7-2017

Educational Magic Tricks Based on Error-Detection Schemes

Ronald I. Greenberg
Loyola University Chicago, Rgreen@luc.edu

Educational Magic Tricks Based on Error-Detection Schemes

Ronald I. Greenberg
Loyola University
Department of Computer Science
820 N. Michigan Ave.
Chicago, Illinois 60611-2147, USA
rig@cs.luc.edu

ABSTRACT

Magic tricks based on computer science concepts help grab student attention and can motivate them to delve more deeply. Error detection ideas long used by computer scientists provide a rich basis for working magic; probably the most well known trick of this type is one included in the CS Unplugged activities. This paper shows that much more powerful variations of the trick can be performed, some in an unplugged environment and some with computer assistance. Some of the tricks also show off additional concepts in computer science and discrete mathematics.

KEYWORDS

computer science education; computational thinking; magic; outreach; public engagement; unplugged activities; discrete mathematics; error detection; error correction; parity checks; pigeonhole principle; permutations; counting principles; modular arithmetic; multidimensional representations; bijections; probability; analysis of algorithms

1 INTRODUCTION

Using magic[1] tricks for computer science education and outreach has been advocated by a number of previous authors. For example, Curzon and McOwan report on presenting 3-hour-long magic shows to gifted students [3]. A series of SIGCSE special sessions has also presented magic tricks that utilize computational thinking concepts and have strongly engaged large audiences of computer scientists [6, 7, 9]. A large pool of computing-related magic tricks also can be found at the "Computer Science For Fun" website, particularly through the "Magic of Computer Science" page [4]. Finally, the trick presented below as "Version 2a" was used by the author (and occasionally a student assistant) in some of the outreach presentations described in [15] that reached several thousand students. Students generally expressed great fascination with the trick and typically wanted to repeat it if time permitted. In surveys of over 200 students, 79% rated the "magic tricks" portion as "Good" or "Very Good" (as opposed to "Poor" or "Fair").[2]

One trick that has become particularly well known through the CS Unplugged collection of activities is based on using parity checks for error detection [1, pp. 35–37]. This activity is recommended for ages 7 and up and may well amaze older students as well, but the trick is quite simple and might not impress sophisticated viewers. This paper shows that much more powerful variations of the trick can be performed, some in an unplugged environment and some with computer assistance. These tricks also show off additional computing concepts besides parity checking.

The remainder of this paper first explains the CS Unplugged error detection trick and then provides variations that are suggested to be performed (with an explanation each time) as an escalating series of tricks. (While the CS Unplugged trick is actually an error *correction* scheme, it is titled as error *detection* for the simple parity-check scheme on which it is based. The other tricks presented in this paper also perform *correction*.)

For all versions of the trick described below, a demo may be viewed at http://rig.cs.luc.edu/~rig/errdetectmagic/errdetect.html; simply append `?v=1`, `?v=2a`, etc. to select the desired version of the trick. Downloading the single source file will also provide a full HTML/JavaScript implementation that should run locally in any modern browser, and a copy of the file will be archived with a version of this paper under http://ecommons.luc.edu/cs_facpubs. (The demo does not perform the role of the magician but does provide all the other steps.)

2 VERSION 1: CS UNPLUGGED

The CS Unplugged setup works as follows, with a magnetic board and 36 magnetic tiles that are colored on one side only:

(1) A volunteer is asked to lay out a 5x5 grid of the magnetic tiles on the board with a "random" mixture of colored and uncolored sides showing.
(2) The magician casually adds a sixth row and column "to make it a bit harder". (This statement is actually untruthful, so it might be better to simply say "to make it a bit larger".)
(3) The volunteer flips a tile while the magician looks away.
(4) The magician looks back at the board and announces which tile was flipped.

The secret of this trick is that the magician adds an extra tile at the end of each row with the exposed side chosen so that the number of colored tiles in the row is even. Then the magician adds an extra

[1] The term magic in this paper does not refer to any supernatural effects or even sleight of hand. Nonetheless, use of this term is well within several dictionary definitions of "magic", and the tricks presented in this paper are not unlike many other "magic" tricks presented in the references and other sources that are based purely on mathematical or scientific phenomena. Furthermore, there are certain elements of showmanship involved that we might think of as constituting "sleight of mind".

[2] Some of the presentations used instead, or additionally, the 1–125 number-guessing magic trick at [10]. Only a small portion of the total students reached were surveyed, due to the complicated requirements of research involving human subjects when interviewing students under age 18.

