Engineering Data Reduction for Nested Dissection

Wolfgang Ost* Christian Schulz† Darren Strash‡

Abstract

Many applications rely on solving sparse linear systems, which can be sped up significantly by permuting the matrix to minimize the number of non-zeros introduced by factorization—the *fill-in*. Equivalently, one can compute an elimination order of the graph that minimizes the number of introduced edges, for which the fast but inexact nested dissection algorithm is often used in practice. In this paper, we engineer new data reduction rules for the minimum fill-in problem, which significantly reduce the size of the graph while producing an equivalent (or near-equivalent) instance. By applying both new and existing data reduction rules exhaustively before nested dissection, we obtain improved quality *and* at the same time large improvements in running time on a variety of instances. For example, on road networks, where nested dissection algorithms are typically used as a preprocessing step for shortest path computations, our algorithms are on average six times faster than Metis while computing orderings with less fill-in.

1 Introduction

Solving sparse linear systems of equations is a fundamental task in scientific computing with a variety of applications, such as computational fluid dynamics, electrical flows, structural engineering, economic modeling and circuit simulation [15, 52]. Another important application is solving Laplacian systems which is, among many other use cases, needed to gain insights on the spectral properties of a given network by examining the eigenvalues and eigenvectors of the graph Laplacian [52]. *Sparse linear systems*, of the form $Ax = b$, can in principle be solved by direct methods [16, 28]. Such methods decompose the matrix A into factors that simplify solving the system. The drawback is that such factors can become dense, having many more non-zeros than the original matrix [16, 28, 49]. Solving the system then becomes prohibitively expensive in terms of storage and computation time. The number of new non-zeros introduced by factorization is called the *fill-in*. By reordering the system, fill-in can be significantly reduced, leading to sparse factors [16, 28, 49]. Thus, a problem of central importance is to reduce the fill-in as much as possible to reduce computation time and storage overhead. For symmetric positive definite matrices, which can be factored by Cholesky factorization [28], we can reorder rows and columns by a symmetric permutation $PAP^\top$ [28, 49]. The minimum fill-in problem is to find a permutation matrix P, such that the number of non-zeros introduced during factorization is minimized.

Yannakakis [55] showed that the problem is NP-complete. Hence, heuristic algorithms such as the minimum degree algorithm [49, 54], nested dissection [22] or combinations of both that work on a graph representation of the input matrix are typically used in practice. More precisely, a symmetric matrix can be represented by an undirected graph. In this graph nodes represent rows and columns of the matrix. There is an edge $\{u, v\}$ in the graph if the matrix element $a_{u,v}$ is not zero. An elimination step in the matrix is reflected in the graph by removing the node corresponding to the eliminated column and connecting its neighborhood to form a clique. The added edges provide an upper bound to the number of non-zeros introduced in an elimination step.

On the other hand, many NP-hard graph problems have been shown to be fixed-parameter tractable (FPT): large inputs can be solved efficiently and provably optimally, as long as some parameter of the input is small. Over the last two decades, significant advances have been made in the design and analysis of FPT algorithms for a wide variety of graph problems. This has resulted in a rich algorithmic toolbox that is by now well-established and described in several textbooks and surveys, e.g. [13, 39]. Few of the new techniques are implemented and tested on real datasets, and their practical potential is far from understood. However, recently the engineering part in area has gained some momentum [1, 14, 31–35, 40, 53]. Surprisingly, the minimum fill-in problem also admits a wide range of simple data reduction techniques that have not yet been successfully used in practice.

Our Results. We engineer a new node ordering algorithm that employs novel and existing data reduction

*Faculty of Computer Science, University of Vienna, Austria, wolfgang.ost@univie.ac.at.

†Faculty of Mathematics and Computer Science, Heidelberg University, Germany, christian.schulz@informatik.uni-heidelberg.de. Partially supported by DFG grant SCHU 2567/1-2. Corresponding author.

‡Department of Computer Science, Hamilton College, USA, dstrash@hamilton.edu.

rules before using a nested dissection algorithm. After the nested dissection algorithm terminates, reductions are undone to compute the final node ordering. By applying data reduction rules exhaustively we obtain improved quality *and* at the same time large improvements in running time on a variety of instances. Note that this directly translates to improvements for typical applications. Overall, we arrive at a system that outperforms the state of the art significantly. For example, on road networks, where nested dissection algorithms are typically used as a preprocessing step for shortest path computations [18, 29], our algorithms are on average six times faster than Metis while computing orderings with less fill-in.

2 Preliminaries

In the following we consider an undirected graph $G = (V, E)$, where V are the vertices and E are the edges. We use $|V| = n$ and $|E| = m$. $\Gamma_G(v) := \{u : \{v, u\} \in E\}$ denotes the *neighborhood* of a node v. The set $\Gamma_G[v] := \Gamma_G(v) \cup \{v\}$ is the *closed neighborhood* of v in G. For a set of nodes $A \subseteq V$ we define its neighborhood $\Gamma_G(A) := \left(\bigcup_{x \in A} \Gamma_G(x)\right) \setminus A$. When clear from the context we omit G and write $\Gamma(x)$, $\Gamma[x]$ and $\Gamma(A)$, respectively.

