Solving systems of linear equations through zero forcing set

Jianbo Wang* Chao Xu† Siyun Zhou‡

Abstract

Let $\mathbb{F}$ be any field, we consider solving $Ax = b$ for a matrix $A \in \mathbb{F}^{n\times n}$ of m non-zero elements and $b \in \mathbb{F}^n$. If we are given a zero forcing set of A of size k, we can solve the linear equation in $O(mk + k^\omega)$ time, where ω is the matrix multiplication exponent. As an application, we show how the lights out game in an $n \times n$ grid is solved in $O(n^3)$ time, and then improve the running time to $O(n^\omega \log n)$ by exploiting the repeated structure in grids.

1 Introduction

Solving the linear system $Ax = b$ has been a fundamental problem in mathematics. We consider the problem in its full generality. Given a field $\mathbb{F}$, a matrix $A \in \mathbb{F}^{n\times n}$ and a vector $b \in \mathbb{F}^n$, we are interested to find an $x \in \mathbb{F}^n$ such that $Ax = b$. Additionally, a zero forcing set associated with A is also given.

As arguably the most important algorithmic problem in linear algebra, there are many algorithms designed for solving systems of linear equations, we refer the reader to [13]. However, most works are for matrices over the real, complex or rational number fields. As this work is concerned with general fields, we will describe some known general algorithms below. The Gaussian elimination is a classic method for solving systems of linear equations, which requires $O(n^3)$ time. If the system is of full rank, one can reduce the problem to a single matrix multiplication, by taking $x = A^{-1}b$. In general cases, the more recent LSP decomposition, which is a generalization of the LUP decomposition, allows the running time to match the matrix multiplication [15, 16]. If one views the non-zeros of the matrix as an adjacency matrix of a graph, the property of the graph can be then used to speed up the algorithm. For example, if the graph is planar, then the nested-dissection technique can generate an $O(n^{3/2})$ time algorithm when the field is real or complex [19]. Later, it was extended to any non-singular matrix over arbitrary fields, and the running time was improved to $O(n^{\omega/2})$ [3], where $\omega < 2.3728596$ is the matrix multiplication constant [2]. More recently, Fomin et al. made a major breakthrough in developing a fast algorithm that has a polynomial dependency on treewidth, pathwidth, and tree-partition width of the bipartite graph generated from the incidences of the row and the column [11, 12]. In particular, when restricted to square matrices, there exist an $O(k^2n)$ time algorithm for solving $Ax = b$ if a width k path decomposition or tree-partition is given, and an $O(k^3n)$ time algorithm if a tree decomposition of width k is given. These algorithms are fairly complicated, but share one similarity with our work in taking the advantage of properties of the graph to improve the running time of the algorithm.

Zero forcing sets were first studied in [1] on graphs relating to the maximum nullity of a matrix, and the results were later expanded to the directed graphs [4]. The zero forcing set captures some “core" information of the linear system, or to be more specific, the system $Ax = b$ is uniquely determined by the values of x on a zero forcing set, which implies that one just needs to observe part of x to recover

*jbwang962@gmail.com, University of Electronic Science and Technology of China, School of Computer Science and Engineering.

†the.chao.xu@gmail.com, University of Electronic Science and Technology of China, School of Computer Science and Engineering.

‡zhousiyun@std.uestc.edu.cn, University of Electronic Science and Technology of China, School of Mathematical Sciences.

Algorithm	Structural Requirement	Running Time	Algebraic Requirement
[19]	planar	$O(n^{3/2})$	positive definite
[3]	planar	$O(n^{\omega/2})$	square, non-singular
[12]	pathwidth/tree-partition width k	$O(k^2(n+p))$	
[12]	treewidth k	$O(k^2(n+p))$	
This Work	zero-forcing number k	$O(km+k^\omega)$	

Figure 1.1: Known algorithms using graph structure corresponding to the $n \times p$ matrix.

its entirety. This idea leads to the independent discovery of zero forcing set by physicists for control of quantum systems [8, 20]. Later it was shown to be applicable to Phasor Measurement Units (PMU) for monitoring power networks [9], and also discovered as a graph searching technique [23]. Most studies in zero forcing set concentrate on its algebraic and combinatorial properties. On the computational front, finding the smallest zero forcing set of both the undirected and directed graphs is NP-hard [21,23], and some exact algorithms have been proposed [6, 7].

