Generative magic and designing magic performances with constraint programming

Guilherme de Azevedo Silveira

Accepted: 28 April 2022 / Published online: 21 May 2022 © The Author(s) 2022

Abstract
Professional magicians employ the use of interesting properties of a deck of cards to create magical effects. These properties were traditionally discovered through trial and error, the application of heuristics or analytical proofs. Those proofs come from diverse mathematical areas such as the set, number and graph theory. We discuss the limitations of relying on humans for such methods and present how professional magicians can use constraint programming as a computer-aided design tool (CAD) to search for desired properties in a deck of cards. Furthermore, we implement a solution in Python by making use of generative magic to design a new effect, demonstrating how this process broadens the level of freedom a magician can decree to their volunteers while retaining control of the outcomes of the magic. Finally, we demonstrate that the model can be easily adapted to multiple languages by presenting multiple variations of the effect supporting American English and Brazilian Portuguese.

Keywords Constraint · Generative design · Computer aided design · Constraint programming · Generative magic · Magical performance

1 Introduction

A central problem in the performing arts of magic concerns designing new effects, which are easily reproduced but also complex enough, so they are not easily figured out by spectators. Magicians traditionally use trial-and-error procedures that take time and are limited to only specific situations, such as the creation process behind Poker Night at the Improv [1].

Others create heuristics such as the System for Arranging Cards for Any Spelling Combination [2], providing a procedure for a performer to set up 13 cards for a magical effect. A guest will freely think of one of the prepared cards, for example the “six of hearts”. The cards are shuffled by the magician. The guest deals one card for each letter spelled, 11 cards for the “six of hearts”. The last card dealt will be the one which was spelled. The problem with heuristics in magic design is that it might not work under different circumstances.
Mathematically inclined magicians publish proofs of properties on a deck of cards, such as the properties of a Faro Shuffle [3–6], a deck control movement described in Appendix A.3. Programmers create closed source code exploring possibilities on a memorised deck, creating a card effect where the volunteer can make some free and some controlled choices as in the Poker Formulas [1].

Even after an effect is published, it might not be replicated by magicians who perform them in other languages since many card effects use characteristics from their own cultures. One such example is the language bias present in many spellings of card effects, such as the value and suit in U.S. English, which are fundamental parts of the Spelling a Card [2]. Other effects use double meanings of words such as Jacks or clubs. Memorised decks, such as the Aronson Stack [7], are built around card spelling characteristics from the U.S. English language. Memorised and stacked decks are described in Appendix A.4.

Moreover, nationality bias diminishes the reach of magical effects beyond the creator’s cultural bubble. Poker is a central theme in many magical effects [8–12], although many countries have their own games [13], which better represent their identities.

Therefore, designing effects based on the properties of a deck of cards is traditionally a time-consuming process limited by the cultural aspects, biases and experience of a creator.

1.1 Mentalism

In mentalism, a volunteer makes \( n \) choices, and the magician reveals a matching prediction [14]. While some effects allow for free choices, others will force the volunteer to a predetermined selection.

One such example is the use of basic mathematics to force any number. The performer places a closed deck of cards on the table. A volunteer is chosen and asked to choose a secret number between 1 and 100. The magician asks for the volunteer to perform a number of operations in their head:

1. Choose a number between 1 and 100.
2. Add 6.
3. Multiply it by 2.
4. Add the secret number.
5. Divide by 3.
6. Subtract the secret number.

The mentalist asks the volunteer for the resulting number. Suppose it is 4. The volunteer removes the cards from the deck and deals four cards face up. The fourth card is shown to be a blank card with “thank you” written on it.

The entire performance relies on the fact that the number 4 is forced. This can be seen by writing down the result of the operations performed as in (1).

\[
\frac{(x + 6) \times 2 + x}{3} - x = \frac{3x + 12}{3} - x = 4
\]

Due to the simplicity behind the method in mathematical magic, such as in this example, historically, “Mathematical magic is often viewed with a double disdain. Mathematicians are inclined to regard it as trivial play, magicians to dismiss it as dull magic” [15]. This reinforces the importance of a seemingly impossible effect achieved through a method with simple steps.
1.2 Basic set manipulation

Another mathematical example is the P.A.T.E.O. force [16]. The magician displays \( n \) items and takes turns along with the volunteer to remove items one by one. The remaining item is always the one that the magician had determined ahead of time. For the magic inclined, the method is further explained in Appendix A.1.

In the P.A.T.E.O force, there are two hints to the volunteers that the magician is guiding them to choose a predetermined option. First, the magician is always involved in the process of removing an item, even if it is the volunteer’s turn. Second, the magician makes the final choice. When using the most common styles of forces, volunteers might perceive that they were forced to make a choice, thereby ruining the entire sensation of an uncontrolled environment the magician is trying to build.

Even though a force is an important tool for the professional magician, even the most famous Riffle Force [17] contains hints on its forcing aspect, as it does not reflect how a free choice is made when selecting a card. The Riffle Force is described in Appendix A.2.

This motivates us to ask whether there are methods to provide free choices to volunteers while retaining control of the final result of the effect.

1.3 Our contribution

By defining and limiting the parameter space of the choices of a volunteer, we demonstrate the application of constraint programming to control and infer results from random selections made by such a person. We provided a Python code to replicate these findings [18] and open-sourced a framework [19] that can be used to design and generate magic under these premises.

