Discrete Applied Mathematics 161 (2013) 1699–1706

Contents lists available at SciVerse ScienceDirect

Discrete Applied Mathematics

journal homepage: www.elsevier.com/locate/dam

Bisimplicial edges in bipartite graphs

Matthijs Bomhoff[*], Bodo Manthey

Faculty of Electrical Engineering, Mathematics and Computer Science, University of Twente, P.O. Box 217, 7500 AE Enschede, The Netherlands

ARTICLE INFO

Article history:
Received 16 September 2010
Received in revised form 9 December 2010
Accepted 3 March 2011
Available online 31 March 2011

Keywords:
Bipartite graphs
Random graphs
Algorithms
Gaussian elimination

ABSTRACT

Bisimplicial edges in bipartite graphs are closely related to pivots in Gaussian elimination that avoid turning zeroes into non-zeroes. We present a new deterministic algorithm to find such edges in bipartite graphs. Our algorithm is very simple and easy to implement. Its running-time is $O(nm)$, where n is the number of vertices and m is the number of edges. Furthermore, for any fixed p and random bipartite graphs in the $G_{n,n,p}$ model, the expected running-time of our algorithm is $O\left(n^2\right)$, which is linear in the input size.

1. Introduction

When applying Gaussian elimination to a square $n \times n$ matrix M containing some elements with value zero, the choice of pivots can often determine the amount of zeroes turned into non-zeroes during the process. This is called the *fill-in*. Some matrices even allow Gaussian elimination without any fill-in. Avoiding fill-in has the nice property of bounding the required space for intermediate results of the Gaussian elimination to the space required for storing the input matrix M. This is often important for processing very large sparse matrices. Even when fill-in cannot be completely avoided, it is still worthwhile to avoid it for several iterations, motivating the search for pivots that avoid fill-in.

If we assume subtracting a multiple of one row of M from another turns at most one non-zero into a zero, we can represent the relevant structure of our problem using only $\{0, 1\}$ matrices. (This assumption is quite natural, as it holds with probability one for a random real-valued matrix.) Given such a square matrix M, we can construct the bipartite graph $G[M]$ with vertices corresponding to the rows and columns in M, where the vertex corresponding to row i and the one corresponding to column j are adjacent if and only if $M_{i,j}$ is non-zero. We denote the number of non-zero elements of M by m. Furthermore, we assume M has no rows or columns containing only zeroes, so the associated bipartite graph has no isolated vertices and $n \leq m \leq n^2$. Fig. 1 shows an example.

The $\{0, 1\}$ matrices that allow Gaussian elimination without fill-in correspond to the class of *perfect elimination bipartite* graphs [3]. Central to the recognition of this class of graphs is the notion of a *bisimplicial* edge: a bisimplicial edge corresponds to an element of M that can be used as a pivot without causing fill-in. The fastest known algorithm for finding bisimplicial edges has a running-time of $O(nm)$ for sparse instances and $O(n^{\omega})$ in general [2,6], where $\omega \leq 2.376$ is the matrix multiplication exponent [1]. However, fast matrix multiplication using the algorithm of Coppersmith and Winograd [1] has huge hidden constants, which makes it impractical for applications.

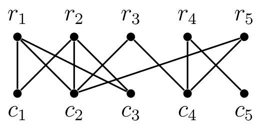
We present a new deterministic algorithm for finding all bisimplicial edges in a bipartite graph. Our algorithm is very fast in practice, and it can be implemented easily. Its running-time is $O(nm)$. In addition, we analyze its expected running-time on random bipartite graphs. For this, we use the $G_{n,n,p}$ model. This model consists of bipartite graphs with n vertices in each

[*] Corresponding author. Fax: +31 53 4894858.
E-mail addresses: m.j.bomhoff@utwente.nl (M. Bomhoff), b.manthey@utwente.nl (B. Manthey).

0166-218X/$ – see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.dam.2011.03.004

$$\begin{pmatrix} 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 \end{pmatrix}$$

a

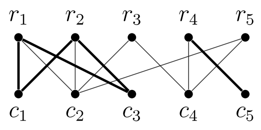


b

Fig. 1. An example of a {0, 1}-matrix M and its bipartite graph $G[M]$.

$$\begin{pmatrix} \boxed{1} & 1 & \boxed{1} & 0 & 0 \\ \boxed{1} & 1 & \boxed{1} & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & \boxed{1} \\ 0 & 1 & 0 & 1 & 0 \end{pmatrix}$$

a



b

Fig. 2. Bisimplicial edges in M and its bipartite graph $G[M]$ (bisimplicial edges are bold, the corresponding matrix entries are dashed).

vertex class, where edges are drawn independently, and each possible edge is present with a probability of p. We show that the expected running-time of our algorithm on $G_{n,n,p}$ graphs for fixed $p \in (0, 1)$ is $O(n^2)$, which is linear in the input size. (The input size of a random $G_{n,n,p}$ graph is $\Theta(n^2)$ with high probability.)

