Memorandum 1931 (December 2010). ISSN 1874-4850. Available from: http://www.math.utwente.nl/publications
Department of Applied Mathematics, University of Twente, Enschede, The Netherlands

Recognizing Sparse Perfect Elimination Bipartite Graphs*

Matthijs Bomhoff

Faculty of Electrical Engineering, Mathematics and Computer Science
University of Twente, P.O. Box 217, 7500 AE Enschede, The Netherlands
m.j.bomhoff@utwente.nl

Abstract. When applying Gaussian elimination to a sparse matrix, it is desirable to avoid turning zeros into non-zeros to preserve the sparsity. The class of perfect elimination bipartite graphs is closely related to square matrices that Gaussian elimination can be applied to without turning any zero into a non-zero. Existing literature on the recognition of this class and finding suitable pivots mainly focusses on time complexity. For $n \times n$ matrices with m non-zero elements, the currently best known algorithm has a time complexity of $O\left(n^3/\log n\right)$. However, when viewed from a practical perspective, the space complexity also deserves attention: it may not be worthwhile to look for a suitable set of pivots for a sparse matrix if this requires $\Omega\left(n^2\right)$ space. We present two new algorithms for the recognition of sparse instances: one with a $O\left(nm\right)$ time complexity in $\Theta\left(n^2\right)$ space and one with a $O\left(m^2\right)$ time complexity in $\Theta\left(m\right)$ space. Furthermore, if we allow only pivots on the diagonal, our second algorithm can easily be adapted to run in time $O\left(nm\right)$.

1 Introduction

Performing Gaussian elimination on sparse matrices may have the unfortunate side-effect of turning zeroes into non-zero values (*fill-in*), possibly even leading to a dense matrix along the way. Clearly, this can be undesirable, for example when working with very large sparse matrices. A natural question therefore is to ask when we can avoid fill-in during the elimination process. Recognizing matrices where fill-in can be avoided and selecting appropriate pivots can decrease the required effort and space for Gaussian elimination. For several special cases, such as symmetric (positive definite) matrices or pivots chosen along the main diagonal, this problem has been treated extensively in literature (see e.g. [1–5]).

The general case of avoiding fill-in on square nonsingular matrices was first treated in detail by Golumbic and Goss in [6]. They first describe the correspondence between matrices that allow Gaussian elimination without fill-in and bipartite graphs. Under the assumption that subtracting a multiple of a row from

* The author gratefully acknowledges 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.

another will always turn at most one element from nonzero to zero, an instance of the problem can be represented by a $\{0,1\}$ matrix M where $M_{i,j} = 1$ denotes that the original matrix contains a non-zero value at element (i,j). 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 vertices i and j are adjacent iff $M_{i,j}$ is nonzero. For example, the matrix shown in Fig. 1(a) corresponds to the bipartite graph shown in Fig. 1(b). Golumbic and Goss called the class of bipartite graphs corresponding to matrices that allow Gaussian elimination without fill-in *perfect elimination bipartite graphs*. This class is characterized using an elimination scheme detailed in the next section. Based on this scheme, they also obtained a first algorithm for the recognition of this class. Improved algorithms for the recognition of this class of graphs and their associated matrices have subsequently been published and are discussed briefly in what follows.

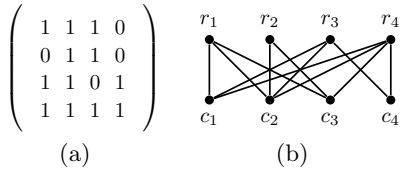

Fig. 1. example $\{0,1\}$-matrix M and its bipartite graph $G[M]$

The correspondence between perfect elimination bipartite graphs and matrices is mainly of practical value for sparse instances: the original motivation for investigating this class of graphs is preserving sparsity during Gaussian elimination on their associated matrices by avoiding fill-in. For the specific case of pivots chosen on the diagonal, Rose and Tarjan [5] have described two algorithms for finding perfect elimination orderings. Their algorithms represent the common trade-off between time and space. One is faster but needs more space, the other is slower but requires storage proportional to the number of non-zero elements. However, for the generic case it appears that efficient algorithms for the recognition of sparse instances have not yet been investigated. The focus in literature so far seems to be only on time complexity for dense instances. The best known algorithms for the generic case are based on a matrix multiplication which may well result in a dense matrix, see e.g. Fig. 2.