This paper extends on generating magical performances with constraint programming [20] by providing extra magic examples and references based on mathematics, expanding on some examples, adding benchmarks for the solutions found, making minor optimisations to the model, adding new figures to illustrate the concepts and effects, proving that multiple results can be found for the same requirements, providing results both in U.S. English and Brazilian Portuguese and providing a new video recording with total freedom on the cuts performed by the volunteer.

1.4 Outline

This paper is organised as follows: Section 1 presents the background information necessary to understand the limitations of traditional creative magical methods. Section 2 proposes a new effect that provides more freedom for the volunteer while retaining control of the outcome by the magician. Section 3 examines how magical effects can be generated through constraint programming, following which Section 4 briefly goes through example code that designs and creates such effects, reports on the results achieved in performing these generative magic effects and discusses their real-life usability before Section 5 presents conclusions.

2 Controlling outcomes through constraint programming

Several spelling effects are described in the literature, where the magician deals a card as the chosen card name is spelt. The traditional deck of cards used in most of the American and European countries consists of 52 cards, divided into four suits (clubs,
spades, hearts and diamonds) with values running from two to ten, plus the values ace, jack, queen and king. Two jokers are sometimes used in card magic performances. It is common to call the card with value jack and suit clubs the jack of clubs.

A simple version of the spelling of a freely chosen card comprises the following procedure.

The magician removes one blue and one red case from their pocket and places them on a table. The magician points to the blue case and announces that it contains one prediction. Next, they open the red one and spread the 52 cards face down over the table, asking a volunteer to remove one card from the spread while hiding it from the magician.

The magician puts away both the remaining 51 cards and the red deck case back into their pocket. From the spectators’ perspective, there is nothing the magician can do from this moment on because both the prediction and the volunteer’s free choice have already been made. The volunteer then reveals the chosen card was, for instance, the one named five of spades.

The performer proceeds to spell the name of the mentioned card, dealing one card from the top of the blue deck for each letter. Since the five of spades is spelt with 12 letters, the magician deals 12 cards face up and reveals the 13th card to be the five of spades, as seen in Fig. 1.

Spreading the remaining 39 cards face up, the magician reminds the volunteer that they could have chosen any card, yet, they both chose the same five of spades.

This effect consists of two parts. The first one is a free selection from the volunteer, which turns out to be a forced card. To achieve this result, one can make use of a force red deck as in Fig. 2. One such deck is a one-way deck that consists of 52 cards having the same red back and the same face. In this example, it is the card named five of spades.

The second part consists in a self-working effect [21] is easy to perform because it only requires the preparation of the blue deck by positioning its five of spades in the 13th position, as seen in Fig. 3.

### 2.1 Convincers and issues

The magician can use false shuffles, such as the Mead and Kennedy false shuffle, on both decks to make the effect stronger [22]. False shuffles temporarily displace some of the cards but end the movement with all cards in their original order prior to the shuffle.

They can follow it with false cuts, such as Bobby Bernard’s False Cut, which do not change the deck order [17].

![Fig. 1](image)

**Fig. 1** The result of cards dealt while spelling five of spades are twelve random cards, the five of spades itself and the remaining 39 cards.
The third method consists of placing the five of spades at the 7th position and executing controlled shuffles, such as the Out-Faro Shuffle [4], which brings the card to the 13th position. The Faro is described in Appendix A.

Even the use of stronger convincers might not be enough for this effect as currently presented since the magician cannot hand the red cards to the volunteers for further inspection. Depending on the force deck in use, it cannot even be spread face up.

2.2 Choices, control and knowledge

While the volunteer made a free choice in the previous effect, the magician controlled its result by giving them no other option but the five of spades. The magician decided beforehand which card would be chosen and placed it in its expected position into the deck. In our example, the five of spades was placed the card at the 13th position.

The selections made by the volunteer can be understood as parameters to the magic effect. Parameters can be randomly chosen from their own parameter space. In the example given, there is one [1,52] space making it a 1-parameter magic effect.
The question raised is whether magical methods exist that allow volunteers to make N random choices over an N-dimensional parameter space while handing them real control over the card sequence and yet allowing the magician to force the result.

In this paper, we aim to confirm that it is indeed possible. A magician can make correct predictions about the outcome of N random choices made by a volunteer based on a set of items such as a 52 card deck by controlling other variables through constraint programming.

3 Constraining for freedom

Our desired effect, Freedom of Spelling, consists of attributing real freedom to the parameters that the volunteer will choose. The magician lays a blue and red case each on the table. The red cards are removed from the deck and fanned, revealing 52 different examinable cards, as in Fig. 4.

They are spread face down on the table. The volunteer chooses three cut points resulting in four packets, as in Fig. 5 and decides a permutation that defines the sequence in which the packets will be put back together.

Therefore, the volunteer has six free choices and real control over the resulting card sequence. The magician emphasises two points. First, the volunteer has made free choices. Second, there are 52 factorial different combinations of a deck of cards. The magician proceeds to reveal one single card turned face down on the blue deck, for example, the ace of spades as in Fig. 6.

One card is dealt face-up on the table for each letter of the ace of spades. The next card at the expected 12th position is the ace of spades itself in the red deck controlled by the volunteer, as shown in Fig. 7.

There are two parts to this effect. In the first part, the volunteer freely makes six choices from a 6-dimensional parameter space. The resulting card sequence is one out of many possibilities.