This work was inspired by an algorithm for the lights out game. In the game, there is a light on each vertex of the graph, and each light is in a state either on (1) or off (0). There is also a button on each vertex. Pressing the button would flip the state of the light of itself and all its (out-)neighbors. The goal is to turn off all the lights. The lights out game is equivalent to solving a system of linear equations $Ax + b = 0$ in $\mathbb{F}_2$, where A is the adjacency matrix of the graph G, and b is the state of the lights and $\mathbb{F}_2$ is the finite field of order 2. The interpretation is that $x_v = 1$ means pressing the button at vertex v, and b_v is the initial state of the light at vertex v. Note that in $\mathbb{F}_2$, $-b = b$, and thus the game is equivalent to solving $Ax = b$. There is a large amount of research in the lights out game, see [10] for a survey. Finding a solution of the lights out game with the minimum number of button presses is NP-hard [5].

We focus on the case where G is an $n \times n$ grid, which corresponds to an $n^2 \times n^2$ matrix. Gaussian elimination would take $O(n^6)$ time. An alternative approach is the light-chasing algorithm [18], which is equivalent to zero forcing in the grid graph. Since the lights in the first row uniquely affect the states of the remaining rows, one can then look up which action on the first row should be taken according to the states of the last row. However, the literature does not provide the strategy for finding the lookup table. Wang claimed there is an $O(n^3)$ time algorithm that given the state of the last row, the state of the first row can be found through solving a linear equation of an n-square matrix [22]. Wang's result is the motivation behind this work. However, there is no proof of its correctness. Alternatively, the algorithm in [12] can be used to obtain an $O(n^4)$ time solver by observing that the pathwidth is n and the path decomposition is computed in linear time.

Our contribution We generalize Wang's method for the lights out game to solving an arbitrary system of linear equations $Ax = b$, and give a formal proof of its correctness. We define a structure, the core matrix B of A, such that one can solve the system of linear equations over B instead of A, and then lift it to a solution of A in linear time. As a consequence, we obtain the following algorithmic result.

Theorem 1.1 (Main) *Given a matrix $A \in \mathbb{F}^{n\times n}$ of m non-zero elements, and a zero forcing set of A of size k where $k \leq n$. A data structure of size $O(k^2)$ can be computed in $O(mk + k^\omega)$ time, such that for each $b \in \mathbb{F}^n$, solving $Ax = b$ takes $O(k^2 + m)$ time.*

Note that the data structure is just the LSP decomposition of the core matrix.

We also show that the lights out game on an $n \times n$ grid can be solved in $O(n^\omega \log n)$ time by finding the core matrix using an alternative method. In addition, we prove some linear algebraic properties of the zero forcing set, which might be of independent interest.

Comparison with previous algorithms If the pathwidth and zero-forcing number are within a constant of each other, the performance of our algorithm can be no worse than the existing algorithms. Actually, our algorithm can be much more efficient when the graph is sparse. In the extreme case where m, the number of edges, is $O(n)$, our algorithm is faster by a factor of k. For example, when the graph is an $n \times n$ grid graph, the pathwidth is n, and so is the zero-forcing number. Since there are n^2 nodes, the algorithm in [12] would then take $O(n^4)$ time, while our algorithm takes a faster $O(n^3)$ time.

On the other hand, zero forcing number can be much larger than the pathwidth. Indeed, the zero forcing number of a star on n vertices is $n-1$, but the pathwidth is $O(1)$. Also, finding a small zero forcing set is difficult, and no good approximation is known so far. In this context, our algorithm may be very limited in practical applications.

2 Preliminaries

Define an index set $[n] = \{1, \ldots, n\}$. We consider an algebraic model, where every field operation takes $O(1)$ time, and each element in the field can be stored in $O(1)$ space.

It is useful to view a vector x indexed by elements X as a function $x : X \to \mathbb{F}$, and thus we can write $x \in \mathbb{F}^X$. Define $\mathrm{supp}(x)$ to be the set of non-zero coordinates of x. For a matrix A and a set of row indices R and a set of column indices C, we define $A_{R,C}$ to be the submatrix of elements indexed by the rows and columns in R and C, respectively. We use $*$ to represent the set of all row indices or column indices, depending on context. For example, $A_{R,*}$ stands for the submatrix of A that is composed of the rows indexed by R, $A_{i,*}$ is the ith row vector of A, and $A_{*,j}$ is the jth column vector. In addition, $A_{i,j}$ is the element in the ith row and jth column of A. For a vector x, x_i is the scalar at index i, and x_I is the subvector formed by the elements in the index set I.

Solving $Ax = b$ is finding a vector x such that $Ax = b$ holds. Given a directed graph $G = (V, E)$ with n vertices, and a field $\mathbb{F}$, we define $\mathcal{M}(G, \mathbb{F})$ to be the set of all matrices $A \in \mathbb{F}^{n \times n}$ such that for all $u \neq v$, $A_{u,v} \neq 0$ if and only if $(u, v) \in E$. We do not impose any restriction on $A_{v,v}$. Let $N^+(u) = \{v \mid (u, v) \in E\}$ be the set of all out-neighbors of u.

