1999

Various pushing methods on grid graphs

Jiaxin Wang
West Virginia University

Various Pushing Methods On Grid Graphs

Jiaxin Wang

A thesis submitted to the College of
Engineering and Mineral Resources
at West Virginia University
in partial fulfillment of the requirements
for the degree of

Master of Science
in
Computer Science

West Virginia University

Department of Computer Science and
Electrical Engineering

Morgantown, West Virginia
1999

Keywords: Grid Graph, Push, Different Push Methods, Determinant

ABSTRACT

Various Pushing Methods On Grid Graphs

Jiaxin Wang

This thesis describes algorithms for determining if a grid of switches can be turned into all-off state from any initial configuration by various methods of activation operation (push). Among these push methods, besides the regular “+” push, “+” push with no center, “X” push, “X” push with no center, and a “V”-typed unbalanced push are studied. The research methods used in this work are mainly linear algebra and algorithm analysis. Results obtained include that the grid m x n is completely solvable with push “+” no center, if and only if co-prime (m+1, n+1).

TABLE OF CONTENTS

TABLE OF PICTURES

ACKNOWLEDGMENTS

At first, I wish to thank my advisor, Dr. William F. Klostermeyer, who led me into the research of such an interesting problem and gave me much encouragement and inspiration. Dr. Frances L VanScoy is greatly appreciated for serving as the chairperson and helping me in preparing the thesis and defense when Dr. Chip was in absence. Many thanks to Dr. Mooney and Dr. Callahan, who serve as members of graduation committee and gave me many suggestions on how to improve my thesis.

My wife Dongmei encouraged and supported me to finish the research.

Finally thank those great scientists who invented the whole system of linear algebra, what a great achievement.

CHAPTER 1 INTRODUCTION

The Problems

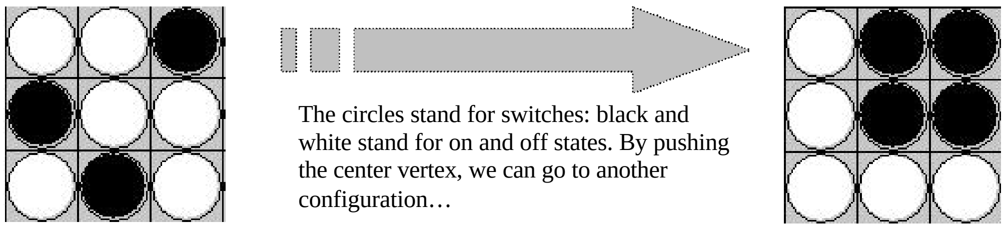
Original Problem

Consider an m x n rectangular array, or grid, of switches, each of which can be in the on or off position. An ***activation*** operation, also called a "***push***" operation, can change the state of the vertex and/or its neighbors. For example, a common "push" method on vertex v will change the state of v and its rectilinear neighbors. In this case, either three, four, or five switches change state, depending on whether the activated switch is in a corner, edge, or interior of the array, respectively. See the following picture for an example.

Given an initial setting, or configuration, of the switches, we seek to determine if there exists a sequence of activations that cause all the switches to be turned off. More generally, we want to know for what values of m and n, that is the size of the grid, it is possible to reach all-off state for every initial on-off configuration.

DEFINITION: We say that the pair (m, n) is ***completely solvable*** if every initial configuration state in an m x n grid can be transformed into the all-off state by a sequence of the pushes.

Figure 1 Picture of 3x3 problem for "+" push



The circles stand for switches: black and white stand for on and off states. By pushing the center vertex, we can go to another configuration…

Different Pushes

We also can use push methods other than the push method mentioned in the above example. Since the push in the last section generally caused "+"-shaped vertices to change states, we call it "Push +", or Push Plus (PP).

DEFINITION: If a push applied on vertex v changes the states of all its four rectilinear neighbors and itself, we call it a ***"+"-shaped push, or push plus.***

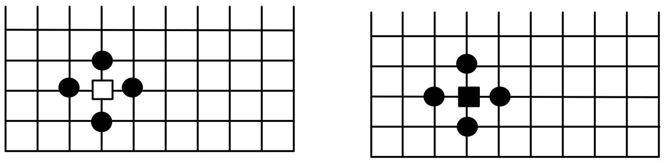


Figure 2 Push Patterns of Push Plus with No Center and Push Plus

In the picture "Push Plus with No Center", the left one shows the orignal configuration, and we apply the push at where the arrow points. The result is the right one, where we can see what vertices are changed – the applied vertex doesn't change state.

DEFINITION: If a push applied on vertex v changes the states of all its four rectilinear neighbors but not itself, we call it a ***"+"-shaped push with no center, or push "+" with no center.***

An example of push "+" with no center is above. In figure "Push Plus with No Center", we see a push on a vertex which doesn't change the state of the vertex itself. This push method can be called, for short, **PPNC**.

DEFINITION: If a push on vertex v changes the state of v, as well as four corner neighbors, i.e., the four vertices at top-left, top-right, bottom-left, and bottom-right place of v, we call it "**Push X**". It is because the five vertices form an "X"-shape; for short, this push is **PX**. If v itself is not changed on the push, we call it "**Push X with No Center**", for short, **PXNC**. These two push patterns are illustrated in above picture.

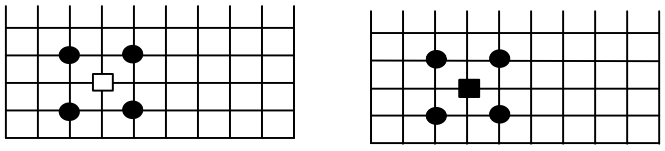


Figure 3 Patterns of Push X with No Center and Push X

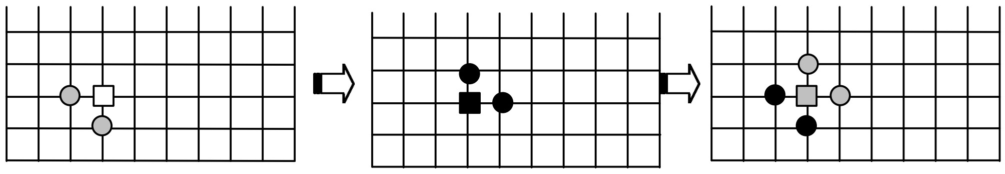
DEFINITION: If we rotate a push pattern by 90 degree, the resulting pattern is the same; we call this type of push a **Balanced Push**, otherwise, an **Unbalanced Push.**

There are various unbalanced push methods. Unlike the four pushes discussed above, these unbalanced pushes are not all symmetric. For example, we can push v to get the states of v and three of its rectilinear neighbors changed, or just two of its neighbors, as in the following figure. Among these many unbalanced pushes, we are interested in what kind of shape of the push can turn the m x n grid into all off state from any initial setting.

DEFINITION: We say that the pair (m, n) is ***completely solvable with certain push method*** if every initial configuration of states in an m x n grid can be transformed into the all-off state by a sequence of certain pushes. Here, the certain push can be Push "+", Push "X", and so on.

Modulo k Pushes

For these different pushes, there is another characteristic our research is interested in — the number of states a vertex can go through before being turned off. It means a vertex does not necessarily just have two states – on and off. It can have three or more states.



The pushes change vertices color from White to Black to Gray and back to White.

Figure 4 Illustration of multi-state push (modulo 3 here)

DEFINITION: The multi-state-based push is called by us "**modulo k**" push, where k is the number of possible states a vertex can go through.

We use different colors to denote the different states of vertices. For example, white is state 0 (off), black is 1, and gray is 2, and so on. Each push increases the affected vertex's number, so changes the color. When each vertex's' color turns into white, we attain the all-off state.

Definition: We say that the pair (m, n) is ***modulo k completely solvable with a certain push*** if every initial configuration state in an m x n grid can be transformed into the all-off state by a sequence of certain pushes over GF (k).

Definition: We say that the pair (m, n) is ***generally solvable with a certain push*** if there exists an integer k, such that every initial configuration states in a m x n grid can be transformed into the all-off state by a sequence of certain pushes over GF (k).

From the definitions, we can see

If a pair (m, n) is modulo k completely solvable with certain push, then it is generally solvable with the same push. Meanwhile, if a pair (m, n) is generally solvable with certain push, there must exist k, such that the pair is modulo k completely solvable.

Note that for mod 2 pushes, it is never necessary to push more than once, because the second push will revert the vertex and its neighbors' states back to the previous values.

Some m x n grids that are not mod 2 completely solvable with certain push may become mod 3 completely solvable with the same push, and vice versa. For example, under plus push ("+"-shaped push), the 4x3 grid is not mod 2 completely solvable, but it is mod 3 completely solvable.

We also want to know

1. For these different push methods, whether they are ***modulo k independent***. **Definition**: A push method is said to be ***modulo k independent*** if for whatever k, the m x n grid is always completely solvable, or never be completely solvable.
2. For one m x n grid with certain push method, can we change k to make it completely solvable when it is not with mod 2 pushes?

We may even want to know for the m x n grids that are not completely solvable, what is the number of the maximum vertices that can be turned off. This is ***Maximization of Switches*** problem, for short, MOS.

Real World Example

A 5x5 example of Push "+" problem may be found in the hand-held electronic computer game "Lights Out". There are also similar games on a 3x3x3 cube. A simple E-version of the game can be found at http://www.csee.wvu.edu/~wfk/coin_flip.html.

Previous Results

For push "+" on a general graph, the problem has been previously studied in 1970s in the context of determining when it is possible to get all the switches off, and in this context is known as *all-parity dominating set problem*. The all-parity dominating set problem has also been studied for classes of graphs such as grids, trees, cycles, and series-parallel graphs. In 1995, Goldwasser and Klostermeyer first described the push problem in graphs, and studied the computational complexity [1]. Approximate algorithms for generarl graphs and grid graphs are given in [1] too. More references can be found as reference items [2,3,4,15] in [1] about the research done in this area.

For problems about grid graphs with the "+" push method, the maximization version of getting as many as possible switches to off state was studied by Goldwasser and Klostermeyer in [1]. A linear algebra approach was undertaken by Goldwasser, Klostermeyer and Trapp to get the general problem solved with $O((mxn)^3)$ performance [3]. H. Ware found a faster algorithm using Fibonacci polynomial which can run at $O(N\log^2 N)$ [2], where N is maximun(m,n).