The volunteers are fooled to believe that there are 52 factorial possible sequences they might generate in this process, while the magician has never explicitly said so. Typically the performer will play with their understanding by truthfully saying, “A fully
shuffled deck has 8 times 10 to the power of 67 possibilities.” The phrases are disconnected, but the volunteers do not perceive them due to their misdirection and misunderstanding of the possibilities generated by a 6-dimensional space parameter.

Part of this misunderstanding is due to the fake shuffles and part due to the natural limitations of real shuffles. Traditional riffle shuffles are required to be performed more than seven times depending on the game being played to provide fairness [23]. In the performance proposed here, the magician only allowed for a much simpler shuffle.

The first cut is made anywhere between positions 2 and 49, the second cut between the first one and 50 and the third one between the second one and 51. Finally, the
volunteer chooses any of the 24 permutations of the four stacks to gather the deck back together in one stack that gives only a subset of all possible card sequence combinations, one in the order of 2 million possibilities. Those parameters spaces will constitute domains to be explored by the resulting algorithm.

Due to the nature of the effect, at one point, the magician needs to know what position the volunteer has cut to. They can count the position by spreading the cards and asking the volunteer where to cut. They can also use number markings on the back of the cards as in *Card Control by the Numbers* [24].

The second part of the effect is to reveal that there is a card in its expected position. If the *ace of spades* is in the 12th position, the magician proceeds to use an index method to reveal the card as a supposed prediction. The *Invisible Deck* [25], the method described earlier, would display the *ace of spades* as the single card face down. The method behind this deck is described in Appendix A.5.

We are left with the question – will there always be one specific card in the expected position, no matter the choice of parameters? The answer is no.

**Theorem 1 (No single card theorem)** There is no single card that will always be at its expected position according to its name in one predetermined language, no matter the decisions of the volunteer.

**Proof** Assume the existence of $x$, a card that will always be at its expected position according to its name, no matter the decisions of the volunteer, in one predetermined language, which is $\text{length}(x) + 1$.

Let $a$, $b$, $c$ and $d$ be the packets such that the number of cards in $a$ is greater than $\text{length}(x) + 1$ and all other packets have at least one card.

If the deck is rebuilt as $[a,b,c,d]$, $x$ is found at position $\text{length}(x) + 1$.

If the deck is rebuilt as $[d,c,b,a]$, $x$ is displaced at least 3 cards from its position $\text{length}(x) + 1$. As it contradicts the definition of $x$, we conclude there is no such a card.

As the model is too restrained by the existence of a single card, we can lose the constraint on the same single card and accept any card in its expected position. Therefore, our new question is whether there will always be at least one card in the expected position no matter the choice of parameters?

An unexpected way to model this problem aids in its solution. The magician has control of two aspects of the effect: the starting deck sequence and the card to be revealed. The first one is a set of 52 variables that start with no constraint. The parameters chosen by the volunteer during the live performance define the card to be revealed.

As the first part of the effect defines six parameters and a sequence of array operations performed on the stack of cards, one can create a set of constraints, which is required to guarantee the existence of one card in its expected position at the end of the process.

In order to achieve it, we define the 6-dimensional parameter spaces. The three cut points are defined in closed intervals, and the first three values of a permutation define the final sequence.

**Definition 1 (space_full)** The entire parameter space consists of:

1. $\text{cut}_1 \in [2,49]$
2. \( \text{cut}_2 \in [\text{cut}_1,50] \)
3. \( \text{cut}_3 \in [\text{cut}_2,51] \)
4. \( \text{sequence} \in S\{1,2,3,4\} \)

The second step consists in extracting the length of each card’s name in the language in which the effect will be performed. For instance, in U.S. English, the *ace of clubs* has a length of 10 while the *king of diamonds* has a length of 14. In Brazilian Portuguese, the same cards, *ás de copas* and *rei de diamantes* - have the length 8 and 14, respectively.

Given the six parameters and starting with a deck numbered \([1, 2, \ldots, 52]\), one can simulate the cuts and deck rebuilding, obtaining the final deck order. For example, cutting to the 10th, 20th and 30th card gives \( \text{cut}_1 = 10, \text{cut}_2 = 20 \) and \( \text{cut}_3 = 30 \). Using the permutation \([3,1,2,4]\) results in the sequence 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, ..., 20, 31, 32, 33, 34, ..., 51, 52.

Note that card 1 ends at position 11 while card 2 ends at position 12. If a card spelt with ten letters begins at position 1, it would end at the expected position 11.

One can now define a constraint in U.S. English for the *ace of clubs* to begin at position 1 in order to end at position 11. Another constraint requires the *king of spades* to begin at position 3, but those constraints do not need to hold at the same time; only one of the 52 generated constraints needs to be satisfied as follows:

1. ace of clubs starts at 1 or
2. two of clubs starts at 1 or
3. ...
4. king of spades starts at 3

Unfortunately, this is not yet enough to generate a magic effect. Satisfying one of such constraints gives us a set of starting deck position rules that work only for this specific point in the parameter space. The code must explore the entire parameter space, generating a set of two million constraints, each one consisting of an Or clause with 52 constraints.

Finally, adding the requirement that all variables are non-repeating integers in the space \([1,52]\), a solver implementation can be used to check for satisfaction and, if possible, a unique solution that works for any set of parameters the volunteer chooses.

If a solution exists, the performer can use that specific order for the starting red deck, perform false shuffles and cuts. The volunteer makes their six choices. The performer looks up a table, as in Poker Formulas [1], to find out which card is at the expected position in the red deck. Other possible solutions for finding out the expected card can be (1) be hinted by a computer or smartwatch and (2) visually check for markings on the back of the cards. Both are traditional solutions in magic described in Appendix A.7. The magician proceeds to reveal that card face down in the blue deck and finally spells the card in the correct position in the red one.