2. Bisimplicial edges

We denote by $\Gamma(u)$ the neighbors of a vertex u and by $\delta(u)$ its degree.

Definition 2.1. An edge (u, v) of a bipartite graph $G = (U, V, E)$ is called *bisimplicial*, if the induced subgraph $G[\Gamma(u) \cup \Gamma(v)]$ is a complete bipartite graph.

Clearly, we can determine in $O(m)$ time if an edge (u, v) is bisimplicial: we simply have to check all edges adjacent to it. So a simple algorithm to find a bisimplicial edge in a bipartite graph G, if one exists, takes $O(m^2)$ time. The bisimplicial edges in our example matrix M and associated graph $G[M]$ are shown in Fig. 2. As mentioned above, Goh and Rotem [2] have presented a faster algorithm based on matrix multiplication that can be implemented in either $O(n^{\omega})$ or $O(nm)$.

We present a different approach that first selects a set of candidate edges. The candidate edges are not necessarily bisimplicial and not all bisimplicial edges are marked as candidates. However, knowing which candidates, if any, are bisimplicial allows us to quickly find all other bisimplicial edges as well. By bounding the number of candidates, we achieve an improved expected running-time. The following observation is the basis of our candidate selection procedure.

Lemma 2.2. *If an edge (u, v) of a bipartite graph $G = (U, V, E)$ is bisimplicial, we must have $\delta(u) = \min_{u' \in \Gamma(v)} \delta(u')$ and $\delta(v) = \min_{v' \in \Gamma(u)} \delta(v')$.*

Proof. Let $(u, v) \in E$ be a bisimplicial edge, and let $A = G[\Gamma(u) \cup \Gamma(v)]$ be the complete bipartite graph it induces. Now assume that there is a vertex $u' \in U_A$ with $\delta(u') < \delta(u)$. Then there must be a $v' \in V_A$ with $u'v' \notin E_A$. But this would mean A is not a complete bipartite graph, leading to a contradiction. □

Translated to the matrix M, this means that if $M_{i,j} = 1$, it can only correspond to a bisimplicial edge if row i has a minimal number of 1s over all the rows that have a 1 in column j and column j has a minimal number of 1s over all the columns having a 1 in row i. In what follows, we will call the row (column) in M with the minimal number of 1s over all the rows (columns) in M the *smallest* row (column). Using this observation, we construct an algorithm to pick candidate edges that may be bisimplicial.

Algorithm 2.3. Perform the following steps:

1. Determine the row and column sums for each row i and column j of M.
2. Determine for each row i the index c_i of the smallest column among those with $M_{i,c_i} = 1$ (breaking ties by favoring the lowest index); or let $c_i = 0$ if row i has no 1.

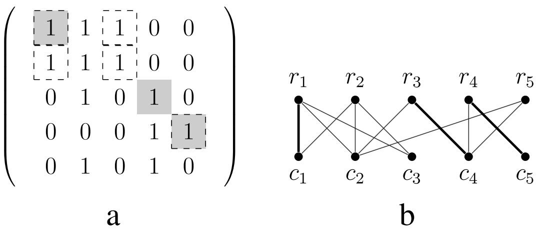


Fig. 3. Selected candidate edges in M (shaded) and its bipartite graph $G[M]$ (bold).

3. Determine for each column j the index r_j of the smallest row among those with $M_{r_j,j} = 1$ (breaking ties by favoring the lowest index); or let $r_j = 0$ if column j has no 1.
4. Mark $M_{i,j}$ as a candidate edge if $c_i = j$ and $r_j = i$.

Clearly, all steps in the algorithm can be performed in $O(n^2)$ time. Furthermore, the last step will mark at most n candidate edges and at least 1. (The reason that we have at least one candidate edge is as follows: let i be the smallest row with the smallest index. Row i will select a column j. Due to the tie-breaking mechanism, column j will also select row i, which leads to a candidate.) The candidate edges marked by this algorithm in our example matrix M are shown in Fig. 3.