ITiCSE '17, July 03-05, 2017, Bologna, Italy

tile at the bottom of each column with the exposed side chosen so that the number of colored tiles in the column is even. In computer science terms, we would say that the magician is ensuring that each row and column has even parity. After the volunteer makes the flip that the magician does not see, the magician looks at the grid of tiles to see which row and column have odd parity, and the tile that was flipped is at the intersection of that row and column.

It is straightforward to extend this trick to a larger $n \times n$ grid, and, when the audience is not too large, I like to do it with an Othello™ set on a table, starting with a 7×7 arrangement of the black and white pieces and then extending to 8×8. Additionally, an interactive demonstration for arbitrary n is available at http://rig.cs.luc.edu/~rig/errdetectmagic/errdetect.html?v=1. (Make the appropriate addition to the query string for the desired value of n after the extension; e.g., the default is the same as appending `&n=6`.)

But the clever observer may recognize that addition of the extra full row and column of tiles is not making the trick harder but rather exploiting a simple rule, especially if the extra row and column are added by computer, or if the magician needs to proceed a bit slowly and deliberately to add the extra tiles.

Even this basic version of the trick can teach about parity checking and the XOR (exclusive or) operation as well as the common technique of identifying a cell in a 2D grid by specifying row and column number (for example to address memory cells). In successive versions of the trick, however, additional ideas will be introduced.

3 EASILY PERFORMED VERSIONS WITH LESS MAGICIAN INTERVENTION

A viewer who knows just a little bit about information theory may realize that the CS Unplugged version of the error detection trick is associating a lot of check bits with a modest amount of data. Specifically, the magician is placing $2n$ check bits on an array of n^2 bits. In principle, viewing the n^2 bits as a linear sequence (e.g., row-major order) and using a Hamming code [11], one would only need to add $\lg(n^2+1) \approx 2 \lg n$ bits (which can be shown to be optimal in the context of transmitting data with single-error correction), but this is not a very easy scheme for a human magician to use.[3] In Section 5, we will see a version of the error detection trick that is difficult to do without computer assistance, but, in this section, we will stick to schemes that are humanly manageable and are more impressive than the CS Unplugged version of the trick. (When we do proceed to a version that relies heavily on the computer, it will appear that we are violating the optimality of Hamming codes; the reality is that we can "beat" Hamming codes because there isn't actually any underlying data being transmitted that we need to retain as we create appropriate checks.)

3.1 Version 2a

The fancier version of the CS Unplugged trick that I have performed to the delight of many student groups may be explored interactively at http://rig.cs.luc.edu/~rig/errdetectmagic/errdetect.html?v=2a or in its original home among other materials for high school outreach presentations [10]. Here, with the default 8×8 grid, the magician requests at most 3 flips in the grid generated by the audience volunteer and/or randomization. (Larger versions can be run by appending, e.g., `&n=10` to the URL.)

The steps of the trick, most readily performed at a computer screen, are as follows:

(1) A random 8×8 grid of black and white tiles is generated, and a volunteer is asked to flip any desired tiles to make sure the pattern is complicated.
(2) The computer marks a set of at most three tiles that the volunteer is asked to flip. (A practiced magician could do this manually, but the computer assist makes it quicker and allows the magician to perform the trick without even looking at the grid at all until step 4.)
(3) The volunteer flips a tile while the magician looks away.
(4) The magician looks back at the board and announces which tile was flipped.

To explain this trick we need to explain how Steps 2 and 4 work. Step 2 is the much more complicated one, and the reader should remember it can be done quickly by the computer.

After Step 1, the computer (or magician) can determine the parities of the first seven rows (i.e., whether each row has an odd or even number of colored tiles). We are guaranteed to find that at least four of these rows have the same parity; let us call this the "majority" parity. (In explaining this to students we introduce a new concept, the generalized pigeonhole principle.) We now make note of the three or fewer rows among the first seven that have the other or "minority" parity; these are rows in which we would like to flip (change) the parity, and we will call them the "flip" rows. Next we apply the same process to the first seven columns, and we similarly find three or fewer "flip" columns.