Some different push methods are related, i.e., one is the complement of another, or combination of others. For example, if the push pattern is to toggle all the vertices at 2-unit distance from the pushed one, it may be composed by 4 PPNC applied on all 4 one-unit close neighbors. Alvy Ray Smith studied the related problem for neighborhoods in cellubar automata in [9].

Complexity

In [1], Goldwasser and Klostermeyer proved the maximization version of the problem for general graphs under the push "+" method to be NP-Complete. However, we still do not know if the maximization of switches in grid graph is also NP-Complete, and if similar proof can be applied to different push methods.

The linear algebra method used to solve the problem runs in $O(N^6)$, where N is max(m,n). We want to know if there are better algorithms for different pushes based on the push itself and the grid's properties as well.

CHAPTER 2 RESEARCH METHODS

The research approaches in this thesis are mainly based on linear algebra and algorithm analysis. Basic knowledge about number theory is applied too, such as the Greatest Common Divisor (GCD) algorithm by Euclid. The arithmetic here is done over the binary field GF(2), if not explicitly specified. When studying modulo k pushes, we do arithmetic over GF(k).

Linear Algebra Method

Construct matrix A

For the m x n grid graph G, the total number of vertices is M = m x n. We number the switches (vertices) from 1 to M from the top-left to bottom right increasing across each row. Denote the M-tuple space over GF(k) by F^M, and define $y = (y_1, \ldots, y_M) \in F^M$ by $y_j = l$, $0<j<n$ if the initial state of vertex j is on and $y_j = 0$ if off. For modulo 2 pushes, y_j can only be 0 and 1.

If we can characterize the connectivity of G by a matrix under certain push, we then can solve any initial setting by solving the linear equation

> Ax = y, where y is the initial configuration, and A means the connectivity characteristics of the graph under certain push.

We define A be A (p, q) = 1 if vertex q will be changed as the result of a push operation on vertex p, otherwise A (p, q) = 0. We can see, for push "+" (over GF(2)), A can be represented by

$$\mathbf{A} = \begin{array}{ccccc} \mathbf{B} & \mathbf{I} & \mathbf{0} & \ldots & \mathbf{0} \\ \mathbf{I} & \mathbf{B} & \mathbf{I} & \ldots & \mathbf{0} \\ \mathbf{0} & \mathbf{I} & \mathbf{B} & \ldots & : \\ : & : & : & \mathbf{B} & \mathbf{I} \\ \mathbf{0} & \mathbf{0} & \ldots & \mathbf{I} & \mathbf{B} \end{array} \qquad \mathbf{B} = \begin{array}{ccccc} 1 & 1 & 0 & \ldots & 0 \\ 1 & 1 & 1 & \ldots & 0 \\ 0 & 1 & 1 & \ldots & : \\ : & : & : & 1 & 1 \\ 0 & 0 & \ldots & 1 & 1 \end{array}$$

where the size of A is (mn x mn), and B is mxm. The sub-matrix above representation is just clearer than a big all numbered matrix. However, it does not imply any structure of the matrix; there even does not always exist such a good form for general push methods, especially for unbalanced pushes.

An example matrix A for the 3x2 grid under push "X" with no center is:

```
Pushing method x no center: only four corners at top-left,
top-right, bottom-left, bottom-right

Matrix A(3,2):
   0   0   0   0   1   0
   0   0   0   1   0   1
   0   0   0   0   1   0
   0   1   0   0   0   0
   1   0   1   0   0   0
   0   1   0   0   0   0

We know the dimension of A(3,2) is actually (3x2)x(3x2),
which is 6x6.
```

For a clearer view of the structure of the matrix A for push PXNC, A can be represented as:

$$\mathbf{A} = \begin{array}{ccccc} \mathbf{0} & \mathbf{B} & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{B} & \mathbf{0} & \mathbf{B} & \dots & \mathbf{0} \\ \mathbf{0} & \mathbf{B} & \mathbf{0} & \dots & : \\ : & : & : & \mathbf{0} & \mathbf{B} \\ \mathbf{0} & \mathbf{0} & \dots & \mathbf{B} & \mathbf{0} \end{array} \qquad \mathbf{B} = \begin{array}{ccccc} 0 & 1 & 0 & \dots & 0 \\ 1 & 0 & 1 & \dots & 0 \\ 0 & 1 & 0 & \dots & : \\ : & : & : & 0 & 1 \\ 0 & 0 & \dots & 1 & 0 \end{array}$$

Once we have the push matrix A, we can use linear algebra to analyze it and find if there are some relationship between the grid size, push method, and the solvability.

If A**x** = **y** does have a solution, say, **x** = $\{x_1, \dots, x_k\}$then $\{j \mid x_j = 1\}$ gives a set of switches(vertices) whose activation will result in the all-off state. We can also study the null space matrix. In [5] and [6], Goldwasser, Klostermeyer, Trapp, and McCague discovered that the null space matrix can be used to get certain periods of recursions for insight into the solution.

Determinant

In order to see if the equation is solvable over GF(k), we can compute the determinant of matrix A. If det (A) = 0, it means A is not invertible, therefore, there is not always a solution for any initial setting **y**. If det (A) mod k = 0, then it is still not always possible to get to the all-off state. By this means, we can determine if an m x n grid graph can be completely solvable or not under certain mod k push.

One problem here is the calculation of the determinant. Since A is mn x mn, the calculation of determinants (by Guassian Elimination) is O(n^3). So the computation is at O(N^6) level, where N = max(m,n); and it requires lots of memory spaces too. Moreover, the result could be very large when m and n are large. For instance, when m=n=13, det (A)=-239121867667810016. Therefore, for larger dimensions, we might have to devise some mechanism for big numbers. Fortunately, since we only care about the result over GF(k), we do not have to obtain the final determinant value, but the value modulo k. Thereby, we can modify the elimination process to adapt the computation into mod k, and then need not worry about big numbers.

It appears that A's determinant can be easily obtained by multiplying all elements at the diagonal line of A after elimination. It is true unless all the diagonal elements are integer. However, when transforming A into the upper-triangle form, we see usually not all the diagonal elements are integer although the determinant may be an integer. Looking at the eigenvalues of the matrix A, we can understand this scenario. For instance, the 3x3 matrix under push "+" push has eigenvector {-1.8284271247461885, -0.41421356237309515, -0.4142135623730946, 0.9999999999999998, 1.0, 1.0000000000000002, 2.414213562373094, 2.414213562373096, 3.8284271247461903}, but the det (A) here is -7, an integer. This makes it harder to do the "modulo k" trick in the process of computing the determinant.

On the other hand, we may still have a way to use the eigenvalues. Since irrational eigenvalues always appears in pairs in this situation, we can multiply them to obtain an integer, and use these integers to discover certain patterns in it. To have the correct pairs of eigenvalues, we have to examine through all the irrational eigenvalues, and that may take O(N^2), or we may modify the process computing eigenvalues to get more detailed information. This can be a future research direction.

One solution here is to magnify the determinant in computation and control the error. By this means, we still can get pretty accurate calculation result after giving up the really big part of the determinant.

The portion of modified determinant is:

```
Get the LU decomposition (or A’s upper triangle form),
for (int j = 0; j < k; j++) {
 {
    final int MODULU = 3;
    final long ERRORCONTROL = 10000;
    d *= A[j][j];
    d = (double) ( (long) (d * ERRORCONTROL) % (MODULU *
ERRORCONTROL));
    d = d / ERRORCONTROL;
 }
```

In above code, A is already in the diagonal form (or upper triangular form). All elements of matrix A and the determinant d are in double for general purposes.

Then we can get the determinant of bigger dimension of m and n, and see if the grid can be completely solvable.

Rank Information

The rank of a matrix is another important way to characterize the matrix. If matrix A is not full rank which means the rank is the same as its dimension (A is square here), then it is not invertible, equivalent to det (A) = 0, then the equation does not have a solution, so the problem cannot be completely solvable. This characteristic of the matrix holds true for modulo k pushes too. That is to say, if matrix A for m x n grid graph under push X is not full rank, then for mod 2 push, mod 3 push… mod k push, the problem is always not completely solvable.

However, on the other hand, even if A is of full rank, we cannot assert that the problem is modulo k completely solvable for any k. The reason is the determinant of A can still be zero modulo k. Nevertheless, this tell us, there must be an integer k, such that the problem is completely solvable over GF(k), because however big the determinant of A is, there exist an k, such that det (A) mod k $\neq$ 0.

Therefore, the rank can tell us different information about the property of the grid graph under a certain push method, regardless of modulo k.

Another thing the rank can tell is the dimension of core space (also called kernel) and range space of A. This tells how many possible equivalence classes of dependent linear vectors of A exist. In the sense of our switch off problem, this tells us at most how many classes of switches cannot be turned off. The determinant method cannot tell us this, but the rank information can.

Computational Issues

There are other characteristics about the matrix, however, as of today, they are not found to be useful in solving the problem. For example, traces, eigenvalues, SVD (Singular Value Decomposition), LU or QR decompositions, and so on, don't reveal any obvious relationship with the solvability result. Some results from trace or eigenvalues will be mentioned, although they don't lead to a quick solution.

Many numerical packages are used to do the computation, among LAPack, MatLab extensions, and other C/C++ based linear algebra packages, we finally chose a simple but yet powerful, straightforward package in Java to attack the problem.

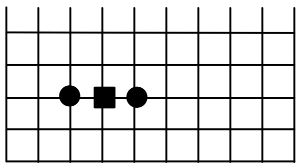
All the major results are calculated in Java language, based on a Java Matrix Package developed by Hicklin, Moler, et al at NIST [10]. The package provides user-level classes for constructing and manipulating real, dense matrices. It can provide sufficient functionality for routine problems, and it has the source code for customization. The package provides the following matrix operations: addition, multiplication, inversion, rank, determinant, eigenvalues, SVD, Cholesky Decomposition, LU/QR decompostion. It doesn't deal with complex number, and not support sparse matrix representation. It cannot handle too big matrices, nor is it a fast algorithm pack, we chose it because it is basically sufficient for our use of calculating the determinant and rank of mid-sized matrices and is customizable.

Algorithmic Approach

The use of linear algebra techniques can give us some results about the push's solvability and thus may hint to us about certain rules on the push. Analyzing the special push property of a certain push, we may be able to see the law under the push, and then we can devise an algorithm to solve the problem.

As stated in the first chapter, we have already known that it is not necessary to activate a switch twice (over GF (2)). We may obtain certain rules by analyzing the push.

