

Recognizing Series-Parallel Matrices in Linear Time

Matthias Walter*¹

¹Department of Applied Mathematics, University of Twente, Enschede, The Netherlands

November 16, 2021

Abstract

A series-parallel matrix is a binary matrix that can be obtained from an empty matrix by successively adjoining rows or columns that are parallel to an existing row/column or have at most one 1-entry. Equivalently, series-parallel matrices are representation matrices of graphic matroids of series-parallel graphs, which can be recognized in linear time. We propose an algorithm that, for an m -by- n matrix A with k nonzeros, determines in expected $\mathcal{O}(m + n + k)$ time whether A is series-parallel, or returns a minimal non-series-parallel submatrix of A . We complement the developed algorithm by an efficient implementation and report about computational results.

1 Introduction

We consider binary matrices $A \in \{0, 1\}^{m \times n}$ and the matroids represented by those (see Truemper's book [5] for relevant matroid concepts). *Series-parallel matroids* are those represented by *series-parallel matrices*, which are defined recursively: every binary m -by- n matrix with $m, n \leq 1$ is series-parallel. For $m \geq 2$ or $n \geq 2$, A is series-parallel if and only if it can be obtained from another series-parallel matrix A' by adjoining a row vector (resp. column vector) that is a copy of a row (resp. column) vector of A' or is a unit or zero vector. The removal of such a row or column is called an *SP-reduction*, more precisely a *copy reduction*, *unit reduction* or *zero reduction*, respectively. In other words, a matrix is series-parallel if there is a sequence of SP-reductions that yields the empty (i.e., 0-by-0) matrix. Matrices for which no SP-reduction is possible are called *SP-reduced*. The main problem of interest is the following.

Problem 1. *Determine whether a given binary matrix is series-parallel.*

A related concept is that of a *series-parallel graph*, which is a graph that can be obtained from the graph with one node and a loop edge by iteratively duplicating an edge or subdividing an edge by a new node. It is easy to see that there is a one-to-one correspondence between the series-parallel graphs and the series-parallel matroids (see Chapter 4.3 in Truemper's book [4] for the 2-connected case). Here, a *copy reduction* (resp. *unit reduction*) of a row corresponds to a contraction of an edge that is in series with (resp. parallel to) another one. Conversely, a *copy reduction* (resp. *unit reduction*) of a column corresponds to the deletion of an edge that is parallel to (resp. in series with) another one. Finally, a *zero reduction* of a row corresponds to the contraction of a cut edge, while a *zero reduction* of a column corresponds to the deletion of a loop.

The recognition problem for series-parallel graphs can be solved in linear time [7]. This immediately yields an almost-linear-time algorithm for Problem 1 by computing a graph G represented by A (see [2, 1]) and then testing whether G is series-parallel.

An important task for solving Problem 1 in linear time is to determine a maximal sequence of SP-reductions that can be applied to a given matrix. This is a useful preprocessing step for every recognition problem for matrix classes that are closed under adjoining zero or unit vectors or copies of existing rows/columns. An example of such a matrix class is that of totally unimodular matrices (see Chapters 19 and 20 in Schrijver's book [3]). In particular, it contributes to improvements for the state-of-the-art implementation of a total unimodularity test [8].

Contribution and outline. Our main contribution is a direct linear-time algorithm. It first finds an inclusion-wise maximal sequence of SP-reductions (see Section 2). In Section 3 we describe an algorithm that computes for a non-series-parallel matrix in linear time a minimal submatrix with the same property. The short Section 4 is about the extension to ternary matrices, i.e., those with entries in $\{-1, 0, +1\}$. In Section 5 we describe our implementation of the algorithm and report about computational results. We conclude the paper by a short discussion the induced hereditary matrix class.

*m.walter@utwente.nl

2 Recognizing series-parallel matrices

The definition of series-parallel matrices is symmetric with respect to rows and columns. Hence, we call their (disjoint) union A 's *elements* and denote them by E . For an element $e \in E$, $A(e)$ denotes the row vector $A_{r,\star}$ if e is row r , while it denotes the column vector $A_{\star,c}$ if e is column c .

We will present an algorithm that sequentially removes elements from the input matrix A until it is SP-reduced. It is easy to see that if an SP-reduction for element $e \in E$ is possible, and another one for $e' \in E$ is carried out, then an SP-reduction for e will also be possible for the reduced matrix. This shows that the order of removal does not matter, and hence A is series-parallel if and only if the SP-reduction procedure terminates with an empty matrix. Due to the simplicity of this algorithm, the only challenge lies in the running time.

2.1 Data structures

In order to achieve its running time, our algorithm relies on a couple of data structures. First, in order to efficiently carry out a sequence of SP-reductions, we store the nonzeros of A in a grid of doubly-linked lists. More precisely, for each nonzero we store pointers to the previous and next nonzeros in the same row and to those in the same column, respectively. We assume that the input matrix A is given in a form that allows the creation of this data structure in linear time. For instance, this is the case if the nonzeros are given as a list that is ordered lexicographically by rows and columns. Moreover, since an SP-reduction would formally cause re-numbering of rows or columns, we actually replace nonzero entries by zeros.

Second, for each element $e \in E$, we store the number of nonzeros of $A(e)$, denoted by $|A(e)|_1$ for convenience. This allows us to identify zero or unit reductions in constant time.

