Recognizing Series-Parallel Matrices in Linear Time

Matthias Walter*[1]

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

February 1, 2023

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 copies of 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 the books of Oxley [15] or Truemper [20] for relevant matroid concepts). *Series-parallel matroids* are those represented by *series-parallel matrices*, which are defined recursively, where m and n are nonnegative integers: 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.*

Series-parallel matroids were invented as generalizations of the well-known *series-parallel graphs*. The latter are those graphs 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. Series-parallel graphs naturally arise in electrical networks [9] and are well studied [3]. Moreover, the application of series-parallel extensions (the inverse operations of SP-reductions) to uniform matroids were studied by Chaourar and Oxley [4]. It is well known that there is a one-to-one correspondence between the series-parallel graphs and the series-parallel matroids. A matrix $A \in \mathbb{F}^{m \times n}$ *represents a matroid* over some field $\mathbb{F}$ in the following sense: the ground set E of the matroid is the set of columns of the matrix $[\mathbb{1} \mid A]$, where $\mathbb{1}$ is the identity matrix of order m, and a subset $Y \subseteq E$ is independent if the corresponding columns are linearly independent over $\mathbb{F}$. A binary matrix A *represents a graph* $G = (V, E)$ with respect to a spanning tree $T \subseteq E$ if the rows of A can be indexed by the edges in T, the columns of A correspond to the non-tree edges $E \setminus T$ and $A_{e,f} = 1$ holds if and only if the fundamental cycle $T \cup \{f\}$ contains edge e. If this is the case, a *copy reduction* (resp. *unit reduction*) of a row corresponds to a contraction of a tree edge that is in series with (resp. parallel to) another edge. Conversely, a *copy reduction* (resp. *unit reduction*) of a column corresponds to the deletion of a non-tree edge that is parallel to (resp. in series with) another edge. Finally, a *zero reduction* of a row corresponds to the contraction of a cut edge (i.e., an edge whose removal disconnects its endnodes), while a *zero reduction* of a column corresponds to the deletion of a loop. See Chapter 4.3 in Truemper's book [19] for proofs.

Not every matrix represents a graph, but one can recognize in almost-linear time whether this is the case [10, 2]. Here, *linear* refers to the number k of nonzeros of the given matrix, where assume $k \geq m, n$ throughout the paper. Moreover, almost linear refers to $\mathcal{O}(k \cdot \alpha(k))$, where α denotes the *inverse Ackermann function* [18]. The term "almost linear" is justified since $\alpha(k) \leq 4$ holds for all practically relevant values of k, namely for $k \leq 2^{65536}$. The recognition problem for series-parallel graphs can be solved in linear time [22] (in the number of nodes and edges of the graph). This immediately yields an almost-linear-time algorithm for Problem 1 by computing a graph G represented by A and then testing whether G is series-parallel.

*m.walter@utwente.nl

Contribution. First, our work improves upon the almost linear running time by removing the factor $\alpha(k)$. Second, it yields a much simpler algorithm since the computation of the graph G is quite involved. Third, our extended algorithm finds, for a given non-series-parallel matrix, a forbidden submatrix as a certificate for not begin series-parallel. It is unclear how to obtain such a certificate via the alternative approach sketched above, in particular because the algorithms [10, 2] are not certifying. Of course, one can always construct such a matrix by successively removing rows or columns and running the recognition algorithm again, but this clearly increases the running time from linear to quadratic. Fourth, our algorithm can – in contrast to the previous approach – determine a maximal sequence of SP-reductions that can be applied to a given matrix (see Section 2). 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. Examples of such a matrix classes are totally unimodular [17, Chapters 19 and 20], balanced [1], perfect [16] and ideal [14] matrices. In particular, the algorithm contributes to improvements for the state-of-the-art implementation of a total unimodularity test [23]. The latter is used by researchers in combinatorial optimization and mixed-integer optimization to analyze problem structure (see, e.g., [5, 6, 7, 8, 12]) since presence of total unimodularity indicates tractable problem (sub-)structures. For instance, mixed-integer programming models with transportation or precedence constraints often contain such matrices.

Outline. In Section 2 we describe our main algorithm. In Section 3 we describe an extension 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.

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 $h : E \to \mathbb{Z}$ 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

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

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

Theorem 2. *For input matrices with k nonzeros, Algorithm 1 finds a maximal sequence of SP-reductions in expected $\mathcal{O}(k)$ 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 at the beginning. 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 8, line 10 or line 16 or e is added to $\mathcal{H}$ in line 21. Moreover, if the SP-reduction causes the removal of nonzeros $\{e, f\}$ in line 12 or line 18, 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 21. 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 21. 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 12 and 18, such elements f are added to $\mathcal{Q}$, which yields a contradiction. We conclude that also invariant (ii) holds.

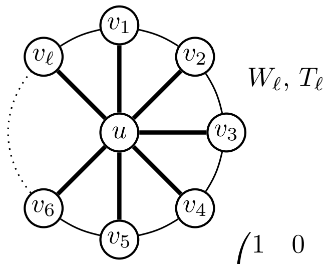
The total number of iterations is bounded by $m + n + k$ since $|\mathcal{Q}| = |E| = m + n$ holds initially, and since further additions to $\mathcal{Q}$ happen at most once per (removed) nonzero. This shows that the algorithm terminates, and by invariants (i) and (ii) with maximal $\mathcal{R}$. It also shows that lines 6, 8, 10, 12, 14, 16 and 21 are each executed at most $m + n + k$ times. Clearly, lines 16 and 18 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 12 and 18 almost surely. We conclude that the overall running time is linear in $m + n + k$ in expectation. $\square$

3 Certifying non-series-parallel matrices

In case a given matrix A is series-parallel, the list of SP-reductions produced by Algorithm 1 yields a certificate, i.e., an easily verifiable reason for begin series-parallel. However, in the case that A is not series-parallel, a user has to trust the correctness of its implementation. Hence, it is desirable to be able to provide a simple reason for this negative outcome as well. The goal of this section is to extend the algorithm as to provide such a certificate in terms of forbidden submatrices. The latter are minimal non-series-parallel matrices, i.e., matrices that are not series-parallel, but for which every proper submatrix is series-parallel.

