visualization of a primitive blocking set. The card values i and j are constrained by $1 \leq i < j < 13$.

The contents of the rows with $\heartsuit$ and $\diamondsuit$ can be interchanged, while still signifying a primitive blocking set. Symmetrically, this holds for the contents of the rows with $\clubsuit$ and $\spadesuit$, and similarly the contents of the bottom two rows can be interchanged with the contents of the top two rows. Note, however, that the structure is often invariant under some of these operations. We need to take this into account when we derive cardinalities from configuration diagrams.

Figure 7 is a graphical interpretation of the dihedral group D_4 [13] as it relates to the card suits from our perspective. Two configurations are to be considered equivalent if one can be turned into the other by interchanging the rows according to the D_4 and relabeling the characters of the alphabet used (i.e., $\{\mathbb{A}, \mathbb{B}, \ldots\}$), but leaving lock annotations untouched.

For each configuration diagram we construct, we are interested in how many distinct primitive blocking patterns it represents. One of the factors in this computation is determined by the cardinality of a subgroup of the D_4. In the example depicted in Table 3 this factor is 4.

We can derive the expression for the value in the row with one blocking

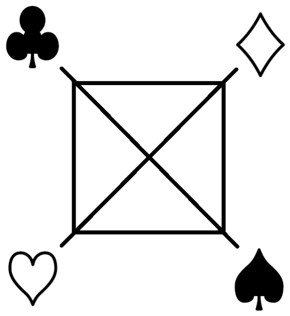

Figure 7: The dihedral group D_4 as it relates to the permutations of card suits.

pattern in Table 2 for the pile of height h from Table 3:

$$3!\binom{h}{4}\sum_{j=2}^{12}\sum_{i=1}^{j-1}4 = 4\times 3!\binom{h}{4}\sum_{j=2}^{12}(j-1) = 1{,}584\binom{h}{4}$$

The factor 4 is the one mentioned above. The factor $3!$ follows from the fact that the three cards indicated with an $\mathbb{A}$ can be dealt in any order before the $\mathbb{A}$-lock. The factor $\binom{h}{4}$ indicates that the primitive blocking set can be asigned to any four of the h locations. Also notice that the number of composite blocking sets, which we really care about, is much smaller; in the case of compositions of one primitive blocking pattern it is just 264, regardless of the pile height.

3.3.2 Compositions of two primitive blocking patterns

There are several ways in which we can combine two primitive blocking patterns. The structure of the primitive blocking sets of the first possibility is shown in Table 4. It is the only way to use two primitive blocking patterns while using only five cards. The expression for the contribution of this composition type to the value of the second row in Table 2 for height $5 \leq h$ is

$$4!\binom{h}{5}\sum_{k=3}^{12}\sum_{j=2}^{k-1}\sum_{i=1}^{j-1}4 = 21{,}120\binom{h}{5}$$

where the factor 4 once again follows from interchanging the suits. The factor $4!$ stems from the fact that the $\mathbb{A},\mathbb{B}$-lock needs to be dealt later than the other four cards, but there are no other requirements other than $1 \leq i < j < k < 13$. The contributions of this type of composition to the row for two primitive blocking patterns in Table 2 for $5 \leq h \leq 7$ are $21{,}120$; $126{,}720$ and $443{,}520$ respectively.

	$\cdots$	i	$\cdots$	j	$\cdots$	k	$k+1$	$\cdots$
$\heartsuit$		$\mathbb{A}$		$\mathbb{B}$		$\mathbb{A}^*, \mathbb{B}^*$		
$\diamondsuit$								
$\clubsuit$							$\mathbb{A}, \mathbb{B}$	
$\spadesuit$							$\mathbb{A}, \mathbb{B}$	

Table 4: A visualization of a composition of two primitive blocking sets using five cards; $1 \leq i < j < k < 13$.

	$\cdots$	i	$\cdots$	j	$\cdots$	k	$k+1$	$\cdots$
$\heartsuit$		$\mathbb{A}$				$\mathbb{A}^*$		
$\diamondsuit$				$\mathbb{B}$		$\mathbb{B}^*$		
$\clubsuit$							$\mathbb{A}, \mathbb{B}$	
$\spadesuit$							$\mathbb{A}, \mathbb{B}$	

Table 5: A visualization of a composition of two primitive blocking sets using six cards; $1 \leq i, j < k < 13$.

Allowing the use of six cards when combining two primitive blocking patterns gives rise only to compositions of the kind displayed in Table 5. The expression of the number of composite blocking patterns involved becomes slightly more complicated. First of all, we need to be careful with the case where $i = j$, because it has more automorphisms than the case where $i \neq j$. Secondly, the number of ways we can order the cards amongst themselves is no longer a simple factorial. Either the $\mathbb{A}$-lock or the $\mathbb{B}$-lock is dealt the last. Therefore, the other lock is dealt earlier, but after the three cards of the same primitive blocking pattern. This means that we can insert the remaining card in any of five positions. The resulting expression is

$$2 \times 3! \times 5 \binom{h}{6} \left(\sum_{k=3}^{12} \sum_{j=2}^{k-1} \sum_{i=1}^{j-1} 4 + \sum_{k=2}^{12} \sum_{j=1}^{k-1} 2 \right) = 60{,}720 \binom{h}{6}$$

and its contribution to Table 2 is $60{,}720$ and $425{,}040$ for $h = 6$ and $h = 7$ respectively.

We can distinguish four different scenarios in which two primitive blocking patterns are combined and together use seven cards. They are displayed in Table 6, Table 7, Table 8, and Table 9.

	$\cdots$	i	$\cdots$	j	$j+1$	$\cdots$	k	$k+1$	$\cdots$
$\heartsuit$		$\mathbb{A},\mathbb{B}$		$\mathbb{A}^*$			$\mathbb{B}^*$		
$\diamondsuit$									
$\clubsuit$					$\mathbb{A}$			$\mathbb{B}$	
$\spadesuit$					$\mathbb{A}$			$\mathbb{B}$	

Table 6: A visualization of a composition of two primitive blocking sets using seven cards; $1 \leq i < j < k < 13$.

	$\cdots$	i	$\cdots$	j	$j+1$	$\cdots$	k	$k+1$	$\cdots$
$\heartsuit$		$\mathbb{A}$		$\mathbb{A}^*,\mathbb{B}$			$\mathbb{B}^*$		
$\diamondsuit$									
$\clubsuit$					$\mathbb{A}$			$\mathbb{B}$	
$\spadesuit$					$\mathbb{A}$			$\mathbb{B}$	

Table 7: A visualization of a composition of two primitive blocking sets using seven cards; $1 \leq i < j < k < 13$.

The structures in Table 6 and Table 7 look nearly identical, but their subtle difference results in a different contribution. In the one in Table 6 one we can deal either the $\mathbb{A}$-lock or the $\mathbb{B}$-lock last. Whichever is chosen, the other lock must be dealt earlier and its three associated cards must be dealt before their lock in any order. Now there are 5×6 ways to place the remaining cards. In contrast, in Table 7 the $\mathbb{A}$-lock cannot be dealt last, because it would mean that the $\mathbb{B}$-lock is not dealt after all the cards in its primitive blocking pattern.

Summing these two contributions, we get:

$$(2+1) \times 5 \times 6 \times 3! \sum_{k=3}^{12} \sum_{j=2}^{k-1} \sum_{i=1}^{j-1} 4 = 2{,}160 \binom{12}{3} = 475{,}200$$

In the computations of the contributions of the last two structures resulting from combining two basic blocking patterns while using seven cards, shown in Table 8 and Table 9, we should use a factor 8 to accomodate the suit symmetry, because all rows are different under all row interchanges and relabelings.

	$\cdots$	i	$\cdots$	j	$\cdots$	k	$k+1$	$k+2$	$\cdots$
$\heartsuit$		$\mathbb{A}$				$\mathbb{A}^*$		$\mathbb{B}$	
$\diamondsuit$								$\mathbb{B}$	
$\clubsuit$				$\mathbb{B}$			$\mathbb{A}, \mathbb{B}^*$		
$\spadesuit$							$\mathbb{A}$		

Table 8: A visualization of a composition of two primitive blocking sets using seven cards; $1 \leq i < k < 12$ and $j \leq k$.

In Table 8 the $\mathbb{B}$-lock cannot be dealt last, so the $\mathbb{A}$-lock must be. Hence, the $\mathbb{B}$-lock must be dealt before the $\mathbb{A}$-lock and the other $\mathbb{B}$ cards must be dealt before the $\mathbb{B}$-lock in any of $3!$ orders. Then there are 5×6 pairs of locations where the remaining $\mathbb{A}$ cards can be placed. The contribution amounts to:

$$5 \times 6 \times 3! \sum_{k=2}^{11} \sum_{j=1}^{k} \sum_{i=1}^{k-1} 8 = 633{,}600$$

	$\cdots$	i	$\cdots$	j	$j+1$	$\cdots$	k	$k+1$	$\cdots$
$\heartsuit$					$\mathbb{A}, \mathbb{B}$		$\mathbb{B}^*$		
$\diamondsuit$					$\mathbb{A}$				
$\clubsuit$		$\mathbb{A}$		$\mathbb{A}^*$				$\mathbb{B}$	
$\spadesuit$								$\mathbb{B}$	

Table 9: A visualization of a composition of two primitive blocking sets using seven cards; $1 \leq i < j < k - 1 < 12$.

In Table 9, either lock can be dealt last. Once again, the other lock must be dealt earlier and the cards of the same primitive blocking pattern are in turn dealt before that in any of $3!$ different orders. Again, the remaining two cards can be placed in 5×6 ways. This results in a contribution of

$$2 \times 5 \times 6 \times 3! \sum_{k=4}^{12} \sum_{j=2}^{k-2} \sum_{i=1}^{j-1} 8 = 2{,}160 \binom{12}{3} = 475{,}200,$$

which gives a total of $2{,}452{,}560$ composite blocking patterns made from two primitive blocking patterns and using seven cards.

Because the highest pile has only seven cards, we do not have to count the number of composite blocking patterns made from two primitive blocking patterns and using eight cards.

3.3.3 Compositions of three primitive blocking patterns

There are three basic structures that together describe all ways to combine three primitive blocking patterns using seven cards or less. The first of these is shown in Table 10. It is an extension of the pattern in Table 4 and uses six cards. There are $\binom{12}{4} = 495$ ways to pick i, j, k and ℓ. The suit symmetry multiplier is 4. The card that serves as the lock for all three primtitive blocking patterns has to be dealt last, but other than that there are no requirements on the order of the cards. The contribution of this type of composition is

$$5!\binom{h}{6}\binom{12}{4} \times 4 = 237{,}600\binom{h}{6}$$

which means $237{,}600$ for the pile of height 6 and $1{,}663{,}200$ for the pile of height 7.

	$\cdots$	i	$\cdots$	j	$\cdots$	k	$\cdots$	ℓ	$\ell+1$	$\cdots$
$\heartsuit$		$\mathbb{A}$		$\mathbb{B}$		$\mathbb{C}$		$\mathbb{A}^*, \mathbb{B}^*, \mathbb{C}^*$		
$\diamondsuit$										
$\clubsuit$									$\mathbb{A}, \mathbb{B}, \mathbb{C}$	
$\spadesuit$									$\mathbb{A}, \mathbb{B}, \mathbb{C}$	

Table 10: A visualization of a composition of three primitive blocking sets using six cards; $1 \leq i < j < k < \ell < 13$.