We can achieve all the desired parity flips, by flipping the tile at the intersection of the first flip row and first flip column, the tile at the intersection of the second flip row and second flip column, etc. If there are actually fewer than three flip rows or fewer than three flip columns, we will run out of rows or columns to use in these pairings, but we can simply go to the last (eighth) row or column if we run out of flip rows or columns, respectively. In this way, we identify a set of at most three tiles that the volunteer is asked to flip, and the result then is that the first seven rows all have the same parity and the first seven columns all have the same parity. Figure 1 provides a screen shot from a sample run at the stage when the desired flips are presented to the volunteer.

Finally, it is straightforward (even mentally) for the magician to perform Step 4 by looking for the row and column among the first seven that has a different parity from the others. (If all seven have the same parity, he knows that the flip occurred in the eighth row; similar reasoning works with the columns.)

Everything described above for version 2a is readily generalized to an $n \times n$ grid, with the number of flips requested by the computer being at most $\lfloor (n-1)/2 \rfloor$. While there is no conceptual change as n increases, the amount of work (mental juggling) for the magician does increase. (It is possible to reduce the work and increase the impressiveness a bit with the next variation.) While the number of flips the computer requests is at most $\lfloor (n-1)/2 \rfloor$, it will sometimes be less, and Appendix A shows how to compute the probability distribution for the number of flips required given a random grid.

[3] The mechanics are actually similar to the workings of the previously mentioned number guessing trick (e.g., [10, 12]), but that trick imposes an organization on the relevant information that is not readily available from looking at just a 2D grid of bits.

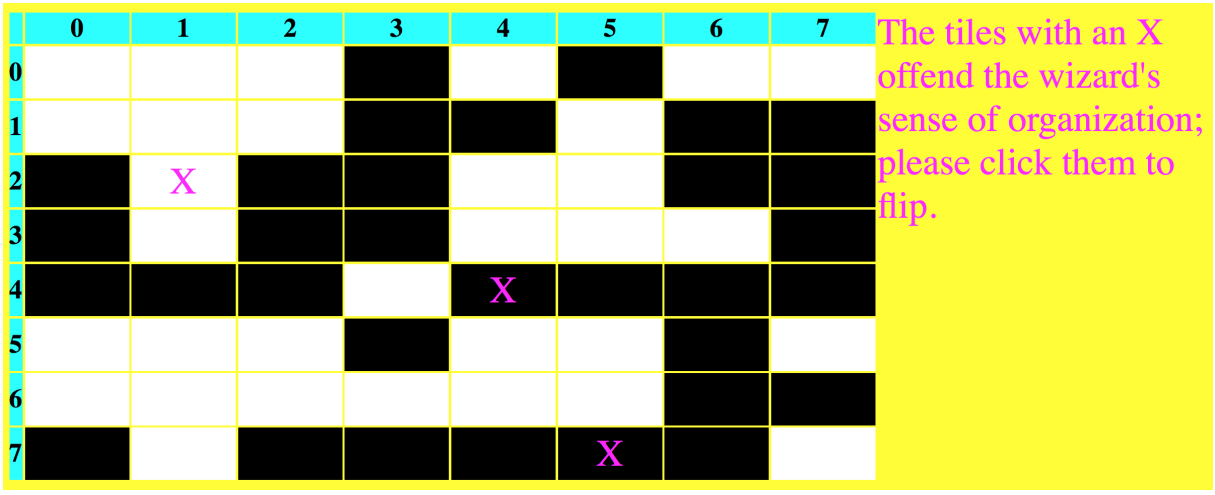

Figure 1: A sample run for Version 2a at the stage where the subject is asked to flip at most three tiles, after which the magician can identify the next tile flipped.

3.2 Version 2b

This version of the trick works for an $n \times n$ grid with $n \equiv 1 \pmod 3$. The basic idea here is for the magician to operate as in Version 2a but with Steps 2 and 4 performed in accordance with a grouping of all but the last row into sets of three (and similarly for the columns). By the pigeonhole principle, at least two of the first three rows have the same parity, so at most one of these rows becomes a flip row. There also is at most one flip row in the next three rows, and so on, for a total of at most $(n-1)/3$ flip rows. Similarly, there are at most $(n-1)/3$ flip columns. Pairing flip rows and columns as in Version 2a (and defaulting to the last row or column if we run out of flip rows or flip columns), the audience volunteer is asked to flip at most $(n-1)/3$ tiles to achieve the same parity within each group of three rows and the same parity within each group of three columns. Again, this calculation for Step 2 can be done with a computer assist.

The magician then can easily complete Step 4 by looking for a minority parity in one of the row groups (otherwise defaulting to the last row), and similarly for the columns.