Then, we briefly review some basic concepts and results related to the zero forcing. Consider the process of coloring a graph. We start with a set of blue colored vertices, denoted by Z. If there exists a blue vertex v with exactly one non-blue out-neighbor, say $u \in V$, then color u blue. The operation of turning u blue is called *forcing*, and we say u is forced by v. If this process ends up with a situation where all vertices are colored blue, then we say the initial set Z is a *zero forcing set*. The *zero forcing number* $Z(G)$ of G is the size of the smallest zero forcing set. We also call Z a zero forcing set of A if $A \in \mathcal{M}(G, \mathbb{F})$ for some field $\mathbb{F}$. The following proposition gives the reason for the name "zero forcing".

Proposition 2.1 ([1, 14]) *For a zero forcing set Z of A, if $x \in \ker(A)$, then $x_Z = 0$ implies $x = 0$. Namely, x vanishing at zero forcing set forces x to be* 0.

The converse is not true in general. As a counterexample, we can simply take A to be a 2×2 identity matrix. Then, $Z = \{1, 2\}$ is the unique zero forcing set of A, and $\ker(A) = \{0\}$. If $x \in \ker(A)$ and $x_1 = 0$, then we have $x = 0$. But $\{1\}$ is clearly not a zero forcing set.

The order of forces for a set Z is not unique, although the final coloring is [1, 14]. For simplicity, we avoid this issue by considering a particular chronological list of forces π, which picks the smallest indexed vertex that can be forced. π is a total ordering of the vertices such that the elements in Z are ordered arbitrarily, and smaller than all elements in $V \setminus Z$. For each $v, u \in V \setminus Z$, $v \leq_\pi u$ if v is forced no later than u. The forcing graph is a graph where there is an edge between u and v if u forces v. It is well known that such graph is a set of node disjoint paths that begin in Z [14]. Hence, if v forces u, we can define $u^\uparrow = v$ to be the *forcing parent*, and correspondingly, u is called the *forcing child* of $u^\uparrow$. A vertex is a *terminal*, if it does not have a forcing child. Let T be the set of terminals, then $|T| = |Z|$. See Figure 2.1 for sequence of forcing and terminal vertices.

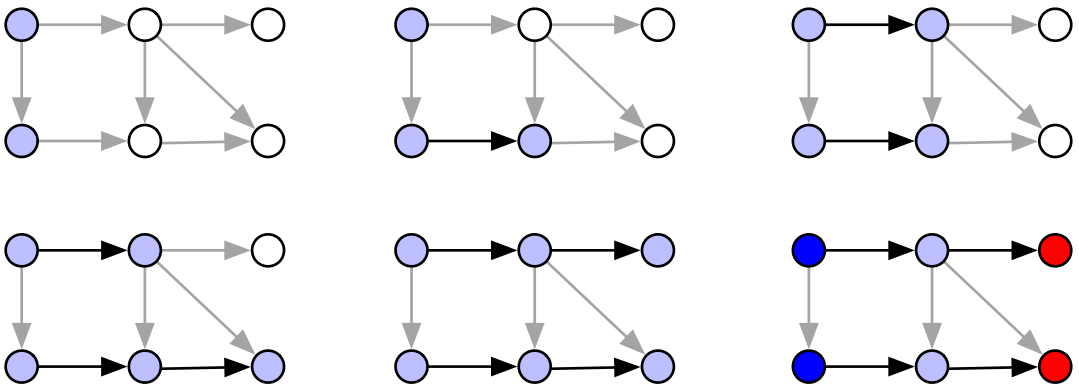

Figure 2.1: The sequence of forcing starting from right most two vertices. In the final figure, the zero forcing set Z is the dark blue vertices, and the terminal set T is the red vertices.

Consider the algorithm FORCING(A, Z, b) as given in Figure 2.2, which takes a matrix A, a vector b, and a zero forcing set Z of A as inputs, and updates x_u for all $u \in V$ in each round. We set $x_u = 0$ during initialization. Then, for each $u \in V \setminus Z$, we update x_u according to $\left(b_{u^\uparrow} - \sum_{v \in N^+(u^\uparrow)\setminus\{u\}} A_{u^\uparrow,v} x_v\right)/A_{u^\uparrow,u}$, iteratively in the forcing order. Note that it is equivalent to setting $x_u \leftarrow \left(b_{u^\uparrow} - A_{u^\uparrow,*}x\right)/A_{u^\uparrow,u}$, since x_u is previously 0.

```
FORCING(A, Z, b)
  x ← 0
  for u ∈ V \ Z ordered by some forcing sequence π
      x_u ← (b_{u↑} − A_{u↑,*} x) / A_{u↑,u}
  return x
```

Figure 2.2: The forcing operation.