The structure displayed in Table 11 is a combination of the structures in Table 6 and Table 7. There are $\binom{12}{3}$ ways to choose i, j and k and the suit symmetry multiplier is 4. The card that serves as both the $\mathbb{B}$-lock and the $\mathbb{C}$-lock needs to be dealt last. The $\mathbb{A}$-lock must be dealt after the other cards marked with $\mathbb{A}$, which can be in any order. The remaining two cards are both marked with $\mathbb{B}$ and $\mathbb{C}$ and can be placed in 5×6 ways. This results in a contribution of

$$5 \times 6 \times 3!\binom{12}{3} \times 4 = 158{,}400$$

	$\cdots$	i	$\cdots$	j	$j+1$	$\cdots$	k	$k+1$	$\cdots$
♡		$\mathbb{A}, \mathbb{B}$		$\mathbb{A}^*, \mathbb{C}$			$\mathbb{B}^*, \mathbb{C}^*$		
♢									
♣					$\mathbb{A}$			$\mathbb{B}, \mathbb{C}$	
♠					$\mathbb{A}$			$\mathbb{B}, \mathbb{C}$	

Table 11: A visualization of a composition of three primitive blocking sets using seven cards; $1 \leq i < j < k < 13$.

to the number of composite blocking patterns created from three primitive blocking patterns in the pile of height 7.

	$\cdots$	i	$\cdots$	j	$\cdots$	k	$\cdots$	l	$l+1$	$\cdots$
♡		$\mathbb{A}$				$\mathbb{B}$		$\mathbb{A}^*, \mathbb{B}^*$		
♢				$\mathbb{C}$				$\mathbb{C}^*$		
♣									$\mathbb{A}, \mathbb{B}, \mathbb{C}$	
♠									$\mathbb{A}, \mathbb{B}, \mathbb{C}$	

Table 12: A visualization of a composition of three primitive blocking sets using seven cards; $1 \leq i < k < l < 13$ an $1 \leq j < k$.

The last composite blocking pattern consisting of three primitive blocking patterns is shown in Table 12. If the card that serves as the $\mathbb{A}$-lock and $\mathbb{B}$-lock is dealt last, then there are $3!$ ways to order the cards which are associated with primive blocking pattern $\mathbb{C}$, and there are 5×6 ways to place the remaining two cards. If the card that serves as the $\mathbb{C}$-lock is dealt last, then there are $4!$ ways to order the cards associated with primitive blocking patterns $\mathbb{A}$ and $\mathbb{B}$. After that there are six positions to place the remaining card. This results in the expression

$$(5 \times 6 \times 3! + 6 \times 4!) \sum_{l=3}^{12} \sum_{k=2}^{l-1} \sum_{j=1}^{l-1} \sum_{i=1}^{k-1} 4 = 2{,}2494{,}800$$

which rounds up the corresponding entry in Table 2.

3.3.4 Compositions of four primitive blocking patterns

There is only one type of composition of four primitive blocking sets using only seven cards, illustrated in Table 13, and this is a further extension of the structures in Table 4 and Table 11.

	...	i	...	j	...	k	...	ℓ	...	m	$m+1$	...
$\heartsuit$		$\mathbb{A}$		$\mathbb{B}$		$\mathbb{C}$		$\mathbb{D}$		$\mathbb{A}^*, \mathbb{B}^*, \mathbb{C}^*, \mathbb{D}^*$		
$\diamondsuit$												
$\clubsuit$											$\mathbb{A}, \mathbb{B}, \mathbb{C}, \mathbb{D}$	
$\spadesuit$											$\mathbb{A}, \mathbb{B}, \mathbb{C}, \mathbb{D}$	

Table 13: A visualization of the composition of four primitive blocking sets using seven cards; $1 \leq i < j < k < \ell < m < 13$.

The total number of compositions of four primtitive blocking patterns this structure represents is:

$$6!\binom{12}{5} \times 4 = 2{,}280{,}960$$

3.4 Composite Restricted Blocking Pattern Aggregation

As was briefly mentioned in Section 3.3, different piles do not share locations, so when we combine composite restricted blocking patterns that are not co-restricted, we do not have to take into account the exact locations the cards are assigned to anymore. We only must not assign the same card to more than one pile, meaning that it suffices to check that the underlying composite blocking sets are disjoint for the composite blocking patterns to be combined.

As a result, we can simply aggregate co-restricted composite restricted blocking patterns with the same associated composite blocking set which were constructed from the same number of primitive blocking patterns v (because we need to know the signs in the inclusion-exclusion formula). We assign a weight to a tuple consisting of a number indicating the associated pile, the composite blocking set, and the number v. Table 14 shows how much this reduces the amount of data we need to process

This suggests we may need to perform roughly 2.7×10^{13} evaluations, which, taking into account that each such evaluation and its subsequent registration requires some small, but non-negligible amount, of computation, still takes too long.

Pile height	4	5	6	7
Composite blocking patterns	1,585	29,041	448,801	9,105,361
Composite blocking sets	265	1,145	4,137	21,605

Table 14: The number of composite blocking patterns per pile compared to the number of composite blocking sets.

3.4.1 Quadruples of tuples

The most straightforward way to combine the tuples formed in the aggregation process is to generate every quadruple consisting of one tuple associated with every pile pile, and for each such quadruple compute the contribution to the final answer according to the inclusion-exclusion formula. Each contribution consists of three components:

I The cardinality x of the set of card deals represented by the quadruple. The four tuples in the quadruple tell us *which*, but, more importantly, also *how many* cards have their location fixed. On the remainder of the cards there is no restriction. So, given the four composite restricted blocking sets A, B, C and D, is computed as follows:

$$x = \begin{cases} (52 - |A| - |B| - |C| - |D|)! & \text{if } A \cap B \cap C \cap D = \emptyset \\ 0 & \text{otherwise} \end{cases}$$

II The sign of the contribution. This depends on the parity of the sum of the number of primitive blocking patterns used in the composite blocking sets in the quadruple.

III A multiplier indicating how many compositions each composite blocking set in the quadruple represents as a consequence of aggregation.

3.4.2 Factorial base counter

Because every summand is a multiple of some factorial $k!$ with $k \in [30, 52]$ and these factorials have very large decimal (or even base 2^{64}) representations, which can be relatively cumbersome to add and multiply, we decided to work with a factorial base counter [14].

To avoid having to perform a huge number of carry operations, instead of limiting the coefficient c_k of $k!$ in the factorial base representation to be in $[0, k]$, we allow c_k to be confined to $[-2^{63}, 2^{63} - 1]$. Under these constraints the factorial base numbers do not have a unique representation, but we do not need them to, as we are not performing comparisons, but only additions and subtractions. Once a coefficient c_k overflows 2^{62} or underflows -2^{62}, we make sure to carry over to or borrow from c_{k+1}.

At the end of the computation we perform a clean-up operation to reduce our representation to a canonical factorial base representation starting at the least significant coefficient c_{30} and moving forward, carrying over or borrowing from the preceding digits. Because the outcome will be a positive number, this procedure will terminate.

3.4.3 Bitwise representation and nested loops

If we would nest four loops (one associated with each of the four piles of height at least 4), and use a 52-bit binary representation of the composite blocking sets using 64-bit integers, we could check for each quadruple whether the four composite blocking sets are disjoint in a very limited number of operations. Since we also maintain how many cards each composite blocking set contains and how many primitive blocking patterns were involved in its creation, and because of the use of a factorial base counter, we can process each quadruple very fast. However, as was pointed out in Section 3.4, there are about 2.7×10^{13} such quadruples and even though we can win some time by backtracking when we encounter non-empty intersections of sets iterated over by the outer loops, there are too many quadruples left to evaluate within an acceptable amount of time on a modern desktop computer.

3.4.4 Exploiting symmetry

A significant factor can be won by exploiting the symmetry among the suits, which we already encountered while analyzing the primitive blocking pattern compositions. Say we iterate over the composite blocking sets associated with the pile of height 7 in the outer loop. Two composite blocking sets that are equivalent under the proper suit transformations (i.e., swapping the red suits, swapping the black suits, swapping the black suits with the red suits and combinations thereof), contribute exactly the same to the final answer, so we can aggregate them and give a representative of the equivalence class their total weight.

If we apply this to the pile of height 4, it reduces the computation time by a factor nearly 4, as each of its composite blocking patterns except for the empty set shares its equivalence class with three others. If we apply it to the piles of height 5, 6 and 7, we get closer to the theoretical upper bound 8 (the number of elements in the D_4), because most of their composite blocking sets contain more cards and therefore there are more opportunities to break the symmetry. The number of composite blocking sets to evaluate for the pile of height 7 can be reduced from $21{,}605$ to $4{,}830$, obtaining a factor slightly less than 4.5.

There is even more to gain if we would be able to perform this reduction on combinations of composite blocking sets. As it turns out it is worthwhile to do this for a combinations of two piles, associated with the outer two loops. It pays off to precompute these combinations and iterate over them in one outer loop. We will briefly touch upon this again in Section 3.4.8.

Exploiting the symmetry for combinations of three composite blocking sets in a way that would still improve the time complexity, seems infeasible.

3.4.5 Lookup tables and dynamic programming

Of the four nested loops we will be able to eliminate the inner loop by using a lookup table. We will first analyze how this could work when we choose the pile of height 4 to be associated with the inner loop. In Section 3.4.1 we mentioned the three factors needed to compute the contribution of a quadruple to the final answer. We can easily compute I, the cardinality of the set of card deals denoted by the quadruple, because it is almost always the same. Only for the empty set the number of fixed cards will be the same as for the first three composite blocking sets in the quadruple; for all others the number of fixed cards will be 4 less. We can also easily compute II, the sign of the contribution, because the parity of the number of primitive blocking patterns used is either the same (again, only in the case of the empty set) or the opposite of the parity for the first three composite blocking sets in the quadruple. Also III, the number of compositions of primitive blocking patterns each composite blocking set associated with the pile of height 4 represents, is always the same, again with the empty set as a special case.

Finally, we need to compute the number of composite blocking sets that do not intersect with the union of the first three sets in the quadruple. Recalling the structure of the primitive blocking sets, as shown in Table 3, we present a *dynamic programming* approach to quickly determine the count.