For example, with a push like the right picture, pushing v will change v itself and its left and right neighbors only. We can call it "3-dot push". We see this push can go row by row and turn "on" vertices off. If it can make it for one line/row off, it can make all the rows in grid off. So, if we can know for what n, 1 x n can be turned off, then all m x n grids are completely solvable.



This simple push on vertex v changes the states of v and its left and right neighbors.

Figure 5 An example of Three-Dot Push Pattern

Regarding the above example, where n = 1 (1x1), it is completely solvable. When n=2, if only one is on, then there is no way to get that one off because each time we push on one vertex, the other one will change state too, so there'll be always one vertex on. Then the 1x2 is not completely solvable. Accordingly, we can show for 1x3 and 1x4 it is completely solvable, but not for 1x5. It is not hard to see the rule here: for all n, n mod 3 = 2, the problem on grid m x n is not completely solvable; otherwise, it is completely solvable. This is due to the restriction of the push itself. Therefore, if we know the properties of the push, we can easily get the result.

We are also interested in the symmetry of the pushes, the behavior of the pattern when the push hits the edge or corner, and so on.

CHAPTER 3 RESULTS

In this chapter, we present the results along the approaches discussed in the last chapter. For each push method, we at first compute the matrix A, then obtain the ranks, determinants, sometimes with a specific grid as an example. Then we provide computational results about solvability information of grids with sizes up to 25 or 50. Proofs are given for some pushing methods; some rules obtained from the results still remain conjecture. Finally, we discuss the operational feature of the push, and provide a much faster algorithm based on it.

Push "+" with No Center

Although this push is only one vertex different from the push "+" method, the solvability is very different.

Matrix A

The matrix is of the following form:

$$A = \begin{matrix} \mathbf{B} & \mathbf{I} & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{I} & \mathbf{B} & \mathbf{I} & \dots & \mathbf{0} \\ \mathbf{0} & \mathbf{I} & \mathbf{B} & \dots & : \\ : & : & : & \mathbf{B} & \mathbf{I} \\ \mathbf{0} & \mathbf{0} & \dots & \mathbf{I} & \mathbf{B} \end{matrix} \qquad \mathbf{B} = \begin{matrix} 0 & 1 & 0 & \dots & 0 \\ 1 & 0 & 1 & \dots & 0 \\ 0 & 1 & 0 & \dots & : \\ : & : & : & 0 & 1 \\ 0 & 0 & \dots & 1 & 0 \end{matrix}$$

where A is mn x mn, and B is n x n.

An example of A for 2x2 grid is:

$$A(2,2) = \begin{matrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{matrix}$$

Determinants and Patterns

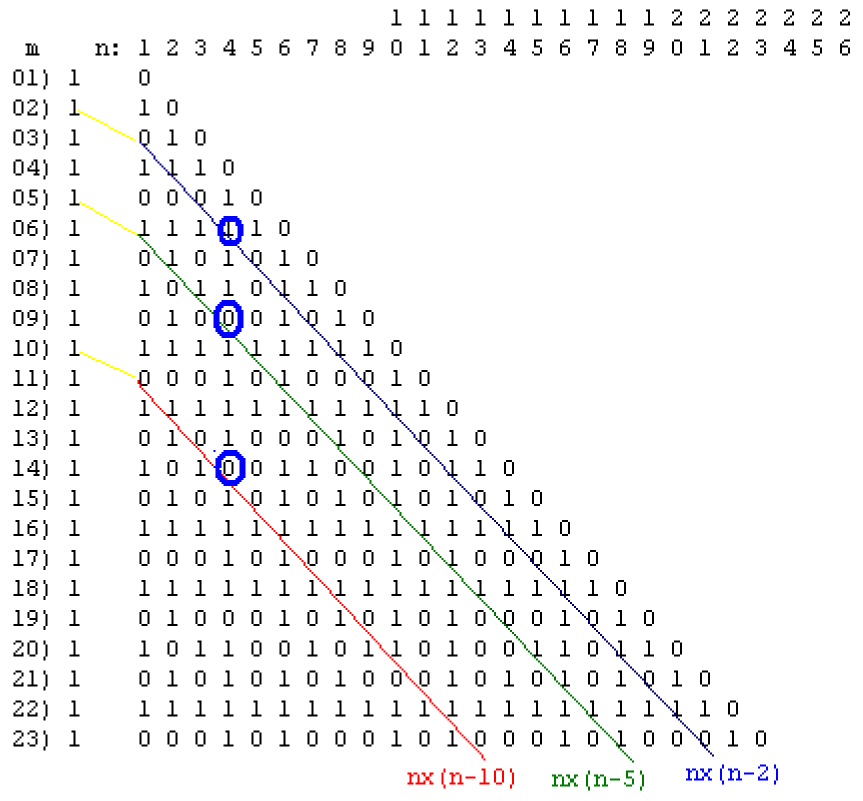
As the computational cost for larger grids is too big, the dimension of grids studied here are restricted to 25. We know for a 20x20 grid, the dimension of A is

400x400, for a 50x50 grid, A is 2500x2500; it'll take too much time to compute all the results. The determinants of the matrices of grid 1x1 to 25x25 and then from 26x25 to 50x1 are listed below (the 1 after right parenthesis is not part of the result, it is used later):

```
This is the list of determinants of grids (m, n) where m is
from 1 to 50, but n is from 1 to 25 if m < 26, then from 25
to 1 when m > 25.
The first column is the index of m, the “all 1” column
following the index is added for clearer view of the pattern.
You can think the column is the column 0.
                           1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2
 m    n: 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6
01) 1    0
02) 1    1 0
03) 1    0 1 0
04) 1    1 1 1 0
05) 1    0 0 0 1 0
06) 1    1 1 1 1 1 0
07) 1    0 1 0 1 0 1 0
08) 1    1 0 1 1 0 1 1 0
09) 1    0 1 0 0 0 1 0 1 0
10) 1    1 1 1 1 1 1 1 1 1 0
11) 1    0 0 0 1 0 1 0 0 0 1 0
12) 1    1 1 1 1 1 1 1 1 1 1 1 0
13) 1    0 1 0 1 0 0 0 1 0 1 0 1 0
14) 1    1 0 1 0 0 1 1 0 0 1 0 1 1 0
15) 1    0 1 0 1 0 1 0 1 0 1 0 1 0 1 0
16) 1    1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0
17) 1    0 0 0 1 0 1 0 0 0 1 0 1 0 0 0 1 0
18) 1    1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0
19) 1    0 1 0 0 0 1 0 1 0 1 0 1 0 0 0 1 0 1 0
20) 1    1 0 1 1 0 0 1 0 1 1 0 1 0 0 1 1 0 1 1 0
21) 1    0 1 0 1 0 1 0 1 0 0 0 1 0 1 0 1 0 1 0 1 0
22) 1    1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0
23) 1    0 0 0 1 0 1 0 0 0 1 0 1 0 0 0 1 0 1 0 0 0 1 0
24) 1    1 1 1 0 1 1 1 1 0 1 1 1 1 0 1 1 1 1 0 1 1 1 1 0
25) 1    0 1 0 1 0 1 0 1 0 1 0 0 0 1 0 1 0 1 0 1 0 1 0 1 0
26) 1    1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1
27) 1    0 1 0 1 0 0 0 1 0 1 0 1 0 1 0 1 0 1 0 0 0 1 0 1
28) 1    1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
29) 1    0 0 0 0 0 1 0 0 0 1 0 1 0 0 0 1 0 1 0 0 0 1
30) 1    1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
31) 1    0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1
32) 1    1 0 1 1 0 1 1 0 1 0 0 1 1 0 1 1 0 1 1
33) 1    0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 0 0 1
34) 1    1 1 1 0 1 0 1 1 0 1 1 1 0 0 1 1 1
35) 1    0 0 0 1 0 1 0 0 0 1 0 1 0 0 0 1
36) 1    1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
37) 1    0 1 0 1 0 1 0 1 0 1 0 1 0 1
38) 1    1 0 1 1 0 1 1 0 1 1 0 0 1
```

```
39) 1    0 1 0 0 0 1 0 1 0 1 0 1
40) 1    1 1 1 1 1 1 1 1 1 1 1
41) 1    0 0 0 1 0 0 0 0 0 1
42) 1    1 1 1 1 1 1 1 1 1
43) 1    0 1 0 1 0 1 0 1
44) 1    1 0 1 0 0 1 1
45) 1    0 1 0 1 0 1
46) 1    1 1 1 1 1
47) 1    0 0 0 1
48) 1    1 1 1
49) 1    0 1
50) 1    1
```

From the listed results, we can see several patterns:



```
                                   1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2
 m     n: 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6
01) 1     0
02) 1     1 0
03) 1     0 1 0
04) 1     1 1 1 0
05) 1     0 0 0 1 0
06) 1     1 1 1 1 1 0
07) 1     0 1 0 1 0 1 0
08) 1     1 0 1 1 0 1 1 0
09) 1     0 1 0 0 0 1 0 1 0
10) 1     1 1 1 1 1 1 1 1 1 0
11) 1     0 0 0 1 0 1 0 0 0 1 0
12) 1     1 1 1 1 1 1 1 1 1 1 1 0
13) 1     0 1 0 1 0 0 0 1 0 1 0 1 0
14) 1     1 0 1 0 0 1 1 0 0 1 0 1 1 0
15) 1     0 1 0 1 0 1 0 1 0 1 0 1 0 1 0
16) 1     1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0
17) 1     0 0 0 1 0 1 0 0 0 1 0 1 0 0 0 1 0
18) 1     1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0
19) 1     0 1 0 0 0 1 0 1 0 1 0 1 0 0 0 1 0 1 0
20) 1     1 0 1 1 0 0 1 0 1 1 0 1 0 0 1 1 0 1 1 0
21) 1     0 1 0 1 0 1 0 1 0 0 0 1 0 1 0 1 0 1 0 1 0
22) 1     1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0
23) 1     0 0 0 1 0 1 0 0 0 1 0 1 0 0 0 1 0 1 0 0 0 1 0
```

Figure 6 Examine the results under PPNC

In the following we examine diagonal lines constituted by the results from grids nx(n-k). Let k be 0, 1, 2, and let n change, we can obtain a diagonal line, as seen in the above figure.