In the case of $n = 7$, the volunteer is asked to flip at most two tiles. (The analysis of the probability distribution for the number of tiles needing to be flipped is again deferred to Appendix A.)

Note that when n is odd, greater care is required in computing the parity of each row and column. It is necessary to consistently count either the number of white tiles or consistently count the number of black tiles, whereas one can count either when n is even. For this reason, the magician may prefer to work with $n = 10$, while limiting the number of organizing flips to 3 as was the case in Version 2a that only worked with $n = 8$. (There also is an increased likelihood here relative to Version 2a of needing fewer than 3 flips, as shown in Appendix A.) An interactive demo for this version defaulting to a 10×10 grid may be explored at http://rig.cs.luc.edu/~rig/errdetectmagic/errdetect.html?v=2b and different values of $n \equiv 1 \pmod 3$ may be utilized by appending, e.g., &n=7.

4 "CHEATING" TO ACHIEVE FEWER MAGICIAN-REQUESTED FLIPS

Garcia and Ginat have occasionally performed tricks involving a secret communication between the two of them, for example crafting a sentence so that the number of words conveys some information [8]. They refer to this as "cheating", and the same terminology is adopted here to describe secret communication from the computer or an assistant to the magician. The trick variations in Sections 3 do not involve any such cheating even though a computerized assistant provides a convenient way for the magician to quickly designate flips that he desires to organize the grid. (While a nimble magician could bypass the computerized assistant, I suggest using it to demonstrate that the magician does not even need to look at the grid until after the last flip and is certainly not memorizing anything about the grid arrangement or receiving any communications.) Having completed versions of the trick as in Section 3, however, the magician may magnify the feat performed by using a bit of subtle cheating as described below.

4.1 Version 3

An interactive demo for this version of the trick with a 10×10 grid can be explored at http://rig.cs.luc.edu/~rig/errdetectmagic/errdetect.html?v=3. In Version 2b, with $n = 10$, we can reveal one secret flip in the 10×10 grid after performing three organizing flips. Now we will explain how to reveal two flips, while staying at three organizing flips but using a bit of subtle communication of information from the computerized assistant.

The steps in this trick are as follows, with the magician not looking at the grid until Step 5:

(1) A random 10×10 grid of black and white tiles is generated, and a volunteer is asked to flip any desired tiles to make sure the pattern is complicated.
(2) The volunteer flips and remembers a tile.
(3) The computer presents at most three tiles, one at a time, that the volunteer is asked to announce and flip.

(4) The volunteer flips and remembers another tile.
(5) The computer prompts the audience to provide a "drumroll", and the magician looks back at the grid and announces the tile flipped at Step 4.
(6) The computer may prompt the audience to provide additional drumrolls (0, 1 or 2), and the magician announces the tile flipped at Step 2.

The reveal in Step 5 works just as in Version 2b, but one may observe that there is some flexibility in how the organizing flips are done, and that is the main source of communication to the magician. Specifically, there will typically be three flip rows and three flip columns, but we can use any ordering of the three flip rows and any ordering of the three flip columns before proceeding to pair them and proceed through the organizing flips. Since there are six possible orderings of the three flip rows and six possible orderings of the three flip columns, the computerized assistant can communicate enough information to discriminate between $6 \times 6 = 36$ possibilities. Here we are introducing the mathematical concept of how to count permutations as well as the multiplication principle for combining the information from the rows and the information from the columns.

(We can also communicate at least as much information if there are fewer than three flip rows or fewer than three flip columns. For example, if there are only two flip rows, we can use the last row as a third distinct flip row. If there is only one flip row, we can pick any other row to be used twice as a flip row, and by choosing an appropriate one from among the first six rows available we provide enough information to discriminate between six possibilities. Finally, if there is are no flip rows, we can use the last row as a flip row and use any other of the first six rows twice according to which of the six possibilities we wish to communicate.)

With the information communicated through the choice of organizing flips, we are nearly able to specify which of the 100 tiles in the grid was the last flip before the organizing flips. We can complete the cheating communication by communicating a number from one to three, which is sufficient to discriminate among $36 \times 3 = 108$ possibilities. Our subtle way to do this is through the number of drumrolls prompted at the beginning of Step 6. (The drumroll prompted at the beginning of Step 5 is just to get the audience to practice and to deflect attention from the true communication.)