The following lemmas establish a few more characteristics of the candidate edges.

Lemma 2.4. *Let i, j, j' be such that the following properties hold:*

1. *$M_{i,j} = 1$ and $M_{i,j'} = 1$ and*
2. *columns j and j' contain an equal number of 1s and*
3. *(i, j) is bisimplicial.*

Then (i, j') is also bisimplicial and columns j and j' are identical. Due to symmetry, the same holds if we exchange the roles of rows and columns.

Proof. If columns j and j' are not identical, but contain an equal number of 1s, then there is some row i' such that $M_{i',j} = 1$ and $M_{i',j'} = 0$. In that case (i, j) cannot be bisimplicial, so columns j and j' have to be identical. But then (i, j) and (i, j') both have to be bisimplicial due to symmetry. □

Lemma 2.5. *If (i', j') is bisimplicial, then there are $i \le i'$ and $j \le j'$ such that rows i and i' are identical, columns j and j' are identical, and (i, j) is bisimplicial and selected as a candidate by Algorithm 2.3.*

Proof. Let $j \le j'$ be the column with (1) the lowest index, (2) $M_{i',j} = 1$, and (3) an equal number of 1s to column j'. As (i', j') is bisimplicial, we know three things from Lemmas 2.2 and 2.4: first, (i', j) is also bisimplicial. Second, columns j and j' are identical. Third, columns j and j' are smallest columns in row i'. Due to symmetry, there is also such a row $i \le i'$ equal to row i' with the lowest index and (i, j') bisimplicial. As (i', j) and (i, j') are bisimplicial, rows i and i' are identical and columns j and j' are identical, also (i, j) must be bisimplicial. Furthermore, by construction, column j must be the smallest column in row i with the lowest index, and row i must be the smallest row in column j with the lowest index. Thus, (i, j) is selected as a candidate. □

Using Algorithm 2.3 as a subroutine, we can construct Algorithm 2.6 to find all bisimplicial edges of $G[M]$. Finding all bisimplicial edges instead of just a single one can be beneficial in practice when performing Gaussian elimination as not every possible pivot may preserve numerical stability.

Algorithm 2.6. Perform the following steps:

1. Determine candidates using Algorithm 2.3.
2. Test each candidate for bisimpliciality.
3. For each candidate (i, j) marked as bisimplicial, mark also each (i', j') as bisimplicial for each row i' with an equal number of non-zeroes as row i and $M_{i',j} = 1$ and column j' with an equal number of non-zeroes as column j and $M_{i,j'} = 1$.

Theorem 2.7. *Algorithm 2.6 finds all bisimplicial edges in time $O(n^3)$.*

Proof. Step 1 marks up to n candidates in time $O(n^2)$. Each of these candidates can be checked for bisimpliciality in time $O(n^2)$, so Step 2 can be completed in time $O(n^3)$. Finally, Step 3 marks all non-candidate bisimplicial edges as can be seen from Lemmas 2.4 and 2.5. For a single candidate (i, j) that is found to be bisimplicial, all relevant rows i' and columns j' can be found in time $O(n)$. A total of $O(n^2)$ additional edges can be marked as bisimplicial during this step and every non-candidate edge is considered at most once. Thus, this step can also be completed in time $O(n^2)$. □