### 4 The Z3Solver solution

Starting with a deck in any order, one needs to simulate the card movements by slicing and joining arrays. Listing 1 creates a sequence and decides where each card in the original order of 1 to 52 ends.
For example, a number 15 in the second position of the returned NumPy [26] array means that the 15th card from the original deck ends up at the 2nd position of the deck.

Listing 1: A function that simulates the cuts and joins.

```python
def simulate(cuts: List[int], deck: List[int]):
    stacks = np.array([range(cuts[0]),
                       range(cuts[0], cuts[1]),
                       range(cuts[1], cuts[2]),
                       range(cuts[2], 53))]
    return np.concatenate(stacks[[*deck]])
```

Using Z3Solver [27], a SMT solver, 52 integer variables are created representing the starting position of their respective cards, as shown in Listing 2. The first 13 variables stand for the ace, 2, 3, ..., 10, jack, queen and king of clubs. The next 39 variables represent the same cards in the suits of hearts, spades and diamonds.

Listing 2: Defining all 52 card variables.

```python
names = map(retrieve_card_name, range(1, 53))
all_vars = list(map(z3.Int, names))
```

Every card must have a constraint limiting its position to [1,52], and no two cards can occupy the same starting position.

The next step to constrain the original deck order to the requirements of a given set of 6 parameters is to simulate the card movements with a sample deck. Then, use its output to generate the required constraints as in Listing 3.

Listing 3: Simulates the slices, joins and return the proper constraints.

```python
def freedom_of_spelling(all_vars: List[z3.Int],
cuts: List[int],
sequence: List[int]):
deck = simulate(cuts, sequence)
return spelling_rules(all_vars, deck)
```

The 52 conditions can be generated by going over each card and extracting its name and length. By using the number that finishes at the expected position in the deck array, we define where such a card should be placed at the beginning of the effect, as in Listing 4.
This allows the exploration of a single point in the parameter space; therefore, it is required to explore the 6-dimensional discrete parameter space by invoking freedom_of_spelling and generating a new rule for each execution as in Listing 5.

4.1 Redesigning for a solution

4.1.1 Multiple outs

The performer has only one out so far. For example, the ace of spades must be at position 12 so that the magician reveals it after spelling the last letter. Such constraint can be relaxed by allowing the performer two outs. If the ace of spades is at position 11, the reveal is made during the spelling of the last letter. The volunteers are never aware of the two possible outcomes. The relaxed constraint says that a card with \( n \) letters can be at position \( n \) or \( n + 1 \).
4.1.2 Unique set of constraints

All card names in U.S. English have a length between 10 and 15, which should give an upper bound of $52 \cdot 51 \cdot 50 \cdot 49 \cdot 48 \cdot 47 = 1$ unique constraints. Therefore, if different starting parameters end up with the same cards in these positions, the redundant constraints are removed in Python, prior to registering duplicate constraints to the solver.

4.1.3 The parameter space

Although all cuts are possible during performances, magicians know that volunteers do not make their cut on the first few cards even when given a free choice. Therefore, limiting the parameter space as follows achieves a similar magical effect.

**Definition 2** *(space*_optimized*) The optimized parameter space consists of:

1. $cut_1 \in [7, 21]$
2. $cut_2 \in [\max(cut_1, 12), 26]$
3. $cut_3 \in [\max(cut_2, 25), 39]$
4. $sequence \in S\{1, 2, 3, 4\}$

After all three optimizations, 2,475 unique constraints for U.S. English are created in eight minutes and analyzed, thus generating one of many sequences of the deck that satisfies the desired properties.

4.2 Baseline

To provide a baseline, phase 1 consisted of running three times the rule generation part of the method for both the entire parameter space *(space*_full, as per Definition 1) and the optimized parameter space *(space*_optimized, as per Definition 2). We performed runs with different seeds that were repeated in the following phases. The mean results were 102.39 minutes (95% CI [101.41, 103.38]) and 8.88 minutes (95% CI [8.78, 8.99]), respectively.

4.3 Multiple U.S. English solutions

Phase 2’s goal was to try and prove that multiple solutions to the problem exist. All three runs for *(space*_full) failed to provide a solution in under four hours, as expected. Meanwhile, the *(space*_optimized) runs resulted in one failure and two successes with a mean time of 8.83 minutes (95% CI [5.56, 12]). Although those results hinted in the direction of dependence of good seed to succeed using the current model, it allowed the finding of two valid solutions.

The resulting stack orders, *Freedom of Spelling in U.S. English*, are presented in Tables 1 and 2.

One common property in both solutions is that there is always at least one card in its expected position if no cuts are made. In both of them, that card, the *seven of spades*, is at position 13. Note that each card have two possible expected positions, and in this case, both are at the position representing the card’s length.
Theorem 2 (Matching card from the top) Any solution to the proposed system will always have at least one card at its expected position in the original stack order.

Proof Assume the solution $s$, a permutation of all 52 cards. By using the parameters $[7,12,25,1,2,3]$, the deck is back in its original stack order, effectively consisting of a false cut. As $s$ is a solution to the system, we find that at least one card must be in the original stack order that matches its expected position. □

As simple as it is to demonstrate the existence of such property amongst all solutions, it does not imply the existence of a single card. As Table 2 shows, the three of diamonds has...
a spelling length of 15 and is at position 16, allowing the magician to choose between two cards to use in the case of a false cut.