Look at the 4th column, we can see the value at diagonal line nx(n-10) is the result of the logic "AND" operation of values at diagonal line nx(n-5) and nx(n-2) (circled values). In fact, we can obtain the following:

- All n x n (diagonal) lines are 0s
- All n x (n-1) lines are 1s
- All n x (n-2) lines are of pattern (01)…
- All n x (n-3) lines are of pattern (110)*, if adding a 1 before it.
- All n x (n-4) lines are of pattern (10)* too, if adding a 1 before it, the same as n x (n-2)
- All n x (n-5) lines are of pattern (11110)*, if adding a 1 before it.
- All n x (n-6) lines are of pattern (100010)*, logic "and" of n x (n-2) and n x (n-3) pattern, e.g., 101010 & 110110 = 100010.
- All n x (n-7) lines are of pattern (1111110)*…
- All n x (n-8) lines are of pattern (10)*…
- All n x (n-9) lines are of pattern (110)*…
- All n x (n-10) lines are of pattern (1010001010)*…

We can see each diagonal line is determined by the difference between m and n. Say, let d = m – n, then the pattern is result of the logic "add" operation on the patterns related to the prime factors of d.

The diagonal pattern of grid n x (n-d) is the logic "AND" operation on the patterns of n x (n-p), where p is the prime factor of d.

If looking at the result column-by-column and row-by-row, we can see similar patterns exist too. If we want to get the pattern for m x n, we only need to look at (m+1)'s prime factors. By applying the logic "AND" operation to patterns of (m+1)'s prime factors, we can obtain the result pattern of (m+1). If (m+1) itself is prime, then the pattern is (1…10)*, and there are exactly m 1s and only the last is zero in one period of the pattern.

SUMMARY: **The pattern of m x n is**

1. **(1..10)*, where there are exactly m 1s in one period, if m+1 is a prime number;**
2. **the logic AND operation on patterns of the prime factors of m+1, if m+1 is a composite number.**

This result can be proved later with operational analysis, and a simpler rule will be deducted too.

One interesting property of the matrices under this push method is **the determinants are either 1 or 0**. In fact, it could be –1, but it does not matter because we only need the result modulo k. This nice property makes this push modulo k independent. It means, whatever k is, if an m x n grid is completely solvable over GF(2), it is completely solvable over GF(k).

Ranks

In order to discover more about this push method, we studied the ranks of the matrices from 1x1 grid to 25x25 then from 26x25 to 50x1. The partial results from A(1,1) to A(6,6) are:

```
Matrix A(1,1): ----------------------- rank = 0
Matrix A(2,1):  det(A)=-1 modulo 3 = -1
Matrix A(2,2): ----------------------- rank = 2
Matrix A(3,1): ----------------------- rank = 2
Matrix A(3,2):  det(A)=-1 module 3 = -1
Matrix A(3,3): ----------------------- rank = 6
Matrix A(4,1):  det(A)=1 modulo 3 = 1
Matrix A(4,2):  det(A)=1 modulo 3 = 1
Matrix A(4,3):  det(A)=1 modulo 3 = 1
Matrix A(4,4): ----------------------- rank = 12
Matrix A(5,1): ----------------------- rank = 4
Matrix A(5,2): ----------------------- rank = 8
Matrix A(5,3): ----------------------- rank = 14
Matrix A(5,4):  det(A)=1 modulo 3 = 1
Matrix A(5,5): ----------------------- rank = 20
Matrix A(6,1):  det(A)=-1 modulo 3 = -1
Matrix A(6,2):  det(A)=1 modulo 3 = 1
Matrix A(6,3):  det(A)=-1 modulo 3 = -1
Matrix A(6,4):  det(A)=1 modulo 3 = 1
Matrix A(6,5):  det(A)=-1 modulo 3 = -1
Matrix A(6,6): ----------------------- rank = 30
.
.
.
```

The partial results from A(25,1) to A(25,25), which will contribute to line 25 in the triangular list, are below:

```
Matrix A(25,1): ----------------------- rank = 24
Matrix A(25,2):  det(A)=-1 modulo 3 = -1
Matrix A(25,3): ----------------------- rank = 74
Matrix A(25,4):  det(A)=1 modulo 3 = 1
Matrix A(25,5): ----------------------- rank = 124
Matrix A(25,6):  det(A)=-1 modulo 3 = -1
Matrix A(25,7): ----------------------- rank = 174
Matrix A(25,8):  det(A)=1 modulo 3 = 1
```

```
  Matrix A(25,9): ----------------------- rank = 224
  Matrix A(25,10):  det(A)=-1 modulo 3 = -1
  Matrix A(25,11): ----------------------- rank = 274
  Matrix A(25,12): ----------------------- rank = 288
  Matrix A(25,13): ----------------------- rank = 324
  Matrix A(25,14):  det(A)=-1 modulo 3 = -1
  Matrix A(25,15): ----------------------- rank = 374
  Matrix A(25,16):  det(A)=1 modulo 3 = 1
  Matrix A(25,17): ----------------------- rank = 424
  Matrix A(25,18):  det(A)=-1 modulo 3 = -1
  Matrix A(25,19): ----------------------- rank = 474
  Matrix A(25,20):  det(A)=1 modulo 3 = 1
  Matrix A(25,21): ----------------------- rank = 524
  Matrix A(25,22):  det(A)=-1 modulo 3 = -1
  Matrix A(25,23): ----------------------- rank = 574
  Matrix A(25,24):  det(A)=1 modulo 3 = 1
  Matrix A(25,25): ----------------------- rank = 600
  .
  .
```

The final part of results from A(45,6) down to A(50,1) is below:

```
  .
  .
  Matrix A(45,1): ----------------------- rank = 44
  Matrix A(45,2):  det(A)=-1 modulo 3 = -1
  Matrix A(45,3): ----------------------- rank = 134
  Matrix A(45,4):  det(A)=1 modulo 3 = 1
  Matrix A(45,5): ----------------------- rank = 224
  Matrix A(45,6):  det(A)=-1 modulo 3 = -1
  Matrix A(46,1):  det(A)=-1 modulo 3 = -1
  Matrix A(46,2):  det(A)=1 modulo 3 = 1
  Matrix A(46,3):  det(A)=-1 modulo 3 = -1
  Matrix A(46,4):  det(A)=1 modulo 3 = 1
  Matrix A(46,5):  det(A)=-1 modulo 3 = -1
  Matrix A(47,1): ----------------------- rank = 46
  Matrix A(47,2): ----------------------- rank = 92
  Matrix A(47,3): ----------------------- rank = 138
  Matrix A(47,4):  det(A)=1 modulo 3 = 1
  Matrix A(48,1):  det(A)=1 modulo 3 = 1
  Matrix A(48,2):  det(A)=1 modulo 3 = 1
  Matrix A(48,3):  det(A)=1 modulo 3 = 1
  Matrix A(49,1): ----------------------- rank = 48
  Matrix A(49,2):  det(A)=-1 modulo 3 = -1
  Matrix A(50,1):  det(A)=-1 modulo 3 = -1
```

Complete results will be found online at http://columbia.me.washington.edu/jw/thesis/index.html or http://www.csee.wvu.edu/~wfk/index.html for links.

We observe that the dimension of the range space is related to the greatest common divisor of m+1 and n+1 of grid mxn. For example, the rank of matrix of grid 5x3 is 14, while the full rank should be 15; the difference is 1, which is equal to GCD (5+1,3+1)-1 = 1. As of 5x2, rank (A(5,2)) = 8, full rank should be 10, GCD (5+1, 2+1) = 3, 3-1 = 2. This works for 25x12, where GCD (25+1, 12+1) = 13, and range space dimension is 12, this coincides with rank(A(25,12))=25*12-(13-1)=288, while 25x12=300.

The possible biggest range space for m occurs only at m x m grids, where GCD (m+1, m+1) = m+1, and the dimension of the range space is m. In terms of pushing switches, this means there are m partitions of the not-off-able configurations. However, from this we cannot tell the maximum number of switches that are left on in m x m grids. One may suspect m/2, m/4 or m/logm is the upper bound. I personally think m/4 is more likely to be the upper bound. That is the number of vertices left on will be at least m/4.

THEOREM: With PPNC, nxn grids are not generally solvable, for any n.

That the matrices of n x n grids are not full rank is determined by the construction of A. In the following example of A(3,3), we can see where the 1s are, and devise a method to get rid of the 1s by Gaussian elimination.

```
Matrix A(3,3):
   0   1   0   1   0   0   0   0   0    -- the 1st row
   1   0   1   0   1   0   0   0   0
   0   1   0   0   0   1   0   0   0
   1   0   0   0   1   0   1   0   0
   0   1   0   1   0   1   0   1   0    -- the 5th row
   0   0   1   0   1   0   0   0   1
   0   0   0   1   0   0   0   1   0
   0   0   0   0   1   0   1   0   1
   0   0   0   0   0   1   0   1   0   -- the last row
```

We can see that using the 1st row, we can make the first part of row 5 zero; and using the last row, we can make the last part of row 5 zero. The tour de force steps to get an all-zero row can be the following:

1. Subtract 1st row from row n+2
2. Subtract (updated) row n+2 from row 2n + 3, which is 2(n+1)+1
3. Substract row 2n+3 from row 3(n+1)+1
4. Repeat it until we hit k(n+1)+1, where k=n-1. This is the last row as (n-1)(n+1)+1 = nxn.

Since A is symmetric, the above steps can be done from bottom-up too.

If we can prove A(n,n) is invertible, then grid nxn is generally solvable under PPNC.

Proof: As of the characteristic of the push, we can see A has the following property:

- ✓ ***The first n+1 elements of the first row (starting from 1), are the same as the first n+1 elements of row n+2.*** This is because of the construction of A. There should be 2 and only 2 1s in the first n+1 elements of row 1 (except when n = 1, the extreme case). One is the vertex itself and another one is the right neighbor of the vertex.
- ✓ ***The elements (k-1)n+k to kn+k of row (k-1)n+k, are the same as the elements (k-1)n+k to kn+k of row kn+k+1, 1<k<n.***

The property is true because if a push on vertex v can change the states of the right and bottom neighbors of v, a push on vertex v's right-bottom corner vertex can also change the states of the same two vertices.

Based on these properties, by multiplying by -1 and adding row (k-1)n+k to row kn+k+1 repetitively, we can make the last row (row nxn) all zero. This means det (A) = 0.

Therefore, we know why n x n grids are always not generally solvable with push "+" no center (PPNC). ÿ

CORRALLARY: **With PPNC, nxn grids are not completely solvable for any n**.

This can be seen directly from the definitions of the two terms.