We split the task up into four parts. In each part we consider a different suit as the one containing the lock card. Figure 8 illustrates a partial configuration that is represented by some choice of composite blocking sets in the outer three loops. The red cards are still undetermined and we wonder

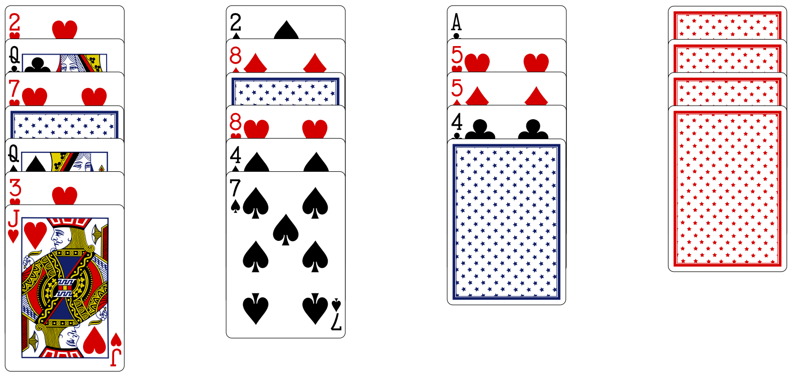

Figure 8: A partial configuration which is represented by some choice of composite blocking sets for the piles of heights 5, 6 and 7.

n	1	2	3	4	5	6	7	8	9	10	11	12	13
$\heartsuit$		x	x		x		x	x			x		
$\diamondsuit$					x			x					
$\clubsuit$	x			x								x	
$\spadesuit$		x		x			x					x	
$s_\heartsuit(n)$	0	1	1	1	2	2	3	3	3	4	5	5	6
$t_\heartsuit(n)$	0	0	0	1	1	1	1	1	4	8	8	13	13
$s_\diamondsuit(n)$	0	1	2	3	4	4	5	6	6	7	8	9	10
$t_\diamondsuit(n)$	0	1	1	4	4	4	9	9	15	22	22	31	31
$s_\clubsuit(n)$	0	0	1	2	2	3	4	5	6	7	8	9	9
$t_\clubsuit(n)$	0	0	1	1	3	3	3	8	14	14	22	22	22
$s_\spadesuit(n)$	0	1	1	2	2	3	4	4	5	6	7	8	8
$t_\spadesuit(n)$	0	0	1	1	3	3	3	7	12	12	19	19	19

Table 15: A dynamic programming solution for computing the number of primitive blocking sets that can be made without using the cards marked by x, applied to the partial configuration of Figure 8.

how many possible choices of composite blocking sets there are left for the pile with height 4.

In Table 15 we demonstrate the dynamic programming approach to answering this question. The letter x marks a card that is already in use. Focusing on the suit of hearts, $s_\heartsuit(n)$ equals the number of hearts cards with value smaller than n that have not been used. It is 0 for $n = 1$ and for $n > 1$ it is $s_\heartsuit(n-1) + 1$ if the $n - 1$ of hearts has not been used and

$s_\heartsuit(n-1)$ otherwise. Furthermore, $t_\heartsuit(n)$ equals the number of primitive blocking sets for which the value of the lock card is at most n. It is 0 for $n=1$. When neither the $\clubsuit(n+1)$ nor the $\spadesuit(n+1)$ is used, $t_\heartsuit(n) = t_\heartsuit(n-1) + s_\heartsuit(n)$, otherwise it is $t_\heartsuit(n-1)$. We treat the case where no primitive blocking pattern is present in the pile of height 4 as a special case. We can fill the table in time linear in the number of cards per suit, whereas a trivial algorithm would take $O(n^2)$ time.

We can preprocess all these values and store them in a large lookup table, so we can extract them in $O(1)$ time when we need them. However, we cannot store the answer for every 52-bit integer, because of the immense amount of memory required and the enormous time needed to fill the table.

Looking back at the dynamic programming solution, notice that to compute the number of primitive blocking sets with the lock card being of a specific suit, we only require information about the unused cards of that suit and the values for which both cards of the other color have not been used (the *bitwise OR* proves to be a useful operation here). This means we can make a 26-bit addressed lookup table instead, at the cost of having four lookups instead of one. In fact, our lookup table only requires 2^{24} entries, as the presence of the king of our suit and presence of the aces of the other color are irrelevant.

n	1	2	3	4	5	6	7	8	9	10	11	12	13
$u_\heartsuit(n)$	0	0	0	0	0	0	0	0	3	9	9	19	19
$u_\diamondsuit(n)$	0	0	0	3	3	3	13	13	28	49	49	85	85
$u_\clubsuit(n)$	0	0	0	0	1	1	1	11	26	26	54	54	54
$u_\spadesuit(n)$	0	0	0	0	1	1	1	7	17	17	38	38	38

Table 16: A dynamic programming solution for computing the number of composite blocking sets that can be made from 5 cards, applied to the partial configuration of Figure 8.

Somewhat surprisingly perhaps, we can apply this approach to the pile of height 5 instead of to the pile of height 4 without complicating matters much further. We have previously distinguished two structures that apply to the pile of height 5. The first was the single primitive blocking pattern, the second was the structure shown in Table 4. We can create a second 24-bit addressed lookup table which is filled similar to the first one. The only difference is that a function u will replace the function t and it differs from t in that we will be adding binomial coefficients $\binom{s_\heartsuit(n)}{2}$ instead of just

$s_\heartsuit(n)$. Table 16 shows u for the configuration in Figure 8 and Table 15.

We now use twice as much space and precomputation- and lookup time than when using the pile of height 4, but because the number of composite blocking sets for the pile of height 5 is well over four times larger than the number of composite blocking sets for the pile of height 4, the total time spent is still smaller.

We cannot combine the two tables by simply adding the values pertaining to the different structures, because the values derived from these tables must be added to neighboring digits in a factorial base, meaning that the ratio of the contributions depends on their exact position, which is not known in advance and can in fact vary.

3.4.6 A shattered lookup table

Since the approaches for piles of height 4 and 5 are so similar, a natural idea would be to extend it to piles of height 6. However, this does not work, because although the structure from Table 10 *can* be treated similarly to the structures we worked with before, we have not found a way to also accomodate the composite blocking sets displayed in Table 5 in a 24-bit addressed lookup table.

n	1	2	3	4	5	6	7	8	9	10	11	12	13
$v_{\heartsuit\diamondsuit}(n)$	0	0	0	3	3	11	11	11	29	57	57	102	102
$v_{\clubsuit\spadesuit}(n)$	0	0	1	1	5	14	14	34	64	64	120	120	120

Table 17: A dynamic programming solution for computing the number of composite blocking sets that can be made from two primitive blocking sets and using six cards, applied to the partial configuration of Figure 8.

We can use a 36-bit addressed lookup table by keeping track of the number of unused cards in two suits of the same color separately, similar to before, this time adding the product of two such values instead of just one number or a binomial coefficient. We have a function $v_{\heartsuit\diamondsuit}(n)$ (shown in Table 17), which is 0 for $n = 1$, and for $n > 1$ is $v_{\heartsuit\diamondsuit}(n-1)$ in case one or both of the cards $\clubsuit(n+1)$ and $\spadesuit(n+1)$ has been used, and $v_{\heartsuit\diamondsuit}(n-1) + s_\heartsuit(n) \times s_\diamondsuit(n)$ otherwise. For this table we only need two lookups per iteration instead of four. However, a table with 2^{36} entries is far too large.

This is where we introduce the concept of a *shattered lookup table*. A key insight into its construction is that $v_{\heartsuit\diamondsuit}(n)$ depends solely on $v_{\heartsuit\diamondsuit}(n-1)$,

$s_\heartsuit(n)$, $s_\diamondsuit(n)$, two bits indicating the absence of the red cards of value n and a single bit indicating the absence of both black cards of value $n+1$.

We break the lookup table into one *heading* table and several *tailing* tables. The heading table will be indexed by the first q bits of each of the three rows in the 36-bit addressed table and will contain 2^{3q} entries with values $v_{\heartsuit\diamondsuit}(q)$. We assume it is easy to count bits in a number, for example by means of a small lookup table. This is how the values of $s_\heartsuit(q+1)$ and $s_\diamondsuit(q+1)$ can be recovered.

There will be $(q+1)^2$ tailing tables with $2^{3\times(12-q)}$ entries each, giving the values $v_{\heartsuit\diamondsuit}(12) - v_{\heartsuit\diamondsuit}(q)$, assuming a different pair of offsets for $s_\heartsuit(q+1)$ and $s_\diamondsuit(q+1)$ in each table.

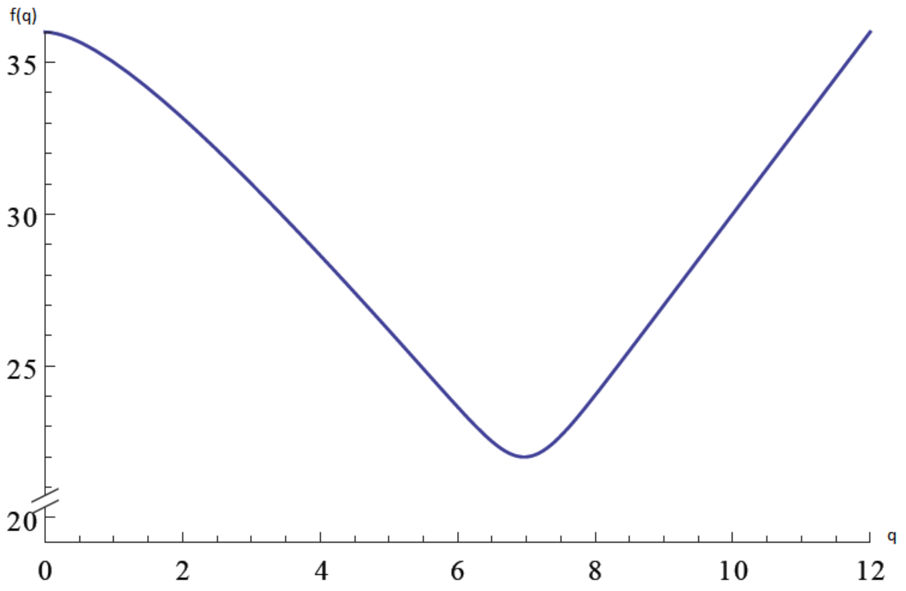

Figure 9: The graph of $f(q) = \log_2(2^{3q} + 2^{36-3q}(q+1)^2)$ [15].

What remains to be seen is for which value of q this approach works best, and whether it works better than having one table with 2^{36} entries at all. We want to minimize $2^{3q} + 2^{36-3q}(q+1)^2$ for $q \in [0, 12]$, because we want to minimize memory usage. Figure 9 quite clearly shows a minimum near $q = 7$. So we can create a heading table with 2^{21} entries and 64 tailing tables with 2^{15} entries each, resulting in 2^{22} entries in total. This is very reasonable.

3.4.7 Intermezzo: Generalizing shattered lookup tables

The approach with the shattered lookup table can be generalized. We can make more cuts to make different trade-offs between memory requirements and running time, also for larger instances of this problem and for similar problems. Applying this strategy to eliminate the loop iterating over the primitive blocking sets associated with the pile of height 7, however, is complicated by the structure shown in Table 9.

We will take a look at larger instances of the problem, where the deck has $n + 1$ cards per suit. In the case where we want to make only one cut, we try to minimize $f_n(q) = 8^q + 8^{n-q}(q+1)^2$ for $0 \leq q \leq n$ and $q \in \mathbb{N}$. For our analysis however, we will consider $q \in \mathbb{R}$. It is easy to see that q_n^*, the value of q for which $f_n(q)$ is minimal, satisfies $\frac{n}{2} \leq q_n^* < n$. It is also clear that f_n is unimodal on the interval $[0, n]$, i.e., it is monotonically decreasing in the interval $[0, q_n^*]$ and monotonically increasing in the interval $[q_n^*, n]$. Hence, the cut should be made at either $\lfloor q_n^* \rfloor$ or $\lceil q_n^* \rceil$, depending on the values of $f_n(\lfloor q_n^* \rfloor)$ and $f_n(\lceil q_n^* \rceil)$.