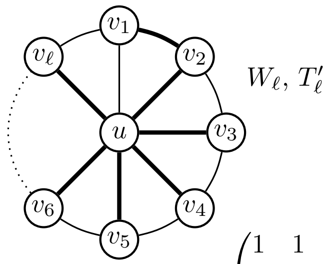
These matrices also appear in the context of the recognition of graphic matroids. A *graphic matroid* is defined by a connected undirected graph $G = (V, E)$ in which a subset $F \subseteq E$ of edges is *independent* if it does not contain a cycle. The *bases* of the matroid are the maximal independent sets, that is, the spanning trees of G. The graph G together with one of its spanning trees T defines a *representation matrix* $M(G, T) \in \{0, 1\}^{T \times (E \setminus T)}$ via $M(G, T)_{e,f} = 1$ holds if and only if the tree edge $e \in T$ lies on the unique cycle in $T \cup \{f\}$.

It is worth to mention that in this context the SP-reductions correspond directly to the graph operations. For instance, the addition of a unit column (indexed by f) with the 1-entry in row e corresponds to the addition of an edge f that is parallel to the tree edge e. Similarly, the addition of a copy f of a row e corresponds to a replacement of e by the a path of length 2, i.e., e and f will be in series.



$$M_\ell \coloneqq M(W_\ell, T_\ell) = \begin{pmatrix} 1 & 0 & 0 & 0 & \cdots & 0 & 1 \\ 1 & 1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 1 & 1 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 1 & 1 & \ddots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & 1 & 0 \\ 0 & 0 & 0 & 0 & \cdots & 1 & 1 \end{pmatrix}$$

(a) Wheel graph W_ℓ with spanning tree T_ℓ that consists of all spoke edges, together with the representation matrix.



$$M'_\ell \coloneqq M(W_\ell, T'_\ell) = \begin{pmatrix} 1 & 1 & 0 & 0 & \cdots & 0 & 1 \\ 1 & 1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 1 & 1 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 1 & 1 & \ddots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & 1 & 0 \\ 0 & 0 & 0 & 0 & \cdots & 1 & 1 \end{pmatrix}$$

(b) Wheel graph W_ℓ with spanning tree T'_ℓ that consists of all but one spoke edge as well as one cycle edge, together with the representation matrix.

Figure 1: Wheel graph W_ℓ with two different spanning trees T_ℓ and T'_ℓ as well as the two corresponding representation matrices M_ℓ and M'_ℓ.

In Figure 1 so-called *wheel graphs* and the *wheel matrices* of order ℓ are depicted. They consist of a cycle of length $\ell \geq 3$ plus one node that is connected to every other node via *spoke* edges. In the figure, two different spanning trees T_ℓ and T'_ℓ together with the representation matrices M_ℓ and M'_ℓ are depicted.

These matrices are required as a starting point in Truemper's total unimodularity testing algorithm [19]. There, in order to test for graphicness, a submatrix representing W_3 must be found. The intuitive reason is that W_3 is the smallest 3-connected graph, and Truemper's graphicness test first finds such a submatrix and then iteratively grows it to a sequence of larger (graphic) submatrices. This sequence is later used to efficiently search for certain matrix decompositions. This *graph minor* W_3 is determined from any W_ℓ by successively deleting a spoke edge and contracting a cylce edge.

Notice that the matrices 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 delayed as it follows 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 [20], where only the case of connected matroids is discussed.

Corollary 4. *A binary matroid is either series-parallel or contains the graphic matroid of the wheel graph 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

We now introduce a graph-theoretic viewpointthat is important for the detection of wheel matrices. A binary matrix $A \in \{0,1\}^{m \times n}$ gives rise to a *bipartite graph*, denoted by $\mathrm{BG}(A)$, 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 $\mathrm{BG}(M_\ell)$ 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 $\mathrm{BG}(A)$ 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 1s.

In his paper [19], Truemper describes a way to enforce finding a longer cycle (if one exists): first, one grows the submatrix consisting of 1s to an inclusion-wise maximal one, indexed by rows X and columns Y. Then, one searches for a shortest path P from X to Y in $\mathrm{BG}(A)$ 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-1s submatrix).

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

Figure 2: An SP-reduced matrix A with inclusion-wise maximal all-1s submatrix in the upper left part. Every path in $\mathrm{BG}(A)$ 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 $\operatorname{rank}(A_{X^1,Y^2}) + \operatorname{rank}(A_{X^2,Y^1}) = 1$ and $|X^i| + |Y^i| \geq 2$ holds for $i = 1, 2$. After reordering of rows and columns, A looks as in Figure 3. The decomposition of A into A^1 and A^2 is called a 2-*sum decomposition*. For matroids, it 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 2 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 1s, while C is the (smallest) submatrix containing all blue 1s.

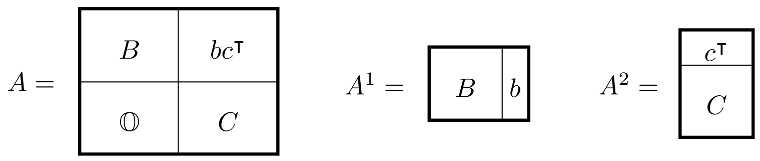


Figure 3: Partitioned version of a 2-separable matrix A that is decomposed into a 2-sum of A^1 and A^2 at the last column and first row, respectively. Note that it is required that B and C each have at least 2 elements and that b and c are nonzero, i.e., $bc^\intercal$ has rank 1.

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 3, 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 identical to a column d of B. We can remove the column d from B and attach it to C. Then the lower-left submatrix (in Figure 3) 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 $\bar{A} = A^\intercal$ 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 $B_{r',\star} = \mathbb{O}$ 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. □