$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \times \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 4 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

Fig. 2. sparse M may lead to dense $Q = MM^T$

New results. In this paper, we present two algorithms for efficient recognition of sparse instances (where 'sparse' is used to indicate $m \ll n^2$): one with a $O(nm)$ time complexity in $\Theta(n^2)$ space and one with a $O(m^2)$ time complexity in $\Theta(m)$ space. We also show how our second algorithm can be adapted to solve the problem in time $O(nm)$ if only pivots on the diagonal are allowed.

The remainder of this paper is organized as follows: The next section describes the class of perfect elimination bipartite graphs as well as existing literature on algorithms for its recognition. The third section describes a new version of the algorithm by Goh and Rotem for the recognition of perfect elimination bipartite graphs that has been adapted to achieve a time complexity of $O(nm)$ instead of $O(n^3)$. The section after that describes a new recognition algorithm with a time complexity of $O(m^2)$ and a space complexity of $\Theta(m)$. Finally, we present a discussion on other possible improvements as well as a few brief conclusions regarding our results.

2 Perfect Elimination Bipartite Graphs

An edge uv of a bipartite graph is called *bisimplicial* if the neighbors of its endpoints $\Gamma(u) \cup \Gamma(v)$ (where $\Gamma(u)$ denotes the neighbors of u) induce a complete bipartite graph. Using this notion, perfect elimination bipartite graphs were first defined by Golumbic and Goss in [6] as follows:

Definition 1. *A bipartite graph $G = (U, V, E)$ is called* perfect elimination bipartite*, if there exists a sequence of pairwise nonadjacent edges $[u_1v_1, \ldots, u_nv_n]$ such that u_iv_i is a bisimplicial edge of $G - \{u_1, v_1, \ldots, u_{i-1}, v_{i-1}\}$ for each i and $G - \{u_1, v_1, \ldots, u_n, v_n\}$ is empty. Such a sequence of edges is called a* (perfect elimination) scheme*.*

This definition is based on the following theorem:

Theorem 1. *If uv is a bisimplicial edge of a perfect elimination bipartite graph $G = (U, V, E)$, then $G - \{u, v\}$ is also a perfect elimination bipartite graph.*

This theorem immediately implies a simple $O(n^5)$ algorithm for the recognition of perfect elimination bipartite graphs that also leads to an elimination scheme in case the graph is perfect elimination bipartite. Let us introduce the notion of row and column sets R_i and C_j defined as follows:

$$R_i = \{j \in \{1 \ldots n\} | M_{i,j} \neq 0\}$$
$$C_j = \{i \in \{1 \ldots n\} | M_{i,j} \neq 0\}$$

In other words: R_i contains the column numbers of elements in row i that have a non-zero value in M. Using these, we can describe the algorithm by Golumbic and Goss, shown in Algorithm 1. The algorithm basically performs n iterations, during each of which all remaining edges are completely checked for bisimpliciality.

Algorithm 1 original recognition algorithm by Golumbic and Goss

```
1: I ← {1...n}
2: J ← {1...n}
3: while I ≠ ∅ do
4:    f ← false
5:    for all (i, j) ∈ I × J do
6:       if M_{i,j} = 1 then
7:          g ← true
8:          for all (k, l) ∈ (C_j ∩ I) × (R_i ∩ J) do
9:             if M_{k,l} = 0 then
10:               g ← false
11:         if g = true then
12:            f = true, x ← i, y ← j
13:   if f = false then
14:      return false {G[M] is not perfect elimination bipartite}
15:   I ← I \ x
16:   J ← J \ y
17: return true {G[M] is perfect elimination bipartite}
```

In [7], Goh and Rotem present a faster recognition algorithm based on the following: A row $M_{a,*}$ is said to *majorize* a row $M_{b,*}$ if for each $1 \leq j \leq n$ we have $M_{a,j} \geq M_{b,j}$. According to this definition, every row majorizes itself.