### 4.4 Brazilian Portuguese solution

Phase 3 consisted of exploring space\textsubscript{optimized} three times for Brazilian Portuguese, with three successes with a mean time of 9.73 (95% CI [9.29, 10.17]). Other solutions exist for card name length tables using different languages. One of the results, Freedom of Spelling in Brazilian Portuguese, is presented in Table 3.

Theorem 2 reminds us of the existence of a card in its expected position, no matter which language is chosen, as long as a false cut might be performed. For this Brazilian Portuguese solution, we find the seis de ouros at position 11 and the quatre de paus at position 13.

### 4.5 Improving the success rate

As the current model performance was unstable and could not be well determined under the current constraints, Phase 4 consisted of two swipes of space\textsubscript{optimized} of 500 seeds, each for U.S. English, looking for two types of errors – a four hour time out and out of memory errors.

In the first part of Phase 4, a change to the model consisted of replacing a set of constraints with the use of a single distinct clause for all 52 variables. Although a decrease in the mean time of 8.83 minutes was expected, the result was a substantial increase in computational time with a mean time of 11.56 minutes (95% CI [11.54,11.58]).

On the other hand, it was accompanied by a perfect success rate, with all attempts finding a solution in under four hours. Its result time histogram can be seen in Fig. 8.

The second part of Phase 4 used the same U.S. English model from Phases 1 to 3, which defined a rule where at most one card must have each value between 1 and 52.

### Table 3 One stack order for Freedom of Spelling in Brazilian Portuguese, from the deck top to its face

|   |   |   |   |   |   |
|---|---|---|---|---|---|
| 1 | sete de copas | 14 | quatro de espadas | 27 | dez de paus |
| 2 | dama de espadas | 15 | oito de copas | 28 | cinco de ouros |
| 3 | reis de copas | 16 | cinco de espadas | 29 | nove de copas |
| 4 | seis de paus | 17 | dama de ouros | 30 | nove de paus |
| 5 | reis de ouros | 18 | tres de ouros | 31 | valete de paus |
| 6 | cinco de copas | 19 | seis de espadas | 32 | dama de paus |
| 7 | dois de copas | 20 | dez de espadas | 33 | as de copas |
| 8 | oito de espadas | 21 | oito de ouros | 34 | nove de espadas |
| 9 | nove de ouros | 22 | quatro de ouros | 35 | dois de ouros |
| 10 | dois de espadas | 23 | cinco de paus | 36 | tres de paus |
| 11 | seis de ouros\textsuperscript{1} | 24 | sete de paus | 37 | valete de ouros |
| 12 | as de paus | 25 | dez de ouros | 38 | valete de espadas |
| 13 | quatre de paus\textsuperscript{2} | 26 | dama de copas | 39 | valete de copas |

\textsuperscript{1} The seis de ouros is a matching card in the original stack order
\textsuperscript{2} The quatre de paus matches with its spelling length plus one
A solution was found 93.2% of the time (95% CI [0.90, 0.95], Wilson score), with a mean time of 29.94 minutes (95% CI [25.99, 33.90]). Figure 9 shows the long-tailed histogram of those successes. Two types of failures were registered; 31 failures were due to the four-hour time limit, and three were due to out of memory errors.
We conclude that simplifying the number of clauses not only result in a lesser probability of failure, but the usage of the distinct clause implies that the running time from random runs is more likely lower than the original approach ($p < .001$, two-sided Wilcoxon rank sum test).

### 4.6 The final model for U.S. English

With this new information, we change the model to explore lesser constrained domains while using simpler constraints provided by Z3Solver. Phase 5 consists of three scenarios of 50 passes over space_full within the 4 hour time constraint.

The first scenario provided the baseline using the original set of constraints. Out of 50 runs, 36 timed out, and 14 stopped on out of memory errors providing a 0% success rate (95% CI [0%, 7.13%]).

The second scenario used the distinct clause known from the previous experiments to run slower but with a better success rate. With an 18% success rate (95% CI [9.77%, 30.79%]), all failed attempts were due to time outs. We achieved those nine successes in a mean time of 177.77 minutes (95% CI [146.61, 208.93]).

The third scenario tried to loose even more the requirements on the effects. Based on the previous pass, the card $x$ must be at the exact length($x$) + 1 position instead of allowing the two outs. All 50 runs timed out.

Even though this does not rule out the possibility of a solution for this third scenario, considering the binary variable of being a solution, 50 random with 0% success rate provides the same 95% CI [0%, 7.13%] found earlier on the first scenario.

The less constraining result achieved during the second scenario of this phase allows for any cuts, as long as the performer is allowed two outs. The resulting stack can be found in Table 4.