The following result formalizes the relation between the solution to a given linear system and the corresponding zero forcing set.

Proposition 2.2 *After the value of x_u is updated in* FORCING(A, Z, b)*, we have $A_{u^\uparrow,*}x = b_{u^\uparrow}$ for all $v \leq_\pi u$. In particular, if $Ax = b$ has a solution where $x_Z = 0$, then* FORCING(A, Z, b) *finds such solution.*

Proof: The proof is by induction. The trivial case is when no x_u has been updated, then the conclusion is vacuously true. If some x_u is updated by FORCING(A, Z, b), we let x' be the vector before the update, and x'' be the one after the update. Then we get $x'_v = x''_v$ for all $v \neq u$, and $x'_u = 0$. And $A_{u,*}x'' = A_{u,*}x' + A_{u^\uparrow,u}x''_u = A_{u,*}x' + A_{u^\uparrow,u}(b_{u^\uparrow} - A_{u^\uparrow,*}x')/A_{u^\uparrow,u} = b_{u^\uparrow}$. Also, for all $v <_\pi u$, $A_{v^\uparrow,*}x'' = A_{v^\uparrow,*}x' = b_{v^\uparrow}$. □

For some fixed A and Z, FORCING(A, Z, b) is a linear transform with respect to b. Moreover, the running time of FORCING(A, Z, b) is $O(m)$, where m is the number of non-zero elements in A.

Theorem 2.3 *Let Z be a zero forcing set of A, and L be the matrix such that $Lb =$* FORCING(A, Z, b)*. The following statements hold.*

1. *The columns in $V \setminus Z$ are linearly independent.*
2. *If $Ax = b$, $Ay = b$, and $x_Z = y_Z$, then $x = y$.*
3. *Given $x' \in \mathbb{F}^V$ such that $\mathrm{supp}(x') \subseteq Z$. If $Ax = b$ for some x such that $x_Z = x'_Z$, then $x = L(b - Ax') + x'$.*

Proof: The proofs of the first two statements can be found in [17]. Though for the second statement, only the $b = 0$ version was proven in [17], its proof still works for a general b. For the third statement, if $Ax = b$ where $x_Z = x'_Z$, then $A(x' + (x - x')) = b$, or in other words, $A(x - x') = b - Ax'$. Hence by Proposition 2.2, $L(b - Ax') = x - x'$, and therefore $(x - x') + x' = x$. □

The *LSP decomposition* of $A \in \mathbb{F}^{m\times n}$ takes the form $A = LSP$, where $L \in \mathbb{F}^{m\times m}$ is a lower triangular matrix with value 1 in the diagonals, $S \in \mathbb{F}^{m\times n}$ can be reduced to a upper triangular matrix if all zero rows are deleted and elements in the main diagonal after the zero row deletion is non-zero, and $P \in \mathbb{F}^{n\times n}$ is a permutation matrix. The LSP decomposition can be found in $O(mnr^{\omega-2})$ time [16], where r is the rank of A. Given the LSP decomposition of an $n \times n$ matrix A, solving $Ax = b$ can be achieved by solving $Ly = b$ and $SPx = y$ based on the back substitution [15], which takes $O(n^2)$ time.

3 Solving $Ax = b$ through zero forcing set

In this section, we first introduce the *core matrix* that serves as the key part of our algorithm, followed by its theoretical guarantees. Based on the core matrix, we then present the detailed algorithm for solving $Ax = b$ with a given zero forcing set of size k, as well as the corresponding computational analysis.

For a clear exposition, we set up the instances we are working with. Throughout this section, we fix a directed graph $G = (V, E)$, a field $\mathbb{F}$, a matrix $A \in \mathcal{M}(G, \mathbb{F})$, a zero forcing set Z of G, a forcing order π, and the terminals T under the forcing order. Since the operations are performed on indices, without loss of generality, we assume that $V = [n]$ and $Z = [k]$.

Let $L(b) = \textsc{Forcing}(A, \pi, b)$. Given that L is a linear transform, we abuse the notation and let L be the matrix that induces the transform. Let $R = I - AL$. It can be observed that $\text{supp}(Rb) \subseteq T$ and $\text{supp}(Lb) \subseteq V \setminus Z$ for all b. Moreover, we have that $|V| = n$, $|Z| = k$, and the number of non-zero elements in A is m.

3.1 Core matrix

As Theorem 2.3 suggests, the solution to $Ax = b$ can be obtained by knowing only x_Z. Hence, it is natural to think of finding the correct x_Z instead of acquiring the full information of x for solving $Ax = b$.