Theorem 2. *[7] Let M be an $n \times n$ $\{0,1\}$ matrix representing a bipartite graph $G = (U, V, E)$. Let ℓ_i be the number of rows in M that majorize row i and let s_j be the sum of the entries in column j of M. Then $M_{i,j} = 1$ and $\ell_i = s_j$ iff the edge $u_i v_j$ is a bisimplicial edge of G.*

The values ℓ_i can be easily determined using the matrix $Q = MM^T$: ℓ_i is equal to the number of elements in the row $Q_{i,*}$ that are equal to $Q_{i,i}$ (including $Q_{i,i}$ itself). Once the matrix Q is computed, finding a bisimplicial edge can be done in $O\left(n^2\right)$ operations. If a bisimplicial edge is found, Q can be updated in $O\left(n^2\right)$ operations to the matrix Q' associated with $G' = G - \{u, v\}$ for the next iteration. After at most n iterations, the algorithm terminates, so the total time complexity of the algorithm is $O\left(n^3\right)$, a significant improvement over the $O\left(n^5\right)$ naive implementation. This algorithm is shown in Algorithm 2 (The notation M^{ij} is used to denote the (i, j) minor of M). As it needs to compute and store the matrix Q, its space complexity is $\Theta\left(n^2\right)$.

In [8], Spinrad obtains an algorithm with time complexity $O\left(n^3/\log n\right)$ using a notion of edges that may soon become suitable pivots during subsequent iterations as well as the faster matrix multiplication algorithm by Coppersmith and Winograd [9].

3 Goh-Rotem on Sparse Instances

By adapting the way calculations are performed, as well as the data structures, we obtain a new implementation of Algorithm 2 with time complexity $O(nm)$:

Algorithm 2 recognition algorithm by Goh and Rotem

simplicial_found ← **true**
compute the matrix $Q = (Q_{i,j})$ where $Q = MM^T$
$\forall j \in \{1 \dots n\} : s_j \leftarrow \sum_{i=1}^{n} M_{i,j}$
while there exists an $s_j \neq 0$ and simplicial_found **do**
  $\forall i \in \{1 \dots n\}$: let ℓ_i be the number of entries in row i of Q which are equal to $Q_{i,i}$
  if there exists a nonzero entry $M_{i,j}$ in M where $s_j = \ell_i$ **then**
    Compute the matrix $D = (d_{k,l})$ where $d_{k,l} = M_{k,j} \cdot M_{l,j}$
    $Q \leftarrow (Q - D)^{ii}$ {Q is now equal to $(M^{ij})(M^{ij})^T$}
    $\forall k \in \{1 \dots n\} : s_k \leftarrow s_k - M_{i,k}$
    $s_j \leftarrow 0$
  else
    simplicial_found ← **false**
return simplicial_found

an improvement for sparse graphs. Using the row and column sets we determine the matrix $Q = MM^T$ as

$$Q_{i,j} = |R_i \cap R_j| \quad . \tag{1}$$

Based on this new formulation, we arrive at the following lemma that will be used below to derive the time complexity of our new algorithm:

Lemma 1. *An upper bound on the sum of the elements in Q is given by*

$$\sum_{i,j} Q_{i,j} \leq nm \quad . \tag{2}$$

Proof.

$$\sum_{i,j} Q_{i,j} = \sum_{i,j} |R_i \cap R_j| \leq \sum_{i} \sum_{j} |R_j| = nm$$

□

Besides the matrix Q, we require an additional $n \times (n+1)$ matrix B the values of which are defined by

$$B_{i,k} := |\{j | j \in 1 \dots n, Q_{i,j} = k\}| \quad . \tag{3}$$

I.e., $B_{i,k}$ contains the number of elements in row i of Q that have the value k. After computation of Q, the matrix B can be computed in time $O\left(n^2\right)$. Furthermore, without increasing the time complexity of an algorithm, we can keep B up to date if we perform any updates to elements of Q. Using B, we can easily determine the value of ℓ_i as

$$\ell_i = B_{i,Q_{i,i}} \quad . \tag{4}$$

Using our set-based calculation of Q and the new matrix B, we can adapt the original algorithm by Goh and Rotem and arrive at our new version shown in Algorithm 3. Apart from our use of the sets I and J to denote the rows and columns that are still part of M during the current iteration instead of taking minors of the involved matrices, the working of the algorithm is still basically identical to Algorithm 2. However, the additional bookkeeping of B and the upper bound on the sum of the elements in Q enable us to achieve an improved time complexity for sparse instances.