For a set of nodes $V' \subseteq V$ we define the set of edges with both endpoints in V' as $E(V') := E \cap (V' \times V')$. A graph $S = (V', E')$ is said to be a *subgraph* of $G = (V, E)$ if $V' \subseteq V$ and $E' \subseteq E(V')$. We call S an *induced* subgraph when $E' = E(V')$. For a set of nodes $U \subseteq V$, $G[U]$ denotes the subgraph induced by U.

A graph G is *triangulated* or *chordal*, if for every cycle of four or more nodes, there is an edge connecting two non-consecutive nodes in the cycle. A *triangulation* of a graph $G = (V, E)$ is a set of edges T, such that $(V, E \cup T)$ is a triangulated graph. A triangulation is *minimal* if no proper subset is also a triangulation. If there is no triangulation T' with $|T'| < |T|$, then T is a *minimum triangulation*. A *clique* is a set of vertices $K \subseteq V$ such that $\forall u, v \in K$ where $u \neq v$ $\{u, v\} \in E$. A vertex $v \in V$ is *simplicial* if $\Gamma(v)$ is a clique. A graph G is said to have a *perfect elimination ordering* if there is an ordering of vertices $v_1 v_2 \cdots v_n$ such that each vertex v_i is simplicial in the subgraph $G[\{v_{i+1}, \ldots, v_n\}]$ induced by vertices later in the ordering.

In this work, we consider several related partitioning problems. The *graph partitioning problem* asks for *blocks* of nodes $V_1, \ldots, V_k$ that partition V; that is, $V_1 \cup \cdots \cup V_k = V$ and $V_i \cap V_j = \emptyset$ for $i \neq j$. A *balancing constraint* demands that $\forall i \in \{1..k\} : |V_i| \leq L_{\max} := (1 + \epsilon)\lceil |V|/k \rceil$ for some parameter ϵ. In this case, the objective is often to minimize the total *cut* $\sum_{i<j} |E_{ij}|$ where $E_{ij} := \{\{u, v\} \in E : u \in V_i, v \in V_j\}$. The set of cut edges is also called an *edge separator*. A node $v \in V_i$ that has a neighbor $w \in V_j, i \neq j$, is a *boundary node*. The *node separator problem* asks to find blocks, V_1, V_2 and a separator S that partition V such that there are no edges between the blocks. Again, a balancing constraint demands $|V_i| \leq (1+\epsilon)\lceil |V|/k \rceil$. However, there is no balancing constraint on the separator S. The objective is to minimize the size of the separator $|S|$. We call V_1 and V_2 the *components* and the induced subgraphs $G[S \cup V_i]$ the *leaves* of S. A separator that is also a clique is a *separation clique*.

In general, a *multilevel approach* consists of three main phases: coarsening, initial solution, and uncoarsening. These phases are typically adjusted depending on the optimization problem that is tackled. In the coarsening phase, contraction should quickly reduce the size of the input. Contraction is stopped when the graph is small enough so a problem can be solved by some other potentially more expensive algorithm, producing the initial solution. In the uncoarsening phase, contractions are iteratively undone and local search is used on all levels to improve a solution. The intuition behind the approach is that a good solution at one level of the hierarchy will also be a good solution on the next finer level so that local search will quickly find a good solution.

Parameterized Complexity and Data Reduction Rules. Many times, tighter analysis of an algorithm is possible by considering the running time in terms of an *input parameter*, generally denoted by k, which is independent of the input size n. The field of parameterized complexity investigates theoretical algorithms involving such input parameters. Following the framework of Downey and Fellows [19], we say a problem is fixed-parameter tractable (FPT) if it can be solved in time $f(k) \cdot \text{poly}(n)$, where k is a (hopefully small) input parameter and $\text{poly}(n)$ is a polynomial-time function of the input size n that does not include k.

Tightly connected to fixed-parameter tractability is the concept of *data reduction rules* and kernelization. Normally discussed in terms of a decision problem, a data reduction rule maps a problem instance (in our case a pair (G, k) where G is the graph and k is the minimum fill-in) to an new instance (G', k') of smaller size, such that (G', k') is a 'yes' instance if and only if (G, k) is a 'yes' instance.

The Node Ordering Problem. Given a matrix $A \in \mathbb{R}^{n \times n}$ and a column vector $b \in \mathbb{R}^n$ we want to solve the system of linear equations given by $Ax = b$. This is usually accomplished by first factoring the matrix A. For symmetric matrices the Cholesky decomposition can be used which factorizes A into a lower triangular matrix L and its transpose $L^\top$ such that $A = LL^\top$. An extension of the simple Cholesky decomposition is

to reorder the rows and columns of A prior to the factorization. This is done by applying a permutation matrix P to rows and columns of the matrix A which leads to $PAP^\top = LL^\top$. For large sparse matrices it is crucial to choose a good permutation matrix P in order to reduce the fill-in during the factorization which reduces both the amount of memory needed to store the factors as well as the number of operations needed to factorize the matrix. A permutation matrix can also be expressed as a permutation vector which maps each row respectively column to a rank in $\{1, \ldots, n\}$. The matrix A can be viewed as a graph $G = (V, E)$ such that $V := \{1, \ldots, n\}$ and there exists an edge for every non-zero entry in A which does not lie on the diagonal: $E := \{\{i, j\} : i \neq j \wedge A[i, j] \neq 0\}$. Elimination of a column and row in A is reflected in G by eliminating the corresponding node and connecting its neighborhood to form a clique. Finding a permutation matrix for A then corresponds to finding an elimination order of nodes in G, which is called a *node ordering*.