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 $\mathrm{BG}(A')$ 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 1s and let $X$ and $Y$ be the row and
               column sets of $B$, respectively.
11             Define $A''$ to be $A'$ with the entries of $B$ replaced by 0s.
12             Using breadth-first search, search for a (shortest) path $P$ from $X$ to $Y$ in $\mathrm{BG}(A'')$.
13             if $P$ exists then
14                 Let $c$ be the column that comes directly after $P$'s starting node from $X$.
15                 Let $r' \in X$ be such that $A_{r',c} = 0$.
16                 Let $r$ be the row that comes directly before $P$'s end node from $Y$.
17                 Let $c' \in Y$ be such that $A_{r,c'} = 0$.
18                 Let $B'$ be the submatrix indexed by all rows and columns of $P$, row $r'$ and column $c'$.
19                 return "$A$ is not series-parallel" together with $B'$.
20             else
21                 The nodes reachable from $X$ induce a 2-separation of $A$ with parts $A^1$ and $A^2$.
22                 Let $i \in \{1,2\}$ be such that $A^i$ has the minimum number of elements.
23                 return output of recursive call of Algorithm 2 for $A^i$.
24             end
25         end
26     else
27         The nodes reachable from the source node of the search induce a 2-separation of $A$ with parts
           $A^1$ and $A^2$.
28         Let $i \in \{1,2\}$ be such that $A^i$ has the minimum number of elements.
29         return output of recursive call of Algorithm 2 for $A^i$.
30     end
31 end
```

Theorem 6. *Let $A \in \{0,1\}^{m \times n}$ have k nonzeros. Then Algorithm 2 determines in expected $\mathcal{O}(k)$ 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 $\mathrm{BG}(A)$ 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 $\mathrm{BG}(A)$ 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-1s 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. The bipartite graph $\mathrm{BG}(A)$ has $m+n \in \mathcal{O}(k)$ nodes and k edges, and hence the breadth-first search runs in $\mathcal{O}(k)$ time. Hence, all lines except for the recursive calls in lines 23 and 29 can be carried out in time $\mathcal{O}(k)$. To bound the overall running time, let $f(k)$ denote the running time of the algorithm for matrices with k nonzeros (and at most k rows and columns). 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. This yields the recurrence relation $f(k) = \mathcal{O}(k) + f(k/2)$, which yields $f(k) = \mathcal{O}(k)$. □

A few remarks can be made. First, Algorithm 2 can be modified[1] 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.

Proof of Theorem 3. It follows from Theorem 6 that every matrix that is not series-parallel must contain a wheel matrix. Moreover, wheel matrices are SP-reduced, and hence they are not series-parallel. This concludes the proof. □

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 be 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 $h : E \to \mathbb{Z}$ by $h' : E \to \mathbb{N}$ 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 these adaptions 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.

In addition, it is possible that the input matrix $A \in \{-1, 0, +1\}^{m \times n}$ is ternary SP-reduced but $\text{supp}(A)$ is not binary SP-reduced.

In this case, no zero- or unit reduction can be possible for $\text{supp}(A)$ since the same reduction would be applicable to A as well. Hence, there must exist a binary copy reduction of $\text{supp}(A)$ that does not correspond to a ternary copy reduction of A. Without loss of generality, we consider only row copy reductions. Thus, there exist rows r and r' such that $A_{r,\star} \neq A_{r',\star}$ and $A_{r,\star} \neq -A_{r',\star}$ but $\text{supp}(A)_{r,\star} = \text{supp}(A)_{r',\star}$ holds. Hence, there must exist columns c and c' such that $A_{r,c} = A_{r',c}$ and $A_{r,c'} = -A_{r',c'}$. This implies that A contains (up to scaling of rows/columns) the matrix

$$N_2 := \begin{pmatrix} -1 & 1 \\ 1 & 1 \end{pmatrix}$$

as a submatrix. Clearly, N_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.

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

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 N_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 $\operatorname{supp}(A')$ obtaining either $\mathcal{R}'$ or submatrix $B'$ of $\operatorname{supp}(A')$.
7      if $\operatorname{supp}(A')$ 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 $\operatorname{supp}(A'(e)) = \operatorname{supp}(A'(e'))$ 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 $\mathcal{O}(k)$ time whether A is series-parallel and if not, finds a signed wheel- or N_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 N_2-submatrix.*