Algorithm 3 adapted Goh-Rotem algorithm

Require: $Q = 0$ $\{Q$ is a $n \times n$-matrix$\}$
Require: $B = 0$ $\{B$ is a $n \times (n+1)$-matrix$\}$
1: $I \leftarrow \{1 \dots n\}$
2: $J \leftarrow \{1 \dots n\}$
3: **for all** $(i,j) \in I \times J$ **do**
4: $\quad Q_{i,j} \leftarrow |R_i \cap R_j|$
5: $\quad B_{i,Q_{i,j}} \leftarrow B_{i,Q_{i,j}} + 1$
6: $\forall j \in J : s_j \leftarrow |C_j|$
7: **while** $I \neq \emptyset$ **do**
8: $\quad f \leftarrow$ **false**
9: $\quad$ **for all** $i \in I$ **do**
10: $\quad\quad$ **for all** $j \in R_i$ **do**
11: $\quad\quad\quad$ **if** $B_{i,Q_{i,i}} = s_j$ **then**
12: $\quad\quad\quad\quad f \leftarrow$ **true**, $x \leftarrow i, y \leftarrow j$
13: $\quad$ **if** $f =$ **false then**
14: $\quad\quad$ **return false** $\{G[M]$ is not perfect elimination bipartite$\}$
15: $\quad \forall i \in I : R_i \leftarrow R_i \setminus y$
16: $\quad$ **for all** $j \in J$ **do**
17: $\quad\quad$ **if** $x \in C_j$ **then**
18: $\quad\quad\quad s_i \leftarrow s_i - 1$
19: $\quad\quad C_j \leftarrow C_j \setminus x$
20: $\quad$ **for all** $(i,j) \in C_y \times C_y$ **do**
21: $\quad\quad B_{i,Q_{i,j}} \leftarrow B_{i,Q_{i,j}} - 1$
22: $\quad\quad Q_{i,j} \leftarrow Q_{i,j} - 1$
23: $\quad\quad B_{i,Q_{i,j}} \leftarrow B_{i,Q_{i,j}} + 1$
24: $\quad \forall i \in I : B_{i,Q_{i,x}} \leftarrow B_{i,Q_{i,x}} - 1$
25: $\quad I \leftarrow I \setminus x$
26: $\quad J \leftarrow J \setminus y$
27: **return true** $\{G[M]$ is perfect elimination bipartite$\}$

Theorem 3. *The time complexity of Algorithm 3 is* $O(nm)$.

Proof. From Lemma 1 we know the sum of the elements of Q is bounded by $O(nm)$. This implies the initialization of the matrices Q and B in the loop on line 3 can be completed within time $O(nm)$. This leaves us with the task of establishing the same bound on the main loop from line 7 on down. Clearly, the

main loop is executed up to n times, either finding and processing a pivot, or returning false during each iteration. Within the main loop, the first loop on line 9 processes each of the $O(m)$ edges in constant time. If a suitable pivot is found, we first update the R_i and C_j sets in lines 15 and 16. This can be done in time $O(m)$.

After that, we have to update the matrices Q and B in the loop on line 20. Every iteration of this inner loop decreases some element of Q by one. As none of the elements are decreased below zero, Lemma 1 again gives us a bound of $O(nm)$ on the number of iterations of this inner loop over the course of the entire algorithm.

Finally, the loop on line 24 decreases $O(n)$ values of B after which I and J are updated to reflect the removal of the pivot row and column; all of this can be done in time $O(n)$.

So for both the initialization and the iteration phase of the algorithm we found a bound of $O(nm)$ on the time complexity. □

The space complexity of Algorithm 3 is $\Theta(n^2)$ as we need to compute and store the matrices Q and B.

4 Avoiding Matrix Multiplication

A possible disadvantage of recognition algorithms based on matrix multiplication is the amount of space required to store the result of the matrix multiplication. Even if an original sparse matrix M is stored efficiently using $\Theta(m)$ space, the result of the multiplication may be a dense matrix requiring $\Theta(n^2)$ space (see Fig. 2). Avoiding matrix multiplication thus seems to be required in order to improve the space complexity. To do this, we started over from the algorithm originally presented by Golumbic and Goss for the recognition of perfect elimination bipartite graphs. Algorithm 1 proceeds in up to n iterations. In every iteration, every edge is checked against possibly all other edges to determine if it is bisimplicial. To check an edge uv for bisimplicity, we need to verify that $G[M]$ contains all edges $u'v'$ with $u' \in \Gamma(v)$ and $v' \in \Gamma(u)$. By performing this every iteration, we obtain a time complexity of $O(n^5)$.