The *deficiency* $D_G(x)$ of a node x in a graph G is the set of distinct pairs of nodes in $\Gamma_G(x)$, that are not themselves neighbors: $D_G(x) := \{\{a, b\} \mid a, b \in \Gamma_G(x),\ a \neq b,\ a \notin \Gamma_G(b)\}$. When clear from the context we omit G and write $D(x)$. Eliminating a node x from a graph $G = (V, E)$ results in the *elimination graph* $G_x := (V \setminus \{x\}, E(V \setminus \{x\}) \cup D_G(x))$, which is obtained by removing x and its incident edges from G, and connecting the neighborhood of x to a clique. We call this process an *elimination step*. The elimination graph obtained by eliminating a sequence of nodes $X = x_1 x_2 \cdots x_m$ is denoted by $G_X := (\ldots((G_{x_1})_{x_2})\ldots)_{x_m}$.

A node ordering of a graph $G = (V, E)$ with $n = |V|$ is a bijection $\sigma : \{1, 2, \ldots, n\} \to V$, that defines a sequence of elimination graphs $G^{(1)}G^{(2)}\ldots G^{(n)}$, where $G^{(i)} := (G^{(i-1)})_{\sigma(i)}$ if $i = 1, \ldots, n$ and G if $i = 0$. In $G^{(n)}$, all nodes have been eliminated. The *fill-in* of an ordering σ is the number of edges added during the elimination process, denoted by $\phi(G, \sigma) := \sum_{i=1}^{n} |D_{G^{(i-1)}}(\sigma(i))|$. We let $\Sigma(G) = \arg\min_\sigma\{\phi(G, \sigma)\}$ be some minimum fill-in ordering of a graph G, with the corresponding minimum fill-in $\Phi(G) = \phi(G, \Sigma(G))$. Note that

$$\Phi(G) \geq \Phi(G^{(1)}) \geq \ldots \geq \Phi(G^{(n-1)}). \tag{2.1}$$

An ordering σ of a graph $G = (V, E)$ generates a triangulation $T(\sigma)$ of G, such that the graph $(V, E \cup T(\sigma))$ is chordal. $T(\sigma)$ is the set of edges added during the elimination process and $|T(\sigma)| = \phi(G, \sigma)$. A minimum fill-in ordering $\Sigma(G)$ generates a *minimum triangulation* $T(\Sigma(G))$, where $\Phi(G) = |T(\Sigma(G))|$ [44]. If G is triangulated, then its minimum triangulation is the empty set and it has a perfect elimination order, i.e., $\Phi(G) = 0$. We use the following notation for node orderings: $\sigma = x_1 x_2 \cdots x_n$ corresponds to $\sigma(1) = x_1, \sigma(2) = x_2, \ldots, \sigma(n) = x_n$. We write $x\Sigma(G_x)$ if x is to be eliminated before the nodes in G_x. To denote nodes ordering where a set of nodes $P = \{p_1, p_2, \ldots, p_n\}$ are eliminated in any order, we use P in the notation instead of $p_1 p_2 \cdots p_n$. For example, $P\Sigma(G_P)$ is an ordering in which the nodes in P are eliminated in any order before the nodes in G_P.

3 Related Work

There has been a *huge* amount of research on graph partitioning, node separators and minimum fill-in ordering; we refer the reader to the overviews [10, 12, 50] for preliminary material in this area. Here, we focus on issues closely related to our main contributions and previous work on the node ordering problem.

Yannakakis proved that the problem of finding a minimum fill-in ordering is NP-complete [55]. Exact algorithms have been introduced in the context of nonserial dynamic programming [8, 9], but they are not practical for large matrices due to their exponential running time [49]. The fastest such algorithm is due to Fomin et al. [21], with running time $\mathcal{O}^*(1.7347^n)$, where $\mathcal{O}^*$ hides polynomial factors. Parameterized algorithms offer a promising alternative to algorithms that are exponential in the input size. In particular, the problem is fixed-parameter tractable [36], when the input parameter k is the minimum fill-in. The fastest-known FPT algorithm for the problem is due to Fomin and Villanger [20], with running time $\mathcal{O}\left(2^{\mathcal{O}(\sqrt{k}\log k)} + k^2 nm\right)$ that is subexponential in the minimum fill-in k. Here, the additive $\mathcal{O}(k^2 nm)$ is the time to compute a kernel of size k^2 using data reduction rules, using the algorithm of Kaplan et al. [37]. This is the smallest known kernel for the problem. Despite these theoretical improvements, in practice, the minimum fill-in problem is extremely hard to solve exactly. Indeed, in the Second Parameterized Algorithms and Computational Experiments Challenge (PACE 2017), even when using generalized variants of the reduction rules of Bodlaender et al. [11], the winning solver for the minimum fill-in problem only solved 54 out of 100 instances [17].