For simplicity's sake, let us consider only decks with an odd number of cards per suit. We compute the following:

$$f_n(\frac{n}{2} + k) = 2^{\frac{3}{2}n-3k-2}(n^2 + (4k+4)n + 4k^2 + 8k + 4 + 2^{6k+2})$$

$$f_n(\frac{n}{2} + k + 1) = 2^{\frac{3}{2}n-3k-5}(n^2 + (4k+8)n + 4k^2 + 16k + 16 + 2^{6k+8})$$

$$f_n(\frac{n}{2}+k)-f_n(\frac{n}{2}+k+1) = 2^{\frac{3}{2}n-3k-5}(7n^2+(28k+4)n+28k^2+48k+16-7\times 2^{6k+5})$$

Without the strictly positive multiplier $2^{\frac{3}{2}n-3k-5}$ of $f_n(\frac{n}{2} + k) - f_n(\frac{n}{2} + k + 1)$, we are left with a quadratic function of n. For any k its graph is a parabola which opens upwards and has its focal point below the horizontal axis, since $c - \frac{b^2+1}{4a} = 28k^2 + 48k + 16 - 7 \times 2^{6k+5} - \frac{(28k+4)^2+1}{4\times 7} < 0$. Combining these facts, we can conclude that the vertex of each parabola is also located below the horizontal axis — a somewhat unusual tactic perhaps, but substituting $\frac{-b}{2a}$ would be unnecessarily messy.

Therefore, for every choice of $k \in \mathbb{N}$, there exists an $n_{k+1} \in \mathbb{R}$, such that for $n > n_{k+1}$ we have $f_n(\frac{n}{2} + k + 1) < f_n(\frac{n}{2} + k)$. The gaps between the subsequent n_k's become exponentially larger, due to the exponentially increasing negative offset $7 \times 2^{6k+5}$. Using the abc-formula we find that

n_{k+1} corresponds to the positive root:

$$\sqrt{2^{6k+5} + \frac{32}{49}} - 2k - \frac{12}{7}$$

This means n_k equals:

$$\sqrt{2^{6k-1} + \frac{32}{49}} - 2k + \frac{2}{7}$$

Conversely, $q^*_{\bar{n_k}}$ is roughly $\frac{\bar{n_k}}{2} + \frac{\log_2(\bar{n_k})}{3} + \frac{1}{6}$. The first ten *transition points* are given in Table 18.

k	n_k	$\bar{n_k}$	$q_{\bar{n_k}}$
1	4.000	4	3
2	41.548	42	23
3	356.325	358	182
4	2888.595	2890	1449
5	23160.761	23162	11586
6	185352.086	185354	92683
7	1482896.686	1482898	741456
8	11863267.489	11863268	5931642
9	94906247.910	94906248	47453133
10	759250105.280	759250106	379625063

Table 18: The transition points where $f(\frac{n}{2} + k)$ is first at least as good as $f(\frac{n}{2}+k-1)$ (only taking into account even values of n). The second column gives the analytical transition point, whereas the third column gives the practical transition point.

When we aim to make two cuts, at positions q and r (with $2q \leq r$), we want to minimize:

$$2^q + 2^{r-q}(q+1)^2 + 2^{n-r}(r+1)^2$$

As the number of cuts c increases, it becomes increasingly tricky to solve the problem with calculus. However, notice that for a given n and a chosen r we do not have to vary q, since the optimal r for a given q is is known from the case with one cut.

Based on the intuition that each cut cut_i with $i \in [0, cuts - 1]$ in an optimal configuration will obey $|cut_i - i\frac{n}{cuts+1}| = O(\log(n))$, we present an algorithm to compute the optimal cuts with time complexity $O(\log(n)^{cuts})$. Pseudocode for the algorithm is shown in Figure 10.

```
pair solve(n, chunks, last_size, memory, delta) {
    if(chunks == 1)
        return (1, [n])
    best_pair = (infinity, [])
    for(this_size = last_size;
        this_size >= 0 && this_size * chunks <= n;
        this_size += delta) {
        this_pair = solve(n − this_size,
            chunks − 1,
            this_size,
            memory + 8**this_size
            * (n − this_size + 1)**2,
            1)
        if(this_pair.first < best_pair.first) {
            best_pair = this_pair
            best_pair.second += [this_size]
        } else {
            break
        }
    }
    return best_pair
}
pair solution = solve(N, CHUNKS, N / CHUNKS, 0, −1)
```

Figure 10: A pseudo code implementation of an algorithm to compute optimal cuts for the generalized shattered lookup table problem.

For smaller values of n we can speed this up slightly by computing the solution analytically when *chunks* equals 2 (assuming the mathematical operations can be performed in constant time for such values of n).

Table 19 lists some optimal solutions for two, three and four cuts, which were computed with the given algorithm. Keeping in mind that the algorithm works with values that are relatively close to each other, a useful trick to make it work for relatively large values of n without having to work with very long numbers, is to represent the summands as pairs (a, b), indicating $a \times 2^b$.

3.4.8 Final notes on exploiting symmetry

Because we replaced the loop associated with the pile of height 6 by a number of lookups, it makes sense to apply the exploitation of the suit symmetry as described in Section 3.4.4 to a combination of the piles of height 5 and height 7.

When we do not exploit any symmetry, there are $10{,}665{,}653$ combinations of one set associated with the pile of height 5 and one set associated

n	2 cuts	3 cuts	4 cuts
3	$[2, 1, 0]$		
4	$[2, 1, 1]$	$[2, 1, 1, 0]$	
5	$[3, 1, 1]$	$[2, 1, 1, 1]$	$[2, 1, 1, 1, 0]$
6	$[3, 2, 1]$	$[3, 1, 1, 1]$	$[3, 1, 1, 1, 0]$
7	$[3, 2, 2]$	$[3, 2, 1, 1]$	$[3, 1, 1, 1, 1]$
8	$[4, 2, 2]$	$[3, 2, 2, 1]$	$[3, 2, 1, 1, 1]$
9	$[4, 3, 2]$	$[4, 2, 2, 1]$	$[3, 2, 2, 1, 1]$
10	$[5, 3, 2]$	$[4, 3, 2, 1]$	$[4, 2, 2, 1, 1]$
11	$[5, 3, 3]$	$[4, 3, 2, 2]$	$[4, 3, 2, 1, 1]$
12	$[5, 4, 3]$	$[5, 3, 2, 2]$	$[4, 3, 2, 2, 1]$
13	$[6, 4, 3]$	$[5, 3, 3, 2]$	$[4, 3, 2, 2, 2]$
14	$[6, 4, 4]$	$[5, 4, 3, 2]$	$[5, 3, 2, 2, 2]$
15	$[6, 5, 4]$	$[5, 4, 3, 3]$	$[5, 3, 3, 2, 2]$
16	$[7, 5, 4]$	$[6, 4, 3, 3]$	$[5, 4, 3, 2, 2]$
17	$[7, 5, 5]$	$[6, 4, 4, 3]$	$[5, 4, 3, 3, 2]$
18	$[7, 6, 5]$	$[6, 5, 4, 3]$	$[6, 4, 3, 3, 2]$
19	$[8, 6, 5]$	$[7, 5, 4, 3]$	$[6, 4, 3, 3, 3]$
20	$[8, 6, 6]$	$[7, 5, 4, 4]$	$[6, 4, 4, 3, 3]$
100	$[36, 32, 32]$	$[28, 25, 24, 23]$	$[23, 20, 19, 19, 19]$
1000	$[337, 332, 331]$	$[255, 249, 248, 248]$	$[205, 200, 199, 198, 198]$

Table 19: Optimal solutions for various larger instances of the shattered lookup table problem.

with the pile of height 7 such that the sets are disjoint. This is significantly less than $1{,}145 \times 21{,}605 = 24{,}737{,}725$.

If we only exploit symmetry in the pile of height 7, there are $2{,}527{,}065$ combinations of one set associated with the pile of height 5 and one set associated with the pile of height 7 such that the sets are disjoint.

When we exploit the symmetry of both the pile of height 5 and the pile of height 7 together (it does not matter for the outcome whether we first exploit the symmetry in the pile of height 7 or not, but it will be faster if we do), and disregard the combinations where the sets are not disjoint, $1{,}700{,}305$ cases reamin to iterate over. This means we can attribute a factor $\frac{10{,}665{,}653}{1{,}700{,}305} \approx 6.27$ of reduction in the number of cases to our exploitation of suit symmetry.

3.5 Results

The final implementation of our algorithm ran in under 85 seconds on a regular desktop computer. It determined the number of distinct type I blocked Klondike Solitaire deals to be a number with 66 decimal digits:

952,628,704,275,585,774,781,929,328,392,634,005,105,712,508,518,715,873,034,240,000,000

When we divide this number by 52!, we get the reduced fraction

$$\frac{21649537217724514188460974601}{1833045925909798584842457600000}$$

as the fraction of type I blocked Klondike Solitaire deals. This amounts to approximately 1.181069% of all games.

3.6 Extension to Type II

A natural extension of blocked Klondike Solitaire deals of type I, is the class of configurations in which one or more primitive blocking sets are intertwined in a way that a lock card might not necessary lock up the remainder of its primitive blocking set within its own pile, but the lock cards together do lock up the remainders of their primitive blocking sets. We call these configurations deals of type II.

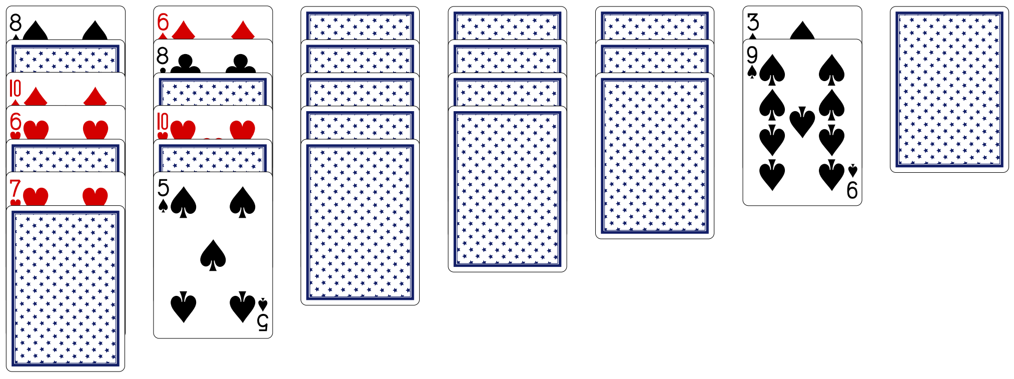

Figure 11: A schematic representation of a Klondike Solitaire deal of type II. Three of the primitive blocking sets that can be discerned in this image are {♠3, ♠5, ♡6, ♢6}, {♠3, ♠9, ♡10, ♢10} and {♡6, ♡7, ♣8, ♠8}.

A schematic representation of a set of deals that are all of type II, but not all of which are of type I, is displayed in Figure 11. From the positioning of the primitive blocking sets {♠3, ♠5, ♡6, ♢6}, {♠3, ♠9, ♡10, ♢10} and $\{\heartsuit 6, \heartsuit 7, \clubsuit 8, \spadesuit 8\}$ alone we can conclude that these games cannot be finished, even though no primitive blocking pattern might be present: The $\spadesuit 5$ cannot be played until one of the cards $\heartsuit 6$ and $\spadesuit 3$ has been freed, but for this to be possible the $\heartsuit 7$ or the $\spadesuit 9$ must be moved. The $\heartsuit 7$ cannot be moved until the $\clubsuit 8$ been freed, which requires the $\spadesuit 5$ to be moved first. The $\spadesuit 9$ cannot be moved until one of the red tens has been freed, but for this to be possible either the $\heartsuit 7$ or the $\spadesuit 5$ must be moved. From this it follows that neither the $\heartsuit 7$, nor the $\spadesuit 5$, nor the $\spadesuit 9$ can ever be moved.