Third, we maintain a queue \mathcal{Q} that contains all candidate elements for SP-reductions. The main iteration of the algorithm consists in finding out whether an element extracted from \mathcal{Q} admits an SP-reduction. If this is the case, the reduction will be carried out, which may imply the addition of other elements to the queue.

Fourth, a hash table \mathcal{H} is used in order to identify copy reductions in constant time. The corresponding hash function $h : E \rightarrow \mathbb{Z}$ shall depend on $A(e)$ only. Moreover, we frequently update the hash value of an element $e \in E$ after a nonzero entry has been removed, which means that after one entry of $A(e)$ is modified, re-computing $h(e)$ shall be done in constant time.

We present such a hash function h that requires randomization. Let $(p, q) \in \mathbb{N}^n \times \mathbb{N}^m$ be a vector obtained by rounding a vector randomly chosen from a sphere in \mathbb{R}^{m+n} of sufficiently large radius R , intersected with the first orthant. We define

$$h(e) := \begin{cases} p^\top A(e)^\top & \text{if } e \text{ is a row element,} \\ q^\top A(e) & \text{if } e \text{ is a column element.} \end{cases} \quad (1)$$

Notice that for row elements $e \in E$, $A(e)$ is a row vector, while it is a column vector for column elements. By the choice of p and q , $h(e)$ is almost-surely collision-free for large radius R . Moreover, if a 1-entry of $A(e)$ is turned into a 0-entry, $h(e)$ decreases by a corresponding entry of p or q . Hence, the hash value of an element can be updated in constant time.

2.2 Reduction algorithm

With these data structures at hand, we can now state our recognition algorithm.

Algorithm 1: Finding a maximal sequence of SP-reductions.

Input: Matrix $A \in \mathbb{Z}^{m \times n}$

Output: Maximal sequence of SP-reductions

```

1 Initialize list representation of  $A$ .
2 Initialize empty hash table  $\mathcal{H}$  for keys  $e \in E$  and values  $A(e)$ .
3 Initialize queue  $\mathcal{Q}$  with all  $e \in E$ .
4 Initialize the set  $\mathcal{R} := \emptyset$  of recorded SP-reductions.
5 Compute  $h(e)$  for all  $e \in E$ .
6 while  $\mathcal{Q}$  is not empty do
7   | Extract element  $e$  from  $\mathcal{Q}$ .
8   | if  $|A(e)|_1 = 0$  then
9     |   Add  $e$  to  $\mathcal{R}$  and mark it as zero reduction.
10  | else if  $|A(e)|_1 = 1$  then
11  |   | Add  $e$  to  $\mathcal{R}$  and mark it as unit reduction.
12  |   | Let  $f \in E$  be such that  $\{e, f\}$  are row and column indices of the 1-entry of  $A(e)$ .
13  |   | Remove nonzero  $\{e, f\}$  from  $A$ , add  $f$  to  $\mathcal{Q}$  if necessary, update hash value of  $f$ , and remove  $f$ 
14  |   | from  $\mathcal{H}$  if necessary.
15  | else
16  |   | Check via  $\mathcal{H}$  whether there is an element  $e' \in E$  such that  $A(e) = A(e')$ .
17  |   | if  $\mathcal{H}$  contains element  $e' \in E$ :  $A(e) = A(e')$  then
18  |   |   | Add  $e$  to  $\mathcal{R}$  and mark it as copy reduction for  $e'$ .
19  |   |   | for each  $f$  such that  $A(e)_f = 1$  do
20  |   |   |   | Remove nonzero  $\{e, f\}$  from  $A$ , add  $f$  to  $\mathcal{Q}$  if necessary, update hash value of  $f$ , and
21  |   |   |   | remove  $f$  from  $\mathcal{H}$  if necessary.
22  |   |   | end
23  |   | else
24  |   |   | Add  $e$  to  $\mathcal{H}$ .
25  |   | end
26 end
27 return  $\mathcal{R}$ 

```

Theorem 2. For input matrices $A \in \{0, 1\}^{m \times n}$ with k nonzeros, Algorithm 1 finds a maximal sequence of SP-reductions in expected $\mathcal{O}(m + n + k)$ time.

Proof. We first show that the algorithm actually finds a maximal sequence of SP-reductions. It is easy to see that all modifications of A correctly reflect the recorded SP-reductions. We claim that the following invariants are satisfied for all elements $e \in E$ throughout the algorithm:

- (i) either $e \in \mathcal{Q}$ or $e \in \mathcal{H}$ or $e \in \mathcal{R}$;
- (ii) if an SP-reduction for e is possible for A , then $e' \in \mathcal{Q}$ holds for some element with $A(e') = A(e)$.

Since we initialize \mathcal{Q} as E , both statements are satisfied. Now consider an iteration of the main loop in which element $e \in E$ was extracted from \mathcal{Q} . Either e is added to \mathcal{R} in line 9, line 11 or line 17 or e is added to \mathcal{H} in line 22. Moreover, if the SP-reduction causes the removal of nonzeros $\{e, f\}$ in line 13 or line 19, then it is ensured that $f \in \mathcal{Q}$ and $f \notin \mathcal{H}$ hold. This establishes invariant (i).