For graphs with a perfect elimination order, the problem can be solved in $\mathcal{O}(|V| + |E|)$ time [47]. Tinney and Walker [54] introduced a heuristic algorithm where the next column to eliminate is selected based on the number of non-zeros. This algorithm is known as the *minimum degree algorithm*, since a node of minimum degree is eliminated at each step [49]. There have been several improvements to this algorithm, both in its design and implementation [23, 25, 26]. The minimum

degree algorithm spends a significant part of its time in updating node degrees. Most of the improvements to the minimum degree algorithm are thus focused on reducing the number of nodes to update [26]. Amestoy et al. [2] introduced an approximate minimum degree algorithm in which the degree update is not performed exactly. The *minimum deficiency algorithm* is a greedy algorithm similar to the minimum degree algorithm [49, 54]: at every step the node with the smallest deficiency is eliminated. If the graph to be ordered has a perfect elimination ordering, the minimum deficiency algorithm finds it. However, finding the deficiency of a node is expensive, so the algorithm is slower than the minimum degree algorithm [49].

In 1973, George [22] introduced an algorithm to produce orderings for regular finite element meshes, called nested dissection. This algorithm computes a node separator, and then recursively orders the partitions before the separator. George and Liu generalized the algorithm to work on arbitrary graphs [24]. The fastest and most widely used nested dissection implementation is in the highly-optimized graph partitioning software package, called Metis, due to Karypis and Kumar [38]. In practice, nested dissection is combined with algorithms such as the minimum degree algorithm: once the subgraphs are small enough, they are ordered by the minimum degree algorithm [4, 5, 38]. A similar approach based on multisectors instead of bisectors was presented by Ashcraft and Liu [5]. LaSalle and Karypis [41] gave a shared-memory parallel algorithm to compute node separators used to compute fill-reducing orderings. Within a multilevel approach they evaluate different local search algorithms indicating that a combination of greedy local search with a segmented FM algorithm can outperform serial FM algorithms. On road networks nested dissection is used as preprocessing step for shortest path computations [29]. The authors use degree-2 preprocessing to speed up their nested dissection algorithm.

Minimum fill-in is closely related to the notions of tree width and tree depth. The tree width of a chordal graph is one less than the size of its maximum clique. The tree width of a graph G is the minimum tree width of a chordal graph that contains G. We can obtain the tree width of G by computing a triangulation T of $G = (V, E)$ that minimizes the size of the maximum clique of the chordal graph $(V, E \cup T)$. The tree depth of a graph is the minimum height of an elimination tree of the graph. An elimination tree is a spanning tree of the triangulated graph and is defined by a node ordering. We are interested in finding a node ordering with minimum fill-in, i.e., a triangulation of minimum size, and do not evaluate our algorithm in terms of tree width and tree depth.

4 Advanced Node Ordering

We now outline our reduced nested dissection algorithm and describe our reductions in detail. For completeness, we outline the standard nested dissection algorithm in Algorithm 1 in Algorithm 4 as implemented for example in Metis [38]. We extend the nested dissection by transforming the input graph G into a (smaller) equivalent graph G' using our reduction rules. We apply reductions in a fixed order and each reduction is applied exhaustively, i.e., the graph is reduced as much as possible by each reduction. Then, we apply nested dissection on the reduced graph G' to obtain an ordering σ. After the nested dissection algorithm returns the ordering σ, the ordering of the reduced graph is then transformed to an ordering of the input graph σ'. We now explain the data reduction rules that we use.

Algorithm 1: UnreducedNestedDissection(G)

input : Undirected graph $G = (V, E)$
output: Ordering σ

```
1 if |G| ≥ recursion limit then
2     V1, V2, S ← Separator(G)
3     foreach G' in (G[V1], G[V2], G[S]) do
4         σ' ← UnreducedNestedDissection(G')
5         σ ← σσ'
6 else
7     σ ← MinDegree(G)
8 return σ
```

4.1 Data Reduction Rules.

A *data reduction rule* transforms an input graph G into a smaller, reduced graph G'. This new smaller problem instance is generally equivalent to the original, and can be solved in less time. The solution on G' can then be transformed into a node ordering of the nodes of G. If the running time of the transformations is small, solving the problem on G in this way will be faster than a direct approach.

We use four exact and two inexact reduction rules. The *simplicial node reduction* eliminates nodes whose neighborhood is already a clique. These nodes can be ordered first in a minimum fill-in ordering, since they do not contribute to the fill-in. The *indistinguishable node reduction* and *twin reduction* contract sets of nodes with equal closed and open neighborhood, respectively. When any node in such a set is eliminated, then the other nodes become simplicial. Thus, such sets can be ordered together. With *path compression* we replace any path of nodes with degree 2 by a single degree-

2 node. If one node on the path is eliminated, then its degree-2 neighbors can be eliminated next in a minimum fill-in ordering.

Degree-2 elimination is an inexact reduction rule that eliminates nodes of degree 2. This reduction turns out to be exact if none of the eliminated nodes are also separators. Lastly, *triangle contraction* contracts adjacent nodes of degree 3 that share at least one neighbor.

To our knowledge, only the indistinguishable node reduction has been used in practice in combination with nested dissection. While linear time algorithms for ordering chordal graphs are known, it appears that the special structure of simplicial nodes is not exploited in non-chordal graphs. There are two well-known reductions we do not discuss here. First, connected components can be ordered separately. For our test instances this reduction was not useful. Second, cut-vertices can be ordered last. We do not use this reduction in our implementation: in finite element meshes and similar graphs such cut-vertices are rare. In social networks, we observe that, after simplicial node reduction, the largest biconnected component is close to the size of the full graph. We now describe the reduction rules in greater detail. Proofs of the statements can be found in Appendix A.