Operational Analysis

This push method has a good feature: the push changes all the vertices that are 1 unit distance from the pushed vertex. It can move two vertices at corners of a small grid without changing other vertices. We use this operational feature to examine what dimension of the grid is completely solvable.

In fact, we've known the nxn grids are not generally solvable. The reason is if a corner vertex is on, there is no way to push the grid to the all-off state. We can only move the "on" vertex to the opposite corner.

For grids m x m-1

If m = n+1, then all mxn grids can be turned off from any initial configuration.

Proof: *The idea is to show any ON point in the grid can be turned OFF by the push. There are three kinds of ON points:*

1) *Corner points, only 4.*
2) *Edge points, 2 * (m + n)-4*
3) *Inner points, (m-1) * (n-1)*

Case 1: **All ON corner vertices can be turned OFF by a sequence of pushes.**

Op1: Push the vertex beside the corner one (which is ON) along the m edge (the longer side).

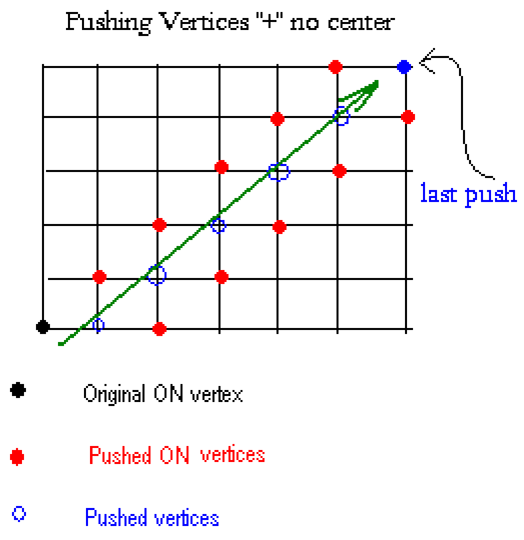


Figure 7 Turn an ON corner vertex into OFF

Result 1: The corner vertex is OFF, and two vertices beside the pushed one are ON now.

Op2: Push the vertex adjacent to both the ON ones at the other side.

Result 2: The ON vertices move diagonally to the opposite corner.

Op3: When the two ON ones reach the edges, (they should reach the edge at the same time because of the dimension of the grid), push the corner.

Result 3: All are OFF. ÿ

Case 2: **All ON edge-vertices can be turned OFF by a sequence of pushes.**

Op1: Using the same push method above on the originally ON vertex, push it until there is only one ON vertices on the side edge[1]. It is always doable, and the resulting ON vertex should be at the same position. (The red one at the right edge is the only ON one after this step.)

Result 1: If the original one is at kth place on the edge, now it should be at the same distance from the corner between the original and the side edge.

[1] If both reach different edges, it will be a corner case just by one more push at the corner.

Op2: Do the same thing for this ON vertex, and let only one vertex on the opposite edge (to the edge contains the original vertex).

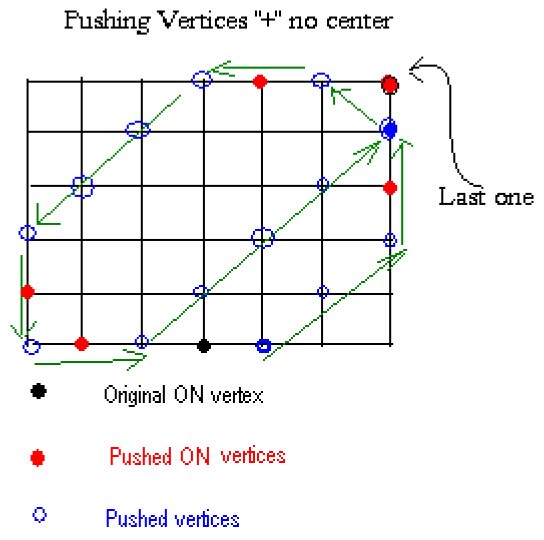


Figure 8 Moving around the vertices on the edges

Result 2: Because of the dimension of the grid, the position of the ON vertex on the top edge is 1 unit near the corner (the top-right corner).

Op3: Repeat op1, op2, until the ON vertices are moved to the corner, either the top-right one or the left-bottom one.

Result 3: The only ON one now is at the corner.

Op4: Apply algorithm in case 1 and we can get the ON corner vertex pushed off.

The illustration shows an ON edge vertex could be pushed many times to become a corner one. Each time when it reaches the opposite edge, the distance to the corner is reduced by 1. This is due to the inherent property of the grid, m=n+1. So finally, the ON edge vertex will be turned into an ON corner vertex. ÿ

Case 3: **All ON inner vertices can be turned OFF by a sequence of pushes.**

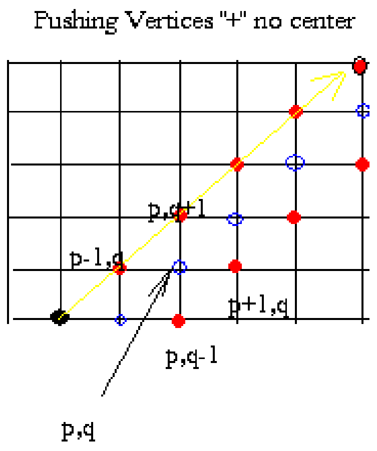


Figure 9 Move an ON edge vertex into the corner

This is a relatively simple, because we already have a simple algorithm to get all inner vertices in the grid pushed to the last edge (whatever one). The algorithm is used for approximation of MOS, Goldwasser and Klostermeyer have more details in [1].

For cases m>n+1

For other mxn grids (m>n+1), the result can be determined by:

- Let d = m-n.
- Compute the prime factors of d, noted as d'.

For each d', if m mod d' = (d'-1), then result = 0, otherwise result = 1.

Along the lines of the above analysis, we expand this to general cases. Consider in an m x n grid, (m>n), we use PPNC to transfer vertices from each edge to another, especially from the bottom edge to the opposite edge – the top edge. There is a very important characteristic of the push:

For a pair of two vertices (p-1, q) and (p, q-1), the push at (p, q) will shift the pair to (p, q+1), (p+1, q), if the pushed vertex is not on an edge.

This is easy to see from the property of the push, shown in the picture. For pushes near the edge, it is clear that one vertex will disappear because of the edge effect, and another one remains. Both vertices disappear if and only if the push occurs at the corner.

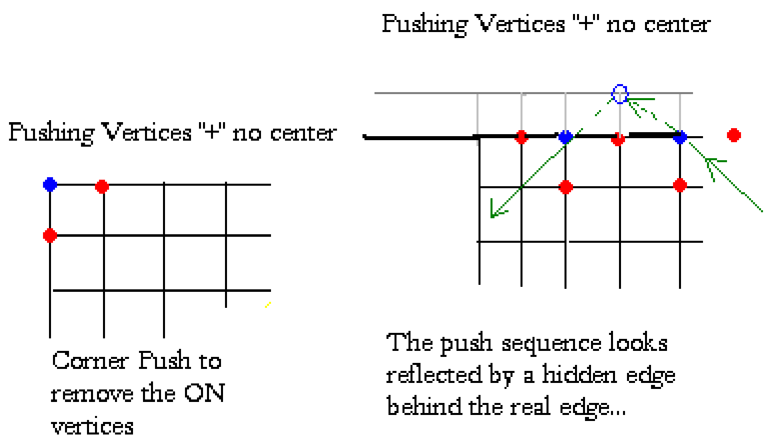

Pushing Vertices "+" no center
Pushing Vertices "+" no center
Corner Push to remove the ON vertices
The push sequence looks reflected by a hidden edge behind the real edge...

Figure 10 Behavior of pushes on edge and corner (PPNC)

Another property of the push on the edge is useful. The trace of pushes in the grid goes diagonal to move one vertex from edge to edge. On the edge, the push can go on[2] by pushing at the neighbor of the only ON vertex on the edge. This, shown above, is as if the push has been reflected by an invisible mirror edge behind.

This idea gives us a clue about simplifying the pushes inside the grid. In fact, **the push sequence in any m x n grid can be viewed as a straight line of pushes (diagonally) being reflected by an (m+1) x (n+1) box.** As illustrated below:

[2] Here I mean by "go on" that the push goes in a different direction – turned by 90 degrees. We can regard the push direction as counterclockwise, and each push on the edge will make the vertices to top-right respective to the edge.

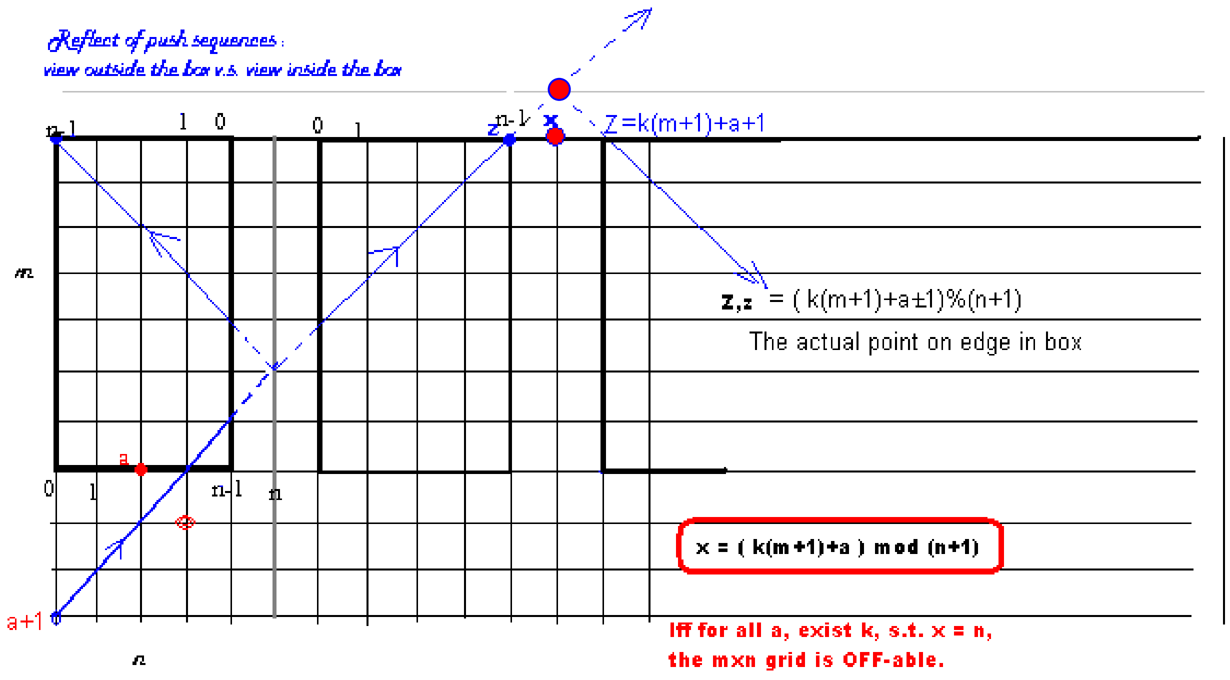

Reflect of push sequences:
view outside the box v.s. view inside the box
n-1
1
0
0
1
z
n-1
x
Z=k(m+1)+a+1
m
Z,z = (k(m+1)+a±1)%(n+1)
The actual point on edge in box
a
0
1
n-1
n
x = (k(m+1)+a) mod (n+1)
a+1
0
n
Iff for all a, exist k, s.t. x = n,
the mxn grid is OFF-able.