The matrix N_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 [21]. In this light, the following corollary is not surprising.

Corollary 9 (Section 4.5 in [20]). *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

We start by describing the changes that we made for our implementation in the C programming language. 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 $i = 1, 2, \ldots, n$. This makes the algorithm deterministic for the price of not running in linear time due to potential hashtable 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). The implemented hash table uses separate chaining via singly linked lists in order to resolve collisions. Hashing of rows is done in an identical way. The implementation is part of the Combinatorial Matrix Recognition Library [24][2]. All the experiments were carried out on a 3.0 GHz Intel Xeon Gold 5217 CPU on a system with 64 GB of RAM memory. During all our experiments, our hashtable implementation never encountered any collisions.

5.1 Matrices from mixed-integer optimization

The most important application of total unimodularity lies in mixed-integer optimization since coefficient matrices of integer programs that are totally unimodular have very attractive properties [13]. Moreover, many substructures of coefficient matrices arising in practice are totally unimodular, e.g., if they encode network flows or (conditional) precedences. For this reason we created a set of ternary matrices for which testing for total unimodularity is of potential interest for mixed-integer programming. We consider the currently 240 benchmark

[2] See discopt.github.io/cmr/ for details on the installation, compilation, instance generation / extraction and reproduction of the results.

Table 1: Results for 33 of 79 ternary coefficient submatrices of mixed-integer-programs from the MIPLIB [11] that are series-parallel. Depicted are characteristics of the ternary submatrix of the constraint matrix as well as the running times of the total unimodularity test with and without Algorithm 1.

Instance	**Ternary submatrix**			**Without**	**With**
	rows	**cols**	**nonzeros**	**Algorithm 1**	**Algorithm 1**
`blp-ar98`	913	16 021	15 806	0.02 s	0.02 s
`blp-ic98`	627	13 640	13 550	0.02 s	0.02 s
`bnatt500`	6525	4024	10 053	0.09 s	0.01 s
`bppc4-08`	20	1456	1454	0.00 s	0.00 s
`csched007`	103	1758	1562	0.01 s	0.01 s
`csched008`	111	1536	1405	0.01 s	0.00 s
`exp-1-500-5-5`	300	990	1480	0.03 s	0.00 s
`fhnw-binpack4-48`	910	3675	3640	0.01 s	0.01 s
`fhnw-binpack4-4`	152	507	494	0.00 s	0.00 s
`glass4`	142	256	263	0.00 s	0.00 s
`lotsize`	725	2985	4175	0.11 s	0.00 s
`mas74`	1	151	150	0.00 s	0.00 s
`mas76`	1	151	150	0.00 s	0.00 s
`mik-250-20-75-4`	120	270	120	0.00 s	0.00 s
`milo-v12-6-r2-40-1`	4914	1764	6296	0.07 s	0.01 s
`n3div36`	4426	22 120	27 960	0.88 s	0.04 s
`neos-2657525-crna`	254	524	676	0.01 s	0.00 s
`neos-2978193-inde`	332	20 800	20 800	0.02 s	0.01 s
`neos-3004026-krka`	4225	17 030	16 900	0.05 s	0.05 s
`neos-3381206-awhea`	4	2375	1900	0.00 s	0.01 s
`neos-3754480-nidda`	300	201	300	0.00 s	0.00 s
`neos-4338804-snowy`	441	1344	2562	0.01 s	0.01 s
`neos-4413714-turia`	952	190 401	190 004	0.10 s	0.10 s
`neos-4647030-tutaki`	2800	12 600	11 200	0.04 s	0.03 s
`neos-4954672-berkel`	1764	567	1035	0.02 s	0.01 s
`neos-787933`	133	236 376	61 944	0.31 s	0.31 s
`neos-848589`	737	550 539	550 539	0.27 s	0.27 s
`neos17`	1	535	50	0.00 s	0.00 s
`pk1`	30	86	60	0.00 s	0.00 s
`proteindesign121hz512p9`	79	159 145	159 133	0.08 s	0.08 s
`proteindesign122trx11p8`	67	127 326	127 315	0.06 s	0.07 s
`supportcase42`	18 439	18 442	34 831	63.51 s	0.03 s
`tr12-30`	360	1080	1068	0.03 s	0.00 s

Table 2: Results for 46 of 79 ternary coefficient submatrices of mixed-integer-programs from the MIPLIB [11] that are not series-parallel. Depicted are characteristics of the ternary submatrix of the constraint matrix and its SP-reduced submatrix as well as the running times of the total unimodularity test with and without Algorithm 1.

Instance	**Ternary submatrix**			**SP-reduced submatrix**			**Without**	**With**
	rows	**cols**	**nonzeros**	**rows**	**cols**	**nonzeros**	**Algorithm 1**	**Algorithm 1**
`30n20b8`	90	18 380	18 438	27	22	54	0.0 s	0.0 s
`50v-10`	50	2013	732	50	183	366	0.0 s	0.0 s
`assign1-5-8`	31	156	260	31	130	260	0.0 s	0.0 s
`b1c1s1`	2848	3872	8112	2080	2192	5728	1.6 s	1.5 s
`beasleyC3`	500	2500	2500	301	426	852	0.2 s	0.0 s
`binkar10_1`	1016	2298	4326	785	1242	3030	0.0 s	0.0 s
`cbs-cta`	10 112	22 326	54 520	244	11 163	22 326	96.6 s	8.6 s
`cost266-UUE`	1389	4161	8151	1332	2052	4104	0.1 s	0.0 s
`drayage-100-23`	405	11 025	22 050	210	10 960	21 920	7.6 s	7.7 s