Let $a_v = A_{*,v}$ be the column of A indexed by v. Define a $k \times k$ matrix $B \in \mathbb{F}^{T\times Z}$ as $B = (RA)_{T,Z}$. In other words, the vth column equals $R(a_v)_T$. The matrix B is called the *core matrix* of A. We will show that for an arbitrary b, $Ax = b$ holds if and only if $Bx_Z = R(b)_T$. Hence, solving the equation $By = R(b)_T$ along with some post processing is sufficient to give a solution of $Ax = b$.

Lemma 3.1 *$Ra_v = 0$ for $v \notin Z$.*

Proof: The forcing algorithm given in Figure 2.2 shows that if $v \neq u$, we will then have $x_v = 1$, and $x_u = 0$. Note that $Ax = a_v$, and therefore $Ra_v = a_v - Ax = 0$. □

Before delving into the key theorem, we introduce some useful notations and definitions that will be repeatedly used in the remaining part of this section. Let $\text{rank}(A) = r \geq n - k$. Define $A' = A_{*,[k]} \in \mathbb{F}^{n\times k}$, and $A'' = A_{*,[n]\setminus[k]} \in \mathbb{F}^{n\times(n-k)}$. To facilitate the analysis, we further decompose A' into a 1×2 block matrix form $A' = [A'_1 \quad A'_2]$ such that $A'_1 = [a_1 \quad \dots \quad a_p]$, $A'_2 = [a_{p+1} \quad \dots \quad a_k]$, $\text{rank}([A'_1 \quad A'']) = r$, $\text{rank}(A'') = n - k$ and $\text{rank}(A'_1) = p = r - (n - k)$. The matrix A can be then rewritten as

$$A = [A'_1 \quad A'_2 \quad A'']. \tag{1}$$

The following result presents the rank-preserving property of A'_1 under the linear mapping R. Based on this property, together with the assumption of the existence of the solution to $Ax = b$.

Lemma 3.2 $\text{rank}(RA'_1) = \text{rank}(A'_1)$.

Proof: For any $s \in \mathbb{F}^n$, there exist $\gamma_{k+1}, \cdots, \gamma_n$, such that

$$s + \sum_{i=k+1}^{n} \gamma_i a_i = Rs,$$

which implies that $(R-I)s \in \text{span}(A'')$. Define another linear mapping Q as

$$Q = R - I, \tag{2}$$

and we have $\text{span}\, Q \subseteq \text{span}\, A''$. Then, we can write RA'_1 as

$$RA'_1 = (Q+I)A'_1 = QA'_1 + A'_1,$$

where $QA'_1 \in \text{span}\, Q \subseteq \text{span}\, A''$. Hence, the columns of QA'_1 and the ones of A'_1 are linearly independent, which immediately gives $\text{rank}(RA'_1) = \text{rank}(A'_1)$. □

The following theorem plays a pivotal role in the theoretical guarantees for our algorithm.

Theorem 3.3 *Given $A \in \mathbb{F}^{n\times n}$ of the form (1). Let $M \in \mathbb{F}^{k\times n}$ ($k \leq n$) such that $A'' \in \ker(M)$, and $\text{rank}(MA'_1) = \text{rank}(A'_1)$. For $b \in \mathbb{F}^n$, suppose that $Ax = b$ has a solution. If $MA'y = Mb$ for some $y \in \mathbb{F}^k$, then there exists $x \in \mathbb{F}^n$ such that $Ax = b$ and $x_Z = y$.*

Proof: Since $A'_2 \in \text{span}(A'_1, A'')$, we can rewrite A'_2 as $A'_2 = A'_1 C'_1 + A''C''$ where $C'_1 \in \mathbb{F}^{p\times(k-p)}$ and $C'' \in \mathbb{F}^{(n-k)\times(k-p)}$ are coefficient matrices. The existence of the solution to $Ax = b$ indicates that $b \in \text{span}(A'_1, A'')$. Similarly, we can write $b = A'_1 C'_b + A''C''_b$ where $C'_b \in \mathbb{F}^p$ and $C''_b \in \mathbb{F}^{n-k}$ are coefficient vectors. Then, we have

$$\begin{aligned} M[A'_1 \quad A'_2]y = M[A'_1 \quad A'_1C'_1 + A''C'']y = M[A'_1 \quad A'_1C'_1]y, \\ Mb = M(A'_1C'_b + A''C''_b) = MA'_1C'_b, \end{aligned}$$

where the last equalities of the above two formulas follow from $A'' \in \ker(M)$. Then, the equation $MA'y = Mb$ can be written as $[MA'_1 \quad MA'_1C'_1]y = MA'_1C'_b$. Decomposing $y \in \mathbb{F}^k$ into $y = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}$ with $y_1 \in \mathbb{F}^p, y_2 \in \mathbb{F}^{k-p}$ further leads to a new homogeneous linear system

$$MA'_1(y_1 + C'_1y_2 - C'_b) = 0.$$

The condition $\text{rank}(MA'_1) = \text{rank}(A'_1)$ gives $\text{rank}(MA'_1) = p$, or to say, MA'_1 is of full rank. Thus, we have $y_1 + C'_1y_2 - C'_b = 0$.