Consider, for the sake of contradiction, a first iteration after which invariant (ii) is violated, i.e., an SP-reduction for \hat{e} is possible, but no $e' \in E$ with $A(e') = A(\hat{e})$ is in the queue. Moreover, for each such e' we have $e' \notin \mathcal{R}$ and thus $e' \in \mathcal{H}$ by the invariant (i). If \hat{e} was extracted from \mathcal{Q} in this iteration, i.e., $\hat{e} = e$ holds, then we must have added \hat{e} to \mathcal{H} in line 22. In particular, the SP-reduction must be a copy reduction for some other element $e' \in E$. Since we argued that $e' \in \mathcal{H}$ holds, we obtain a contradiction to the fact that we reached line 22. Otherwise, \hat{e} must have become SP-reducible, i.e., $\hat{e} = f$ holds for some element $f \in E$ for which $\{e, f\}$ is a nonzero of A . However, in the corresponding lines 13 and 19, such elements f are added to \mathcal{Q} , which yields a contradiction. We conclude that also invariant (ii) holds.

The total number of iterations is bounded by $m + n + k$ since $|\mathcal{Q}| = |E| = m + n$ holds initially, and since further additions to \mathcal{Q} happen at most once per (removed) nonzero. This shows that the algorithm terminates, and by invariants (i) and (ii) with maximal \mathcal{R} . It also shows that lines 7, 9, 11, 13, 15, 17 and 22 are each executed at most $m + n + k$ times. Clearly, lines 17 and 19 are executed at most k times since each time a nonzero is removed from A . Due to the data structures and the properties of the hash function, each of these lines can be executed in constant time, where this holds for lines 13 and 19 almost surely. We conclude that the overall running time is linear in $m + n + k$ in expectation. \square

3 Certifying non-series-parallel matrices

For $\ell \geq 3$ we denote by W_ℓ the *wheel graph with ℓ spokes*, i.e., a cycle of length ℓ plus one node that is connected to all other nodes via an edge. Two of its representation matrices are of interest to us, namely

$$M_\ell := \begin{pmatrix} 1 & 0 & 0 & 0 & \cdots & 0 & 1 \\ 1 & 1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 1 & 1 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 1 & 1 & \ddots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & 1 & 0 \\ 0 & 0 & 0 & 0 & \cdots & 1 & 1 \end{pmatrix}, \quad M'_\ell := \begin{pmatrix} 1 & 1 & 0 & 0 & \cdots & 0 & 1 \\ 1 & 1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 1 & 1 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 1 & 1 & \ddots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & 1 & 0 \\ 0 & 0 & 0 & 0 & \cdots & 1 & 1 \end{pmatrix},$$

which we call *wheel matrices*. Notice that M_ℓ and M'_ℓ differ only in the entry in the first row and second column. It is easy to see that both matrices are SP-reduced. We say that a matrix *contains a wheel-submatrix* if it contains a wheel matrix as a submatrix, possibly after permuting rows or columns. We will show that the following holds.

Theorem 3. *A binary matrix is either series-parallel or it contains a wheel-submatrix.*

The proof is omitted as it will follow from the correctness of our algorithm that returns one of these matrices as a submatrix when confronted with a matrix that is not series-parallel (see Theorem 6). Using matroid operations, the certificate can be simplified even further. The following corollary is a strengthening of Theorem 4.2.13 in Truemper's book [5], where only the case of connected matroids is discussed.

Corollary 4. *A binary matroid is either series-parallel or contains the graphic matroid of W_3 as a minor.*

Proof. The result follows from Theorem 3 by observing that a graphic matroid of W_ℓ with $\ell \geq 3$ contains the graphic matroid of W_3 as a minor. In fact, a binary pivot operation on the entry in the second row and first column of M_ℓ yields M'_ℓ , and M'_ℓ contains $M_{\ell-1}$ as a submatrix. Repeated application yields M_3 after $\ell - 3$ pivots. \square

3.1 Bipartite graph

A binary matrix $A \in \{0, 1\}^{m \times n}$ gives rise to a *bipartite graph*, denoted by $\text{BG}(A)$, which has m nodes on one side R and n nodes on the other side C of the bipartition and those edges $\{r, c\}$ (with $r \in R, c \in C$) for which $A_{r,c} = 1$. It is easy to see that $\text{BG}(M_\ell)$ is a chordless cycle of length 2ℓ . Hence, the basic idea of our recognition algorithm is to first apply Algorithm 1 and then to find a chordless cycle in $\text{BG}(A)$ of length at least 6 in the SP-reduced matrix A . Chordless cycles can be found using breadth-first search. However, it may turn out that all found cycles have length 4, which corresponds to a 2-by-2 matrix with only 1's.

If this is the case, we can use the following approach [4]. Grow the submatrix consisting of 1's to an inclusion-wise maximal one, indexed by rows X and columns Y . Search for a shortest path P from X to Y in $\text{BG}(A)$ without using an edge corresponding to a 1 in this submatrix. If P exists, then the submatrix induced by P and one additional row and column is of type M'_ℓ (the top-left 2-by-2 submatrix of M'_ℓ is part of the all-1's submatrix). However, as can be seen in Figure 1 this need not always be the case.