Again, as per Theorem 2, we find the ace of hearts at position 11 and the eight of diamonds at position 16.

| 1  | jack of clubs | 14 | six of spades | 27 | five of clubs | 40 | ten of clubs |
| 2  | jack of diamonds | 15 | five of hearts | 28 | six of diamonds | 41 | three of spades |
| 3  | five of spades | 16 | eight of diamonds | 29 | ace of spades | 42 | king of hearts |
| 4  | ten of spades | 17 | four of hearts | 30 | two of spades | 43 | six of hearts |
| 5  | two of clubs | 18 | ten of hearts | 31 | five of diamonds | 44 | seven of spades |
| 6  | ace of clubs | 19 | three of diamonds | 32 | two of hearts | 45 | nine of clubs |
| 7  | queen of diamonds | 20 | four of clubs | 33 | king of diamonds | 46 | queen of hearts |
| 8  | four of diamonds | 21 | seven of diamonds | 34 | two of diamonds | 47 | six of clubs |
| 9  | ten of diamonds | 22 | nine of diamonds | 35 | seven of hearts | 48 | ace of diamonds |
| 10 | three of clubs | 23 | eight of hearts | 36 | nine of hearts | 49 | eight of clubs |
| 11 | ace of hearts$^1$ | 24 | three of hearts | 37 | eight of spades | 50 | jack of hearts |
| 12 | queen of spades | 25 | jack of spades | 38 | queen of clubs | 51 | nine of spades |
| 13 | seven of clubs | 26 | four of spades | 39 | king of spades | 52 | king of clubs |

$^1$ The ace of hearts is one matching card in the original stack order

$^2$ The eight of diamonds can be found at its spelling length plus one
4.7 Preshow and performance

During the performance, the simulation function is run once to obtain the matching card for the set of parameters chosen by the volunteer.

A professional magician performed the effect online in U.S. English, Korean, French and Brazilian Portuguese. In a close-up presentation, it is preferred to use no technology such as card markings, explained in Appendix A.6.

One can also use low technology such as a clicker to input the position of the cuts. In this case, a smartwatch or hidden thumper can notify the magician which card is at its expected position. These technologies are already in use in traditional close-up magic routines as described in the same Appendix.

In a performance, Fig. 4 shows the stacked deck from Table 1. A volunteer makes the first cut, as in Fig. 10.

Suppose that the cuts are made at positions 2, 20 and 30. When the volunteer rebuilds the stack using the sequence [3, 2, 1, 4], the resulting 10th to 15th cards can be seen in Fig. 11: the two of spades, five of spades, ten of spades, two of clubs, ace of clubs and queen of diamonds.

In this case, the ten of spades is at position 12, its length plus 1. Also, the queen of diamonds is at position 15, its exact length. Therefore, the magician has two options. Suppose they decide to go with the ten of spades. In a second deck, they display a single card face down, the ten of spades. They spell ten of spades while dealing 12 cards, revealing the ten of spades, as in Fig. 12. A Brazilian Portuguese and American English performance was made available [28] together with the explanation presented during Constraint Programming 2021 [29].

Fig. 10 Controlling the first cut of a volunteer

Fig. 11 Peeking at the rebuilt deck for parameters [2,20,30,3,2,1,4]
4.8 A Minizinc model

The Z3Solver was first chosen due to its integration with Python which allowed for a programmatic line-by-line definition of the model. Minizinc allows the model to be presented in a shorter and more compact form, and therefore a Minizinc version was duly developed.

Listing 5: Exploring the 6-dimensional parameter space.

```python
for cut1 in range(2, 49):
    for cut2 in range(cut1, 50):
        for cut3 in range(cut2, 51):
            for sequence in permutations(range(4)):
                cuts = (cut1, cut2, cut3)
                rule = freedom_of_spelling(all_vars, cuts, sequence)
                rules.append(rule)
```

The parameter space and the lengths of the name of the cards are parameters that allow the user to explore different spaces and languages, as seen in Listing 6.

The variable `position` defines the starting position of each card. A simulation function analogous to Listing 1 is responsible for generating all possible deck results when exploring the space parameter; both are found on Listing 7.
Listing 7: The position variable and helper functions to simulate the deck shuffling process.

```plaintext
array [CARD] of var int: position;

% a range function
function array [int] of int:
    cut(int: from, int: length) =
    from + 1..from + length;

% number of cards in each packet
function array [1..4] of int:
    lengths(array [1..5] of int: cuts,
            array [1..4] of int: seqs) =
    [cuts[seqs[i] + 1] - cuts[seqs[i]] | i in 1..4];

% simulate the deck procedure
function array [1..52] of int:
    simulate(array [1..5] of int: cuts,
              array [1..4] of int: seqs) =
    let {
        array [1..4] of int: len = lengths(cuts, seqs),
    } in cut(cuts[seqs[1]], len[1]) ++
     cut(cuts[seqs[2]], len[2]) ++
     cut(cuts[seqs[3]], len[3]) ++
     cut(cuts[seqs[4]], len[4]);
```

Constraints are added to limit the cards between positions 1 and 52, to ensure they are all at distinct starting positions in the deck. Finally, for all possible point in the parameter space, a set of constraints is added to require at least one card in its expected position, as shown in Listing 8.
Running the model with parameters for U.S. English and the \textit{space\_optimized} parameter space with the default Gecode 6.3.0 and Chuffed 0.10.4 settings did not provide a valid solution in 24 hours of runtime.

\textsc{Z3Solver} allows for a programmatic python creation of the model which translates the constraints into \textit{strings} and send them to their solver. Version 0.6.0 of MiniZinc Python supports a string-based definition. In order to benefit from generating unique constraints as optimized using Python and \textsc{Z3Solver} in Section 4.1.2, one would have to generate the unique constraints in Python, translate them into a \textit{string}, which will then be loaded and translated by MiniZinc into its internal model.
Therefore, as long as the implementation of the model requires an extra step to remove constraint duplicates, the combination of Python with Z3Solver seems to provide an easier solution.