In showing the way that we encode one of the 100 grid tiles by communicating three values with ranges of 6, 6, and 3 (not quite fully utilized), we can also introduce mathematical concepts such as encoding numbers using different choices of number base (radix) and construction of bijections.

4.2 Version 4

As a warmup for the final non-cheating trick in Section 5, we can reveal a flip in a very large grid with a single organizing flip but a heavy cheat. We can actually use an arbitrarily large n for this trick, but we will show it (http://rig.cs.luc.edu/~rig/errdetectmagic/errdetect.html?v=4) and explain it for $n = 11$. In this variation, we reveal a single secret flip after the computerized assistant requests the volunteer to announce and perform a single flip. There are actually a number of ways to do this easily (for example, request a flip to the tile that is in the mirror position across the diagonal of the grid from the secret flip), but a method that is reasonably subtle and likely to puzzle most audience members for a time is to use multiplicative inverses mod 11. Specifically, the computerized assistant can request a flip to tile (r, c) (row and column numbers starting at 0), such that the magician can then reveal the secret flip as being $(5r \bmod 11, 5c \bmod 11)$. Or to guard against the possible tendency of humans to pick $r = c$, one may want to increase the mystery by using two different values, at the expense of doing slightly more difficult arithmetic, for example, arrange that the secret flip will be $(5r \bmod 11, 7c \bmod 11)$; these are the details used in the referenced demo.

5 ORGANIZING ANY GRID WITH ONE FLIP

At this point, an audience that has been led through the above versions of the trick, has seen powerful variations with no "cheating" communication to the magician and extremely powerful tricks with such "cheating". We can now promise to go back to eliminating any opportunities for cheating communication but still increase the impressiveness of the trick. In fact, we will be able now to reveal a secret flip in an arbitrarily large grid by doing just one organizing flip beforehand and no cheating! The only wrinkle in this ultimate variation of the trick is that it is difficult for a human magician to complete it unaided. With practice, it should be manageable in an 8×8 grid, and I propose to create a smartphone app to perform the trick for larger grids. This smartphone app will photograph the grid after the secret flip occurs, but it will be operated by an audience volunteer who will first verify that the phone is in airplane mode so that it will be clear that no cheating communication is occurring. Following is a description of how this trick is performed in theory.

5.1 Version 5

In this version of the trick, we view the tiles as being arranged in a d-dimensional space with four positions in each dimension, i.e., 4^d total tiles. For smooth presentation, we will still display them in an $n \times n$ grid; the default size demonstrated in http://rig.cs.luc.edu/~rig/errdetectmagic/errdetect.html?v=5 is an 8×8 grid representing 4^3 tiles that can be thought of as comprising a 3-dimensional array of $4 \times 4 \times 4$ tiles. More generally, the demo can be run in higher dimensions by appending, e.g., `&d=4` to the URL.

The idea in this version of the trick is a conceptually simple extension of Version 2b with $n = 4$. In each of the d dimensions, we consider the four possible indices and focus on the first three values. We compute the parity for the array slices with values 0, 1, and 2 and pick at most one slice in which a tile must be flipped so that all three slices will have the same parity; if no such flip is needed, we select the slice at value 3. After doing this for each of the d dimensions, we determine the single tile at the intersection of all d slices. This is the single tile we request the volunteer to flip. To reveal the secret flip, we check each of the dimensions for a slice among the first three that has different parity than the others or default to the last slice. That is, revealing the secret flip is done through essentially the same process as determining which single organizing flip is desired.

The difficulty for a human magician is that once we get up to at least three dimensions, the slices are large, and some mental gymnastics must be performed to view where they lie within a simple

two-dimensional presentation. It is, however, quite straightforward mathematically to map between $d-$*dimensional* coordinates of tiles and positions in a two-dimensional grid. Thus, it should be feasible to program a smartphone app to perform this trick.

6 CONCLUSION

Through the medium of magic tricks based on binary error detection/correction, we have shown that students can be entertained and taught about many concepts in discrete mathematics, such as parity checks, the pigeonhole principle, permutations, counting principles, modular arithmetic, multidimensional representations, and bijections. The smartphone app proposed in Section 5 also can provide an interesting programming assignment. Finally, the analyses in the Appendices motivate delving into probability and analysis of algorithms for more advanced students.

ACKNOWLEDGMENTS

The author is supported in part by National Science Foundation grants CNS-1543217 and CNS-1542971.