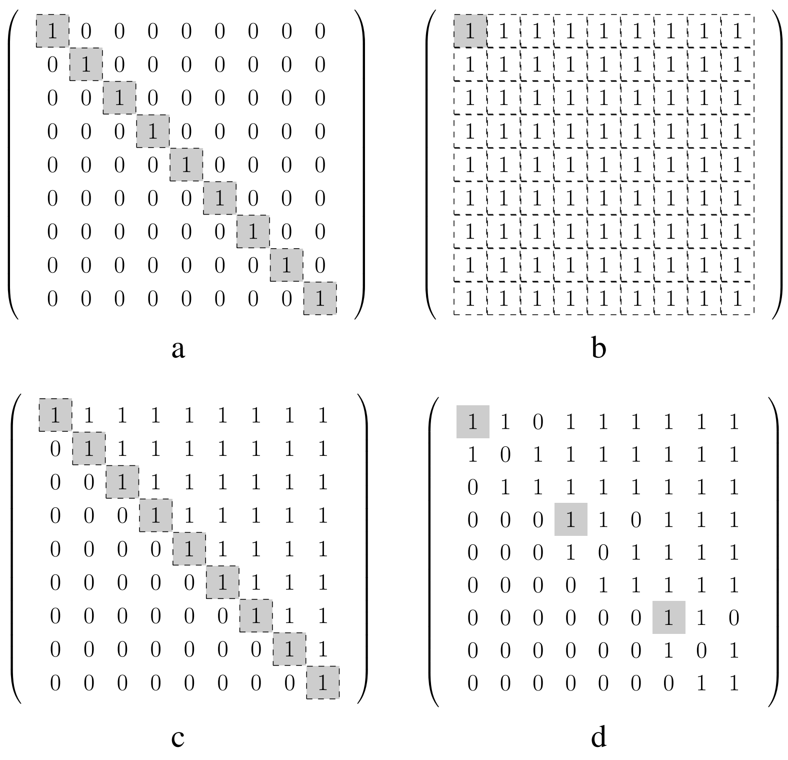


Fig. 4. Several example matrices with bisimplicial (dashed) and candidate (shaded) elements.

To give a bit more insight into the working of Algorithm 2.6, Fig. 4 shows several example matrices with their bisimplicial and candidate edges: Fig. 4(a) and (c) show situations in which candidates and bisimplicial edges are the same. Fig. 4(b) illustrates how a single candidate can be used to identify all edges as bisimplicial. Fig. 4(d) shows how an arbitrarily large matrix can be constructed with $n/3$ candidates and no bisimplicial edges at all.

The running-time of Algorithm 2.6 is dominated by Step 2 in which we have to check all candidates in $O\left(n^2\right)$ time each. As we can find up to n candidates, this leads to a worst-case running-time of $O\left(n^3\right)$. In the next section, we present an improved running-time analysis for sparse instances. After that, we show that our algorithm performs significantly better on random bipartite graphs. The reason for this is that our algorithm will usually select only a few candidate edges.

3. Sparse matrices

Algorithm 2.6 can be implemented such that it makes use of any sparsity in the matrix M. This section describes how a running-time of $O\left(Cm\right)$ can be obtained, where C denotes the number of candidates found in the first phase of the algorithm. As $C \leq n$, the running-time is bounded by $O\left(nm\right)$. We assume the input matrix M is provided in the form of adjacency lists of the corresponding graph $G[M]$: for every row (column) we have a list of columns (rows) where non-zero elements occur.

The first step of Algorithm 2.6 consists of running Algorithm 2.3, which selects the candidates. Algorithm 2.3 itself consists of three steps. The first step, determining the row and column sums, can be completed in time $O\left(m\right)$ by simply traversing the lists. The same holds for the second step: by traversing the adjacency lists the values of c_i and r_j can be determined in time $O\left(m\right)$. Constructing the actual set of candidates from these values can subsequently be done in time $O\left(n\right)$. In total, Algorithm 2.3 determines the set of candidates in time $O\left(m\right)$. After this time, the number C of candidates is known.

Checking a single candidate can be done in time $O\left(m\right)$. Thus, the second step of Algorithm 2.6, which consists of checking all candidates for bisimpliciality, can be performed in time $O\left(Cm\right)$.

Finally, we analyze the third step of Algorithm 2.6, marking the remainder of the bisimplicial edges. For each bisimplicial candidate (i, j), we have to find all rows i' identical to row i and columns j' identical to column j. Due to Lemma 2.4, we can simply traverse the adjacency lists for row i and column j and check the column and row sums. As every row and every column contains at most one candidate, all adjacency lists are traversed at most once. Thus, this takes at most time $O\left(m\right)$ for all candidates together. For each candidate, once all relevant rows i' and columns j' have been determined, we have to mark all combinations (i', j') as bisimplicial. As every edge is considered at most once during this process, this can also be completed in time $O\left(m\right)$.

Summarizing we have that the total running-time of Algorithm 2.6 is $O\left(Cm\right)$ where C is bounded from above by n and known in time $O\left(m\right)$ after the first phase of the algorithm has been completed.

4. A first bound on the number of candidate edges

In order to justify the good practical performance of our algorithm theoretically, we study the behavior of Algorithm 2.6 on random bipartite graphs in the $G_{n,n,p}$ model in the following sections. For such instances, we show that the number of candidates is significantly smaller than n, both with high probability and in expectation. This yields an improved expected running-time on these instances. In this section, we show a logarithmic bound on the expected number of candidates for a fixed value of the parameter p in the $G_{n,n,p}$ model. This also provides tail bounds for the distribution of the number of candidates. The two subsequent sections improve the bound for the expected number of candidates to a constant.