Figure 11 How an ON edge vertex can be pushed OFF

We can see on the top base line, there are two points – z and Z, which indicate 0 and n-1 (mod n+1) for the grid. The coordinate of Z is (m-1,0) while the coordinate of z is (m-1, n-1). If any push can go there, it indicates that vertex on the opposite edge can be turned off. A more accurate criterion is to use the imaginary point above x. so that if the push ever goes there, the original vertex, which incurred the sequence of pushes, can be turned off.

The magic formula then is:

$$\mathbf{X = (k*(m+1)+a) \bmod (n+1)}$$

According to this formula, we claim:

The m x n grid is completely solvable with PPNC if and only if for any vertex a on the edge, 0<a<n+1, there exists an integer k, such that X = (k*(m+1)+a) mod (n+1) = n.

We can see, k only needs to go from 1 to n+1 to cover all possibilities of the mod result. Then from the result, there is directly an algorithm to compute the result of the mxn grid under push "+" with no center (PPNC).

ALGORITHM PPNC 1:

```
Input m, n,
Construct result array Result [n], initialize it to be 0
For integer I = 0 to n-1
        For integer k=1 to n
                Let x = (k*(m+1)+I) mod (n+1)
                If x == n then
                Let Result [I] = 1;
Go over the result array, check if each Result [I] == 1; if all 1, then the grid is
completely solvable by PPNC.
```

The above algorithm clearly runs in O (n^2) time. It is much faster than the matrix rank/determinant computation algorithm, which runs in O $((mxn)^3)$, or $O(N^6)$, where N is max(m,n). This algorithm should have a small constant too. In fact, it takes virtually no time[3] to output the whole result that the rank method takes days to compute.

An even faster one

From the above result, let's examine more about the equation:

$$\mathbf{X = (k*(m+1)+a) \bmod (n+1)}$$

It actually signals a divisibility property– if (m+1) and (n+1) are co-prime or not. It is equivalent to

Grid mxn can be turned all off if and only if GCD (m+1, n+1) = 1.

Applying the well-known Euclidean GCD algorithm, we see the new way to determine if grid mxn is completely solvable only takes log N time.

Push "+"

H. Ware studied Push "+"in [2], and a fast algorithm has already been obtained using Fibonacci Polynomials on GF(2). Here we present the mod 3 result and the rank result. From the rank result, we proposed a conjecture that certain rules exist for the solvability of this push too, and we can have an even faster algorithm.

[3] My experience is as soon as I pressed "Enter" key, the results were instantly printed out.

Ranks

The matrix A for this push can be found in the previous chapter, and also in [1.Goldwasser & Klostermeyer]. The result of full rank of grids from 1x1 to 25x25 then to 50x1 is listed below.

```
  01)1   1
  02)1   0 1
  03)1   1 0 1
  04)1   1 1 1 0
  05)1   0 0 0 1 0
  06)1   1 1 1 1 1 1
  07)1   1 0 1 1 0 1 1
  08)1   0 1 0 1 0 1 0 1
  09)1   1 0 1 0 0 1 1 0 0
  10)1   1 1 1 1 1 1 1 1 1 1
  11)1   0 0 0 1 0 1 0 0 0 1 0
  12)1   1 1 1 1 1 1 1 1 1 1 1 1
  13)1   1 0 1 1 0 1 1 0 1 1 0 1 1
  14)1   0 1 0 0 0 1 0 1 0 1 0 1 0 0
  15)1   1 0 1 1 0 1 1 0 1 1 0 1 1 0 1
  16)1   1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
  17)1   0 0 0 1 0 1 0 0 0 1 0 1 0 0 0 1 0
  18)1   1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
  19)1   1 0 1 0 0 1 1 0 0 1 0 1 1 0 1 1 0 1 0
  20)1   0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1
  21)1   1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1
  22)1   1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
  23)1   0 0 0 1 0 1 0 0 0 1 0 1 0 0 0 1 0 1 0 0 0 1 0
  24)1   1 1 1 0 1 1 1 1 0 1 1 1 1 0 1 1 1 1 0 1 1 1 1 0
  25)1   1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1
  26)1   0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0
  27)1   1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1
  28)1   1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
  29)1   0 0 0 0 0 1 0 0 0 1 0 1 0 0 0 1 0 1 0 0 0 1
  30)1   1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
  31)1   1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0
  32)1   0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0
  33)1   1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1
  34)1   1 1 1 0 1 1 1 1 0 1 1 1 1 0 1 1 1
  35)1   0 0 0 1 0 1 0 0 0 1 0 1 0 0 0 1
  36)1   1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
  37)1   1 0 1 1 0 1 1 0 1 1 0 1 1 0
  38)1   0 1 0 1 0 1 0 1 0 1 0 1 0
  39)1   1 0 1 0 0 1 1 0 0 1 0 1
  40)1   1 1 1 1 1 1 1 1 1 1 1
  41)1   0 0 0 1 0 1 0 0 0 1
  42)1   1 1 1 1 1 1 1 1 1
  43)1   1 0 1 1 0 1 1 0
  44)1   0 1 0 0 0 1 0
  45)1   1 0 1 1 0 1
```

```
46)1   1 1 1 1 1
47)1   0 0 0 1
48)1   1 1 1
49)1   1 0
50)1   0
```

The result of mod 3 determinants is in the followed section. By comparing the results, we can see, the not-full-rank grids can never be turned to all-off state, but the full rank cannot guarantee the grid is completely solvable under mod 3 push. The full rank result here is obtained by calculating the rank directly. It can also be obtained by setting the mod to be large enough, say like 100000 so that the probability that it is dividable by the determinant is small enough to be ignored.

Here is the partial result of full rank with Push Plus:

```
Matrix A(4,4): ----------------------- rank = 14
Matrix A(5,1): ----------------------- rank = 4
Matrix A(5,2): ----------------------- rank = 9
Matrix A(5,3): ----------------------- rank = 14
Matrix A(5,4):  det(A)=55 modulo 10001 = 55
Matrix A(5,5): ----------------------- rank = 23
Matrix A(6,1):  det(A)=1 modulo 10001 = 1
Matrix A(6,2):  det(A)=-7 modulo 10001 = -7
Matrix A(6,3):  det(A)=-7 modulo 10001 = -7
Matrix A(6,4):  det(A)=29 modulo 10001 = 29
Matrix A(6,5):  det(A)=-1183 modulo 10001 = -1183
Matrix A(6,6):  det(A)=2197 modulo 10001 = 2197
Matrix A(7,1):  det(A)=1 modulo 10001 = 1
Matrix A(7,2): ----------------------- rank = 13
Matrix A(7,3):  det(A)=119 modulo 10001 = 119
Matrix A(7,4):  det(A)=279 modulo 10001 = 279
Matrix A(7,5): ----------------------- rank = 34
Matrix A(7,6):  det(A)=-791 modulo 10001 = -791
Matrix A(7,7):  det(A)=-34391 modulo 10001 = -4388
.
.
.
Matrix A(17,1): ----------------------- rank = 16
Matrix A(17,2): ----------------------- rank = 33
Matrix A(17,3): ----------------------- rank = 50
Matrix A(17,4):  det(A)=187055 modulo 10001 = 7037
Matrix A(17,5): ----------------------- rank = 83
Matrix A(17,6):  det(A)=-22018260992 modulo 10001 = -9387
Matrix A(17,7): ----------------------- rank = 118
Matrix A(17,8): ----------------------- rank = 135
Matrix A(17,9): ----------------------- rank = 152
Matrix A(17,10):  det(A)=-6974095590913 modulo 10001 = -1088
Matrix A(17,11): ----------------------- rank = 185
Matrix A(17,12):  det(A)=-9223372036854775808 modulo 10001 =
-8250
```

```
Matrix A(17,13): ---------------------- rank = 220
Matrix A(17,14): ---------------------- rank = 237
Matrix A(17,15): ---------------------- rank = 254
Matrix A(17,16):  det(A)=9223372036854775807 modulo 10001 =
8249
Matrix A(17,17): ---------------------- rank = 287
```

In this list, we can see, for instance, as far as the 17x17 grids, the push cannot turn all initial settings into off state in modulo k case, for any k. The 17x6 grids are not completely solvable under the mod 2 push, but completely solvable under the mod 3 push. The 7x4 grids are not completely solvable under the mod 3 push (because 279 mod 3 = 0).

Looking at the triangular result, we can find certain "patterns" in it. For example, if (n+1) is prime, except for 2, 3 and 5, the whole rows are all 1s, say, the 10th, 12th, 16th, 28th, etc. This means they are generally solvable, and can be completely solvable under certain modulo k push.

However, for all m x n grids, if $m+1 = 2^k$, then the solvability pattern with n changing is { 10110110110110… }. By adding an imaginary 1 before the first column, we can see the pattern is (110)*. If $m+1 = 3^k$, the solvability pattern is (10)*. It can be verified by looking at the 2nd, 8th, and 26th row of the rank result, for example. If $m+1 = 5^k$, we see the pattern is (11110)*.

By induction, we can guess, for grid m x n, the rank result is governed by the following rules:

- **If the prime factors of m+1 don't contain 2, 3 or 5, then the rank is 1 regardless of n.**
- **If the prime factors of m+1 contain 2, 3 or 5, then the solvability pattern (starting from 0) is the logical "and" of 2's pattern, (110)*, 3's pattern, (10)*, and 5's pattern (11110)*.**

If the pattern at m, n is zero, it means this grid is not completely solvable over any GF(k). Otherwise, whether the grid is completely solvable still depends on the value of the determinant of A.