The idea behind our new algorithm is as follows: in Algorithm 1 we check every remaining edge uv against possibly all other edges during every iteration. However, we can shave a factor n from the time complexity if we are checking uv and find an edge $u'v'$ as present in $G[M]$ during some iteration, we avoid checking it for uv again in subsequent iterations. A naive algorithm based on this notion is described in Algorithm 4. Assuming the use of suitable data structures, the time complexity of this algorithm is $O(n^2m)$. Unfortunately, by precomputing for every edge e the set of possible edges E_e that need to be checked, we require a lot more space, instead of less.

Observing the usage of the sets E_e, we see they are all constructed at the beginning and processed one element at a time in arbitrary order. The element under consideration is either removed from the set and followed by another

Algorithm 4 a $O\left(n^2 m\right)$ recognition algorithm

```
1: I ← {1...n}
2: J ← {1...n}
3: ∀e = (i, j) ∈ E : E_e = C_j × R_i
4: while I ≠ ∅ do
5:    f ← false
6:    for all e = (i, j) ∈ E do
7:       g ← true
8:       if i ∉ I ∨ j ∉ J then
9:          g ← false
10:      while (E_e ≠ ∅) ∧ (g = true) do
11:         e' = (i', j') ← arbitrary_element (E_e)
12:         if i' ∉ I ∨ j' ∉ J then
13:            E_e ← E_e \ e'
14:         else if M_{i',j'} = 1 then
15:            E_e ← E_e \ e'
16:         else
17:            g ← false
18:      if g = true then
19:         f ← true, x ← i, y ← j
20:   if f = false then
21:      return false {G[M] is not perfect elimination bipartite}
22:   I ← I \ x
23:   J ← J \ y
24: return true {G[M] is perfect elimination bipartite}
```

element, or it leads to the conclusion that e is not bisimplicial in the matrix that remains in the current iteration and it will be considered again later. If we impose a specific order on the processing of the edges $e \in E_e$, we can do away with precomputing and storing the entire sets E_e and only store the element e' currently under consideration for each edge e.

To implement this, we again represent M using the sets R_i and C_j, but this time we store them as sorted lists, as shown in Fig. 3(a). To perform a pivot and remove the associated row and column, we simply adjust the links in the row and column lists to skip over the removed row and column, as shown in Fig. 3(b). Clearly, such a pivot operation can be implemented in time $O\left(m\right)$, as we can simply pass over all the elements in each of the lists and adjust the links as we pass them. This representation requires $\Theta\left(m\right)$ space.

To check if an element $M_{i,j}$ corresponds to a bisimplicial edge in $G[M]$, we have to test if all edges between the neighbors of its endpoints exist. In terms of the column sets of the matrix M, this means that for every column $k \in R_i$, we must have that $C_j \subseteq C_k$. If we use the sorted list representation, the number of comparisons for each edge e is bounded by $O\left(m\right)$. Every comparison has one of three possible outcomes (see Fig. 4):

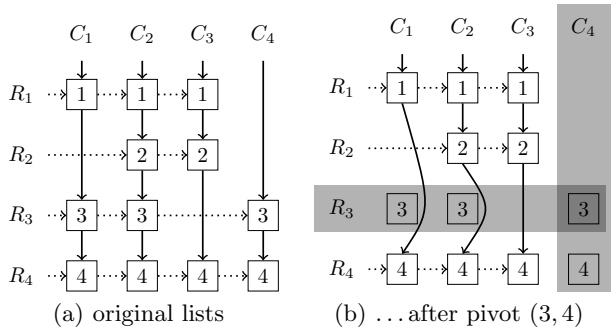

Fig. 3. row and column lists for example matrix M from Fig. 1(a)

1. C_j and C_k both contain the row number: the required edge is present, we can continue checking the next row number
2. C_k contains a number not present in C_j: an additional edge is present, we can continue checking the next row number
3. C_j contains a number not present in C_k: a required edge is missing: e is not bisimplicial in the current matrix

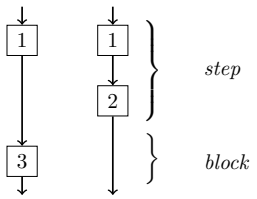

Fig. 4. steps and blocks

We call the first two cases 'steps' (as they can be repeated during a single iteration) and call the third case a 'block' (as it ends the checks for e during this iteration). For a single edge e, steps can occur $O(m)$ times during the algorithm, whereas blocks are limited by $O(n)$ as they can occur only once per iteration. If there are no more comparisons left for any edge e that still remains in M at some point during the algorithm, we have found a suitable pivot. After removing the pivot row and column from M, we simply proceed checking the remaining edges starting at the point where they blocked during the previous iteration. We continue this process until either we have found a complete elimination scheme or we cannot find a bisimplicial edge anymore. This procedure is described in Algorithm 5.