For a fixed value of $p \in (0, 1)$, we consider random bipartite graphs in the $G_{n,n,p}$ model. This means that we have n vertices in each vertex class and each edge is present with a probability of p. Such a random graph corresponds to a stochastic $n \times n\{0, 1\}$ matrix M with $\mathbb{P}\left[M_{i,j} = 1\right] = p$. Let X_i be the (random) i-th row of M, and let $|X_i|$ be the (random) sum of its elements. If we order the X_i vectors according to the number of 1s they contain (breaking ties by favoring lower values of i), we denote by $X_{(1)}$ the row with the least number of 1s, by $X_{(2)}$ the row with the second-to-least number etc.

Lemma 4.1. *Let $\varepsilon = 2\sqrt{\frac{\log n}{pn}}$. Then*

$$\mathbb{P}\left[|X_{(1)}| < (1-\varepsilon)pn\right] \le \frac{1}{n}.$$

Proof. Fix any $i \in \{1, \dots, n\}$. By Chernoff's bound [4], we have

$$\mathbb{P}\left[|X_i| < (1-\varepsilon)pn\right] < \mathrm{e}^{-np\varepsilon^2/2} = \mathrm{e}^{-2\log n} = \frac{1}{n^2}.$$

By a union bound over all rows, we get

$$\mathbb{P}\left[|X_{(1)}| < (1-\varepsilon)pn\right] \le n\mathbb{P}\left[X_i < (1-\varepsilon)pn\right] = \frac{1}{n}. \quad \square$$

Lemma 4.2. *Fix $p \in (0, 1)$. For $k \in o(\sqrt{n}/\log n)$, we have*

$$\mathbb{P}[C > k] \le (1 + o(1)) \cdot n(1-p)^k + \frac{1}{n}.$$

Proof. Choose $\varepsilon = 2\sqrt{\frac{\log n}{pn}}$ as in Lemma 4.1 above. If $|X_{(1)}| \ge (1-\varepsilon)pn$, we have for any column j

$$\mathbb{P}\left[\text{Column } j \text{ has no 1 in rows } X_{(1)}, \dots, X_{(k)} \mid |X_{(1)}| \ge (1-\varepsilon)pn\right] \le (1-p+\varepsilon p)^k.$$

Thus, the probability that in this case, any column does not have a 1 in the k rows with the smallest number of 1s is bounded from above by

$$\mathbb{P}\left[\exists j : \text{Column } j \text{ has no 1 in rows } X_{(1)}, \dots, X_{(k)} \mid |X_{(1)}| \ge (1-\varepsilon)pn\right] \le n(1-p+\varepsilon p)^k.$$

If all columns have at least one 1 in rows $X_{(1)}, \dots, X_{(k)}$, all candidates selected must be among these k rows, as they contain the smallest number of 1s over all the rows in M. Since each row contributes at most 1 candidate, Algorithm 2.3 selects at most k candidates in this case.

By Lemma 4.1, the probability that $|X_{(1)}| < (1-\varepsilon)pn$, i.e., the smallest row contains too few 1s, is bounded from above by $1/n$. Altogether, we get

$$\begin{aligned}\mathbb{P}[C > k] &\le n(1-p+\varepsilon p)^k + \frac{1}{n}\\ &\le n\left((1-p)^k + \sum_{i=1}^{k}\binom{k}{i}(\varepsilon p)^i(1-p)^{k-i}\right) + \frac{1}{n}\\ &\le n\left((1-p)^k + (1-p)^k \cdot \sum_{i=1}^{k}\left(\frac{k\varepsilon p}{1-p}\right)^i\right) + \frac{1}{n}.\end{aligned}$$

The lemma follows because $\sum_{i=1}^{k}\left(\frac{k\varepsilon p}{1-p}\right)^i \in o(1)$ since $\frac{k\varepsilon p}{1-p} \in o(1)$ by our choice of ε and k. $\square$

Lemma 4.2 says that at most $O(\log n)$ candidates are selected with high probability. This bound also holds in expectation: we use $k = 2\log_{(1-p)} \frac{1}{n} = 2\log_{1/(1-p)} n$. If the number of candidate edges exceeds k, then we use the worst-case bound of n. This gives us

$$\begin{aligned}\mathbb{E}[C] &\le k + n\mathbb{P}[C > k] \\ &\le k + n^2(1-o(1))(1-p)^k + 1 \\ &\le (2+o(1))\log_{1/(1-p)} n\end{aligned}$$

and the following theorem and corollary.