We were not able to compute the exact number of card deals of type II, but we performed a Monte Carlo simulation with a Mersenne Twister as a random number generator to get a good estimate [16]. In $10{,}000{,}000$ random deals, $217{,}328$ were of type II, meaning about 2.17% of all deals is expected to contain a conflict of this kind.

The accuracy of this result depends on the quality of the random number generator used. We do think that the result reported is accurate, because the same random number generator and a comparable simulation were applied to estimate the percentage of card deals of type I and in this case even much higher accuracy was attained.

4 Klondike Draws

One of the constraining factors when playing Klondike Solitaire is the fact that not every card from the stock is available at all times, as explained in Section 1.1. In order to get some measure of the limitations imposed by the stock, we study how many permutations of cards can be extracted from it.

As an aside, if we would want to quickly determine whether a given permutation can be extracted from a given stock, we could quite efficiently do so, using a Fenwick Tree [17]. We first relabel the elements, such that the cards in the stock are labeled from 1 through n in increasing order, in $O(n)$ time and space. Next, we build a Fenwick tree on top of a frequency table with n entries of 1, in $O(n \log n)$ time. We keep track of the position of the last extracted card and repeatedly check whether the next card is either a multiple of three cards ahead or a multiple of three cards from the start. If not, we can conclude that the given permutation cannot be extracted from the stock. If we can process the entire permutation, it *can* be extracted. All lookups of cumulatives and all card removals can be done in $O(\log n)$ time per query, hence, the worst running case time of this algorithm is $O(n \log n)$.

4.1 Deriving a Recurrence

Let $A(n, k)$ denote the number of permutations of $n > 0$ elements that can be generated by a so-called *Klondike draw of degree* k. Such a draw works in the following way. Consider the sequence of elements $1, 2, \ldots, n$, in this order. We proceed in *stages*. In every stage at least one number must be removed from the sequence, and appended to the permutation that is being generated; the process terminates as soon as the last element has been removed. In a single stage one is allowed to do the following. First choose t integers $1 \leq i_1 < i_2 < \ldots < i_t \leq n_j$, the number of remaining elements (n in the first stage), all a multiple of k — except for possibly i_t, which may also be equal to n_j, even if this is not a multiple of k. These numbers indicate the available positions. The number t is called the *stage rank*. Then pick the i_1-th element, the $(i_1 - 1)$-th element, the $(i_1 - 2)$-th element, $\ldots$ (as many as one wants, and of course when still available), the i_2-th element, the $(i_2 - 1)$-th element, the $(i_2 - 2)$-th element, $\ldots$ and so on. Of course, elements can only be removed once. Note that when taking, e.g., the i_2-th element, still the original numbering is used.

If during a stage k elements are removed, it is possible to abandon further actions in this stage, and postpone them to the next. The last few elements play a somewhat special role: indeed, they can always be removed.

We clearly have $A(n,1) = n!$, and $A(n,k) = 1$ if $k \geq n$. So we may assume $1 < k < n$. Now the main recursion is:

$$\begin{aligned} A(n,k) &= \lceil n/k \rceil A(n-1,k) + \binom{\lceil n/k \rceil}{2} A(n-2,k) \\ &\quad + \binom{\lceil n/k \rceil + 1}{3} A(n-3,k) + \ldots + \binom{\lceil n/k \rceil + k - 2}{k} A(n-k,k) \\ &= A(n-1,k) + \sum_{i=1}^{k} \binom{\lceil n/k \rceil + i - 2}{i} A(n-i,k). \end{aligned}$$

This can be proven as follows. Suppose that during the *first* stage we want to pick $r \leq k$ elements. We do not want to choose the last element (except perhaps when $r = 1$, which easily leads to the first term from the righthand side of the formula). We then have to choose r elements from the $\lceil n/k \rceil - 1$ elements at the available positions (thereby excluding the last one, if n happens to be a multiple of k), with repetitions allowed. A repetition simulates the case where we keep on taking elements before proceeding to the next multiple of k. This gives a contribution of $\binom{\lceil n/k \rceil + r - 2}{r}$ possibilities. Now one can start a new stage with r elements less. Note that usage of the last element is fully taken care of in the situation with $r = 1$.

$n \backslash k$	1	2	3	4	5	6	7	8	9	10	11	12
1	1	1	1	1	1	1	1	1	1	1	1	1
2	2	1	1	1	1	1	1	1	1	1	1	1
3	6	3	1	1	1	1	1	1	1	1	1	1
4	24	7	4	1	1	1	1	1	1	1	1	1
5	120	30	10	5	1	1	1	1	1	1	1	1
6	720	111	25	13	6	1	1	1	1	1	1	1
7	5,040	624	121	33	16	7	1	1	1	1	1	1
8	40,320	3,162	478	85	41	19	8	1	1	1	1	1
9	362,880	22,050	1,897	431	106	49	22	9	1	1	1	1
10	3,628,800	141,870	11,666	1,745	276	127	57	25	10	1	1	1
11	39,916,800	1,181,970	62,826	7,033	1,426	331	148	65	28	11	1	1
12	479,001,600	9,219,870	340,270	28,483	5,831	865	386	169	73	31	12	1

Table 20: The number of Klondike draws $A(n,k)$ of length n and degree k for $1 \leq n, k \leq 12$. The values for $k < n \leq 2k$ are underlined for clarity.

Table 20 lists the number of Klondike draws of length n and degree k

for small values of n and k as computed using the recurrence given and confirmed with a brute force solution.

A number of special interest is $A(24, 3)$, since there are 24 cards in the stock in Klondike Solitaire, and under standard rules stock cards are flipped over per three. We found that $A(24, 3) = 104{,}244{,}550{,}813{,}439{,}400$, meaning roughly only one in six million permutations of 24 cards can actually be extracted from the stock.

4.2 Klondike Draws with $k < n \leq 2k$

Some patterns emerge within the number jumble of Table 20. For example, when we consider the values $A(n, k)$ for $k < n \leq 2k$ (underlined in the table), we have $\lceil \frac{n}{k} \rceil = 2$ and $\binom{\lceil \frac{n}{k} \rceil + i - 2}{i} = 1$, so our recursive formula reduces to:

$$A(n, k) = A(n-1, k) + \sum_{i=1}^{k} A(n-i, k),$$

from which we can derive the following expression in which the number of terms is independent from n and k:

$$A(n, k) - A(n-1, k) = 2A(n-1, k) - A(n-2, k) - A(n-1-k, k),$$

or equivalently,

$$A(n, k) = 3A(n-1, k) - A(n-2, k) - 1,$$

since $A(n, k) = 1$ for $n \leq k$.

By solving this linear homogeneous recurrence relation with constant coefficients [18], for the same restrictions on n and k we can derive the following closed formula:

$$A(n, k) = 1 + k \times \frac{(3+\sqrt{5})^{n-k} - (3-\sqrt{5})^{n-k}}{2^{n-k}\sqrt{5}}$$

This formula reveals a close relationship with the Fibonacci numbers [19]. In fact, $\frac{3+\sqrt{5}}{2} = (\frac{1+\sqrt{5}}{2})^2 = \phi^2$ and $\frac{3-\sqrt{5}}{2} = (\frac{1-\sqrt{5}}{2})^2 = (1-\phi)^2$, and using Binet's formula [20] we obtain:

$$A(n, k) = 1 + k \times F_{2n-2k},$$

and

$$A(n, k) - A(n-1, k-1) = F_{2n-2k},$$

as can easily be verified along the backslash diagonals within the underlined area.

4.3 Klondike Draws with $2k < n \leq 3k$

For $2k < n \leq 3k$ we know $\lceil \frac{n}{k} \rceil = 3$ and our main recursion reduces to

$$A(n,k) = A(n-1,k) + \sum_{i=1}^{k} (i+1)A(n-i,k).$$

Now let us consider what $A(n,k) - 2A(n-1,k) + A(n-2,k)$ looks like. In the notation below we will keep k fixed and denote $A(n,k)$ as a_n for simplicity.

$$\begin{array}{cccccccc}
3a_{n-1} & +3a_{n-2} & +4a_{n-3} & +5a_{n-4} & \dots & +(k+1)a_{n-k} & & \\
 & -3a_{n-2} & -3a_{n-3} & -4a_{n-4} & \dots & -ka_{n-k} & -(k+1)a_{n-k-1} & \\
 & -3a_{n-2} & -3a_{n-3} & -4a_{n-4} & \dots & -ka_{n-k} & -(k+1)a_{n-k-1} & \\
 & & +3a_{n-3} & +3a_{n-4} & \dots & +(k-1)a_{n-k} & +ka_{n-k-1} & +(k+1)a_{n-k-2} \\
\hline
3a_{n-1} & -3a_{n-2} & +a_{n-3} & +0 & \dots & +0 & -(k+2)a_{n-k-1} & +(k+1)a_{n-k-2}
\end{array}$$

Conveniently, all but five terms are eliminated in the addition, which leads us to the following alternative recursive formula for $2k < n \leq 3k$:

$$\begin{aligned} A(n,k) &= 5A(n-1,k) - 4A(n-2,k) + A(n-3,k) \\ &\quad -(k+2)A(n-k-1,k) + (k+1)A(n-k-2,k). \end{aligned}$$

4.4 Pfaff-Fuss-Catalan as a Lower Bound

In Section 1.1 we mentioned that Klondike Solitaire is sometimes played with at most three passes through the stock. The number of permutations that can be extracted from the stock in a constant number of passes c is a lower bound $L_c(n,k)$ on $A(n,k)$. We will consider the case $c = 1$.

While extracting a permutation we repeatedly flip k cards, a total number of $f = \lceil \frac{n}{k} \rceil$ times and we draw a single card n times, under the constraint that at no point we are to have drawn more cards than we have flipped over. It is easily verified that $L_1(n,k) = L_1(k\lceil \frac{n}{k} \rceil, k)$, so for simplicity we will assume that k divides n.

The situation can be thought of as a rectangular lattice grid on which we want to count the number of monotonic paths from $(0,0)$ to (n,f), permitting only steps $(0,1)$ and $(1,0)$, and without ever going below the line $y = \frac{x}{k}$.

We mimic a proof for the formula for Catalan numbers as it is found on Wikipedia [21]. Figure 12 serves to help illustrate the proof.