In a similar manner, we decompose $x \in \mathbb{F}^n$ into three subvectors $x_1 \in \mathbb{F}^p, x_2 \in \mathbb{F}^{k-p}$, and $x_3 \in \mathbb{F}^{n-k}$. Then $Ax = b$ gives

$$[A'_1 \quad A'_1C'_1 + A''C'' \quad A''] \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = A'_1C'_b + A''C''_b.$$

Taking $x_1 = y_1$, $x_2 = y_2$ yields

$$A'_1y_1 + A'_1C'_1y_2 + A''C''y_2 + A''x_3 = A'_1C'_b + A''C''_b,$$

which can be rearranged as

$$A'_1(y_1 + C'_1y_2 - C'_b) + A''(C''y_2 + x_3 - C''_b) = 0.$$

Since $C''y_2 + x_3 - C''_b = 0$, we get

$$A''(C''y_2 + x_3 - C''_b) = 0.$$

Now, we can set $x_3 = C''_b - C''y_2$, which immediately gives a solution of $Ax = b$ of the form

$$x = \begin{bmatrix} y_1 \\ y_2 \\ C''_b - C''y_2 \end{bmatrix}.$$

□

Theorem 3.4 *Let B be the core matrix of A. If there is a y such that $By = R_{T,*}b$, then there exists x such that $Ax = b$ and $x_Z = y$. Otherwise, $Ax = b$ has no solution.*

Proof: Assume there is a solution to $Ax = b$. Since $\text{supp}(Rx), \text{supp}(R_{T,*}x) \in T$ for all x, we have $\text{rank}(RA'_1) = \text{rank}(R_{T,*}A'_1)$. By Theorem 3.3 with $M = R_{T,*}$, we know there is a solution of $Ax = b$ such that $x_Z = y$.

Otherwise, suppose for contradiction that $Ax = b$ has no solution, but $By = R_{T,*}b$. From the definition of the core matrix B, we get $(RA)_{T,Z}y = (Rb)_T$. Then, we can construct an x^* as

$$x^*_Z = y, \quad x^*_{V/Z} = 0,$$

such that $(RA)_{T,*}x^* = (Rb)_T$ holds. Due to the fact that $\text{supp}(RA) \subseteq T$ and $\text{supp}(Rb) \subseteq T$, we obtain that $RAx^* = Rb$, and from (2) we further arrive at $QAx^* + Ax^* = Qb + b$. Since $QAx^*, Ax^*, Qb \in \text{span}(A)$, we thus have $b \in \text{span}(A)$, which gives a contradiction. □

3.2 The algorithm

We first provide the algorithm FINDCORE for effectively computing the core matrix in Figure 3.1, with the computational cost given in Theorem 3.5.

```
FINDCORE(A, Z, b)
    π, T ← the forcing ordering and terminal set
    for v ∈ Z
        z ← FORCING(A, Z, A_{*,v})
        B_{*,v} ← (A_{*,v})_T − A_{T,*}z
    Compute the LSP decomposition of B
    return B
```

Figure 3.1: Find the core matrix.

Theorem 3.5 *The algorithm* FINDCORE *takes $O(mk + k^\omega)$ time.*

Proof: Computing $L(A_{*,v})$ for all $v \in Z$ takes $O(m(1 + |Z|)) = O(mk)$ time. The computation of $B_{*,v}$ for a $v \in Z$ consists of a vector-vector difference and a matrix-vector product, which can be implemented in $O(m)$ linear time. Thus, computing the core matrix takes $O(mk)$ time in total. The time taken for LSP decomposition is $O(k^\omega)$. □

Next, the algorithm SOLVELINEARSYSTEMGIVENCORE(A, Z, B, b) in Figure 3.2 shows how the solution of $Ax = b$ is obtained using the computed core matrix and all the information about the zero forcing set. By Theorem 3.4, we know there exists a solution that matches y at the zero forcing set. By Theorem 2.3, we obtain the remaining part of the solution through forcing. The corresponding computational cost is provided in Theorem 3.6.

```
SolveLinearSystemGivenCore(A, Z, B, b)
  π, T ← forcing sequence and terminal set T
  z ← Forcing(A, Z, b)
  b' ← b_T − A_{T,*} z
  y ← solution to By = b'
  if y does not exists
      return "NO SOLUTION"
  x' ← the vector where x'_Z = y and 0 everywhere else
  x ← x' + Forcing(A, Z, b − Ax')
  return x
```

Figure 3.2: Solve a linear system $Ax = b$ given a core matrix.