Theorem 4.3. *Fix $p \in (0,1)$ and consider random instances in the $G_{n,n,p}$ model. With a probability of $1 - O(\frac{1}{n})$ and in expectation, Algorithm 2.3 selects at most $(2+o(1))\log_{1-p} \frac{1}{n}$ candidates.*

Corollary 4.4. *For any fixed p, Algorithm 2.6 has an expected running-time of $O\left(n^2 \log_{1/(1-p)} n\right)$ on instances drawn according to $G_{n,n,p}$.*

5. Isolating lemma for binomial distributions

The tie-breaking of Algorithm 2.3 always chooses the row or column with the lowest index. Thus, the probability of the event that row i and column j becomes a candidate edge depends also on the number of rows (or columns) that actually have the minimum number of 1s.

Let us analyze the number of rows (or columns) that attain the minimum number of 1s. At first glance, one might argue as follows: the number of 1s in the rows are independent random variables with binomial distribution. Thus, according to Chernoff's bound, the number of 1s in each row is $np \pm O(\sqrt{n})$ with high probability. Hence, we have roughly np random variables that assume values in an interval of size roughly $O\left(\sqrt{n}\right)$. From this, we would expect that the minimum is assumed by roughly $O\left(\sqrt{n}\right)$ random variables. However, first, this bound does not give us any good bound on the number of candidates. Second, it is far too pessimistic. It turns out that, although relatively many random variables fall into a relatively small interval, the minimum is usually unique: the probability that the minimum is unique is $1 - o(1)$. This resembles the famous isolating lemma [5]. Even stronger, the expected number of random variables that assume the minimum is $1 + o(1)$. The following lemma is the crucial ingredient for this, and it captures most of the intuition.

Lemma 5.1. *Let $k \in \mathbb{N}$, and let $X_1, \dots, X_k$ be independent and identically distributed random variables with values in $\mathbb{Z}$. Let $Y = \min\{X_1, \dots, X_k\}$, and let $Z = |\{i \mid X_i = Y\}|$ be the number of random variables that assume the minimum value. Let $t \in \mathbb{Z}$, $q \in (0,1)$, and $c \in (0,1)$ such that the following properties hold:*

1. $\mathbb{P}[X_i \le t] \le q$ *for any* $i \in \{1, \dots, k\}$.
2. *For every $s > t$, we have* $\mathbb{P}[X_i = s \mid X_i \le s] \le c$.

Then

$$\mathbb{E}[Z] \le \frac{1}{1-c} + k^2 q.$$

Proof. The probability that $Y \le t$ is bounded from above by kq by a union bound over the k events $X_i \le t$. If indeed $Y \le t$, we use the trivial upper bound of $Z \le k$. This contributes the term $k^2 q$. Otherwise, we consider $X_1, X_2, \dots, X_k$ one after the other. Let $Y_i = \min\{X_1, \dots, X_i\}$. Let $Y_0 = \infty$ for consistency. Clearly, we have $Y_k = Y$. For every $i \in \{1, \dots, k\}$, we let an adversary decide whether $X_i \le Y_{i-1}$ or $X_i > Y_{i-1}$.

Fix any $\ell \in \mathbb{N}$, and let $j_0, j_1, \dots, j_\ell$ be the last $\ell + 1$ positions for which the adversary has chosen $X_{j_i} \le Y_{j_i - 1}$. By our choice of j_i, we have $Y_{j_i - 1} = Y_{j_{i-1}}$.

The crucial observation is that $Z \ge \ell + 1$ if and only if $X_{j_i} = Y_{j_{i-1}}$ for all $i \in \{1, \dots, \ell\}$. By independence and assumption, the probability of this is bounded from above by c^ℓ. This essentially shows that the distribution of $Z - 1$ is dominated by a geometric distribution with parameter c. Overall, we obtain

$$\mathbb{E}[Z] \le \sum_{\ell=0}^{\infty} c^\ell + k^2 q = \frac{1}{1-c} + k^2 q$$

as claimed. □

To actually get the result for binomial random variables, we show that the value for c from the lemma above can be chosen arbitrarily small. Intuitively, this is because for binomial distributions, adjacent values have approximately the same probability.