Theorem 4. *The time complexity of Algorithm 5 is* $O(m^2)$.

Algorithm 5 a new $O\left(m^2\right)$ recognition algorithm using $\Theta\left(m\right)$ space

```
1: I ← {1...n}
2: J ← {1...n}
3: Construct R_i and C_j representation
4: while I ≠ ∅ do
5:    f ← false
6:    for all e = (i, j) ∈ E do
7:       g ← true
8:       if i ∉ I ∨ j ∉ J then
9:          g ← false
10:      while g = true and we are not done checking edges do
11:         e' = (i', j') ← the current edge to check
12:         if i' ∉ I ∨ j' ∉ J then
13:            Proceed to the next edge to check (if any) {This can only happen during
               the first iteration of this inner loop}
14:         else if e' blocks then
15:            g ← false
16:         else
17:            Proceed to the next edge to check (if any)
18:      if g = true then
19:         f ← true, x ← i, y ← j
20:   if f = false then
21:      return false {G[M] is not perfect elimination bipartite}
22:   Update R_i and C_j links to perform pivot round (x, y)
23:   I ← I \ x
24:   J ← J \ y
25: return true {G[M] is perfect elimination bipartite}
```

Proof. It is possible to construct our sorted list representation in time $O\left(n^2\right)$. Other initialization, such as the state of the comparisons for every edge e, can easily be done within the same time. Following the initialization, we perform up to n iterations, during each of which we perform a pivot on our rows and columns lists in time $O\left(m\right)$ for a total of $O\left(nm\right)$. During the entire algorithm we perform $O\left(m\right)$ 'steps' and $O\left(n\right)$ 'blocks' for each of the m edges, leading to an overall time complexity of $O\left(m^2\right)$. □

Theorem 5. *The space complexity of Algorithm 5 is* $\Theta\left(m\right)$.

Proof. Our sorted lists representation of M contains m edges. For each edge we need $\Theta\left(1\right)$ space to store its progress with respect to its comparisons against its required neighbors for a total of $\Theta\left(m\right)$. Finally, we have to store the sets I and J to keep track of the rows and columns that still remain, both require $\Theta\left(n\right)$ space. In total, we thus obtain a space complexity of $\Theta\left(m\right)$. □

After establishing its running time and space requirements, we end this section by adapting our new algorithm to the special case of finding a perfect elimination ordering allowing only pivots on the diagonal of the matrix. Rose

and Tarjan have studied this problem and have presented two algorithms for it also focussing on the trade-off between time and space requirements [5]. One of their algorithms has a time complexity of $O(nm)$ and uses $\Theta(nm)$ space, the other one has a time complexity of $O(n^2m)$ but uses only $\Theta(m)$ space.

It is not hard to see that our algorithm can be adapted to consider only a subset of all the edges as pivots: this simply means we only process steps and blocks for these edges while ignoring the other edges. If we test only c edges as allowed pivots in this way, the running time of our algorithm is $O(cm + nm)$ while the space complexity remains $\Theta(m)$. By only allowing pivots on the diagonal ($c = n$) instead of anywhere ($c = m$), we get a time complexity of $O(nm)$ for this restricted case. We thus obtain a single algorithm that combines the best time complexity of Rose and Tarjan with their best space complexity for this restricted problem.

5 Discussion

In the previous sections, we have presented two new algorithms for the recognition of perfect elimination bipartite graphs. Both are aimed at efficient recognition of sparse instances, the trade-off between the two is in the amount of space required, respectively $\Theta(n^2)$ and $\Theta(m)$.