Theorem 3.6 *Given a matrix $A \in \mathbb{F}^{n\times n}$ of m non-zero elements, its zero forcing set of size k, and a core matrix B represented by its LSP decomposition. The system of linear equations $Ax = b$ can be solved in $O(k^2 + m)$ time.*

Proof: Following the algorithm in Figure 3.2, the computation of $b' = R_{T,*}b$ by forcing takes $O(m)$ time. As mentioned earlier, if the LSP decomposition of the matrix is given, then solving a system of linear equations with k variables and k equations takes $O(k^2)$ time. Once the solution of the linear system for the core matrix is attained, we can find the solution to the original problem through forcing in $O(m)$ time. The total running time is thus $O(k^2 + m)$. □

Combining Theorem 3.5 and Theorem 3.6, we finally arrive at our main theorem.

Theorem 1.1 (Main) *Given a matrix $A \in \mathbb{F}^{n\times n}$ of m non-zero elements, and a zero forcing set of A of size k where $k \leq n$. A data structure of size $O(k^2)$ can be computed in $O(mk + k^\omega)$ time, such that for each $b \in \mathbb{F}^n$, solving $Ax = b$ takes $O(k^2 + m)$ time.*

4 Lights out game on a grid

In this section, we show that the lights out game in an $n \times n$ grid can be solved in $O(n^\omega \log n)$ time.

Consider the lights out game on an $n \times n$ grid graph. We number the vertices in position (i, j) with index $(i-1)n + j$, and hence, the vertices are in $[n^2]$. Let $Z = [n]$, which is obviously a zero forcing set. If we apply Theorem 1.1 directly to the adjacency matrix of the grid graph, then in this special case, we will obtain precisely the algorithm of [22]. Since $m = n^2$ and $k = n$, the algorithm takes $O(n^3)$ time. The computational bottleneck is the calculation of the core matrix, as forcing itself only takes $O(n^2)$ time. Fortunately, by exploiting the repeated structure in grids, we can significantly improve the running time of computing the core matrix to $O(n^\omega \log n)$.

The forcing operation for the lights out game is greatly simplified, because the field is $\mathbb{F}_2$. In this case, $x_u = 1$ or $x_u = 0$ can be interpreted as pressing the button at vertex u or not, respectively. The b_u can be understood as the state of light at vertex u, where the value 1 means on, and 0 means off. When operating on the uth vertex, the forcing operation sets $x_u = 1$ if and only if $b'_{u\uparrow} = 1$. Here b' is the state of the board after applying all previous button presses. In other words, the forcing operation is iteratively setting $x_u = b'_{u\uparrow}$.

We then encode the operation of forcing, where we are given the states of the first and second rows and the aim is to compute the force operations on the second row ensuring that the states of the first row can be all 0. The output is the states of the second and the third rows after all of the button presses. To this end, we define such matrix to be $N(n) \in \mathbb{F}_2^{(2n)\times(2n)}$. The vertices of the first row are indexed from

$$\left[\begin{array}{ccc:ccc} 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 \\ \hdashline 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{array}\right]$$

Figure 4.1: The matrix $N(3)$.

1 to n. The vertex of the second row below the vertex i is $n+i$. One can easily verify that N could be written in the following block form

$$N(n) = \begin{bmatrix} N'(n) & I_n \\ I_n & 0_n \end{bmatrix},$$

where $N'(n)$ is the matrix satisfying that $N'(n)_{i,j} = 1$ if and only if $|i-j| \leq 1$.

Now, we define $M = N^{n-1}(n)$, which requires $O(n^\omega \log n)$ computational time using the exponentiation by squaring. Let a_j be the jth column of A. Since $\mathrm{supp}(a_j)$ is a subset of the first two rows if $j \in [n]$, we first let $a'_j = (a_j)_{[2n]}$. Next, we compute $y = Ma'_j$. Let t be the last n coordinates of y, which then yields the desired $R_{T,*}a_j$. Note that we can batch all the multiplications together, that is to say, we can compute $M[a'_j | j \in [n]] = MA_{[n],[2n]}$ in one go, which takes $O(n^\omega)$ running time, and recover the desired core matrix from it. The procedure described above is summarized in Figure 4.2.

```
FindGridCore(n)
  A ← adjacency matrix of an n × n grid
  M ← (N(n))^{n−1}
  return (MA_{[n],[2n]})_{[n],[n]}
```

Figure 4.2: Find core matrix for a grid graph.

At this point, we have provided an $O(n^\omega \log n)$ time algorithm for solving the lights out problem on an $n \times n$ grid. Moreover, the algorithm can also be applied to the $n \times m$ grids with $n \leq m$, and the running time becomes $O(n^\omega \log m + nm)$ accordingly.