Lemma 5.2. *Fix any $p \in (0, 1)$. Let $X_1, \dots, X_k \sim \text{Binom(n, p)}$ be independent random variables distributed according to a binomial distribution with parameters n and p, and let $k \in O(n)$. Let $Y = \min\{X_1, \dots, X_k\}$, and let $Z = |\{i \mid X_i = Y\}|$. Then*

$$\mathbb{E}[Z] \leq 1 + o(1).$$

Proof. We show that the value for c in Lemma 5.1 can be chosen as $c = o(1)$, provided that n is sufficiently large.

Let $t = np - a$ for $a = \sqrt{n} \log n$. According to Chernoff's bound, we have $\mathbb{P}[X_i \leq t] = o(1/k^2)$ for any i. Thus, we can choose $q = o(1/k^2)$ for the application of Lemma 5.1.

Now we choose a slowly growing $x = x(n) \in \omega(1)$. We will give constraints for the function x later on. Our goal is to show that it is possible to choose $c = 2/x = o(1)$. This together with our choice of q yields

$$\mathbb{E}[Z] \leq \frac{1}{1 - 2/x} + qk^2 = 1 + \frac{2/x}{1 - 2/x} + qk^2 = 1 + o(1)$$

as claimed.

Now fix any $s > t$. We have

$$\begin{aligned}
\mathbb{P}[X_i = s \mid X_i \leq s] &\leq \frac{\mathbb{P}[X_i = s]}{\mathbb{P}[X_i \in \{s, s-1, \dots, s-x+1\}]} \\
&= \frac{\binom{n}{s} p^s (1-p)^{n-s}}{\sum_{\ell=s-x+1}^{s} \binom{n}{\ell} p^\ell (1-p)^{n-\ell}} \\
&= \frac{n! \cdot p^s (1-p)^{n-s}}{s! \cdot (n-s)! \cdot \sum_{\ell=s-x+1}^{s} \frac{n!}{\ell! \cdot (n-\ell)!} p^\ell (1-p)^{n-\ell}} \\
&= \frac{1}{\sum_{\ell=s-x+1}^{s} \frac{(1-p)^{s-\ell}}{p^{s-\ell}} \cdot \prod_{i=\ell+1}^{s} \frac{i}{n-i}}. \qquad (1)
\end{aligned}$$

Let us estimate the product within the summation in the denominator. For some appropriately chosen $\varepsilon > 0$, we have

$$\begin{aligned}
\prod_{i=\ell+1}^{s} \frac{i}{n-i} &\geq \left(\frac{s}{n-s}\right)^{s-\ell} \geq \left(\frac{t}{n-t}\right)^{s-\ell} = \left(\frac{p - \frac{a}{n}}{1 - p + \frac{a}{n}}\right)^{s-\ell} \\
&\geq \left((1-\varepsilon) \cdot \frac{p}{1-p}\right)^{s-\ell}.
\end{aligned}$$

The last inequality holds in particular for $\varepsilon = \frac{a}{n} \cdot \left(\frac{1}{1-p} + \frac{1}{p}\right)$ and n large enough such that $p > \log n / \sqrt{n}$. Plugging this into (1) yields

$$\mathbb{P}[X_i = s \mid X_i \leq s] \leq \frac{1}{\sum_{\ell=s-x+1}^{s} (1-\varepsilon)^{s-\ell}} \leq \frac{1}{x \cdot (1-\varepsilon)^x}.$$

The term on the right-hand side is bounded by $2/x$ for $x \leq \ln(1/2)/\ln(1-\varepsilon)$. Thus, we can choose $x = \lfloor \ln(1/2)/\ln(1-\varepsilon) \rfloor = \omega(1)$, which completes the proof. □

6. Constant bound for the number of candidates

Theorem 6.1. *Fix any $p \in (0, 1)$, and let C be the (random) number of candidates if we draw instances according to $G_{n,n,p}$. Then*

$$\mathbb{E}[C] \leq \frac{1 + o(1)}{p}.$$

Proof. Similar to Lemma 4.1, for $p' = (1 - \varepsilon)p$ and $\varepsilon = \frac{\log n}{\sqrt{n}}$, the probability that some row or column in M contains less than np' 1s is $o(1/n)$ by Chernoff's bound [4]. If some row or column of M does have fewer 1s, we simply assume that we have n candidates. This adds only $o(1)$ to our final expected value, which is negligible. For the remainder of the proof we may thus assume that all rows and columns contain at least np' 1s.