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 0 & \color{red}1 & 0 & 0 & \color{red}1 \\ 1 & 1 & 1 & 1 & 0 & \color{red}1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \color{red}1 & \color{red}1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \color{red}1 & \color{red}1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \color{red}1 & \color{red}1 \\ 0 & 0 & 0 & 0 & 0 & 0 & \color{red}1 & 0 & \color{red}1 \\ \color{blue}1 & \color{blue}1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \color{blue}1 & \color{blue}1 & 0 & \color{blue}1 & 0 & 0 & 0 & 0 \\ \color{blue}1 & \color{blue}1 & 0 & 0 & \color{blue}1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Figure 1: An SP-reduced matrix A with inclusion-wise maximal all-1's submatrix in the upper left part. Every path in $\text{BG}(A)$ from the first three rows X to the first four columns Y must use an edge from this submatrix. The edges corresponding to the part reachable from X are colored red, while those reachable from Y are colored blue.

3.2 Separations

In case such a path does not exist, we have found a *2-separation* of A , which is a partitioning of A 's rows into X^1 and X^2 and A 's columns into Y^1 and Y^2 such that $\text{rank}(A_{X^1, Y^2}) + \text{rank}(A_{X^2, Y^1}) = 1$ and $|X^i| + |Y^i| \geq 2$ holds for $i = 1, 2$. After reordering of rows and columns, A looks as in Figure 2. Such a separation induces a *2-sum decomposition* of A into A^1 and A^2 , which corresponds to a 2-sum decomposition involving the corresponding represented matroids. However, we will work only on the matrix level and never exploit any matroid structure. The 2-separation for the example in Figure 1 can be readily seen: the rank-1 submatrix is the 7-by-5 submatrix in the upper left, B is the (smallest) submatrix containing all red 1's, while C is the (smallest) submatrix containing all blue 1's.

$$A = \begin{array}{|c|c|} \hline B & bc^\top \\ \hline \mathbb{O} & C \\ \hline \end{array} \quad A^1 = \begin{array}{|c|c|} \hline B & b \\ \hline \end{array} \quad A^2 = \begin{array}{|c|} \hline c^\top \\ \hline C \\ \hline \end{array}$$

Figure 2: Partitioned version of a 2-separable matrix A that is decomposed into a 2-sum of A^1 and A^2 at the last column and first row, respectively. Note that it is required that B and C each have at least 2 elements and that b and c are nonzero, i.e., bc^\top has rank 1.

One may be tempted to recursively search for a wheel submatrix in both parts of the 2-separation. However, this may lead to an increased running time, and it turns out that both parts will contain such a submatrix.

Lemma 5. *For a 2-sum decomposition of an SP-reduced matrix $A \in \{0, 1\}^{m \times n}$ as in Figure 2, neither of submatrices A^1 and A^2 is series-parallel.*

Proof. By symmetry it suffices to prove the statement for A^1 . For the sake of contradiction, assume that $A^1 = [B \mid b]$ is series-parallel and that among all 2-separations of matrices A with the same number of elements, A^1 has a minimum number of elements. Note that A^1 has at least 3 rows and 3 columns since otherwise there would be an SP-reduction applicable to A . Because A is SP-reduced, the only SP-reductions applicable to $[B \mid b]$ can be column reductions that involve b or a unit row reduction with a 1-entry in b . We distinguish the possible reductions.

Case 1: b is parallel to a column c of B . We can remove the column c from B and attach it to C . Then the lower-left submatrix (in Figure 2) remains a zero matrix and the upper-right matrix remains a rank-1 submatrix, just with one more column than in the given 2-separation. This yields another 2-separation, which violates our assumption on the size of A^1 .

Case 2: b is a unit vector. We can remove the row r in which b has the unique 1-entry from B and attach it to C . This turns the upper-right submatrix into a zero matrix and the lower-left one into a rank-1 submatrix with nonzeros only the row r . This yields a 2-separation of $\bar{A} = A^\top$ into \bar{A}^1 and \bar{A}^2 such that \bar{A}^1 has one element less than A^1 , which contradicts our assumption on A and the 2-separation.

Case 3: for some row r' , $b_{r'} = 1$ and $B_{r', *}$ is \mathbb{O} hold. We can remove the row r' , which effectively sets $b_{r'} = 0$. Unless Case 1 or Case 2 were already applicable before, they must be applicable now, since otherwise

there is no SP-reduction possible. In both cases, we also attach row r' to C (in addition to column c or row r , respectively).

We conclude that such a matrix A with such a 2-separation cannot exist, which completes the proof. \square

Hence, it suffices to only consider the smaller of the two components A^1 , A^2 for a recursive search. Note that A^1 (or A^2) is not necessarily SP-reduced, and hence we need to apply Algorithm 1 again.

3.3 Wheel search algorithm

The previous discussion leads to the following recursive algorithm for searching a wheel-submatrix.

Algorithm 2: Certifying recognition algorithm for binary series-parallel matrices.