4.2 The Simplicial Node Reduction. A node x is *simplicial* if its neighborhood $\Gamma(x)$ is a clique (see Figure 1 for an example). There exists a minimum fill-in ordering where x is eliminated first.

THEOREM 4.1. *Let $G = (V, E)$ be a graph with a simplicial node x. The ordering $x\Sigma(G_x)$ is a minimum fill-in ordering of G.*

This allows us to eliminate all simplicial nodes first by the following procedure: Find any simplicial node x in $G = (V, E)$, eliminate x from G and place it next in the node ordering. If the elimination graph G_x has simplicial nodes, then repeat the procedure for G_x. If every elimination graph in the elimination sequence σ has at least one simplicial node, then $\phi(G, \sigma) = 0$. In this case, σ is a perfect elimination ordering of G. Graphs that admit such an ordering are called *chordal* or *triangulated* graphs [48, 49].

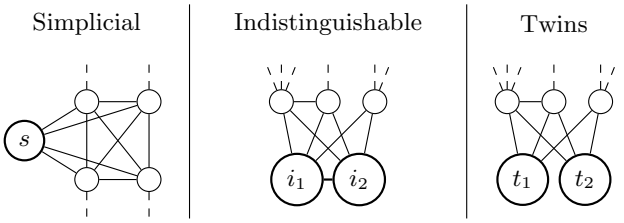

Figure 1: Examples for simplicial nodes, indistinguishable nodes and twins. The neighborhood of s is a clique, so s is simplicial. Nodes i_1 and i_2 are indistinguishable, since they are neighbors and adjacent to all unlabeled nodes, i.e., $\Gamma[i_1] = \Gamma[i_2]$. Nodes t_1 and t_2 are twins, since they are both adjacent to all unlabeled nodes, but not to each other. $\Gamma(t_1) = \Gamma(t_2)$.

REDUCTION 1. (SIMPLICIAL NODE REDUCTION) *Given a graph $G = (V, E)$ and a simplicial node $x \in V$, construct a new graph $G' = G[V \setminus \{x\}]$. $\Phi(G) = \Phi(G')$ and $x\Sigma(G')$ is a minimum fill-in ordering of G.*

4.3 The Indistinguishable Node Reduction. Two nodes a and b are *indistinguishable* if $\Gamma[a] = \Gamma[b]$ (see Figure 1 for an example). Such nodes can be eliminated together: if a and b are indistinguishable nodes, then there exists a minimum fill-in ordering $x_1 \cdots x_i a b x_{i+1} \cdots x_\ell$, where $\{x_1, \ldots, x_i, x_{i+1}, \ldots, x_\ell\} = V \setminus \{a, b\}$. To obtain a reduced graph G', we contract a set of indistinguishable nodes S in G to a single node.

We first establish that indistinguishable nodes stay indistinguishable throughout the elimination sequence. Then, we show that eliminating indistinguishable nodes does in fact lead to minimum fill-in orderings.

LEMMA 4.1. *If a, b are indistinguishable nodes in a graph G, then a and b are indistinguishable in any elimination graph G_x for $x \notin \{a, b\}$.*

THEOREM 4.2. *Let $G = (V, E)$ be a graph with a set of nodes $A \subseteq V$, where $\forall\ a_i, a_j \in A$, $\Gamma[a_i] = \Gamma[a_j]$. There is an ordering $\sigma' = x_1 \cdots x_i A x_{i+1} \cdots x_\ell$, where $V \setminus A = \{x_1, \ldots, x_\ell\}$, such that $\phi(G, \sigma') = \Phi(G)$.*

REDUCTION 2. (INDISTINGUISHABLE NODE REDUCTION) *Given a graph $G = (V, E)$ with indistinguishable nodes $a, b \in V$, construct a new graph $G' = G(V \setminus \{b\})$. Replacing a in $\Sigma(G')$ by ab results in a minimum ordering of G.*

Note, that in the reduced graph G', the deficiency of any node neighboring a set of indistinguishable nodes is different from that of the corresponding node in the original graph G. Thus, we have to optimize the ordering in G' not in terms of the deficiency of a node in G', but in terms of the deficiency of the corresponding node in G. Indistinguishable nodes are commonly used to speed up the minimum degree algorithm [23, 25, 27]. In this context the reduction has been shown to be exact. This reduction is also known as *graph compression* and is used in other variants of nested dissection and the minimum degree algorithm, see for example the algorithms by Ashcraft [3] and Hendrickson and Rothberg [30].

4.4 The Twin Reduction. Two nodes a and b are *twins* if $\Gamma(a) = \Gamma(b)$ (see Figure 1). As indistinguishable nodes, twins can be eliminated together.

THEOREM 4.3. *Let a, b be twins in a graph $G = (V, E)$. There exists an ordering $\sigma' = x_1 \cdots x_i a b x_{i+1} \cdots x_l$, with $x_j \in V \setminus \{a, b\}$, such that $\phi(G, \sigma') = \Phi(G)$.*

We can treat twins similarly to indistinguishable nodes: we obtain a reduced graph by contracting twins. As with Reduction 2, the deficiency of a node neighboring contracted twins in G' is smaller than the deficiency of the corresponding node in G. Thus, orderings of G' should be evaluated not in terms of the deficiency of nodes in G', but in terms of the deficiency of corresponding nodes in G.