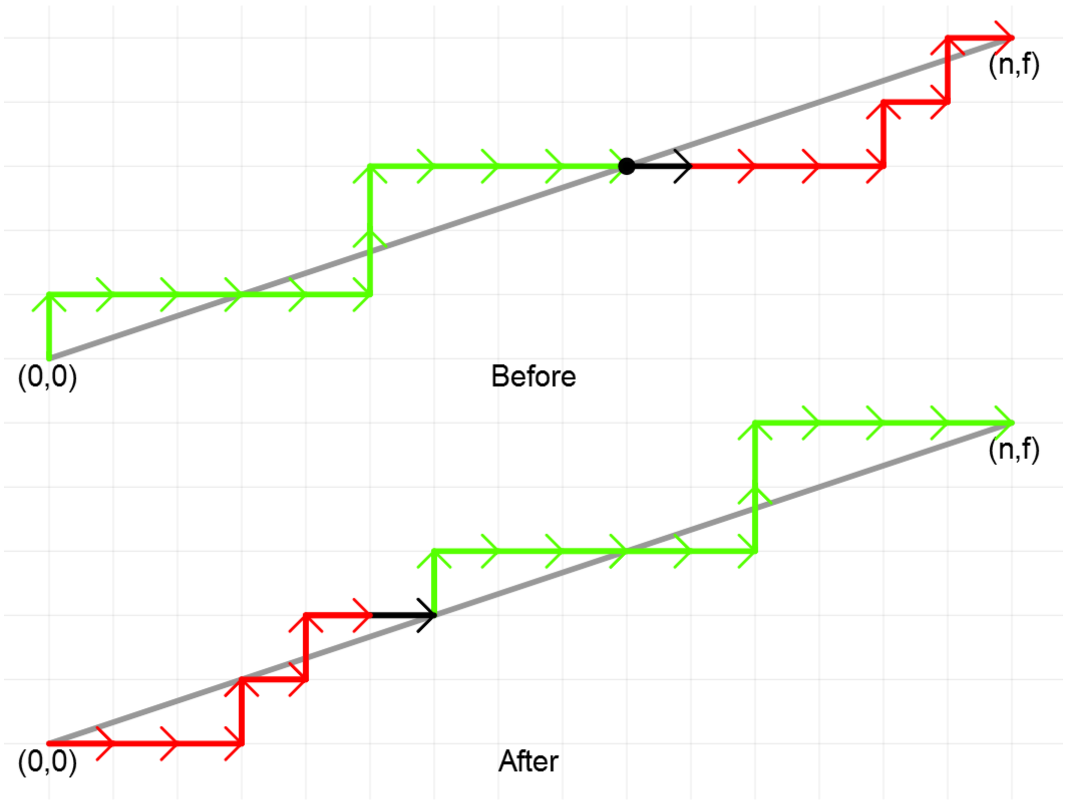

Figure 12: A monotonic path with exceedence 7 (above) and a modified version with exceedence 6.

We define the *exceedence* of a monotonic path as the number of horizontal edges which lie below the line $y = \frac{x}{k}$. In the upper path in Figure 12 this is 7.

In case a monotonic path does not have exceedence 0, there must exist a rightmost horizontal edge starting at the line $y = \frac{x}{k}$. We can swap the part of the path before and after this edge to end up with a path that has exceedence one lower than the original path had. After all, neither the part that was originally on the left, nor the part that was originally on the right had its exceedence altered, but the edge connecting them now has to be located above the line $y = \frac{x}{k}$, while earlier it was not.

The process is reversible: For any path P with exceedence less than n there exists a unique path that results in P when the procedure described is used. In the reverse procedure we identify the first horizontal edge that ends at the line $y = \frac{x}{k}$.

From this we can conclude that the number of monotonic paths from $(0, 0)$ to (n, k) with exceedence 0 equals the number of monotonic paths from $(0,0)$ to (n,k) of any exceedence e with $1 \leq e \leq n$. As a consequence, the number of paths with exceedence 0 must equal the total number of monotonic paths, which is $\binom{n+f}{n}$, divided by $n+1$. Hence,

$$L_1(n,k) = \binom{(k+1)\lceil \frac{n}{k} \rceil}{k\lceil \frac{n}{k} \rceil} \frac{1}{n+1}.$$

In particular, $L_1(24,3) = 420{,}732$.

For $k = 1$, $L_1(n,k) = C_n$, the n-th Catalan number. More generally, $L_c(n,k)$ equals $C^{k+1}_{\lceil \frac{n}{k} \rceil}$, where C^m_n is the Pfaff-Fuss-Catalan sequence [22, 23].

5 Counting No-braider Klondike Solitaire Deals

Consider a rule set for Klondike Solitaire in which all cards in the stock are accessible. We will define a *no-braider* deal as a game which can be won by solely applying the following two types of moves:

1) Moving a card from the stock directly to the foundation.

2) Moving a card from the table directly to the foundation.

In this section we will lay the groundwork for computing the total number of no-braider deals.

5.1 Braid Numbers

Let the vectors $v, w \in \mathbb{N}^*$ each represent a row of piles of cards such that the height of the p-th pile within a row is specified by the p-th element of its respective vector. All cards within a row are considered distinct. Starting with such a row of card piles, we will repeatedly take one of the topmost cards and move it onto a (possibly empty) pile in a second row of cards. We continue until all cards have been moved to the second row. Now, the *braid number* $\mathfrak{B}(v \to w)$ denotes the number of possible configurations of the second row of cards when the heights of the piles in the first row are described by v and the heights of the piles in the second row are described by w.

Figure 13 gives an example of the way cards could be moved from a configuration with pile heights described by the vector $(3, 1, 2)$ to a configuration with pile heights described by the vector $(2, 1, 3)$. The configuration in the lower right is counted only once in $\mathfrak{B}((3, 1, 2) \to (2, 1, 3))$, even though there are multiple ways to reach it. For example, the yellow card could already have been moved in the third step.

The braid number is invariant under permutations of the vectors v and w, that is to say that:

$$\forall \pi_1, \pi_2 : \mathfrak{B}(v \to w) = \mathfrak{B}(\pi_1(v) \to \pi_2(w))$$

Insertion or removal of zero elements does not alter a vector's behavior when it comes to braid numbers. Furthermore — and this is not entirely trivial — the function is symmetric, meaning that $\mathfrak{B}(v \to w) = \mathfrak{B}(w \to v)$ for all v, w.

For $v = (v_1, \ldots, v_k) \in \mathbb{N}^k$, we say that $|v| = k$ and we define $||v||$ as $\sum_{i=1}^{|v|} v_i$. We define the braid number of v and w to be zero whenever $||v|| \neq ||w||$.

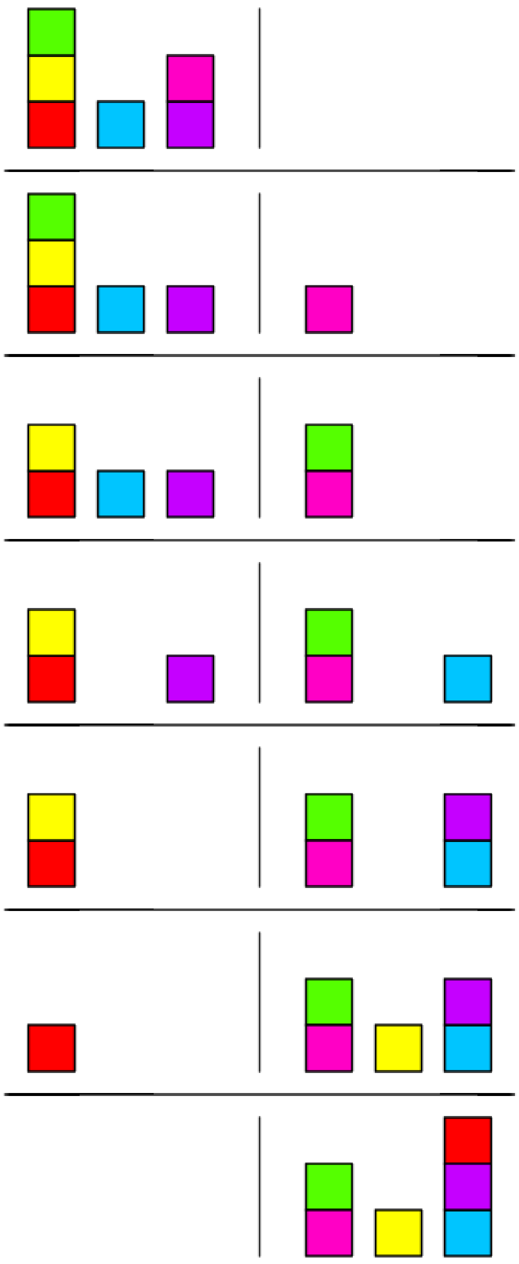

Figure 13: An example of the way cards could be moved from a configuration with pile heights described by the vector $(3, 1, 2)$ to a configuration with pile heights described by the vector $(2, 1, 3)$.

When $v = (v_1) \in \mathbb{N}^1$, $w = (w_1, \ldots, w_k) \in \mathbb{N}^k$ and $||v|| = ||w||$, the following holds true:

$$\mathfrak{B}(v \to w) = \frac{v_1!}{w_1! \cdots w_k!}$$

In particular, when $|v| = |w| = 1$, we have $\mathfrak{B}(v \to w) = 1$.

If $v_i \leq 1$ for $1 \leq i \leq |v|$, then for any w with $||v|| = ||w||$ we have $\mathfrak{B}(v \to w) = ||v||!$.

5.2 A Recursive Definition for $\mathfrak{B}((h_1, h_2) \to (\ell_1, \ell_2))$

Let $h_1 + h_2 = \ell_1 + \ell_2$, where h_1 and h_2 are the heights of the original left and right pile, and ℓ_1 and ℓ_2 are the heights of the final left and right pile. Then we have

$$\begin{aligned} &\mathfrak{B}((h_1, h_2) \to (\ell_1, \ell_2)) = \\ &\quad \sum_{t=0}^{\min(h_1, \ell_1 - 1)} \sum_{s=0}^{\min(h_1 - t, \ell_2)} \binom{s+t-1}{s} \mathfrak{B}((h_1 - s - t, h_2 - 1) \to (\ell_1 - t - 1, \ell_2 - s)) \\ &\quad + \text{similar with } \ell_1 \leftrightarrow \ell_2 \end{aligned}$$

This corresponds with the following. The top element of the original right pile, let us call this element b, ends up in either the final left or right pile. Assume it does so in the left pile, with t elements (that must have come from the original left pile) underneath. Clearly, this t must satisfy $t \leq \ell_1 - 1$ and $t \leq h_1$. The element immediately underneath b (if any) determines how many elements from the original left pile *certainly* must have already been moved: suppose there are $t+s$ of these, where s elements have moved to the right pile. Clearly, $s \leq \ell_2$ and $s \leq h_1 - t$. Note that more elements from the original left pile could have been moved to the final right pile, but one can assume that b was moved now. (If $t = 0$, the formula also holds, though other splits might be also usable.)

The $t + s$ elements could have produced $\binom{s+t-1}{s}$ different pairs of piles, having the topmost extra element in the final left pile fixed. It remains to count how many possibilities there are for moving piles of heights $h_1 - t - s$ and $h_2 - 1$ to piles of heights $\ell_1 - t - 1$ and $\ell_2 - s$.

If $\ell_1 = \ell_2$, we simply get a factor 2.