A PROBABILITIES FOR NUMBER OF MAGICIAN-REQUESTED FLIPS

For more advanced students, the versions of the trick in Section 3 motivate additional analyses regarding the number of organizing flips the magician must request. (The analysis for Version 2b will also be applicable to Version 3 in Section 4.) We already have established upper limits on the number of organizing flips, but sometimes fewer flips will suffice. Here we analyze the probabilities of needing differing numbers of organizing flips. Some of the analysis is common to Versions 2a and 2b, and we start with that portion of the analysis before proceeding in the two different directions.

In either version, let us denote by l the upper limit on the number of organizing flips, i.e., $l = \lfloor (n-1)/2 \rfloor$ for Version 2a and $l = (n-1)/3$ for Version 2b. Also denote by $F(n,f)$ the probability of having exactly f flip rows. The analysis is the same for columns, so that $F(n,f)$ will also denote the probability of having exactly f flip columns. Now the probability of having *at most* f flip rows is

$$S(n,f) = \Sigma_{i=0}^{f} F(n,i) \ .$$

Finally, the probability of needing f flip tiles, $\Pr(f)$ is the probability of having f flip columns and at most f flip rows or vice-versa, minus the intersection of these two events, i.e.,

$$\Pr(f) = 2F(n,f)S(n,f) - (F(n,f))^2 \ .$$

A simplified case is when $f = l$; in that case we see $S(n,l) = 1$, and the probability of needing l flip tiles is $\Pr(l) = 2F(n,l) - (F(n,l))^2$.

A.1 Version 2a

In Version 2a, we assume for simplicity that n is even (with only small modifications otherwise needed). To complete the analysis, we just need to note that

$$F(n,i) = 2\binom{n-1}{i}/2^{n-1} = \binom{n-1}{i}/2^{n-2} \ ,$$

based on choosing exactly i of the first $n-1$ rows to have even parity or choosing exactly i to have odd parity. It does not seem feasible to give simpler expressions for the probability of needing f flip tiles in general, but we note in Table 1 the values for $n = 8$.

Table 1: The probability $\Pr(f)$ that f is the minimum number of magician-requested flips in Version 2a for $n = 8$. (Exact fractional values are given; for uniformity, they are not necessarily in simplest terms.)

f	$F(n,f)$	$S(n,f)$	$\Pr(f)$
0	1/64	1/64	1/4096
1	7/64	8/64	63/4096
2	21/64	29/64	777/4096
3	35/64	64/64	3255/4096

Table 2: The probability $\Pr(f)$ that f is the minimum number of magician-requested flips in Version 2b for $n = 10$.

f	$F(n,f)$	$S(n,f)$	$\Pr(f)$
0	1/64	1/64	1/4096
1	9/64	10/64	99/4096
2	27/64	37/64	1269/4096
3	27/64	64/64	2727/4096

A.2 Version 2b

In Version 2b, we complete the analysis by noting that

$$F(n,i) = \binom{l}{i}\left(\frac{3}{4}\right)^i\left(\frac{1}{4}\right)^{l-i} \ .$$

Again, it does not seem feasible to give simpler expressions for the probability of needing f flip tiles in general, but we note in Table 2 the values for $n = 10$.

Comparing Table 1 for Version 2a ($n = 8$) to Table 2 for Version 2b ($n = 10$), each with a maximum of 3 flip tiles, we see that the latter version improves the probability of getting by with fewer than 3 flips from about 1/5 to about 1/3.

B VERSION 5 COMPUTATIONAL EFFICIENCY

While all the trick versions presented in the paper illustrate computational concepts, most are actually computationally quite simple, such that they can be performed by a human. In the case of Version 5, however, a more involved computational organization is required, and the naive approach is not the most computationally efficient. This can motivate deeper analysis for advanced students, and we analyze here the required running time for a serial algorithm and even the required resources for a parallel algorithm. Recall that there are actually two computations, one to organize the array of tiles and one to reveal the secret flip performed by the audiene volunteer. But these two computations are the same, and we consider here the time to do it once. (Good references for fundamentals of algorithm analysis are [2, 13].)