If the conjecture is true, then this simple way to determine solvability over GF(k) generally is much faster, it runs in O(lg N) time, where N=max(m,n). In fact, we only need to know if m contains prime factors 2, 3 or 5. We don't have to know all prime factors of m, to know if m contains factor 2 and 5 requires constant time, however, to know if m mod 3 = 0, it requires O(lg N) time, because we need to add each digit of m to see if m is divisible by 3. The calculation of 1-0 pattern at nth place in row m is the same too.

CONJECTURE: **The general solvability of grid m x n with Push "+" can be determined in O(lg N) time, where N=max(m,n)**.

However, this is only good for the general solvability; it cannot provide detailed information about the solvability result for a particular modulo k push, especially when the grid is generally solvable. If the grid is even not generally solvable, it is not completely solvable over GF (k), for any k.

Although not every grid mxn is generally solvable, we still can get their eigenvalues. One interesting we observed about the eigenvalues is that the sum of all eigenvalues is equal to the trace of the matrix. As the matrix is so constructed that we know the trace of A is mxn. Therefore we obtain the following conjecture:

CONJECTURE: **The sum of all eigenvalues of A for an mxn grid equals to mxn.**

Determinants with mod

Using the determinants with mod, we can see if the grid can be completely solved under certain modulo push. For example, the triangular table we present here shows little pattern, because mod 2 and mod 3 will change the solvability a lot.

The determinants modulo 3 are presented below for reference too.

```
1)    1
2)    0 0
3)    1 0 1
4)    1 1 0 0
5)    0 0 0 1 0
6)    1 1 1 1 1 1
7)    1 0 1 0 0 1 1
8)    0 0 0 1 0 1 0 0
9)    1 0 0 0 0 1 0 0 0
10)    1 1 1 1 1 1 1 1 1 1
11)    0 0 0 0 0 1 0 0 0 1 0
12)    1 1 1 1 1 0 1 1 1 1 1 0
13)    1 0 1 1 0 0 1 0 1 1 0 1 1
14)    0 0 0 0 0 1 0 0 0 1 0 1 0 0
15)    1 0 1 0 0 1 1 0 0 1 0 1 1 0 1
16)    1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1
17)    0 0 0 1 0 1 0 0 0 1 0 1 0 0 0 1 0
18)    1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1
19)    1 0 0 0 0 1 0 0 0 1 0 1 1 0 1 1 0 1 0
20)    0 0 0 1 0 1 0 0 0 0 0 1 0 1 0 1 0 1 0 1
21)    1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1
22)    1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1
23)    0 0 0 0 0 1 0 0 0 1 0 1 0 0 0 1 0 1 0 0 0 1 0
24)    1 1 0 0 1 1 0 1 0 1 1 1 1 0 1 1 1 1 0 1 1 1 1 0
25)    1 0 1 1 0 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1
```

```
  26)    0 0 0 1 0 0 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0
  27)    1 0 1 0 0 0 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1
  28)    1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
  29)    0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 1 0 1 0 0 0 1
  30)    1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
  31)    1 0 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0
  32)    0 0 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0
  33)    1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1
  34)    1 1 0 0 1 1 1 1 0 1 1 1 1 0 1 1 1
  35)    0 0 0 0 0 1 0 0 0 1 0 1 0 0 0 1
  36)    1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
  37)    1 0 1 1 0 1 1 0 1 1 0 1 1 0
  38)    0 0 0 1 0 1 0 1 0 1 0 1 0
  39)    1 0 0 0 0 1 1 0 0 1 0 1
  40)    1 1 1 1 1 1 1 1 1 1 1
  41)    0 0 0 1 0 1 0 0 0 1
  42)    1 1 1 1 0 1 1 1 1
  43)    1 0 1 1 0 1 1 0
  44)    0 0 0 0 0 1 0
  45)    1 0 1 1 0 1
  46)    1 1 1 1 1
  47)    0 0 0 0
  48)    1 1 1
  49)    1 0
  50)    0
```

We can see grids 2x2, 4x3, etc. are not completely solvable, though the matrices associated with them are of full rank.

In addition, we studied the difference between full rank result and mod k result. It is hard to find any rules for the difference; we still don't know why the determinant for certain grid dimension is zero mod k. Say, for mod 2, we have grids (8,6), (14,12), (15,12), (15,13), (16, 10), (16, 11), (16, 12), (16, 13), (16, 16), (17, 6), … and so on to be zero (mod 2). There may be something underlying the difference.

Operational features

As of today, no operational features have been found for this push. But the rank information suggests there may be some. Because all m x n grids where m is prime are full rank, it seems there are certain modulo k pushes that can turn the grid to the all off state.

Looking at the PPNC case, we can see a similar method may be applied to get the conjectured features. However, in this case, it is more complicated, and the center-on push leaves a trail of center points we need to deal with.

Push “X” with No Center

Matrix A

The associated matrix of Push “X” with No Center (PXNC) is of the following form:

$$\mathbf{A} = \begin{array}{ccccc} \mathbf{0} & \mathbf{B} & \mathbf{0} & \ldots & \mathbf{0} \\ \mathbf{B} & \mathbf{0} & \mathbf{B} & \ldots & \mathbf{0} \\ \mathbf{0} & \mathbf{B} & \mathbf{0} & \ldots & \mathbf{:} \\ \mathbf{:} & \mathbf{:} & \mathbf{:} & \mathbf{0} & \mathbf{B} \\ \mathbf{0} & \mathbf{0} & \ldots & \mathbf{B} & \mathbf{0} \end{array} \qquad \mathbf{B} = \begin{array}{ccccc} 0 & 1 & 0 & \ldots & 0 \\ 1 & 0 & 1 & \ldots & 0 \\ 0 & 1 & 0 & \ldots & : \\ : & : & : & 0 & 1 \\ 0 & 0 & \ldots & 1 & 0 \end{array}$$

An example of A(3,2) will be:

```
Matrix A(3,2): rank = 4
   0   0   0   0   1   0
   0   0   0   1   0   1
   0   0   0   0   1   0
   0   1   0   0   0   0
   1   0   1   0   0   0
   0   1   0   0   0   0
```

It is easy to see the relatively simple structure of these matrices can provide us a simpler result.

Determinants and Ranks

Since all the determinants of the associated matrices with PXNC are either 1 (-1) or 0, there is no need to list twice the result for determinants and ranks. This nice property, as PPNC has, makes it ***modulo k independent***.

The ranks of the matrices of grid from 1x1 to 15x15 and from 16x15 to 30x1 are listed below:

```
1)      0
2)      0 1
3)      0 0 0
4)      0 1 0 1
5)      0 0 0 0 0
6)      0 1 0 1 0 1
7)      0 0 0 0 0 0 0
8)      0 1 0 1 0 1 0 1
9)      0 0 0 0 0 0 0 0 0
10)     0 1 0 1 0 1 0 1 0 1
```

```
  11)     0 0 0 0 0 0 0 0 0 0 0
  12)     0 1 0 1 0 1 0 1 0 1 0 1
  13)     0 0 0 0 0 0 0 0 0 0 0 0 0
  14)     0 1 0 1 0 1 0 1 0 1 0 1 0 1
  15)     0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  16)     0 1 0 1 0 1 0 1 0 1 0 1 0 1 0
  17)     0 0 0 0 0 0 0 0 0 0 0 0 0 0
  18)     0 1 0 1 0 1 0 1 0 1 0 1 0
  19)     0 0 0 0 0 0 0 0 0 0 0 0
  20)     0 1 0 1 0 1 0 1 0 1 0
  21)     0 0 0 0 0 0 0 0 0 0
  22)     0 1 0 1 0 1 0 1 0
  23)     0 0 0 0 0 0 0 0
  24)     0 1 0 1 0 1 0
  25)     0 0 0 0 0 0
  26)     0 1 0 1 0
  27)     0 0 0 0
  28)     0 1 0
  29)     0 0
  30)     0
```
The pattern can be easily seen.

Operational Analysis

THEOREM: An m x n grid is completely solvable under PXNC if and only if both m and n are even.

Proof: We observe that the push at (p, q) only changes vertices (p-1, q-1), (p+1, q+1), (p-1, q+1), (p+1, q-1). Each push will move the on-vertices 2 units, either horizonally or vertically. A vertex can be successfully turned off only when it can be transferred to next-to-corner position, e.g., (2, 2) if the index starting from 1. We can see that only by pushing the corner (1, 1), can we turn (2, 2) off. This means only when both m and n are even, can we transfer all vertices to the "next-to-corner" positions, and get the grid into all-off state. If either m or n is odd, then for all vertices at an odd position, they cannot be transferred to "next-to-corner" position whose coordinates are even. ÿ

Push “X”

Matrix A

The matrix for Push “X” is of the following form:

$$\mathbf{A} = \begin{array}{ccccc} \mathbf{I} & \mathbf{B} & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{B} & \mathbf{I} & \mathbf{B} & \dots & \mathbf{0} \\ \mathbf{0} & \mathbf{B} & \mathbf{I} & \dots & : \\ : & : & : & \mathbf{I} & \mathbf{B} \\ \mathbf{0} & \mathbf{0} & \dots & \mathbf{B} & \mathbf{I} \end{array} \qquad \mathbf{B} = \begin{array}{ccccc} 0 & 1 & 0 & \dots & 0 \\ 1 & 0 & 1 & \dots & 0 \\ 0 & 1 & 0 & \dots & : \\ : & : & : & 0 & 1 \\ 0 & 0 & \dots & 1 & 0 \end{array}$$

An example of the matrix for the 3x2 grid is:

```
Matrix A(3,2):
   1   0   0   0   1   0
   0   1   0   1   0   1
   0   0   1   0   1   0
   0   1   0   1   0   0
   1   0   1   0   1   0
   0   1   0   0   0   1
```

We can see the inner structure of the matrix may suggest similar properties as in the PPNC case.