We proceed by bounding the probability that a row i contains a candidate. To establish an upper bound on this probability, we introduce a game on an unknown matrix M in which our adversary aims to increase the probability of row i containing a candidate as much as possible. For any fixed i, let us consider an unknown $n \times n$ matrix M and let our adversary pick a column j. We set $M_{i,j} = 1$ and let our adversary place additional 1s in column j so that it contains at least np' 1s. The other elements of M (i.e., those not in column j) are subsequently each assigned a 1 with probability p. Based on our assumption, every row and column now contains at least np' 1s. We now determine an upper bound on the maximum probability our adversary can achieve of row i containing a candidate.

The number and placement of 1s in column j is the only element of the game our adversary can influence to maximize the probability of row i containing a candidate. Thus, the optimal strategy is to maximize the probability of (i, j) becoming a candidate. In order to do this, the number of 1s in column j has to be as small as possible (to force row i to select column j), so we may assume our adversary places no more than $np' - 1$ additional 1s for a total of np'. We assume row i thus selects column j.

Now let Z again be a random variable denoting the number of rows containing the smallest number of 1s among all rows having a 1 in column j. Recall that $\mathbb{E}[Z] \le 1 + o(1)$ by Lemma 5.2. The probability of column j selecting row i in our algorithm is now bounded from above by $\mathbb{E}[Z]/np'$, which implies the probability of (i, j) becoming a candidate is also bounded by this probability. Plugging this in, we get

$$\begin{aligned}\mathbb{E}[C] &\le \sum_i \mathbb{P}[\text{row } i \text{ contains a candidate}] + o(1)\\ &= \sum_i \frac{\mathbb{E}[Z]}{np'} + o(1)\\ &\le \frac{\mathbb{E}[Z]}{p} + o(1)\\ &= \frac{1+o(1)}{p}. \quad \square\end{aligned}$$

For any fixed $p \in (0, 1)$, we have a constant bound on the expected number of candidates. This implies the expected running-time of Algorithm 2.6 on random instances of $G_{n,n,p}$ is $O(n^2)$. This expected running-time is linear in the input size.

7. Conclusion

Avoiding fill-in while performing Gaussian elimination is related to finding bisimplicial edges in bipartite graphs. Existing algorithms to find bisimplicial edges are based on matrix multiplication. Their running-time is dominated by the matrix multiplication exponent ($\omega \le 2.376$). We have presented a new algorithm to find such pivots that is not based on matrix multiplication. Instead, our algorithm selects a limited number of candidate edges, checks them for bisimpliciality, and finds all other bisimplicial edges based on that. The worst-case running-time of our algorithm is $O(n^3)$, but the expected running-time for random $G_{n,n,p}$ instances for fixed values of p is $O(n^2)$, which is linear in the input size. The main reason for this difference is that the expected number of candidates is only $\frac{1+o(1)}{p}$.

Besides improving on the expected running-time on random instances, our new algorithm is also very easy to implement in an efficient way. The running-time can be brought down easily to $O(Cm)$, where the number of candidates C is known after time $O(m)$ and is bounded from above by n. Thus, we have a worst-case running-time of $O(nm)$. The combination of ease of efficient implementation and a linear bound on the average-case running-time makes our algorithm very practical.

Existing algorithms for the recognition of perfect elimination bipartite graphs are based on finding a sequence of bisimplicial edges. We ask whether it is possible to extend our new algorithm to a new algorithm for the recognition of perfect elimination bipartite graphs.

Acknowledgements

We gratefully acknowledge the support of the Innovation-Oriented Research Programme 'Integral Product Creation and Realization (IOP IPCR)' of the Netherlands Ministry of Economic Affairs, Agriculture and Innovation.

References

[1] D. Coppersmith, S. Winograd, Matrix multiplication via arithmetic progressions, J. Symbolic Comput. 9 (1990) 251–280.
[2] L. Goh, D. Rotem, Recognition of perfect elimination bipartite graphs, Inform. Process. Lett. 15 (1982) 179–182.
[3] M.C. Golumbic, C.F. Goss, Perfect elimination and chordal bipartite graphs, J. Graph Theory 2 (1978) 155–163.
[4] R. Motwani, P. Raghavan, Randomized Algorithms, Cambridge University Press, Cambridge, United Kingdom, 1995.
[5] K. Mulmuley, U.V. Vazirani, V.V. Vazirani, Matching is as easy as matrix inversion, Combinatorica 7 (1987) 105–113.
[6] J.P. Spinrad, Recognizing quasi-triangulated graphs, Discrete Appl. Math. 138 (2004) 203–213.