5 Conclusion and further research

We presented a novel problem formulation to the design process of new effects in magic. Our implementation proves that new magical effects can be devised and implemented through the use of computer-aided design (CAD) frameworks. We devised and applied one effect using constraint programming, giving the volunteers more freedom while keeping control of the results with the magician.

We presented multiple solutions in American English and Brazilian Portuguese. We released the entire source code as open source [18] and a new project as a library [19]. As both are open source, anyone can run it using a different language, therefore, limiting some of the unintentional biases and boundaries that a magical effect has due to the identity and experience of its creator.

A few questions are left open for further study: (1) What is the minimum amount of cards required to be stacked in position while still retaining control for a specific language? (2) Whether two is the minimum amount of different resulting cards required to perform such control?

Optimizing and simplifying the model can be further researched by using other solvers and trying to remove the constraints from the parameter space. We are exploring other effects, such as allowing a volunteer to choose the language to be used after the performer has handled the deck.

Generative magic can be explored with other areas of mathematics and computer science. Its ultimate research challenge is to search and catalogue the existing effects in magic literature in programming terms so that an engine can co-design new ones.

Appendix: A Mentioned effects

Magic is an uncommon theme in the field of constraint programming. Therefore, this appendix further describes a few of the effects mentioned earlier.

A.1 P.A.T.E.O. force

During a dinner event, the performer writes ‘bottle’ on a piece of paper and leaves it folded over the table in plain sight. Nobody but the performer knows what is written as this is the magician’s prediction. A volunteer is chosen, and the performer selects and names five items for them to play with, such as a napkin, glass, bottle, fork and knife.

The magician points to two of those items. The volunteer is asked to select one amongst the two to remove. With four elements left, it is the volunteer’s turn to point to two items. The magician selects one amongst the two and removes it. This process is repeated until there is only one element, which will always be the prediction element - in this case, the bottle.

For this effect to work, the magician must always select two items that do not include the item predicted by the magician. If the volunteers, during their turn, point to the two
items that do not include the prediction element, the magician can remove any element. If the volunteer points to one item and the prediction item together, the magician must remove the former. As the magician goes first to show how it works, the number of items must be odd for the effect to work.

A.2 Riffle force

A common method of forcing a card is to riffle through the cards and ask a volunteer to utter the word “stop”. At this moment, the magician controls the riffle to drop enough cards and force a predetermined one, the force card.

While the volunteer had freedom to utter “stop” at any moment, their decision has no effect upon the resulting selected card.

A.3 The Faro shuffle

Although called a shuffle, the Faro is a movement that controls the entire deck. A typical deck of 52 cards is split into two stacks of 26 cards. Each stack is interwoven, which results in a single stack where all even cards come from the original top and odd ones from the original bottom. The result might be the inverse according to which card is the first one to interweave.

The magician must practice the precise cut and interweave movements, as to pretend to be doing an uncontrolled shuffle, while in reality controlling the position of the cards.

The simplest usage of a series of Faro shuffles is to force one card into a specific target position.

A.4 Stacked decks

Stacked decks can be partially or completely memorized and used throughout magic performances. The sequence of the cards provides interesting properties that the magician uses throughout their performance.

For example, the cards at positions 10 to 15 in the Aronson Stack are the ace of clubs, ten of spades, five of hearts, two of diamonds, king of diamonds and seven of diamonds. All of them are at their exact spelling position when using the U.S. English language.

A.5 Invisible deck

The Invisible Deck is traditionally performed by first displaying a closed deck case. After asking a volunteer to name any card, the magician opens the case and shows that there is only one card face down. It happens to be the card named by the volunteer.

The widely used Invisible Deck can not be examined after the effect is performed because its gimmick is easily perceived when the cards are manipulated. Michael Close’s variation [30] does not use any gimmick and is fully examinable after the reversed card is revealed.

Invisible decks are used as finishers to show a matching prediction. However, its usage in Freedom of Spelling is innovative as it forces an intermediate outcome.
A.6 Card markings

Magicians will mark the back of each card with one or more symbols that indicate which card it is. There are many systems that allows for a magician to quickly identify the card while remaining invisible for a spectator who casually handles the card and does not know what to look for [24].

While slowly spreading the deck face down, the magician can then look at the cards laying on the table and identify which one is at its expected position.

A.7 Clicker, thumper and smart watches

It is common for magicians to use hidden communication forms so they can receive information without guests realizing. A clicker is a device that when clicked sends information to a receiver, typically a thumper which hits the leg of the magician once for every click.

More modern implementations of hidden communication on the stage involve the a helper of the magician, hidden amongst the guests, who sends information from their own smartphones to the magician’s smart watch.

For example, in the effect presented in this paper, if the cuts and the re-positioning of the packets of cards are performed with marked cards, a hidden helper can input this data in their device, sending the expected result of the magic to the magician.

Acknowledgements  The basis of this work was first presented at CP’2021. We thank the CP and the Constraints reviewers for their suggestions. Thanks to Daniela Mikyung Song for assistance in providing the card designs and illustrations. Thanks for Carlos Eduardo Ferreira and Gabriel Morete de Azevedo for helping with describing the model.

Funding  No funds, grants, or other support was received.

Declarations

Competing interests  The authors have no competing interests to declare that are relevant to the content of this article.