REDUCTION 3. (TWIN REDUCTION) *Given a graph $G = (V, E)$ with twins $a, b \in V$, construct a new graph $G' = G[V \setminus \{b\}]$. Replacing a in $\Sigma(G')$ by ab results in a minimum ordering of G.*

4.5 Path Compression. We now show that a path of nodes with degree 2 can be eliminated together. More formally, let $P = \{a_1, a_2, \ldots, a_k\}$ be a path in a graph $G = (V, E)$ with $\deg(a_i) = 2$ for all $a_i \in P$. There is a minimum fill-in ordering $\Sigma = x_1 \cdots x_i a_1 \cdots a_k x_{i+1} \cdots x_\ell$, where $V \setminus P = \{x_1, \ldots, x_\ell\}$.

We prove this by distinguishing three cases based on which nodes are separation cliques, and using the relationship between minimum triangulation and minimum fill-in orderings. Corollary 1 and Proposition 2 from [49] are central to our proof and we restate them here.

LEMMA 4.2. (COROLLARY 1 FROM [49]) *Let $G = (V, E)$ be a graph with separation clique S with components $C_1, C_2, \ldots, C_k$. Any minimum triangulation T of G contains only edges $e = \{x, y\} \in T$ with x and y in the same component C_j, or edges $e = \{x, y\} \in T$ with $x \in C_j$ and $y \in S$.*

LEMMA 4.3. (PROPOSITION 2 FROM [49]) *Let $C = (V, E)$ be a cycle with $|V| \geq 3$ nodes. Any ordering of C is a minimum fill-in ordering.*

Furthermore, we need to show that nodes with degree 2 in induced cycles of four or more nodes can be eliminated first.

LEMMA 4.4. *Let $G = (V, E)$ be a graph with a node $a \in V$ where $\deg(a) = 2$, $\Gamma(a) \notin E$ and $\{a\}$ is not a separation clique. Then, $a\Sigma(G_a)$ is a minimum ordering of G.*

To prove Lemma 4.4 we establish that there exists a minimum triangulation that does not contain an edge to such a node a.

LEMMA 4.5. *Let G and a be as in Lemma 4.4. There exists a minimum triangulation $\hat{T}$ of G, with $\Gamma(a) \in \hat{T}$ and $\{a, x\} \notin \hat{T}$ for all $x \in V$.*

With these results we now prove our original statement:

THEOREM 4.4. *Let $G = (V, E)$ and $P = \{a_1, \ldots, a_k\} \subseteq V$ such that $G[P]$ is a path graph and $\forall\ a \in P$ $\deg(a) = 2$. Let $\Gamma(P) = \{a_0, a_{k+1}\}$ and $\Gamma(a_i) = \{a_{i-1}, a_{i+1}\}$, $i = 1, \ldots, k$. There exists an ordering $\sigma' = x_1 \cdots x_i a_1 \cdots a_k x_{i+1} \cdots x_\ell$ where $V \setminus P = \{x_1, \ldots, x_\ell\}$, such that $\phi(G, \sigma') = \Phi(G)$.*

Since such sets of nodes P can be eliminated together, we can contract them to a single node. It is possible that in a minimum elimination sequence of a graph G, the degree of $a_1 \in P$ becomes 1. Then, P has to be ordered as $a_1 a_2 \cdots a_k$ to obtain a minimum ordering.

REDUCTION 4. (PATH COMPRESSION) *Given a graph $G = (V, E)$ with a set of nodes $P = \{a_1, \ldots, a_k\}$, where $G[P]$ is a path graph, $N(P) = \{a_0, a_{k+1}\}$ and $\forall\ a \in P$ $\deg(a) = 2$, construct a new graph $G' = (V \setminus \{a_2, \ldots, a_k\}, E')$, where $E' = (E \setminus E(P \cup \{a_{k+1}\})) \cup \{\{a_1, a_{k+1}\}\}$. Replacing a_1 in $\Sigma(G')$ by $a_1 a_2 \cdots a_k$ yields a minimum ordering of G.*

4.6 Degree-2 Elimination. Our first *inexact reduction* removes any vertices of degree 2 that remain after applying the simplicial node and path compression reductions. Since the graph has minimum degree two, these nodes would be eliminated first by the minimum degree algorithm, and therefore (judging by that algorithm's success in practice) these are good candidates for removal. Note if this reduction is used after path compression, then the compressed paths are eliminated.

INEXACT REDUCTION 1. (DEGREE-2 ELIMINATION) *Given a graph $G = (V, E)$ and any node x with degree 2, construct the elimination graph G_x. The potentially non-minimum ordering of G is $x\Sigma(G_x)$. The method is applied recursively while there are nodes with degree 2.*

Note that according to the path compression reduction, this reduction is exact when the vertices are in (induced) cycles of at least three vertices. The proof of Theorem 4.4 reveals general conditions for when degree-2 elimination is exact, which are captured in the following two corollaries.

COROLLARY 4.1. *Let $G = (V, E)$ be a graph. If $x \in V$ is in any cycle $C \subseteq V$ and $\deg(x) = 2$, $x\Sigma(G_x)$ is a minimum ordering of G.*

COROLLARY 4.2. *Let $G = (V, E)$ be a graph. Let $x \in V$ be a separator with $\deg(x) = 2$. $x\Sigma(G_x)$ is a non-minimum fill-in ordering of G.*

Corollaries 4.1 and 4.2 imply that degree-2 elimination is exact if only degree-2 nodes that are part of a cycle are eliminated. In graphs where no degree-2 nodes are separators, degree-2 elimination is therefore exact.