Acknowledgements

Chao like to thank Jephian C.-H. Lin for discussion on algorithmic application of zero forcing set.

References

[1] AIM Minimum Rank – Special Graphs Work Group. Zero forcing sets and the minimum rank of graphs. *Linear Algebra and its Applications*, 428(7):1628–1648, April 2008.

[2] Josh Alman and Virginia Vassilevska Williams. A refined laser method and faster matrix multiplication. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 522–539.

[3] Noga Alon and Raphael Yuster. Matrix sparsification and nested dissection over arbitrary fields. *Journal of the ACM*, 60(4):1–18, August 2013.

[4] Francesco Barioli, Shaun Fallat, H. Hall, Daniel Hershkowitz, Leslie Hogben, Hein Van der Holst, Bryan Shader, Bryan Shader, Bryan Shader, Bryan Shader, and Bryan Shader. On the minimum

rank of not necessarily symmetric matrices: A preliminary study. *The Electronic Journal of Linear Algebra*, 18, January 2009.

[5] Abraham Berman, Franziska Borer, and Norbert Hungerbühler. Lights Out on graphs. *Mathematische Semesterberichte*, 68(2):237–255, October 2021.

[6] Boris Brimkov, Caleb C. Fast, and Illya V. Hicks. Computational approaches for zero forcing and related problems. *European Journal of Operational Research*, 273(3):889–903, 2019.

[7] Boris Brimkov, Derek Mikesell, and Illya V. Hicks. Improved computational approaches and heuristics for zero forcing. *INFORMS Journal on Computing*, 33(4):1384–1399, 2021.

[8] Daniel Burgarth and Vittorio Giovannetti. Full control by locally induced relaxation. *Phys. Rev. Lett.*, 99:100501, Sep 2007.

[9] Nathaniel Dean, Alexandra Ilic, Ignacio Ramirez, Jian Shen, and Kevin Tian. On the power dominating sets of hypercubes. In *2011 14th IEEE International Conference on Computational Science and Engineering*, pages 488–491, 2011.

[10] Rudolf Fleischer and Jiajin Yu. *A Survey of the Game "Lights Out!"*, pages 176–198. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[11] Fedor V. Fomin, Daniel Lokshtanov, Michał Pilipczuk, Saket Saurabh, and Marcin Wrochna. Fully polynomial-time parameterized computations for graphs and matrices of low treewidth. In *Proceedings of the 2017 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1419–1432.

[12] Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, MichaŁ Pilipczuk, and Marcin Wrochna. Fully polynomial-time parameterized computations for graphs and matrices of low treewidth. *ACM Trans. Algorithms*, 14(3), jun 2018.

[13] Gene H Golub and Charles F Van Loan. *Matrix Computations*. Johns Hopkins Studies in the Mathematical Sciences. Johns Hopkins University Press, Baltimore, MD, 4 edition, February 2013.

[14] Leslie Hogben. Minimum rank problems. *Linear Algebra and its Applications*, 432(8):1961–1974, April 2010.

[15] Oscar H. Ibarra, Shlomo Moran, and Roger Hui. A generalization of the fast LUP matrix decomposition algorithm and applications. *Journal of Algorithms*, 3(1):45–56, 1982.

[16] Claude-Pierre Jeannerod. LSP matrix decomposition revisited. Technical Report 2006-28, École Normale Supérieure de Lyon, September 2006.

[17] Franklin H.J. Kenter and Jephian C.-H. Lin. On the error of a priori sampling: Zero forcing sets and propagation time. *Linear Algebra and its Applications*, 576:124–141, September 2019.

[18] C. David Leach. Chasing the Lights in Lights Out. *Mathematics Magazine*, 90(2):126–133, April 2017.

[19] Richard J. Lipton, Donald J. Rose, and Robert Endre Tarjan. Generalized Nested Dissection. *SIAM Journal on Numerical Analysis*, 16(2):346–358, April 1979.

[20] Simone Severini. Nondiscriminatory propagation on trees. *Journal of Physics A: Mathematical and Theoretical*, 41(48):482002, oct 2008.

[21] Maguy Trefois and Jean-Charles Delvenne. Zero forcing number, constrained matchings and strong structural controllability. *Linear Algebra and its Applications*, 484:199–218, 2015.

[22] Zheng Wang (axpokl). 点灯游戏Flip game的$O(n^3)$算法. https://zhuanlan.zhihu.com/p/53646257, 2018.

[23] Boting Yang. Fast–mixed searching and related problems on graphs. *Theoretical Computer Science*, 507:100–113, 2013. Combinatorial Optimization and Applications.