Let us begin by defining some notation and determining the naive serial computation time. We have been working with tiles displayed in an $n \times n$ square corresponding to a d-dimensional quaternary array. That is $n^2 = 4^d$ (i.e., $n = 2^d$), and each tile

is addressed with a d-tuple of coordinates, $x_0, x_1, \ldots, x_{d-1}$, each coordinate having value 0, 1, 2, or 3. We denote by $P_d(C)$ the parity, i.e., XOR, of all the tiles in the d-dimensional quaternary array satisfying condition C. Our task then is to compute $P_d(x_i = v)$ for all $i \in \mathbb{Z}_d$ and $v \in \mathbb{Z}_3$ (with $\mathbb{Z}_m$ being the standard notation for the set $\{0, 1, \ldots, m-1\}$). Let $T(d)$ be the time to compute all these values. The naive approach is to compute each of these values independently as the XOR of $4^{d-1} = n^2/4$ bits, which involves a total of $3d\left(\frac{n^2}{4} - 1\right)$ XOR operations, which is $\Theta(n^2 \lg n)$, where, again, n^2 is the number of tiles.

B.1 Efficient Serial Computation

For more efficient computation, note that for $i \in \mathbb{Z}_{d-1}$,

$$P_d(x_i = v) = \bigoplus_{j=0}^{3} P_d(x_i = v \text{ and } x_{d-1} = j) \ . \tag{1}$$

For any fixed value of j, finding $P_d(x_i = v \text{ and } x_{d-1} = j)$ for all $i \in \mathbb{Z}_{d-1}$ and $v \in \mathbb{Z}_3$ involves computing in a quaternary array of dimension $d-1$, so to complete that computation for all $j \in \mathbb{Z}_4$, time $4T(d-1)$ is sufficient. Once we have done that, the computations indicated in (1) for all $i \in \mathbb{Z}_{d-1}$ and $v \in \mathbb{Z}_3$ can be completed with $3 \cdot 3 \cdot (d-1)$ XOR operations. Finally, we need to compute $P_d(x_{d-1} = v)$ for each $v \in \mathbb{Z}_3$, which is easy to do using results already available in any of the dimensions, e.g.,

$$P_d(x_{d-1} = v) = \bigoplus_{j=0}^{3} P_d(x_1 = j \text{ and } x_{d-1} = v) \ . \tag{2}$$

The computation of (2) for all $v \in \mathbb{Z}_3$ can be completed with just 9 XOR operations. Thus we obtain the following recurrence for the running time with initial condition $T(1) = 0$:

$$T(d) = 4T(d-1) + 9(d-1) + 9 = 4T(d-1) + 9d \ .$$

Rewriting in terms of $n = 2^d$ with $T'(n) = T(\lg n)$, we have

$$T'(n) = 4T'(n/2) + 9 \lg n$$

and $T'(2) = 0$, with solution $T'(n) = \Theta(n^2)$. Thus we shave a $\lg n$ factor off of the naive computation time, and we can see this is asymptotically optimal since we must inspect every one of the n^2 tiles except the one with all coordinates equal to 3.

B.2 Efficient Parallel Computation

For a parallel algorithm, we need some additional notation; let $x_{i,l}$ represent a tuple of l coordinates starting at x_i, i.e.,

$$x_{i,l} = (x_i, x_{i+1}, x_{i+2}, \ldots, x_{i+l-1}) \ .$$

Then for $v' \in \mathbb{Z}_4^{2l}$, let $S(v') \in \mathbb{Z}_4^l$ denote the starting half of the coordinates of v' and $E(v') \in \mathbb{Z}_4^l$ denote the ending half of the coordinates of v'. Further, let $T_l(d)$ and $W_l(d)$ be the time and work (total number of operations) to compute $P_d(x_{il,l} = v)$ for all $i \in \mathbb{Z}_{d/l}$ and $v \in \mathbb{Z}_4^l$, so that $T_1(d)$ time and $W_1(d)$ work suffice for the overall computation we need. Now the key relationships are

$$P_d(x_{il,l} = v) = \bigoplus_{\substack{v' \in \mathbb{Z}_4^{2l} \\ S(v') = v}} P_d(x_{il,2l} = v') \tag{3}$$

and

$$P_d(x_{il+l,l} = v) = \bigoplus_{\substack{v' \in \mathbb{Z}_4^{2l} \\ E(v') = v}} P_d(x_{il,2l} = v') \tag{4}$$

For fixed i and v, each of the computations in (3) and (4) is an XOR of 4^l values, which can be completed in $\Theta(\lg(4^l)) = \Theta(l)$ time with $\Theta(4^l)$ work. Thus, we can relate $T_l(d)$ and $W_l(d)$ to $T_{2l}(d)$ and $W_{2l}(d)$ by using (3) and then (4), each for i even. (Using the equations in sequence avoids a concurrent read to $P_d(x_{il,2l} = v')$ so that the results are valid even on an EREW PRAM.)

$$T_l(d) = T_{2l}(d) + \Theta(l) \tag{5}$$

and

$$W_l(d) = W_{2l}(d) + \Theta((d/l)(4^l)^2) \tag{6}$$

where (6) accounts for using (3) and (4) for d/l values of i and 4^l values of v.