`drayage-25-23`	405	11 025	22 050	210	10 960	21 920	7.9 s	7.6 s
`fiball`	3449	34 218	34 648	221	258	688	0.1 s	0.1 s
`h80x6320d`	238	12 640	18 881	80	3160	6320	3.3 s	0.6 s
`ic97_potential`	1046	205	2092	523	205	1046	0.1 s	0.0 s
`icir97_tension`	368	2494	2027	243	145	1078	0.1 s	0.0 s
`lectsched-5-obj`	33 526	5651	59 340	7880	127	15 760	81.2 s	1.5 s
`mad`	30	220	400	30	200	400	0.0 s	0.0 s
`mc11`	400	3040	3040	400	760	1520	0.2 s	0.0 s
`neos-1122047`	57 791	5000	115 580	57 790	5000	115 580	37.0 s	37.2 s
`neos-1445765`	1024	20 617	20 663	10	12	24	0.2 s	0.0 s
`neos-1582420`	10 080	10 100	22 407	2387	2407	7021	59.5 s	33.6 s
`neos-3024952-loue`	3390	3255	7725	3315	3075	7575	0.1 s	0.1 s
`neos-3046615-murg`	258	274	546	18	32	64	0.0 s	0.0 s
`neos-3083819-nubu`	4719	8644	20 118	3648	6628	17 031	25.1 s	19.3 s
`neos-3627168-kasai`	1190	1448	3038	1022	924	2674	0.1 s	0.1 s
`neos-4387871-tavua`	554	4004	7984	279	1012	2024	0.4 s	0.0 s
`neos-4738912-atrato`	918	6189	13 035	768	2025	5955	350.5 s	125.5 s
`neos-5107597-kakapo`	6498	138	13 008	1698	114	3396	3.4 s	0.1 s
`neos-911970`	59	888	1680	59	840	1680	0.1 s	0.1 s
`nexp-150-20-8-5`	2385	20 115	22 350	150	2119	4238	0.2 s	0.2 s
`opm2-z10-s4`	160 625	6250	321 250	160 625	6250	321 250	862.5 s	861.9 s
`p200x1188c`	200	2376	2376	200	594	1188	0.1 s	0.0 s
`radiationm18-12-05`	33 265	40 623	73 885	14 905	18 372	36 948	695.1 s	28.0 s
`rd-rplusc-21`	125 896	118	124 457	4	4	8	1.6 s	0.1 s
`reblock115`	4715	1150	9430	4706	1140	9412	0.4 s	0.4 s
`rmatr100-p10`	7260	7359	21 877	7260	7259	21 777	17.0 s	13.3 s
`rmatr200-p5`	37 617	37 816	113 048	37 617	37 616	112 848	867.4 s	791.6 s
`rococoB10-011000`	1442	4456	7787	450	2025	4050	0.1 s	0.0 s
`rococoC10-001000`	1088	3117	5396	330	1353	2706	0.0 s	0.0 s
`sct2`	2001	5884	8901	1797	2885	5770	6.8 s	43.2 s
`sp150x300d`	150	600	600	150	253	506	0.0 s	0.0 s
`supportcase18`	120	13 410	14 400	30	30	70	1.0 s	0.0 s
`supportcase26`	870	40	1700	434	40	868	0.1 s	0.0 s
`swath1`	503	6805	19 121	102	6240	18 640	0.6 s	0.3 s
`swath3`	503	6805	19 121	102	6240	18 640	0.6 s	0.3 s
`trento1`	1259	7687	73 662	1189	6415	69 590	13.7 s	5.7 s
`uccase9`	49 061	21 446	47 270	8008	8008	19 327	0.9 s	0.8 s

MIP instances from the state-of-the-art benchmark library MIPLIB 2017 [11]. For each of these instances, we extracted a large ternary submatrix in a greedy fashion: successively remove a row or a column with a maximum number entries not in $\{-1, 0, +1\}$.

For the matrices of the nine instances `enlight_hard`, `gen-ip002`, `gen-ip054`, `markshare2`, `markshare_4_0`, `ns1952667`, `pg`, `pg5_34` and `timtab1` our greedy removal algorithm returned a matrix without any nonzeros. Moreover, for the four instances `neos-4763324-toguru`, `radiationm40`, `square41` and `square47` the total unimodularity test took more than an hour. Finally, for another four instances `buildingenergy`, `neos-873061`, `snp-02-004-104` and `thor50dday` disabling the preprocessing with Algorithm 1 forced the implementation to carry out extremely many 2-sum decompositions (which were actually SP-reductions). Since the resulting matrices of each 2-sum decomposition are created in memory, the code ran into memory problems despite the 64 GB available RAM. Note that this problem does not occur when using Algorithm 1 since only the SP-reduced submatrix is explicitly created in memory for further computations. For a further 144 instance, the computed ternary submatrices did not pass the very first step of the total unimodularity test, which is a test whether the sign pattern of the nonzeros of the given ternary matrix is *balanced*. For details we refer to the respective descriptions in [19, 23].

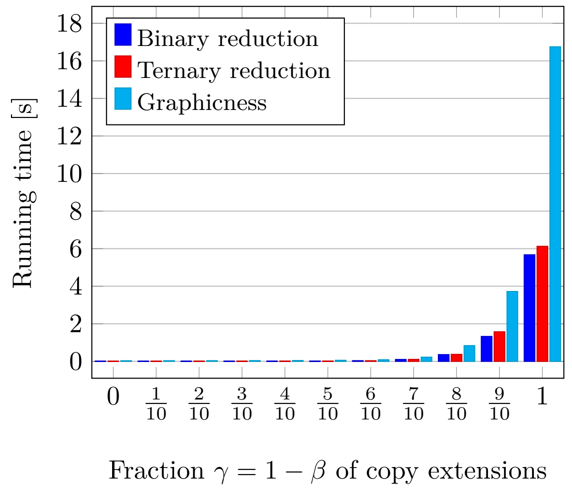
For the remaining 79 matrices we ran the total unimodularity test with and without preprocessing using Algorithm 1. In Tables 1 and 2 we present the sizes of the ternary submatrices, the SP-reduced submatrices and report about the timings of the total unimodularity test in both runs. Note that for the series-parallel submatrices we do not report about the SP-reduced matrices since these are empty.

For series-parallel submatrices, both running times are very short since the total unimodularity test does not need to do anything further besides searching for 2-separations and carrying out 2-sum decompositions. In particular, running times are very similar. Only for larger matrices the preprocessing actually gives a speed-up. The reason is that an explicit application of a 2-sum decomposition is more costly for larger matrices.

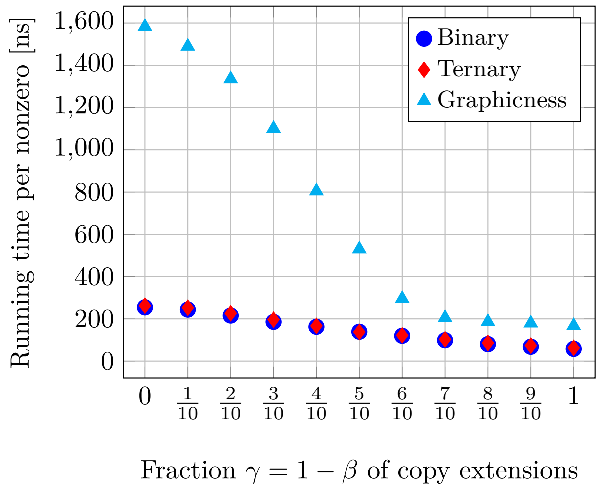