Input: Matrix $A \in \{0, 1\}^{m \times n}$
Output: Either “ A is series-parallel” together with a list of $m + n$ SP-reductions,
or “ A is not series-parallel” together with a wheel-submatrix of A .

```

1 Run Algorithm 1 for  $A$ , obtain  $\mathcal{R}$  and replace  $A$  by the reduced matrix.
2 if  $|\mathcal{R}| = m + n$  then return “ $A$  is series-parallel” together with  $\mathcal{R}$ .
3 else
4   Let  $A'$  be the SP-reduced matrix.
5   Run breadth-first search in  $\text{BG}(A')$  to find a chordless cycle  $C$  of length  $2\ell$  for some  $\ell \in \mathbb{N}$ .
6   if  $C$  exists then
7     Let  $B$  be the submatrix of  $A$  indexed by all rows and columns of  $C$ .
8     if  $\ell \geq 3$  then return “ $A$  is not series-parallel” together with  $B$ .
9     else
10      Grow  $B$  to a maximal submatrix of  $A'$  that contains only 1's and let  $X$  and  $Y$  be the row
11      and column sets of  $B$ , respectively.
12      Define  $A''$  to be  $A'$  with the entries of  $B$  replaced by 0's.
13      Using breadth-first search, search for a (shortest) path  $P$  from  $X$  to  $Y$  in  $\text{BG}(A'')$ .
14      if  $P$  exists then
15        Let  $c$  be the column that comes directly after  $P$ 's starting node from  $X$ .
16        Let  $r' \in X$  be such that  $A_{r',c} = 0$ .
17        Let  $r$  be the row that comes directly before  $P$ 's end node from  $Y$ .
18        Let  $c' \in Y$  be such that  $A_{r,c'} = 0$ .
19        Let  $B'$  be the submatrix indexed by all rows and columns of  $P$ , row  $r'$  and column  $c'$ .
20        return “ $A$  is not series-parallel” together with  $B'$ .
21      else
22        The nodes reachable from  $X$  induce a 2-separation of  $A$  with parts  $A^1$  and  $A^2$ .
23        Let  $i \in \{1, 2\}$  be such that  $A^i$  has the minimum number of elements.
24        return output of recursive call of Algorithm 2 for  $A^i$ .
25      end
26    end
27  else
28    The nodes reachable from the source node of the search induce a 2-separation of  $A$  with parts
29     $A^1$  and  $A^2$ .
30    Let  $i \in \{1, 2\}$  be such that  $A^i$  has the minimum number of elements.
31    return output of recursive call of Algorithm 2 for  $A^i$ .
32  end

```

Theorem 6. *Let $A \in \{0, 1\}^{m \times n}$ have k nonzeros. Then Algorithm 2 determines in expected $\mathcal{O}(m + n + k)$ time whether A is series-parallel and if not, finds a wheel-submatrix of A .*

Proof. We first discuss the correctness of the algorithm and then turn to its running time.

After line 1, A is SP-reduced, which implies that $\text{BG}(A)$ is not a forest. Hence, the breadth-first search in line 5 finds a chordless cycle of length at least 4. As explained in Section 3.1, a chordless cycle of length at least 6 in $\text{BG}(A)$ corresponds to a submatrix M_ℓ for $\ell \geq 3$, which is returned in line 8 if such a cycle is found. Otherwise, path P is searched for in line 12.

Suppose, P exists. By maximality of the submatrix B , in line 14, $A_{X,c}$ is not the all-1's vector, and hence, r' is well defined. Similarly, c' from line 17 is well defined. We claim that the matrix B' is of the form M'_ℓ . To see this, observe that among the rows X , exactly two belong to B' , one with $A_{r',c} = 0$ and one with a 1-entry in column c . Similarly, exactly two of the columns Y belong to B' , one with $A_{r,c'} = 0$ and one with a 1-entry in row r . The fact that P is a shortest X - Y -path ensures that B' has all other required 0/1-entries.

If P does not exist, the 2-separation of A is easily verified. Lemma 5 ensures that we can restrict our search to any of the parts A^1, A^2 . Clearly, the recursion will terminate since A^i in line 23 or line 29 has fewer elements than A .

We now prove that the algorithm runs in expected linear time. First, note that after applying Algorithm 1 for the first time, zero rows/columns are removed, and hence $m, n \leq k$ holds. Hence, all lines except for the recursive calls in lines 23 and 29 can be carried out in time $\mathcal{O}(k)$. Let k_1 and k_2 be the number of nonzeros of A^1 and A^2 , respectively. We obtain

$$k = |X| \cdot |Y| + (k_1 - |X|) + (k_2 - |Y|).$$

For the smaller of the two, k_i , we obtain

$$2k_i \leq k_1 + k_2 = k - |X| \cdot |Y| + |X| + |Y| = k - (|X| - 1) \cdot (|Y| - 1) + 1,$$

which implies $k_i \leq k/2$ since $|X|, |Y| \geq 2$ holds. The geometric series implies that the total effort is linear in k . \square

A few remarks can be made. First, Algorithm 2 can be modified¹ to return an $M_{\ell-1}$ submatrix of M'_ℓ in case $\ell \geq 4$. The modified algorithm will then either return the submatrix M'_3 or a submatrix M_ℓ for $\ell \geq 3$, all of which are *inclusion-wise minimal* non-series-parallel submatrices. Second, it can be modified to either return the sequence of SP-reductions in case A is determined to be series-parallel, or return a wheel-submatrix otherwise, making it a certifying algorithm for the recognition problem.

4 Extension to ternary matrices

Scaling rows or columns of representation matrices by -1 does not change the underlying matroid. While this has no effect for the binary field due to $-1 \equiv +1 \pmod{2}$, it does matter for the ternary field. Our techniques easily extend to this case by allowing to scale rows and columns by -1 . Hence, a ternary series-parallel matrix is constructed from a ternary 1-by-1 matrix by successively adding zero rows/columns, (negated) unit row/column vectors or (negated) copies of existing rows/columns. We call the corresponding reductions *ternary SP-reductions*.