5.3 A Case Study: $\mathfrak{B}((h, h) \to (h, h))$

We can compute an upper bound $U(h)$ for $\mathfrak{B}((h, h) \to (h, h))$ relatively easily: For $0 \leq i \leq h$, there are $\binom{h}{i}$ ways to choose i cards from the leftmost starting pile to move to the leftmost end pile. There are also $\binom{h}{i}$ ways to choose i out of h locations in the leftmost end pile where these cards should be put. The order in which these i cards are mapped to their new locations is fixed. Now there are $\binom{h}{h-i}$ ways to choose which $h-i$ cards from the rightmost starting pile will end up in the leftmost end pile and $\binom{h}{h-i}$ ways to choose which positions the remaining cards from the leftmost starting pile will occupy in the rightmost end pile. Since everything else is fixed and $\binom{h}{i} = \binom{h}{h-i}$, we end up with $U(h) = \sum_{i=0}^{h} \binom{h}{i}^4$. This happens to equal ${}_4F_3(-h,-h,-h,-h;1,1,1;1)$, where ${}_pF_q(a_1,\ldots,a_p;b_1,\ldots,b_q;z)$ is the generalized hypergeometric series [15].

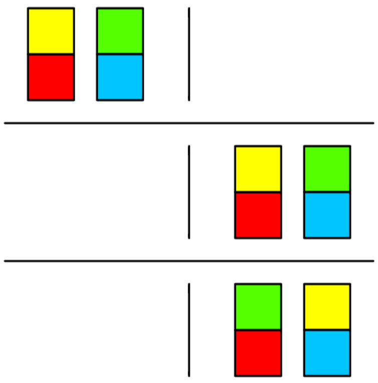

Figure 14: Neither of the configurations on the right-hand side can be obtained from the configuration on the left.

To demonstrate that $U(h)$ is not in general a tight upper bound, we take $h = 2$ and compare $U(2) = 18$ with $\mathfrak{B}((2,2) \to (2,2)) = 16$. The difference stems from two unattainable end configurations (one of these is depicted at the bottom in Figure 14, the other is that same configuration with the piles interchanged) in which the original top cards are on top once again, each covering the card that was originally at the bottom of the other pile. The resulting cyclic dependency among the four cards violates the rules set forth, but is not properly taken care of in the computation of $U(h)$.

A preliminary brute force algorithm unexpectedly brought a very interesting pattern to light regarding the braid numbers of the form $\mathfrak{B}((h,h) \to (h,h))$. We then conjectured:

$$\mathfrak{B}((h,h) \to (h,h)) = Q_h^2 \times 2^{h \bmod 2},$$

where $Q_h \in \mathbb{N}$ depends on h.

The speculated pattern in the values for $\mathfrak{B}((h,h) \to (h,h))$ might hint at the existence of a manageable underlying structure yet to be discovered.

5.4 An Algorithm to Compute Braid Numbers

So far, we have not found an elegant one-line recursive definition of $\mathfrak{B}(v, w)$, but we did design a fairly efficient algorithm based on memoization [24], which greatly outperforms any brute force solution.

We use a double-layered depth-first search, with both layers governed by the same function and subject to the same memoization method. In the outer layer we choose any possible matching between cards currently on top in the starting piles and bottommost open locations in the end piles, including a possibility for each top card to not be matched in this *round*. It is imperative that the current top cards in the starting piles, if they are not matched during this round, will not be matched with the locations currently available to them in any subsequent round, so as to not double count configurations. To enable this, each of the starting piles has a bit vector of length equal to the number of end piles associated with it. Whenever a location in one of the end piles is filled, the forbid-bits associated with this end pile must be set to 0, and whenever a card currently at the top on one of the starting piles is matched, all of its forbid-bits must be set to 0. We have to be careful what we manage on the fly and what we manage at the end of a round (we do not want to match two cards to locations within the same pile within a single round, for example).

In the inner layer of our nested depth-first search we iterate — by means of recursion — over the starting piles in order to match the card currently on top to an available location in the end piles. We also allow this card to not be matched. To avoid an infinite recursion, we disallow rounds in which no cards are matched at all.

We drastically reduced the number of states to store and at the same time significantly increased the probability of a hit during a memoization lookup by normalizing states before starting a new matching, by sorting the starting piles and their associated forbid-bits.

Table 21 shows braid numbers for configurations in which cards are moved from two piles of height h to two new piles of height h. The rightmost column serves to reveal the pattern we discovered. The sequence has been added to The Online Encyclopedia of Integer Sequences as $A214623$ [25]. Table 22 lists braid numbers for configurations in which there are three, four and five piles of height h respectively, from which cards are moved to equally many end piles.

h	$\mathfrak{B}((h,h) \to (h,h))$	$2^{h \pmod 2} \times Q_h^2$
0	1	1^2
1	2	2×1^2
2	16	4^2
3	128	2×8^2
4	1,156	34^2
5	10,952	2×74^2
6	107,584	328^2
7	1,083,392	2×736^2
8	11,115,556	$3{,}334^2$
9	115,702,472	$2 \times 7{,}606^2$
10	1,218,289,216	$34{,}904^2$
11	12,948,910,592	$2 \times 80{,}464^2$
12	138,708,574,096	$372{,}436^2$
13	1,495,661,223,968	$2 \times 864{,}772^2$
14	16,218,468,710,656	$4{,}027{,}216^2$
15	176,727,219,273,728	$2 \times 9{,}400{,}192^2$
16	1,933,956,651,447,076	$43{,}976{,}774^2$
17	21,243,204,576,601,928	$2 \times 103{,}061{,}158^2$
18	234,121,111,199,439,424	$483{,}860{,}632^2$
19	2,587,943,032,046,002,688	$2 \times 1{,}137{,}528{,}688^2$

Table 21: Braid numbers for a configuration in which cards are moved from two piles of height h to two new piles of height h.

5.5 A Graph Theoretical Interpretation

We can look at braid numbers in a graph theoretical context. Let $G_1(V, E_1)$ and $G_2(V, E_2)$ be forests of directed paths. A directed edge connects two vertices which represent cards placed directly on top of one another in the associated pile configuration. The direction of an edge signifies which of the two cards must be moved first or must have been moved first. In such a graph no vertex has an in-degree greater than one and no vertex has an out-degree greater than one. Let g_1 and g_2 be vectors describing the sizes of all weakly connected components in G_1 and G_2 respectively. Now $\mathfrak{B}(g_1 \to g_2)$ is the number of permutations $\pi \in S_{|V|}$ for which the graph $G(V, E_1 \cup \pi(E_2))$ is acyclic, where with $\pi(E_2)$ we mean a permutation with respect to the underlying set of vertices V of the edge set E_2.

h	$\mathfrak{B}((h)^3 \to (h)^3)$	$\mathfrak{B}((h)^4 \to (h)^4)$	$\mathfrak{B}((h)^5 \to (h)^5)$
0	1	1	1
1	6	24	120
2	504	28,800	2,620,800
3	64,368	82,616,832	218,855,808,000
4	9,565,236	300,656,771,136	
5	1,578,247,416	1,269,472,367,476,224	
6	279,149,957,136		
7	51,881,173,065,504		
8	10,006,245,507,952,548		
9	1,986,141,173,595,782,616		

Table 22: Braid numbers for a configuration in which cards are moved from three, four or five piles of height h to as many new piles of height h.

5.6 A Case Study: $\mathfrak{B}(R_{2,k} \to R_{2,k})$

Orthogonal to the case where there are two piles of height h as both the starting and the end configuration, we examine the situation where we start and end with k piles of height 2. We use the notation $R_{2,k}$ for the corresponding vector.

The graph theoretical analogue of $\mathfrak{B}(R_{2,k} \to R_{2,k})$ is the number of acyclic compositions (as defined above) of two 1-regular directed graphs on $2k$ vertices. It is easily demonstrated that each such composition is 2-regular.

Every 2-regular directed multigraph has an underlying undirected multigraph which is composed entirely of disconnected cycles [26]. This fact allows us to derive a recursive formula for $\mathfrak{B}(R_{2,k} \to R_{2,k})$.

Figure 15 and 16 show a 1-regular directed graph and a 2-regular directed multigraph respectively. The underlying undirected multigraph of the graph in Figure 16 consists of three cycles. The multigraph itself however, contains only a single cycle ($ABEF$). This cycle should be interpreted as a cyclic dependency, constituting an invalid composition in the context of braid numbers.

5.6.1 A recursive definition and a closed formula

Let $a_k = \mathfrak{B}(R_{2,k} \to R_{2,k})$. We can derive a recursive definition for a_k as follows. The leftmost end pile corresponds to a directed edge in the com-

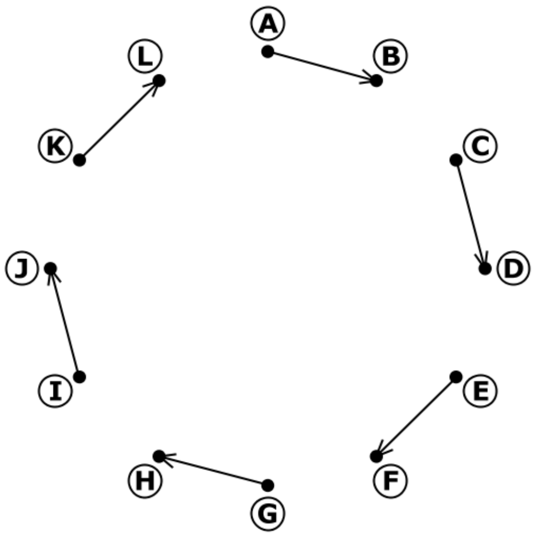

Figure 15: A 1-regular directed graph.

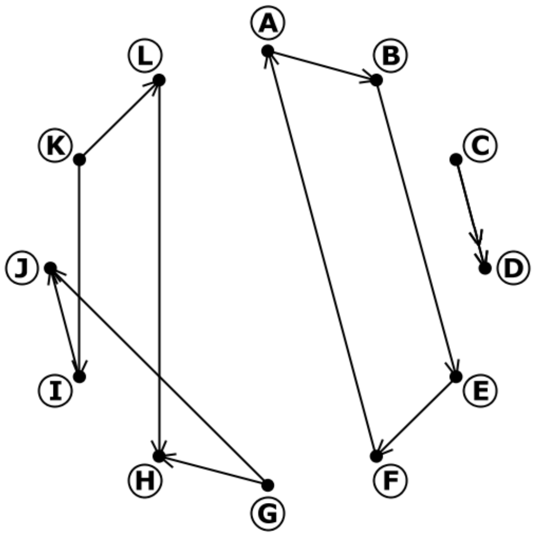

Figure 16: A 2-regular directed multigraph. The double arrow from node C to node D denotes a double edge.

posite graph; it is part of a cycle in the underlying undirected graph which further consists of $\binom{k}{t}$ edges corresponding to t out of k starting piles and $\binom{k-1}{t-1}$ edges corresponding to other end piles, for $1 \leq t \leq n$. These edges alternate in the cycle. Each of the remaining $2t-1$ edges can be oriented in any of two directions with regard to the edge corresponding to the leftmost end pile. Exactly one of the 2^{2t-1} possibilities forms a cycle in the directed graph. Each of the other (undirected) cycles can be completed to form an acyclic composition in exactly a_{k-t} ways. This derivation translates to the following recursive definition of a_k:

$$\begin{aligned}
a_k &= \sum_{t=1}^{k} a_{k-t}\binom{k}{t}\binom{k-1}{t-1}t!(t-1)!(2^{2t-1}-1) \\
&= \sum_{t=0}^{k-1} a_t\binom{k}{k-t}\binom{k-1}{k-t-1}(k-t)!(k-t-1)!(2^{2k-2t-1}-1) \\
&= \sum_{t=0}^{k-1} a_t\frac{k!(k-1)!}{(t!)^2}(2^{2k-2t-1}-1) \\
&= \sum_{t=0}^{k-1} a_t\left(\frac{k!!(k-1)!!}{(t!!)^2}-\frac{k!(k-1)!}{(t!)^2}\right)
\end{aligned}$$

k	$\mathfrak{B}(R_{2,k} \to R_{2,k})$
0	1
1	1
2	16
3	504
4	28,800
5	2,620,800
6	348,364,800
7	63,707,212,800
8	15,343,379,251,200
9	4,707,627,724,800,000
10	1,792,664,637,603,840,000

Table 23: Braid numbers for a configuration in which cards are moved from k piles of height 2 to k new piles of height 2.