Open Access  This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article’s Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article’s Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

References

1. Hartling, P. (2016). In order to amaze. Germany: Self-published.
2. Hugard, J., Crimmins, J.J., & Gravatt, G.G. (1974). Encyclopedia of card tricks. New York: Dover Publications.
3. Diaconis, P., Graham, R.L., & Kantor, W.M. (1983). The mathematics of perfect shuffles. Advances in Applied Mathematics, 4(2), 175–196. https://doi.org/10.1016/0196-8858(83)90009-X. Accessed 2021-04-16.
4. Minch, S. (1994). The collected works of Alex Elmsley Vol. 2. Tahoma, California: L & L Publishing.
5. Morris, S.B., & Hartwig, R.E. (1976). The generalized faro shuffle. *Discrete Mathematics, 15* (4), 333–346. https://doi.org/10.1016/0012-365X(76)90047-9.
6. Morris, S.B. (1975). The basic mathematics of the faro shuffle. *Pi Mu Epsilon Journal, 6*(2), 85–92.
7. Aronson, S. (1979). *A stack to remember*. United States of America: Self-published.
8. Ortiz, D. (1986). *Darwin Ortiz on casino gambling: The complete guide to playing and winning*, 1st edn. New York: Dodd, Mead & Co.
9. Marlo, E. (1974). *The ten hand poker stack*. Chicago: Self-published.
10. Morris, S.B. (1975). The basic mathematics of the faro shuffle. *Pi Mu Epsilon Journal, 6*(2), 85–92.
11. Regal, D. (1999). *Close-up & personal*. Seattle, Washington: Hermetic Press. OCLC: 45272617.
12. Hugard, J., & Braue, F. (1979). *The royal road to card magic*. London, United Kingdom: Farber and Farber. OCLC: 25036735.
13. Arts, N. (1999). *Enciclopedia de Los Juegos de Cartas*. Barcelona, Spain: RobinBook. OCLC: 807845266.
14. Cassidy, R. (1984). *The art of mentalism*. United States of America: Collectors’ Workshop.
15. Gardner, M. (1956). *Mathematics, magic and mystery*. New York: Dover Publication.
16. Miller, H. (1978). *Baker’s Bonanza*, 2nd edn. Devon, England: Supreme Magic Publication.
17. Giobbi, R., Giobbi-Ebnoether, B., & Hatch, R. (1996). *Roberto Giobbi’s Card College*. Seattle, Washington: Hermetic Press, Inc. OCLC: 937750892.
18. de Azevedo Silveira, G. (2021). Freedom of Spelling source code. https://github.com/guilhermesilveira/freedom-of-spelling.
19. de Azevedo Silveira, G. (2021). Generative Magic. https://github.com/guilhermesilveira/generative_magic/.
20. de Azevedo Silveira, G. (2021). Generating magical performances with constraint programming. In L.D. Michel (Ed.) 27th *international conference on principles and practice of constraint programming (CP 2021)*. *Leibniz International Proceedings in Informatics (LIPIcs)*. https://doi.org/10.4230/LIPIcs.CP.2021.10, https://drops.dagstuhl.de/opus/volltexte/2021/15301, (Vol. 210 pp. 10–11013). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik.
21. Gravatt, G. (1936). *Encyclopedia of self-working card tricks*. United States of America: Quality Magic Company.
22. Close, M. (2004). *Closely guarded secrets*, 2nd edn. Toronto, Canada: MichaelClose.com.
23. van Zuylen, A., & Schalekamp, F. (2004). The Achilles’ heel of the GSR SHUFFLE: A note on New Age Solitaire. *Probability in the Engineering and Informational Sciences, 18*(03). https://doi.org/10.1017/S0269964804183034. Accessed 02 Nov 2021.
24. Pete, M., & Close, M. (2019). *The PM Card System*. Toronto, Canada: MichaelClose.com.
25. Bobo, J.B. (1947). *Watch this one!* California: Lloyd E. Jones.
26. Harris, C.R., Millman, K.J., van der Walt, S.J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N.J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M.H., Brett, M., Haldane, A., del Río, J.F., Wiebe, M., Peterson, P., …. Oliphant, T.E. (2020). Array programming with NumPy. *Nature*, 585(7825), 357–362. https://doi.org/10.1038/s41586-020-2649-2.
27. de Moura, L., & Björner, N. (2008). Z3: An efficient SMT solver. In D. Hutchison, T. Kanade, J. Kittler, J.M. Kleinberg, F. Mattern, J.C. Mitchell, M. Naor, O. Nierstrasz, C. Pandu Rangan, B. Steffen, M. Sudan, D. Terzopoulos, D. Tygar, M.Y. Vardi, G. Weikum, C.R. Ramakrishnan, & J. Rehof (Eds.) *Tools and algorithms for the construction and analysis of systems*. https://doi.org/10.1007/978-3-540-78800-3_24. Accessed 09 Apr 2021, (Vol. 4963 pp. 337–340). Berlin, Heidelberg: Springer.
28. de Azevedo Silveira, G. (2021). Generative magic performance example with constraint programming. https://youtu.be/5Dm8484dpCY Accessed 24 Nov 2021.
29. de Azevedo Silveira, G. (2021). CP2021 “Generating magical performances with constraint programming”. https://www.youtube.com/watch?v=Qbloc15mb_E. Accessed 24 Nov 2021.
30. Close, M. (1996). *Workers number 5*. United States of America: Self-published.

**Publisher's note** Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

**Supplementary information** The entire codebase used to generate the results found on this paper, including its output, were shared on Github and deposited at Zenodo. The videos demonstrating the performance in American English and Brazilian Portuguese were shared on YouTube and deposited at Zenodo.