Besides improving time and space complexity, another interesting aspect of algorithmic performance is the possibility of parallelization. From the algorithm of Goh and Rotem, it is not too hard to see that finding a single bisimplicial edge can be done in polylog time given a polynomial number of processors: matrix multiplication can be performed in polylog time [10] as well as the post-processing to determine the values of ℓ_i and check the individual matrix elements for bisimpliciality. Our new algorithm can be parallelized on $O(m^2)$ processors to find a bisimplicial edge in polylog time as all checks for all edges can be performed in parallel and subsequently combined in polylog time to find a bisimplicial edge if one exists. It is however unclear if it is also possible to use a polynomial number of processors to run the entire recognition process in polylog time: all currently known recognition algorithms are based on finding an elimination sequence of n bisimplicial edges and this appears to be an inherently sequential process. A fundamentally different approach might be necessary in order to achieve more parallelism and obtain a polylog time approach for the entire recognition process.

Another subject for further investigation is that of minimizing fill-in when it cannot be avoided completely. For symmetric positive definite matrices with pivots chosen along the main diagonal, minimizing the fill-in in the associated chordal graphs has been shown to be NP-hard [11]. Furthermore, for fill-in in chordal graphs an approximation algorithm has been developed [12]. As far as we know, the complexity of minimizing fill-in for general matrices and perfect elimination bipartite graphs is unknown. Considering the practical applications of minimum elimination orderings, obtaining results on the complexity in the general case, as well as either a polynomial time algorithm or an approximation algorithm for minimizing fill-in seem to be good topics for further research.

6 Conclusions

In current literature, the fastest known algorithm for the recognition of general perfect elimination bipartite graphs is the algorithm by Spinrad in [8] with a time complexity of $O\left(n^3/\log n\right)$. We have presented two new algorithms focussed specifically on sparse instances. Our first algorithm is an adaption of the algorithm by Goh and Rotem with a time complexity of $O\left(nm\right)$, leading to an improvement for instances with $m = o\left(n^2/\log n\right)$ (all but the densest instances). The second algorithm we have presented is not based on some form of matrix multiplication and is as such able to do away with the $\Omega\left(n^2\right)$ space complexity associated with it. This algorithm has a time complexity of $O\left(m^2\right)$ and a space complexity of just $\Theta\left(m\right)$. For instances with $m = o\left(n\sqrt{n\log n}\right)$ this algorithm is faster than the algorithm by Spinrad while requiring less space. We have also shown how the restricted problem where only pivots on the diagonal are allowed can be solved in time $O\left(nm\right)$ using an adapted version of our algorithm.

Interesting subjects for further study might be algorithms that parallelize better as well as problems related to the minimum fill-in on general square matrices, such as its complexity, (exact) algorithms and approximations.

References

1. D. Rose. A graph-theoretic study of the numerical solution of sparse positive definite systems of linear equations. In Ronald Read, editor, *Graph Theory and Computing*, pages 183–217. Academic Press, New York, 1972.
2. Loren Haskins and Donald J. Rose. Toward characterization of perfect elimination digraphs. *SIAM J. Computing*, 2(4):217–224, 1973.
3. D. J. Kleitman. A note on perfect elimination digraphs. *SIAM J. Computing*, 3(4):280–282, 1974.
4. Donald J. Rose, R. Endre Tarjan, and George S. Lueker. Algorithmic aspects of vertex elimination on graphs. *SIAM J. Computing*, 5(2):266–283, 1976.
5. Donald J. Rose and Robert Endre Tarjan. Algorithmic aspects of vertex elimination on directed graphs. *SIAM Journal on Applied Mathematics*, 34(1):176–197, 1978.
6. Martin Charles Golumbic and Clinton F. Goss. Perfect elimination and chordal bipartite graphs. *J. Graph Theory*, 2(2):155–163, 1978.
7. L. Goh and D. Rotem. Recognition of perfect elimination bipartite graphs. *Inform. Process. Lett.*, 15(4):179–182, 1982.
8. Jeremy P. Spinrad. Recognizing quasi-triangulated graphs. *Discrete Appl. Math.*, 138(1-2):203–213, 2004.
9. Don Coppersmith and Shmuel Winograd. Matrix multiplication via arithmetic progressions. *J. Symbolic Comput.*, 9(3):251–280, 1990.
10. Christos H. Papadimitriou. *Computational Complexity*. Addison-Wesley Publishing Company, Inc., Reading, MA, 1994.
11. Mihalis Yannakakis. Computing the minimum fill-in is NP-complete. *SIAM J. Alg. Disc. Meth.*, 2(1):77–79, 1981.
12. Assaf Natanzon, Ron Shamir, and Roded Sharan. A polynomial approximation algorithm for the minimum fill-in problem. In *STOC '98: Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 41–47, New York, NY, USA, 1998. ACM.