Among the more interesting instances with non-series-parallel ternary submatrices we can find more with longer running times. Here, after carrying out 2-sum decompositions, further algorithms are applied, among them the enumeration of 3-separations which is responsible for the asymptotic running time of the total unimodularity test. Here, in several cases, preprocessing reduces the running time significantly. Moreover, it rarely slows down the computation, except for one remarkable instance: for `sct2`, detailed inspection reveals that the version without preprocessing spends about 2.5 s for finding 2-separations 3750 times, while the SP-reduction algorithm is only called 348 times, which requires 40.1 s. Profiling indicated that after carrying out the SP-reductions, subsequent decisions are made differently than in the run without preprocessing.

These instances show that submatrices arising in practical mixed-integer programs admit SP-reductions, and are sometimes even series-parallel. Moreover, we can conclude that in most cases the application of Algorithm 1 is advantageous.

5.2 Random matrices



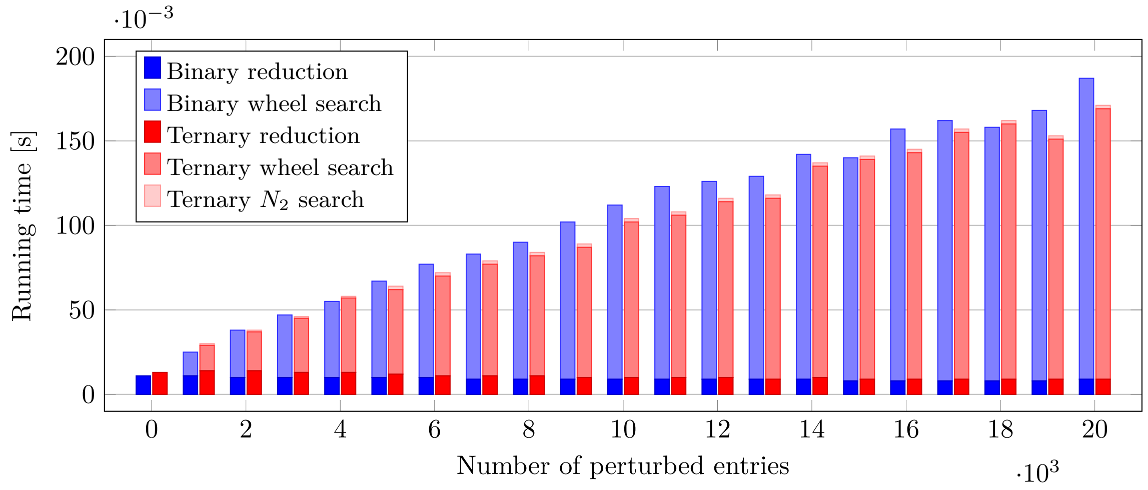
(a) Effect of reduction type on running time.



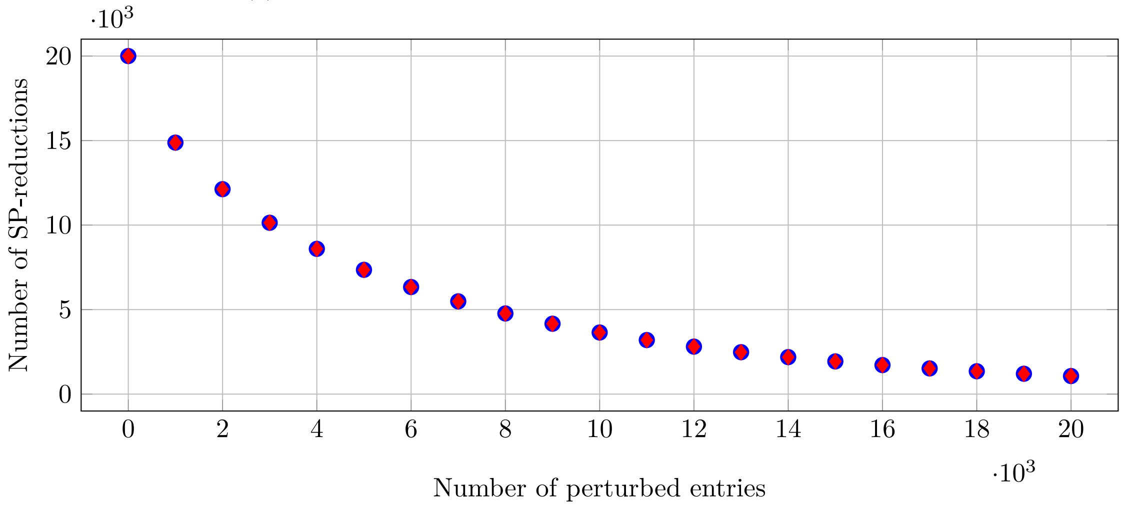
(b) Effect of reduction type on running time per nonzero.

Figure 4: Results for random series-parallel matrices with $\alpha = \delta = 0$, $p = 1$, $\beta = 1 - \gamma$ and varying values for $\gamma \in [0, 1]$. Averaged over 1000 matrices per configuration.

In order to investigate the factors that have impact on the running time of our algorithms we generated random N-by-N matrices in a structured way. The matrices are parameterized by four factors α, β, γ and δ

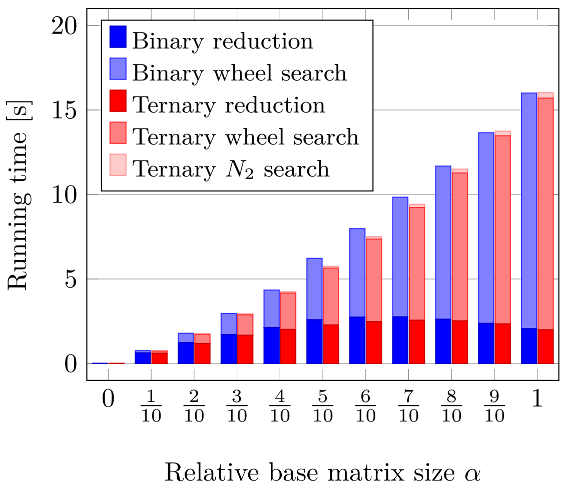


(a) Effect of number of perturbed entries on running time.



(b) Effect of number of pertubed entries on the number of possible SP-reductions.

Figure 5: Results for random series-parallel matrices with $\alpha = 0$, $\beta = \gamma = 0.5$, $p = 1$ and varying values for $\delta \in [0, 2]$. Averaged over 1000 matrices per configuration.



(a) Effect of base matrix size on running time.

(b) Effect of base matrix size on running time per nonzero.

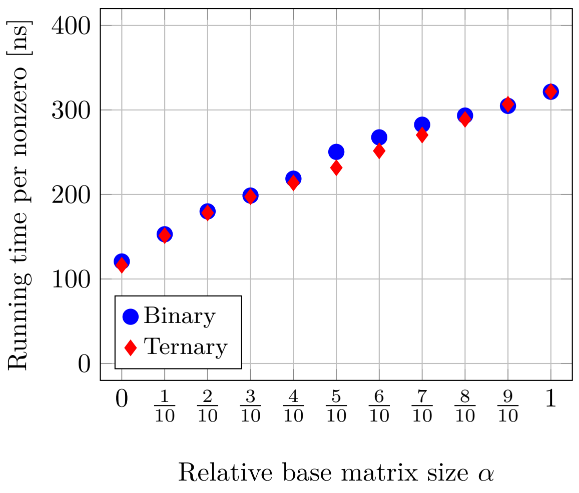
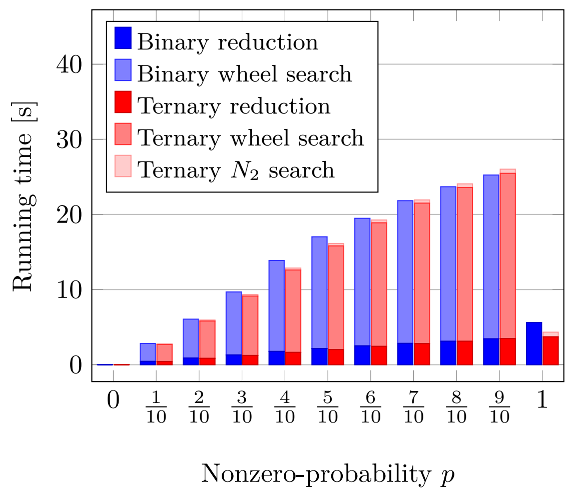
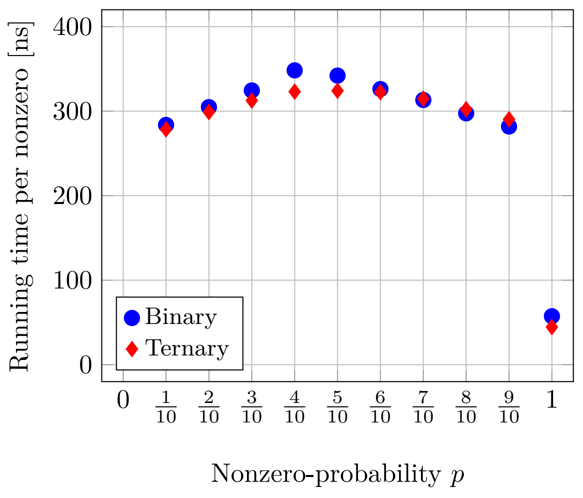


Figure 6: Results for random series-parallel matrices with $\delta = 0$, $p = 0.5$, $\beta = \gamma = \frac{1}{2}(1 - \alpha)$ and varying values for $\alpha \in [0, 1]$. Averaged over 100 matrices per configuration.



(a) Effect of nonzero probability on running time.



(b) Effect of nonzero probability on running time per nonzero.

Figure 7: Results for random series-parallel matrices with $\alpha = 1$, $\beta = \gamma = \delta = 0$ and varying values for $p \in [0, 1]$. Averaged over 10 matrices per configuration.

and a probability p. The first factor $\alpha \in [0, 1]$ indicates the portion of an $(\alpha \cdot N)$-by-$(\alpha \cdot N)$ *base matrix* whose entries were drawn from a uniform distribution. For binary matrices, p indicates the probability of a 1, while for ternary matrices, $+1$ and -1 each occur with probability $p/2$. With high probability, this base matrix does not admit SP-reductions, unless p is close to 0 or 1 or α is close to 0.

This base matrix is extended by subsequently adding $(\beta \cdot N)$ unit rows, $(\beta \cdot N)$ unit columns, $(\gamma \cdot N)$ copies of rows and $(\gamma \cdot N)$ copies of columns to it. 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. To obtain the right matrix size always have $\alpha + \beta + \gamma = 1$. The final step in our generation procedure is perturbation as proposed by one the anonymous reviewers: in order to destroy a small amount of series-parallel structure we flip $(\delta \cdot N)$-many entries. To stay within the (memory) capacity of our computational environment we carried out all our experiments for $N = 10\,000$.

Series-parallel matrices. The first set of random matrices has $\alpha = 0$ (technically, there exists a 1-by-1 base matrix), varying β, $\gamma = 1 - \beta$ and $\delta = 0$. Hence, all these matrices are series-parallel, and hence $2N$ reductions are carried out each time. The results are shown in Figure 4, also in comparison to our implementation of the graphicness algorithm due to Bixby and Wagner [2]. It is easy to see that our specialized algorithm performs significantly better than the graphicness test. However, we would like to emphasize that the main motivation for Algorithm 1 is not to test for being series-parallel faster than the graphicness test, but also to compute the (largest) SP-reduced submatrix in case of a non-series-parallel matrix, as done for the matrices from mixed-integer programming. In particular, the graphicness test does not produce such a submatrix. Hence, we omit the comparison for subsequent matrices since these are mostly non-series-parallel.