4.7 Triangle Contraction. For our next and final inexact reduction, we consider contracting the nodes of a triangle. We assume that simplicial node reduction and degree-2 elimination have already been applied and the minimum degree is 3. Consider two adjacent nodes $a, b \in V$ where $\deg(a) = \deg(b) = 3$ and $|\Gamma(a) \cap \Gamma(b)| \geq 1$, i.e., nodes a and b share at least one neighbor, forming a triangle. If $|\Gamma(a) \cap \Gamma(b)| = 2$, then a and b are indistinguishable and can be contracted. Now, consider the case where $|\Gamma(a) \cap \Gamma(b)| = 1$. Eliminating node a does not increase the degree of node b, and vice versa. After eliminating a, $|D(b)| \leq 2$, i.e., eliminating b only inserts two edges into the graph. Since this fill-in is small, we eliminate b as soon as a was eliminated, and vice versa. Thus, we contract nodes a and b.

Inexact Reduction 2. (Triangle Contraction) *Given a graph $G = (V, E)$ and adjacent nodes a, b with $\deg(a) = \deg(b) = 3$ and $|\Gamma(a) \cap \Gamma(b)| = 1$, construct a new graph $G' = (V \setminus \{a\}, E \setminus (\cup_{x \in \Gamma(a)} \{a, x\}) \cup_{x \in \Gamma(a)} \{x, b\})$. Replacing b by ba in $\Sigma(G')$ yields a potentially non-minimum ordering of G.*

5 Implementation Details

To apply simplicial node reduction (Reduction 1), we iterate through nodes in order by non-decreasing degree. To test if a node x is simplicial, we iterate through the neighbors $y \in \Gamma(x)$. If $|\Gamma(y) \cap \Gamma(x)| = \deg(x) - 1$ for all y, then x is simplicial. When a node is found to be simplicial, we mark it as removed and adjust the degrees of its neighbors accordingly. Removed nodes are ignored when testing the other nodes. The order in which simplicial nodes are found yields their elimination order. Since we only evaluate each node once in a single pass, this method may introduce new simplicial nodes that remain in the graph. However, in practice we find that most simplicial nodes are eliminated in a single pass. Deciding if a node v is simplicial takes time $O(\deg(v)^2)$. For graphs where $\deg(v) = O(n)$ this implies a total time for simplicial node reduction of $O(n^3)$. To avoid this case, we introduce a parameter Δ and only test nodes v that have degree $\deg(v) \leq \Delta$. The total time for simplicial node reduction is then $O(n\Delta^2)$.

The indistinguishable node and twin reductions (Reductions 2 and 3) are similar in their implementation and are based on the algorithms by Ashcraft [3] and Hendrickson and Rothberg [30]. For both reductions we first compute a hash of the neighborhood of each node x_i as $h_c(x_i) = \sum_{y_j \in \Gamma[x_i]} j$ and $h_o(x_i) = \sum_{y_j \in \Gamma(x_i)} j$ We only compare the neighborhoods directly if the hashes of two candidates are equal. To detect indistinguishable nodes, we now go through all pairs (u, v) of adjacent nodes and, if $h_c(u) = h_c(v)$, test if $\Gamma[u] = \Gamma[v]$. Detecting and contracting sets of indistinguishable nodes in this way takes time $O(m)$. To detect twins, we first sort the list of hashes h_o. We then go through the list, and, for pairs of nodes (u, v) with equal hash and degree, test if $\Gamma(u) = \Gamma(v)$. In the worst case, if all hashes are equal and all nodes have the same degree, our implementation takes time $O(mn + n \log(n))$.

In path compression (Reduction 4) and degree-2 elimination (Inexact Reduction 1), nodes to contract or eliminate are detected in time $O(n)$. The reduced graph is then built in time $O(m)$. We order sets of nodes contracted by to path compression starting at the end whose neighbor is eliminated first. Nodes removed during degree-2 elimination appear in the final ordering as they are removed from the graph.

We detect set of nodes A to be contracted in triangle contraction (Inexact Reduction 2) by the following procedure: Let x be some node with $\deg(x) = 3$. Add x to A. Then we repeat the following procedure: If x has a neighbor y with $\deg(y) = 3$ and $|\Gamma(x) \cap \Gamma(y)| \geq 1$, add x and y to A. Let $a \in (\Gamma(x) \cap \Gamma(y))$. Let $z \in \Gamma(y), z \notin A$. If $\deg(z) = 3$ and $a \in \Gamma(z)$, add z to A. Otherwise, stop. Repeat the procedure with the neighbors of z. This reduction can be implemented in time $O(m)$. In the ordering of the input graph, nodes in A are ordered as they are added to A.