Reduction algorithm. Algorithm 1 naturally extends as well: we can replace our hash function $h : E \rightarrow \mathbb{Z}$ by $h' : E \rightarrow \mathbb{N}$ defined via $h'(e) := |h(e)|$. If there exist elements $e, e' \in E$ with $A(e) = -A(e')$, we will have $h(e) = -h(e')$ and thus $h'(e) = h'(e')$. Hence, we will almost surely have collisions (only) for copies and for negated copies. Clearly, also the check for equality of $A(e)$ and $A(e')$ has to be adapted to also detect negated copies. However, neither of the changes affects the asymptotic runtime. For the remainder of this section we thus consider Algorithm 1 to be modified accordingly.

Certificates. By definition, every SP-reduction for a ternary matrix A induces a corresponding SP-reduction for the (binary) support matrix $\text{supp}(A)$. Hence, if even $\text{supp}(A)$ admits no (binary) SP-reduction, then any corresponding certificate also shows that A is not ternary series-parallel. In other words, for $\ell \geq 3$, the submatrices M_ℓ or M'_ℓ of $\text{supp}(A)$ constitute certificates that can be found in expected linear time.

Let us consider a matrix $A \in \{-1, 0, +1\}^{m \times n}$ that is ternary SP-reduced. If $\text{supp}(A)$ is not binary SP-reduced, then the only possible (binary) SP-reductions for $\text{supp}(A)$ must be copy reductions that do not correspond to ternary copy reductions for A . This implies that A contains (up to scaling of rows/columns) the matrix

$$M_2 := \begin{pmatrix} -1 & 1 \\ 1 & 1 \end{pmatrix}$$

as a submatrix. Clearly, M_2 is not series-parallel. We call a submatrix of A a *signed B*-submatrix if it contains a submatrix that can be turned to B by multiplying rows or columns with -1 .

¹By removing the second row and the first column of a matrix $M_{\ell t}$.

Algorithm 3: Certifying recognition algorithm for ternary series-parallel matrices.

Input: Matrix $A \in \{-1, 0, 1\}^{m \times n}$

Output: Either “ A is series-parallel” together with a list of $m + n$ ternary SP-reductions,
or “ A is not series-parallel” together with a signed wheel- or M_2 -submatrix of A .

```

1 Run modified Algorithm 1 for  $A$  obtaining  $\mathcal{R}$ .
2 if  $|\mathcal{R}| = m + n$  then
3   | return “ $A$  is series-parallel” together with  $\mathcal{R}$ .
4 else
5   | Let  $A'$  arise from  $A$  by applying all SP-reductions  $\mathcal{R}$ .
6   | Run Algorithm 2 for  $\text{supp}(A')$  obtaining either  $\mathcal{R}'$  or submatrix  $B'$  of  $\text{supp}(A')$ .
7   | if  $\text{supp}(A')$  is not series-parallel then
8     |   Let  $B''$  be the signed wheel-submatrix of  $A$  corresponding to  $B'$ .
9     |   return “ $A$  is not series-parallel” together with  $B''$ .
10  | else
11  |   Let  $e, e' \in E$  with  $\text{supp}(A'(e)) = \text{supp}(A'(e'))$  be the elements of the first SP-reduction in  $\mathcal{R}'$ .
12  |   Let  $f, f' \in E$  be such that  $A'(e)_f = A'(e')_f \in \{-1, +1\}$  and  $A'(e)_{f'} = -A'(e')_{f'} \in \{-1, +1\}$ .
13  |   Let  $B''$  be the submatrix of  $A'$  induced by  $e, e', f$  and  $f'$ .
14  |   return “ $A$  is not series-parallel” together with  $B''$ .
15  | end
16 end

```

By the discussion above, correctness of Algorithm 3 follows naturally.

Theorem 7. *Let $A \in \{-1, 0, 1\}^{m \times n}$ have k nonzeros. Then Algorithm 3 determines in expected $\mathcal{O}(m + n + k)$ time whether A is series-parallel and if not, finds a signed wheel- or M_2 -submatrix of A .*

Structure. We obtain the following characterization of ternary series-parallel matrices.

Theorem 8. *A ternary matrix is either series-parallel or it contains a signed wheel- or M_2 -submatrix.*

The matrix M_2 represents the uniform matroid of rank 2 with 4 elements, denoted by U_4^2 , which is the unique forbidden minor for binary matroids [6]. In this light, the following corollary is not surprising.