Noting that $T_d(d) = W_d(d) = 0$, we can iterate (5) and (6):

$$T_1(d) = \sum_{j=1}^{\lg d - 1} \Theta(2^j)$$

and

$$W_1(d) = \sum_{j=1}^{\lg d - 1} \Theta\left((d/2^j)(4^{(2^j)})^2\right) \ .$$

In each case, the term with $j = \lg d - 1$ dominates, and we find $T_1(d) = \Theta(d)$ and $W_1(d) = \Theta(4^d)$, i.e., time $\Theta(\lg n)$ and work $\Theta(n^2)$ for n^2 tiles. This result matches the lower bound on work to compute even a single $P_d(x_i = v)$ as per the serial analysis, and it matches the lower bound on time for even a randomized CREW PRAM [5] and comes very close to the lower bound of $\Omega(\lg n / \lg \lg n)$ on the still more powerful randomized CRCW PRIORITY PRAM [14]

REFERENCES

[1] Tim Bell, Ian H. Witten, Mike Fellows, Robyn Adams, Jane McKenzie, and Sam Jarman. 2015. *CS Unplugged: An enrichment and extension programme for primary-aged students.* http://csunplugged.org/wp-content/uploads/2015/03/CSUnplugged_OS_2015_v3.1.pdf.

[2] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2009. *Introduction to Algorithms* (third ed.). MIT Press.

[3] Paul Curzon and Peter W. McOwan. 2008. Engaging with Computer Science Through Magic Shows. In *13th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education.* ACM SIGCSE, 179–183.

[4] Paul Curzon, Peter W. McOwan, et al. 2011. The Magic of Computer Science. http://www.cs4fun.org/magic. (2011). Accessed March 8, 2016.

[5] Martin Dietzfelbinger, Miroslaw Kutylowksi, and Rüdiger Reischuk. 1994. Exact Lower Bounds for Computing Boolean Functions on CREW PRAMs. *J. Comput. System Sci.* 48, 2 (1994), 231–254.

[6] Daniel D. Garcia and David Ginat. 2012. Demystifying Computing with Magic. In *SIGCSE '12.* Association for Computing Machinery, 83–84.

[7] Daniel D. Garcia and David Ginat. 2013. Demystifying Computing with Magic, continued. In *SIGCSE '13.* Association for Computing Machinery, 207–208.

[8] Daniel D. Garcia and David Ginat. 2016. Presentation associated with [9] of a trick not described in the written publication. (March 2016).

[9] Daniel D. Garcia and David Ginat. 2016. Demystifying Computing with Magic, part III. In *SIGCSE '16.* Association for Computing Machinery, 158–159.

[10] Ronald I. Greenberg. 2010. Activities (from high school presentation materials). http://www.illinoiscomputes.org/hspresent/what/activities. (Jan. 2010).

[11] R. W. Hamming. 1950. Error Detecting and Error Correcting Codes. *Bell System Technical Journal* 26, 2 (April 1950), 147–160.

[12] C. Heeren, T. Magliery, and L. Pitt. 1998. MATHmaniaCS Lesson 1: Binary Numbers. http://www.mathmaniacs.org/lessons/01-binary. (1998). Accessed 3/10/16.

[13] Joseph JáJá. 1992. *An Introduction to Parallel Algorithms.* Addison-Wesley.

[14] Miroslaw Kutylowski and Thomas Schwöppe. circa 1998. A lower bound for PARITY on randomized CRCW PRAMs. http://citeseerx.ist.psu.edu/viewdoc/versions?doi=10.1.1.46.5019 accessed 1/15/17. (circa 1998).

[15] Steven McGee, Ronald I. Greenberg, Dale F. Reed, and Jennifer Duck. 2013. Evaluation of the IMPACTS Computer Science Presentations. *The Journal for Computing Teachers* (Summer 2013), 26–40. International Society for Technology in Education, www.iste.org.