Determinants

The determinants mod 3 of the matrices of grid from 1x1 to 25x25 and from 26x25 to 50x1 are listed below (the 1 after the right parenthesis is not part of the result, it is used later):

```
01)1    1
02)1    1 0
03)1    1 1 0
04)1    1 1 1 0
05)1    1 0 1 1 0
06)1    1 1 1 1 1 1
07)1    1 1 0 1 1 1 0
08)1    1 0 1 1 0 1 1 0
09)1    1 1 1 0 1 1 0 1 0
10)1    1 1 1 1 1 1 1 1 1 1
11)1    1 0 0 1 0 1 0 0 1 1 0
12)1    1 1 1 1 1 0 1 1 1 1 1 1
13)1    1 1 1 1 1 1 1 1 1 1 1 0 1
14)1    1 0 1 0 0 1 1 0 0 1 0 0 1 0
15)1    1 1 0 1 1 1 0 1 0 1 0 1 0 0 1
```

```
16)1    1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1
17)1    1 0 1 1 0 1 1 0 1 0 0 1 1 0 1 1 0
18)1    1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1
19)1    1 1 0 0 1 1 0 1 0 1 1 1 1 0 1 1 1 1 0
20)1    1 0 1 1 0 1 1 0 1 0 0 1 1 0 1 1 0 1 1 0
21)1    1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1
22)1    1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
23)1    1 0 0 1 0 1 0 0 0 0 0 1 1 0 1 1 0 1 1 0 1 1 0
24)1    1 1 1 0 1 1 1 1 0 1 1 1 1 0 1 1 1 1 0 1 1 1 1 0
25)1    1 1 1 1 1 0 1 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1
26)1    1 0 1 1 0 1 1 0 0 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1
27)1    1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
28)1    1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1
29)1    1 0 1 0 0 1 1 0 0 1 0 1 1 0 1 1 0 1 0 0 1 1
30)1    1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
31)1    1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
32)1    1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1
33)1    1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1
34)1    1 1 1 0 1 1 1 1 0 1 1 1 1 0 1 1 1
35)1    1 0 0 1 0 0 1 0 1 1 0 1 1 0 1 1
36)1    1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
37)1    1 1 1 1 1 1 1 1 1 1 1 1 1 1
38)1    1 0 1 1 0 0 1 0 1 1 0 1 1
39)1    1 1 0 0 0 1 1 1 0 1 1 1
40)1    1 1 1 1 1 1 1 1 1 1 1
41)1    1 0 1 1 0 0 1 0 1 1
42)1    1 1 1 1 1 1 1 1 1
43)1    1 1 0 1 1 1 1 1
44)1    1 0 1 0 0 1 1
45)1    1 1 1 1 1 0
46)1    1 1 1 1 1
47)1    1 0 0 1
48)1    1 1 1
49)1    1 1
50)1    1
```

There seem to be certain patterns in the data, but it is hard to see it because the determinant information is not as essential as rank information.

Ranks

The ranks of the matrices of grid from 1x1 to 25x25 and from 26x25 to 50x1 are listed below (the 1 after the right parenthesis is not part of the result; it is used for clearer understanding of the patterns you'll see later.):

```
01)1    1
02)1    1 0
03)1    1 1 1
04)1    1 1 1 0
05)1    1 0 1 1 0
06)1    1 1 1 1 1 1
```

```
07)1    1 1 1 1 1 1 1
08)1    1 0 1 1 0 1 1 0
09)1    1 1 1 0 1 1 1 1 0
10)1    1 1 1 1 1 1 1 1 1 1
11)1    1 0 1 1 0 1 1 0 1 1 0
12)1    1 1 1 1 1 1 1 1 1 1 1 1
13)1    1 1 1 1 1 1 1 1 1 1 1 1 1
14)1    1 0 1 0 0 1 1 0 0 1 0 1 1 0
15)1    1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
16)1    1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
17)1    1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0
18)1    1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
19)1    1 1 1 0 1 1 1 1 0 1 1 1 1 0 1 1 1 1 0
20)1    1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0
21)1    1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
22)1    1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
23)1    1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0
24)1    1 1 1 0 1 1 1 1 0 1 1 1 1 0 1 1 1 1 0 1 1 1 1 0
25)1    1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
26)1    1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1
27)1    1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
28)1    1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
29)1    1 0 1 0 0 1 1 0 0 1 0 1 1 0 1 1 0 1 0 0 1 1
30)1    1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
31)1    1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
32)1    1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1
33)1    1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
34)1    1 1 1 0 1 1 1 1 0 1 1 1 1 0 1 1 1
35)1    1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1
36)1    1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
37)1    1 1 1 1 1 1 1 1 1 1 1 1 1 1
38)1    1 0 1 1 0 1 1 0 1 1 0 1 1
39)1    1 1 1 0 1 1 1 1 0 1 1 1
40)1    1 1 1 1 1 1 1 1 1 1 1
41)1    1 0 1 1 0 1 1 0 1 1
42)1    1 1 1 1 1 1 1 1 1
43)1    1 1 1 1 1 1 1 1
44)1    1 0 1 0 0 1 1
45)1    1 1 1 1 1 1
46)1    1 1 1 1 1
47)1    1 0 1 1
48)1    1 1 1
49)1    1 1
50)1    1
```

From the rank information, we can see for m x n grid:

- A is full rank if m+1 mod 3 ≠ 0 and m+1 mod 5 ≠ 0
- A is not full rank only if (m+1 mod 3 = 0 or m+1 mod 5 = 0) and (n+1 mod 3 = 0 or n+1 mod 5 = 0).

Therefore, we have the following conjecture:

CONJECTURE: **The m x n grid is not completely solvable with Push "X" over GF(k), for any k, if ((m+1 mod 3 = 0 or m+1 mod 5 = 0) and (n+1 mod 3 = 0 or n+1 mod 5 = 0).**

There are some m x n grids for which the push problem is not completely solvable over GF(k), with certain k. However, the grids can be completely solvable for a different k.

Some Unbalanced Pushes

An unbalanced push is a push that is neither left-right or up-down symmetric. There are many forms of unbalanced pushes. Some of the pushes have the state of the activated vertex itself changed. We call these pushes, **with center**; otherwise **with no center**. The pushes with center usually have more changes to make grids completely solvable in that they provide more control on vertices.

There are some unbalanced pushes that are simple to analyze. These pushing patterns only change the states of the pushed vertex's neighbors whose positions are in half of the whole space. For example, pushing v only changes v's top neighbors' states. These pushes can be easily found either completely solvable for all mxn cases, or not, depending on if the pushed vertex itself gets pushed or not.

More complicated unbalanced push patterns can be anything but symmetric and can happen anywhere, not restricted to the close neighbors of pushed vertex. They are hard to analyze using the operational method.

Among all unbalanced pushes, some push patterns are just left-right or top-bottom symmetric, we only need to study one case.

Left-Right Symmetric Pushes

The definition of left-right symmetric push refers to a push pattern which is only left-right symmetric (not up-down symmetric).

Meanwhile, an up-down symmetric push can be regarded as a left-right symmetric push too because if we transpose the grid and the push pattern as well, we see the same pushing problem except the row and column dimensions are changed.

An LR symmetric push is usually simpler because the problem can be confined in one line. Since the push pattern is not symmetric top-down, we can easily find out whether the push can transfer all on vertices to off, except the bottom line (from top to bottom). If not, then the problem is not completely solvable. If yes, we need only study one line - bottom line.

For example, a “V”-shaped push, where pushing on vertex v changes the left, right and bottom neighbors of v, is a unbalanced LR symmetric push. We need only know how it performs in one line, because for each initial configuration in a grid, we can transfer it into a state where only vertices on the bottom line are on. In this case, the push we want to study is actually a one-line pushing pattern. It is not hard to see the rule governing the “V” push is similar to the rule governing PXNC.

It is not hard to see the rules governing the “V” push are:

- For m x n grid, the push switch problem is completely solvable if and only if m mod 3 $\neq$ 2.

We can derive the same result from the push’s associated matrix A. The determinant of A will be zero only when m mod 3 = 2. By Guassian elimination, we can easily see this pattern.

CHAPTER 4 CONCLUSION

The summary of important results in this thesis is:

- ✓ **Under PPNC, grid mxn can be completely solvable if and only if GCD (m+1, n+1) = 1, independent with modulo k push.**
- ✓ **Under PXNC, grid mxn is completely solvable if and only if both m and n are even.**
- ✓ **(Conjecture)Under PP, grid mxn can be generally solvable,**
  - **If the prime factors of m don't contain 2, 3 or 5, regardless of n.**
  - **If the prime factors of m contain 2, 3 or 5, then the solvability pattern (starting from 0) is the logical "and" of 2's pattern, (110)*, 3's pattern, (10)*, and 5's pattern (11110)*.**
- ✓ **(Conjecture)Under PP, grid mxn cannot be completely solvable regardless of modulo k push if the pattern above at nth place is zero.**
- ✓ **(Conjecture)Under PX, grid mxn is not completely solvable with Push "X" over GF(k) if ((m+1 mod 3 = 0 or m mod 5 = 0) and (n+1 mod 3 = 0 or n+1 mod 5 = 0).**

REFERENCES

1. Goldwasser and W. Klostermeyer, *Maximization Versions of 'Lights Out' Games in Grids and Graphs*; J. Congressus Numerantium, vol. 126, 1997, pp. 99-111.
2. H. Ware. *Divisibility Properties of Fibonacci Polynomials over GF(2)*; MS Report, Dept. of Statistics and Computer Science, West Virginia University, August 1997
3. J. Goldwasser, W. Klostermeyer, and G. Trapp, *Characterizing Switch Setting Problems*; Linear and Multilinear Algebra, vol. 43, no. 1-3 (1997), pp. 121-136
4. Donald E. Knuth, *The Art of Computer Programming*, second edition, vol. 1 and 2, Addison-Wesley, Reading, Massachusetts, 1973, 1981.
5. J. Goldwasser, W. Klostermeyer, G. Trapp, and C.Q. Zhang, *Setting Switches on a Grid*; Technical Report #95-20, Department of Statistics and Computer Science, WVU, 1995
6. W. McCague; *Using Circulant Adjacency Matrices to Analyze Coin Flipping Problems;* MS Report, Dept. of Statistics and Computer Science, West Virginia University, April 1996
7. Alfred V. Aho, Hohn E. Hopcroft, and Jeffery D. Ullman. *The Design and Analysis of Computer Algorithm*, Addison-Wesley, Reading, Massachusetts, 1974
8. G. Strang, *Linear Algebra and its Applications*, Academic Press, New York, 1976
9. Alvy R. Smith, *Simple Computation-Universal Cellular Spaces and Self-Reproduction*, 9th IEEE FOCS Conference Record, pp. 269-277, October 1968
10. Joe Hicklin, Cleve Moler, Peter Webb, Ronald F. Boisvert, etc. *JAMA: A Matrix Package*. http://math.nist.gov/javanumerics/jama/.