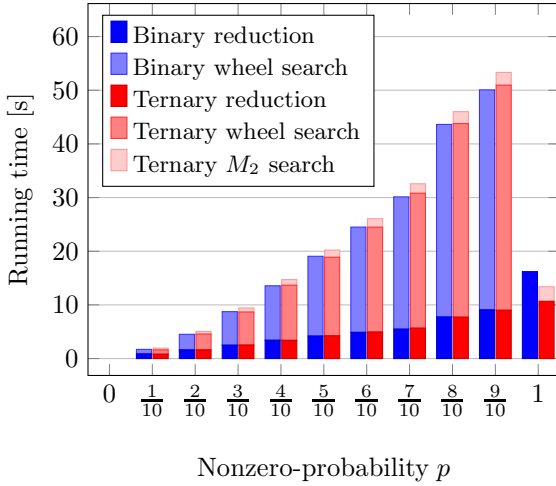
Corollary 9 (Section 4.5 in [5]). *A matroid is either series-parallel or contains the graphic matroid W_3 or the uniform matroid U_4^2 as a minor.*

5 Computational study

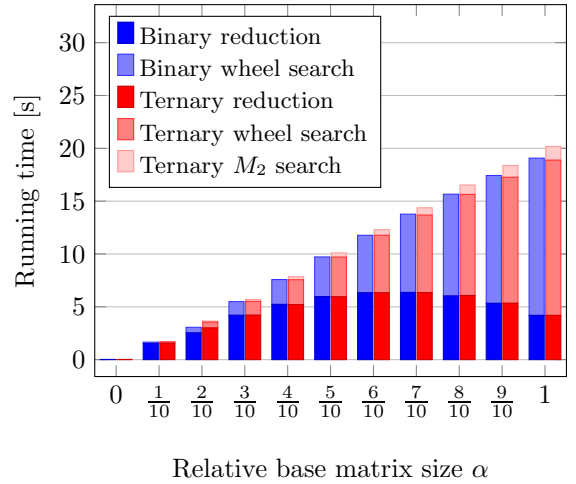
Implementation. We start by describing the changes that we made for our implementation. First, we decouple hashing of rows from hashing of columns by maintaining two hashtables. Second, instead of a random vector $p \in \mathbb{N}^n$ for hashing of columns we use a deterministic one, defined via $p_i := 3^i \bmod 2^{62}$ for $i = 1, 2, \dots, n$. This makes the algorithm deterministic for the price of not running in linear time due to potential hashtable collisions. However, in our experiments we never encountered any collisions. Note that our integer data types would in principle admit the range $[-2^{63}, 2^{63}] \cap \mathbb{Z}$. However, we use one bit less to detect overflows, which allows us to ensure that vectors x and $-x$ receive the same hash value (see Section 4). Hashing of rows is done in an identical way.

The implementation is part of the Combinatorial Matrix Recognition Library [9]. The experiments can be repeated by building with the `cmake` option `-DGENERATORS=on`, compiling via `make`, and then finally calling the Python script `experiments/series_parallel.py 32 100`, where the 32 triggers generation of matrices such that at most 32 GB memory are occupied and the 100 indicates the number of repetitions for each matrix class.

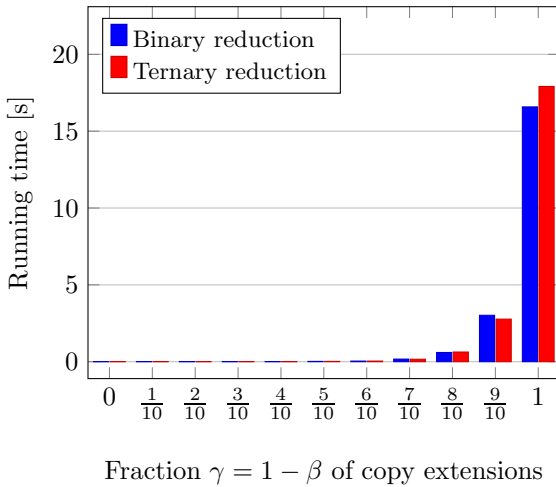
Instances. We tested our code for random binary matrices of size 16384-by-16384 that were generated in a structured way: start with a $16384 \cdot \alpha$ -by- $16384 \cdot \alpha$ -matrix (for $\alpha = 0$ we create a 1-by-1 matrix) whose entries were generated independently uniformly at random with probability p for a nonzero. For ternary matrices, we use $p/2$ for a +1 entry and $p/2$ for a -1 entry. Then, this *base matrix* is extended by adding $16384 \cdot \beta$ unit rows, $16384 \cdot \beta$ unit columns, $16384 \cdot \gamma$ copies of rows and $16384 \cdot \gamma$ copies of columns in random order. For unit vectors the place of the unique 1-entry is chosen uniformly at random. Similarly, for copied vectors the source vector of the copy is chosen uniformly at random among the available ones. We always have $\alpha + \beta + \gamma = 1$.



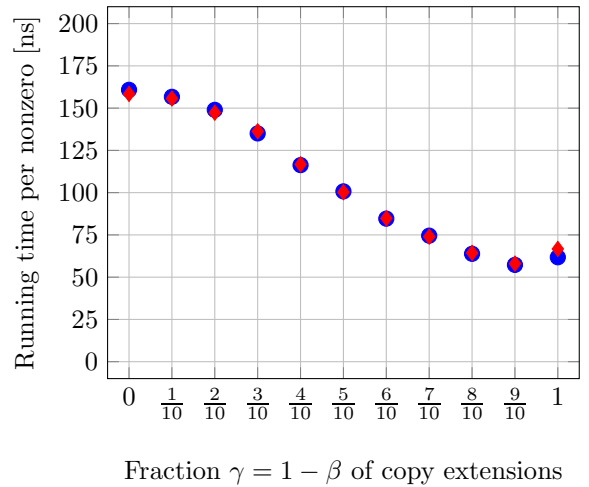
(a) Matrices with $\alpha = 1$, $\beta = 0$, $\gamma = 0$ and different nonzero-probabilities p .