Now let us turn to the actual analysis of our algorithm's behavior. The visible running time increase in case of a larger number of copy reductions seems to be surprisingly high. Hence, we also depict the average running time per nonzero. As Figure 4b 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 one nonzero.

Perturbed series-parallel matrices. For the second set we kept $\alpha = 0$, $\beta = \gamma = 0.5$ and $p = 1$, but now varied δ, i.e., we considered random perturbations of series-parallel matrices. Results are shown in Figure 5. Clearly, the number of possible SP-reductions decreases with increasing δ, as Figure 5b shows. This is natural since the first perturbed entries most likely destroys the corresponding row and the corresponding column reduction at once. The running times of Algorithm 1 are not affected drastically, in particular since the overall number of nonzeros of the matrices do not change much. However, the it is apparent that the running time of Algorithm 2 is dominated by the size of the SP-reduced submatrix.

Matrices with varying base matrix sizes. For the third set we kept $\delta = 0$, $p = 0.5$, varied α and set $\beta = \gamma = \frac{1}{2}(1 - \alpha)$, i.e., we considered different sizes of base matrices that were extended to size N-by-N. For all the generated matrices, exactly $(1 - \alpha) \cdot 2N$ many SP-reductions reductions could be applied, i.e., the all generated base matrices were SP-reduced. The results are shown in Figure 6. While the effort for Algorithm 2 increases with α, the running time for Algorithm 1 starts to decrease again for larger α, which can be justified by the fact that fewer reductions must actually applied. As in the previous experiments, the overall running time is dominated by that of Algorithm 2. In particular, searching an N_2-submatrix takes almost no time. Finally, while our implementation was slower for ternary series-parallel matrices than for binary ones, the situation is reversed for non-series-parallel matrices. We do not have an explanation for this, but the differences are also not very significant.

Matrices without SP-reductions. Our last set of randomly generated matrices has $\alpha = 1$, i.e., an N-by-N base matrix, $\beta = \gamma = \delta = 0$, and varying nonzero probabilities p. For $p = 0$ and for $p = 1$ (in the binary case) the resulting matrices were the all-zeros and all-ones matrices, respectively, and thus series-parallel. In all other cases, all of the generated matrices were SP-reduced. Moreover, all returned forbidden submatrices were M_3 or M_3', except for the ternary case with $p = 1$, where N_2 was returned. Further results are shown in Figure 7. Both, the effort for Algorithm 1 for determining that the matrix is SP-reduced, and the effort for Algorithm 2 for finding a wheel submatrix grow with the increasing nonzero probability, which underlines our previous explanation that the number of nonzeros is the driving factor. As before, the overall running time is dominated by the search for wheel submatrices. The jump for $p = 1$ is due to the special structure: for binary matrices, the algorithm has to apply SP-reductions, which is generally fast, while for ternary matrices it has to determine that no ternary SP-reduction is necessary and inspect a single binary SP-reduction in order to find an N_2-submatrix.

6 Conclusion

Our presented algorithm for the problem of recognizing series-parallel matrices is not only theoretically fast. We have shown experimentally that is is also very effective in practice and can deal with large matrices. In fact the measured running times were in the same order of magnitude as elementary operations such as transposing the matrix (assuming that it is given in a sparse data structure). This means that with this direct test one can recognize series-parallel matrices much faster than by determining the graph first and testing the latter for being series-parallel.

Moreover, our algorithm and is certifying, and actually computing a certificate for not being series-parallel requires linear time as well. In practice, this step actually needs most of the running time. As a byproduct, our algorithm is able to determine a maximal non-SP submatrix. This allows to use it as a preprocessing for more complex algorithms such as the recognition of totally unimodular matrices. In fact we can conclude that for several matrices from mixed-integer programming the bottleneck of the total unimodularity test was indeed the application of SP-reductions.