Table 23 shows the braid numbers $\mathfrak{B}(R_{2,k} \to R_{2,k})$ for $0 \leq k \leq 10$. We confirmed the values for $k \leq 7$ with an implementation of the algorithm described in Section 5.4. The sequence has been added to The Online Encyclopedia of Integer Sequences as $A214624$ [25].

Without further proof, we conjecture that for $k \geq 0$, the following closed formula holds:

$$a_k = (2k)! - k^2(2k-2)! = \frac{3k-2}{4k-2}(2k)!$$

This expression suggests a much more straightforward alternative recurrence relation, in which $a_0 = 1$ and for $k > 0$:

$$a_k = a_{k-1}\frac{2k(2k-3)(3k-2)}{3k-5}$$

5.7 Application of Braid Numbers to Klondike Solitaire

The total number of no-braider Klondike Solitaire games when all the cards in the stock are accessible (which is in essence equivalent with flipping one card each time) equals the braid number $\mathfrak{B}(((1)^{25}, 2, 3, 4, 5, 6, 7) \rightarrow (13, 13, 13, 13))$. Here we consider every stock card to be its own pile of height 1. Computing this number with the algorithm from Section 5.4 would take too long. Luckily, we can treat piles with a single card in a special way. A starting pile of height 1 cannot be part of any cyclic dependency, hence in our case 25 of the cards involved can be placed anywhere without constraints.

Let us first examine a smaller instance of the problem. Say we want to compute $\mathfrak{B}((1, 4, 5) \rightarrow (4, 6))$. The card in the pile of height 1 could go to any of the four locations in the leftmost end pile, where it can be interleaved with any of $\mathfrak{B}((4, 5) \rightarrow (3, 6)) = 2{,}544$ possibilities to assign the nine remaining cards. Similarly, it can go to any of the six locations in the rightmost pile, where it can be interleaved with any of $\mathfrak{B}((4, 5) \rightarrow (4, 5)) = 3{,}316$ possibilities to assign the remaining cards. Hence, $\mathfrak{B}((1, 4, 5) \rightarrow (4, 6)) = 4 \times 2{,}544 + 6 \times 3{,}316 = 30{,}072$.

When k piles with a single card are in play, we have to incorporate the number of ways in which these k cards can be ordered, $k!$, and the number of ways a single such ordered collection of k cards can be assigned to k locations within the end piles. If there are ℓ end piles, this can be expressed as a summation over products of ℓ binomial coefficients; in our case:

$$\mathfrak{B}(((1)^{25}, 2, 3, 4, 5, 6, 7) \rightarrow (13, 13, 13, 13)) =$$
$$25! \times \sum_{a+b+c+d=25} \binom{13}{a}\binom{13}{b}\binom{13}{c}\binom{13}{d} \times$$
$$\mathfrak{B}((2, 3, 4, 5, 6, 7) \rightarrow (13-a, 13-b, 13-c, 13-d))$$

This is still somewhat computationally intensive, but using the framework for which we laid the groundwork in this section and the current state of our understanding of braid numbers, we think we will very soon be able to compute the number of no-braider Klondike Solitaire games within a reasonable amount of time.

As an aside, we know from Section 4 how many permutations can be extracted from the stock when flipping cards per three, and it is tempting to consequently consider the stock as one pile of height 24 and multiply the result by $A(24, 3)$. However, this logic is flawed, just like it is incorrect to state that $\mathfrak{B}((2,2) \to (2,2)) = 16 = \binom{4}{2} \times \mathfrak{B}((4) \to (2,2)) = 6 \times 6$.

6 Conclusions and Future Research

We studied the problem of counting the number of Klondike Solitaire games in which not a single move can be made, and we presented a very fast Dynamic Programming solution. The algorithm scales linearly[1] in each of the relevant dimensions (the number of piles on the tableau, the number of initially accessible cards in the stock and the number of cards per suit). The value computed for the standard version of Klondike Solitaire corroborates the result reported by Donkersteeg and Kosters [10].

We successfully extended this approach to also count the number of games in which a single ace can be moved to the foundation, after which no moves are possible anymore. Together these values constitute a lower bound of $0.3645\ldots\%$ of games that cannot be won. It would be feasible to push this limit further in a similar vein in future research.

We established another lower bound on the number of Klondike Solitaire games that cannot be won by counting starting configurations in which cards within a single pile on the tableau can together block the game. We carefully examined the structures and symmetries of composite blocking patterns, and subsequently designed an algorithm based on: the Inclusion-Exclusion principle; lookup tables; a factorial base counter with large-range coefficients; and a hybrid between a lookup table and a dynamic programming approach. The latter, coined a *shattered lookup table*, can be generalized to accomodate trade-offs between space and time under comparable circumstances, and we briefly touched upon this. We believe that the lower bound of $1.181069\ldots\%$ of games that cannot be completed because of these blocking configurations might correspond to the claim by Yan, Diaconis, Rusmevichientong and Van Roy [6] of an upper bound of solvable games of 98.81%, for which a faster implementation was under way as of 2005.

We established that $2.17\ldots\%$ of all games[2] cannot be completed due to the presence of one or more blocking patterns from a wider class of blocking patterns, including patterns that span more than a single pile, by means of a Monte Carlo simulation with a Mersenne Twister [16] as random number generator. We tried to devise an algorithm to compute this number exactly within a reasonable amount of time, but we were as of yet unsuccessful.

We investigated to what extent the stock limits game play in Klondike Solitaire. We discovered a recurrence for the number of permutations that

[1]Not taking into account the added complexity of additions of large numbers.

[2]See Section 3.6 for a discussion on the accuracy of the result.

can be extracted from a stock, given the number of cards in the stock and the number of cards per flip. We derived a closed formula for cases where the number of cards in the stock is more than once and at most twice the number of cards per flip. We found an elegant recursion for the cases where the number of cards in the stock is more than twice and at most three times the number of cards per flip. For the general case we demonstrate that a lower bound is given by the Pfaff-Fuss-Catalan sequences. Tight bounds on the number of permutations that can be extracted from a Klondike Solitaire stock for larger numbers of cards are still to be established.

We discovered a class of games that can trivially be completed by performing only moves directly to the foundation, when playing with the rule that every card in the stock is accessible at all times (coined *no-braider games*). Looking into smaller cases, we found that the relation is symmetrical and we discovered two fascinating patterns. The first pattern concerns situations in which all starting piles and all ending piles are of height 2, for which we found both a recursive definition and a closed formula, but we have not yet formally proved equality. The second pattern regards situations in which we have two starting piles and two ending piles, all of equal height. We found that the number of possible configurations appears to be either a square or twice a square, depending on whether the height is even or odd respectively. We cannot explain the second pattern. Also, we would be interested in an elegant interpretation of the closed formula for the first pattern.

We were not yet able to compute the number of no-braider games for the version of Klondike Solitaire where the piles on the tableau are of height 1 through 7 and the stock contains 24 cars which are flipped per three, due to time constraints, but we believe that given our current insights and understanding, it will be only a matter of time before we will be able to do so.

Bibliography

[1] S.S. Skiena, The Algorithm Design Manual, Springer, Corrected ed., 1997.

[2] R. Bellman, Dynamic Programming, Princeton University Press, Princeton, New Jersey, 1957.

[3] H.S. Wilf, Generatingfunctionology, Academic Press, 1994.

[4] J. Levin, Solitaire-y confinement, 2008, `http://www.slate.com/id/2191295`, retrieved Jun. 2012.

[5] Microsoft, Earnings Release FY12 Q2, 2012-01-19.

[6] X. Yan, P. Diaconis, P. Rusmevichientong, B. Van Roy, Solitaire: Man Versus Machine, Solitaire: Man Versus Machine, Advances in Neural Information Processing Systems 17, MIT Press, pp. 1553–1560, 2005.

[7] R. Bjarnarson, P. Tadepalli, A. Fern, Searching Solitaire in Real Time, International Computer Games Association Journal 30(3), pp. 131–142, 2007.

[8] Oberon Games and Microsoft Corporation, Microsoft Windows, Version 6.1 (Build 7600).

[9] U. Latif, The probability of unplayable solitaire (Klondike) games, 2004, `http://www.techuser.net/klondikeprob.html`, retrieved Dec. 2010.

[10] P.B. Donkersteeg, Klondike strategies using Monte Carlo techniques, Master Thesis, 2010, retrieved Dec. 2010.

[11] C. Aguilar, Vectorized Playing Cards 1.3, Licensed under LGPL 3, `http://code.google.com/p/vectorized-playing-cards/`, retrieved Feb. 2012.

[12] Wikipedia – Inclusion-exclusion principle, `http://en.wikipedia.org/wiki/Inclusion_exclusion`, retrieved Feb. 2012.

[13] Wikipedia – Dihedral group, `http://en.wikipedia.org/wiki/Dihedral_group`, retrieved Aug. 2012.

[14] Wikipedia – Factorial number system,
http://en.wikipedia.org/wiki/Factorial_base,
retrieved Apr. 2012.

[15] Wolfram Alpha LLC. 2009. Wolfram—Alpha.
http://www.wolframalpha.com, retrieved May 2012.

[16] A. Fog, Pseudo random number generators,
http://www.agner.org/random/, retrieved Feb. 2012.

[17] P.M. Fenwick, A New Data Structure for Cumulative Frequency Tables, Software: Practice and Experience, Vol. 24, 1994, pp. 327–336.

[18] D.H. Greene, D.E. Knuth, Mathematics for the Analysis of Algorithms, 2nd ed., Birkhäuser, 1982, p. 17.

[19] L.E. Sigler, (trans.) Fibonacci's Liber Abaci. Springer-Verlag. Chapter II.12, 2002, pp. 404–405.

[20] M. Livio, The Golden Ratio: The Story of Phi, the World's Most Astonishing Number. New York, Broadway Books, 2002, p. 108.

[21] Wikipedia – Catalan numbers,
http://en.wikipedia.org/wiki/Catalan_numbers,
retrieved Jul. 2012.

[22] R.L. Graham, D.E. Knuth, O. Patashnik, Concrete Mathematics. Addison-Wesley, Reading, MA, 1990, pp. 200, 347.

[23] R.P. Stanley, Enumerative Combinatorics, Vol. 2, Cambridge Studies in Advanced Mathematics 62, Cambridge University Press, Cambridge, 1999, p. 212.

[24] Wikipedia – Memoization,
http://en.wikipedia.org/wiki/Memoization,
retrieved May 2012.

[25] The Online Encyclopedia of Integer Sequences,
http://oeis.org/, retrieved Jul. 2012.

[26] Regular Graph, Wolfram MathWorld,
http://mathworld.wolfram.com/RegularGraph.html,
retrieved Jun. 2012.