(b) Matrices with $p = 0.5$ and $\beta = \gamma = \frac{1-\alpha}{2}$.



(c) Series-parallel matrices with $\alpha = 0$, $p = 1$ and $\gamma = 1 - \beta$.



(d) Series-parallel matrices with $\alpha = 0$, $p = 1$ and $\gamma = 1 - \beta$.

Figure 3: Running times for different types of randomly generated matrix classes. Results for binary (resp. ternary) matrices are indicated in blue (resp. red).

Results. All benchmarks have been done on a 3.0 GHz Intel Xeon Gold 5217 CPU on a system with 64 GB of RAM memory. In our experiments we generated 100 matrices for every relevant parameter setting and report about averaged results.

Figure 3 shows the running times of the different algorithms for three classes of matrices. These first two matrix classes are typically not series-parallel, except for $p = 1$ in Figure 3a and $\alpha = 0$ for Figure 3b. A first observation is that the reduction (Algorithm 1) typically takes less time than the search for a wheel submatrix (Algorithm 2 without line 1).

The running time increase in Figure 3a is clearly due to the fact that larger p implies more nonzeros which has direct impact on the algorithm. The deviation for $p = 1$ stems from the fact that the matrix has all entries equal to 1 in the binary case (which requires more effort since there are reductions that have to be carried out) and in $\{-1, +1\}$ for the ternary case (for which an M_2 -submatrix is found).

It stands out that the running time for the reduction algorithm is non-monotonic with its maximum near $p = 0.7$. A possible explanation is that matrices with larger α are denser and hence cause more effort, while matrices with α closer to 1 need to carry out fewer SP-reductions since $\beta + \gamma = 1 - \alpha$ is small.

A running time of 50 seconds may seem quite large for such a simple problem. However, it is worth noting that the matrices corresponding to Figure 3a with $p = 0.9$ had about 242 million nonzero entries for which the linked list representation already requires more than 10 GB memory.

In Figure 3c we depict the results for series-parallel matrices with different fractions of unit extensions (resp. copy extensions). Since the running time increase seems to be surprisingly high, we also depict the average running time per nonzero. As Figure 3d shows, this value even decreases with a larger number of copy reductions. This can be explained by the fact that the algorithm spends some time per SP-reduction and some

time per nonzero that is processed for such a reduction. Obviously, a unit reduction requires to process only 1 nonzero.

6 Hereditary matrix classes

All SP-reductions correspond to 1- or 2-separations (see Truemper’s book [5] for a definition of a k -separation for general k) in which the number of rows plus the number of columns of either the matrix B or C is at most 2. Generally, *hereditary* classes of matrices, i.e., those that are closed under taking submatrices are of interest. Figure 1 shows that the class of (binary) matrices that admit a 1- or 2-separation is not hereditary since the matrix in the figure contains the 3-connected matrix M_3 as a submatrix. Hence, a natural question is that for the largest hereditary class of 1- or 2-separable matrices. Clearly, such a class must not contain wheel matrices. Now, Theorem 3 readily implies that every non-empty matrix that does not contain a wheel submatrix must have a 1- or 2-separation, which yields the following corollary.

Corollary 10. *The largest hereditary subclass of binary 1- or 2-separable matrices is that of binary series-parallel matrices.*

We would like to conclude this paper with the question for a result similar to Corollary 10 only for 1-, 2- or 3-separable matrices.

Acknowledgements. This publication is part of the project *Making Mixed-Integer Programming Solvers Smarter and Faster using Network Matrices* (with project number *OCENW.M20.151* of the research programme *NWO Open Competition Domain Science – M* which is (partly) financed by the Dutch Research Council (NWO)).

References

- [1] Robert E. Bixby and Donald K. Wagner. An almost linear-time algorithm for graph realization. *Mathematics of Operations Research*, 13(1):99–123, 1988.
- [2] Satoru Fujishige. An efficient pq-graph algorithm for solving the graph-realization problem. *Journal of Computer and System Sciences*, 21(1):63 – 86, 1980.
- [3] Alexander Schrijver. *Theory of Linear and Integer Programming*. John Wiley & Sons, Inc., New York, NY, USA, 1986.
- [4] Klaus Truemper. A Decomposition Theory for Matroids. V. Testing of Matrix Total Unimodularity. *Journal of Combinatorial Theory, Series B*, 49(2):241–281, 1990.
- [5] Klaus Truemper. *Matroid decomposition (revised edition)*. Academic Press, 1998.
- [6] William T. Tutte. A homotopy theorem for matroids, II. *Transactions of the American Mathematical Society*, 88(1):161–174, 1958.
- [7] Jacobo Valdes, Robert E. Tarjan, and Eugene L. Lawler. The recognition of Series Parallel digraphs. *SIAM Journal on Computing*, 11(2):298–313, 1982.
- [8] Matthias Walter and Klaus Truemper. Implementation of a unimodularity test. *Mathematical Programming Computation*, 5(1):57–73, 2013.
- [9] Matthias Walter and Klaus Truemper. CMR – Combinatorial Matrix Recognition Library, 2021. Documentation: discopt.github.io/cmr, Source code: github.com/discopt/cmr/.