Acknowledgements. The author is grateful to the anonymous reviewers whose comments led to improvements of this manuscript, in particular for the suggestions of further matrix classes. 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] Claude Berge. Balanced matrices. *Mathematical Programming*, 2(1):19–31, 1972. doi:10.1007/BF01584535.

[2] Robert E. Bixby and Donald K. Wagner. An almost linear-time algorithm for graph realization. *Mathematics of Operations Research*, 13(1):99–123, 1988. doi:10.1287/moor.13.1.99.

[3] Andreas Brandstädt, Van Bang Le, and Jeremy P. Spinrad. *Graph Classes: A Survey.* Society for Industrial and Applied Mathematics, 1999. doi:10.1137/1.9780898719796.

[4] Brahim Chaourar and James Oxley. On series-parallel extensions of uniform matroids. *European Journal of Combinatorics*, 24(7):877–879, 2003. doi:10.1016/S0195-6698(03)00093-3.

[5] Wei Chen, Milind Dawande, and Ganesh Janakiraman. Integrality in Stochastic Inventory Models. *Production and Operations Management*, 23(9):1646–1663, 2014. doi:10.1111/poms.12104.

[6] Michele Conforti, Samuel Fiorini, Tony Huynh, and Stefan Weltge. Extended formulations for stable set polytopes of graphs without two disjoint odd cycles. *Mathematical Programming*, 192(1):547–566, 2022. doi:10.1007/s10107-021-01635-0.

[7] Matthew D. Dean, Krishna Murthy Gurumurthy, Felipe de Souza, Joshua Auld, and Kara M. Kockelman. Synergies between repositioning and charging strategies for shared autonomous electric vehicle fleets. *Transportation Research Part D: Transport and Environment*, 108:103314, 2022. doi:10.1016/j.trd.2022.103314.

[8] Paul Deuker and Ulf Friedrich. Recognizing integrality of weighted rectangles partitions. *Optimization Online*, 2023. Optimization Online 2021/11/8703. URL: https://optimization-online.org/?p=18353.

[9] Richard J. Duffin. Topology of series-parallel networks. *Journal of Mathematical Analysis and Applications*, 10(2):303–318, 1965. doi:10.1016/0022-247X(65)90125-3.

[10] Satoru Fujishige. An efficient PQ-graph algorithm for solving the graph-realization problem. *Journal of Computer and System Sciences*, 21(1):63 – 86, 1980. doi:10.1016/0022-0000(80)90042-2.

[11] Ambros Gleixner, Gregor Hendel, Gerald Gamrath, Tobias Achterberg, Michael Bastubbe, Timo Berthold, Philipp M. Christophel, Kati Jarck, Thorsten Koch, Jeff Linderoth, Marco Lübbecke, Hans D. Mittelmann, Derya Ozyurt, Ted K. Ralphs, Domenico Salvagnin, and Yuji Shinano. MIPLIB 2017: Data-Driven Compilation of the 6th Mixed-Integer Programming Library. *Mathematical Programming Computation*, 2021. doi:10.1007/s12532-020-00194-3.

[12] Gregory Henselman and Pawel Dłotko. Combinatorial invariants of multidimensional topological network data. In *2014 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, pages 828–832. IEEE, 2014. doi:10.1109/GlobalSIP.2014.7032235.

[13] Alan J. Hoffman and Joseph B. Kruskal. Integral boundary points of convex polyhedra. In *Linear Inequalities and Related Systems*, volume 38, pages 223–246. Princeton University Press, 1956.

[14] Alfred Lehman. On the width-length inequality. *Mathematical Programming*, 16(1):245–259, 1979. doi:10.1007/BF01588263.

[15] James G. Oxley. *Matroid theory*, volume 1997. Oxford University Press, 1992.

[16] Manfred W. Padberg. A characterization of perfect matrices. In *Topics on Perfect Graphs*, volume 88 of *North-Holland Mathematics Studies*, pages 169–178. North-Holland, 1984. doi:10.1016/S0304-0208(08)72932-3.

[17] Alexander Schrijver. *Theory of Linear and Integer Programming.* John Wiley & Sons, Inc., New York, NY, USA, 1986.

[18] Robert E. Tarjan. Efficiency of a good but not linear set union algorithm. *Journal of the ACM*, 22(2):215–225, 1975. doi:10.1145/321879.321884.

[19] Klaus Truemper. A Decomposition Theory for Matroids. V. Testing of Matrix Total Unimodularity. *Journal of Combinatorial Theory, Series B*, 49(2):241–281, 1990. doi:10.1016/0095-8956(90)90030-4.

[20] Klaus Truemper. *Matroid decomposition (revised edition).* Academic Press, 1998. URL: http://www.utdallas.edu/~klaus/mbook.html.

[21] William T. Tutte. A homotopy theorem for matroids, II. *Transactions of the American Mathematical Society*, 88(1):161–174, 1958. doi:10.2307/1993244.

[22] Jacobo Valdes, Robert E. Tarjan, and Eugene L. Lawler. The recognition of Series Parallel digraphs. *SIAM Journal on Computing*, 11(2):298–313, 1982. doi:10.1137/0211023.

[23] Matthias Walter and Klaus Truemper. Implementation of a unimodularity test. *Mathematical Programming Computation*, 5(1):57–73, 2013. doi:10.1007/s12532-012-0048-x.

[24] Matthias Walter and Klaus Truemper. CMR – Combinatorial Matrix Recognition Library, 2023. Documentation: https://discopt.github.io/cmr, Source code: https://github.com/discopt/cmr.