6 Experimental Evaluation

Methodology. We implemented the reductions in C++ and compiled using g++ 8.3.0 with optimization flag `-O3`. Additional implementation details can be found in Section 5. We use Metis (version 5.0) [38] to perform nested dissection. All running times were measured on a machine with four Intel Xeon E7-8867 v3 processors (16 cores, 2.5 GHz, 45 MB L3-cache) and 1000 GB RAM. The machine is running 64-bit Debian 10 with Linux kernel version 4.19.67. Our implementation runs on a single core. For each graph and set of parameters we average the results of ten repetitions. We use nested dissection in Metis with default parameters. Our reference is Metis without reductions. We also compare our result with orderings from the `gord`-program from the software package Scotch (version 6.0.6) [46]. In evaluating our orderings we focus on the number of non-zeros in the matrix factors and the running time of the ordering algorithm. We obtain the number of non-zeros with the `gotst`-program from Scotch. This program performs a Cholesky factorization and reports statistics

on the elimination process. Some of our plots are performance profiles. These plots relate the running times or quality of all algorithms to the fastest/best algorithm on a per-instance basis. For each algorithm A, these ratios are sorted in increasing order. The plots show $\left(\frac{t_{\text{fastest}}}{t_{\text{A}}}\right)$ (in case of running time) or $\left(\frac{\phi_{\text{best}}}{\phi_{\text{A}}}\right)$ on the y-axis. A point close to zero shows that the algorithm was considerably slower/worse than the fastest/best algorithm.

We run compare our algorithm to the winning solver of the PACE 2017 challenge [17]. We run the solver using OpenJDK 11.0.6 with a time limit of 24 hours. The exact solver outputs the fill-edges of a minimum triangulation. We compare our solutions by computing the number of fill-edges.

Instances. We evaluate our algorithm on the large undirected graphs from [43]. These graphs include social networks, citation networks and web graphs compiled from [6] and [42]. These are complex networks of up to 1.38 million vertices with low diameter and are scale-free, having few high-degree nodes, many low-degree nodes. We also use the graphs from Walshaw's graph partitioning archive [51], which are mostly meshes and similar graphs, which are medium-sized networks of up to 448K vertices, generally have small degree, and are fairly symmetric, and road networks obtained from [7], which have up to 50.9 million vertices and uniformly low degree. Properties of our benchmark instances can be found in the appendix of the technical report [45]. We also evaluate our algorithm on the public and hidden instances of the second PACE challenge [17], and compare the results to the winning submission by Kobayashi and Tamaki.

Parameters and Abbreviations. We apply the reductions in a fixed order on each recursion level. The reductions are specified by their first letter; Δ for triangle contraction. We add a number to the configuration to specify the degree limit on simplicial nodes used for social networks. For example, $SD18$ means simplicial node reduction is applied before degree-2 elimination, with the degree limit set to 18 on the social network dataset. Note that we never use Reductions 4 and 1 together. After degree-2 elimination, path compression cannot reduce the graph and degree-2 elimination eliminates any nodes contracted by path-compression. Thus, using all reductions equates to the configuration $SITD\Delta$. Nodes with high degree can cause simplicial node reduction (Reduction 1) to be slow. Social networks tend to contain high-degree nodes, so we limit the degree of simplicial nodes on these graphs. On meshes and road networks such nodes do not cause problems. Thus, we do not limit the degree for meshes or road networks. See the technical report [45] for details on the choice of the degree limit. We use the default parameters for nested dissection in Metis. For Scotch we choose the default ordering strategy (option `-cq`), which emphasizes quality over speed.

6.1 Experimental Results. We now look at the performance of different reductions when used as a preprocessing step before running Metis. The time reported for our algorithm is the overall running time needed, i.e., compute the kernel, run Metis on the kernel, convert the solution on the kernel to a solution on the input graph. Figure 2 compares the results for different combinations of reductions and graph classes. We look at each graph class separately, i.e. social networks, mesh-like networks, and road networks. See the technical report [45] for results for each instance for configuration $SID\Delta 12$.

Social Networks. We first look at social networks. In general, reducing the graph before nested dissection yields significant speedups on most instances over nested dissection without any reductions. At the same time the number of non-zeros is also reduced.

With configuration $SID\Delta 12$ we obtain a speedup of 1.5 on average (see Table 2); the improvement in number of non-zeros is 1.06. This configuration yields the highest speedup and improvement in quality, on average. Note, that for the other configurations, the average speedup is greater than 1.35 on average. The social networks can be reduced to 57% of their original size, on average (see Table 2). Out of all graphs and configurations we observe the largest speedup of 3.92 for the instance `as-22july06`. The smallest speedup for this graph is 1.72 with configuration $SITP12$. Only two out of 21 of the social graphs do not benefit from the reductions in terms of speedup: on the instances `eu-2005` and `as-skitter` nested dissection with reductions is always slower than nested dissection without reductions. For `as-skitter` the speedup lies between 0.74 and 0.91, for `eu-2005` between 0.81 and 0.95. Out of all graphs and configurations the lowest speedup is 0.75 for instances `as-skitter` and `p2p-Gnutella04`, with configuration $SITP12$ in both cases. With configuration $SD18$ we observe a speedup of 1.03 for `p2p-Gnutella04`.

The largest improvement in number of non-zeros out of all graphs and configurations is 1.31 relative to Metis for the instance `coAuthorsCiteseer` with configuration $SITP12$. The speedup is 1.85 for this graph and configuration. Only on the instance `coPapersCiteseer` the number of non-zeros is not reduced when applying reductions. For this graph the number of non-zeros is 4% above that of Metis with configuration $SD18$. Here, the speedup is 1.19. On 13 of the social graphs the

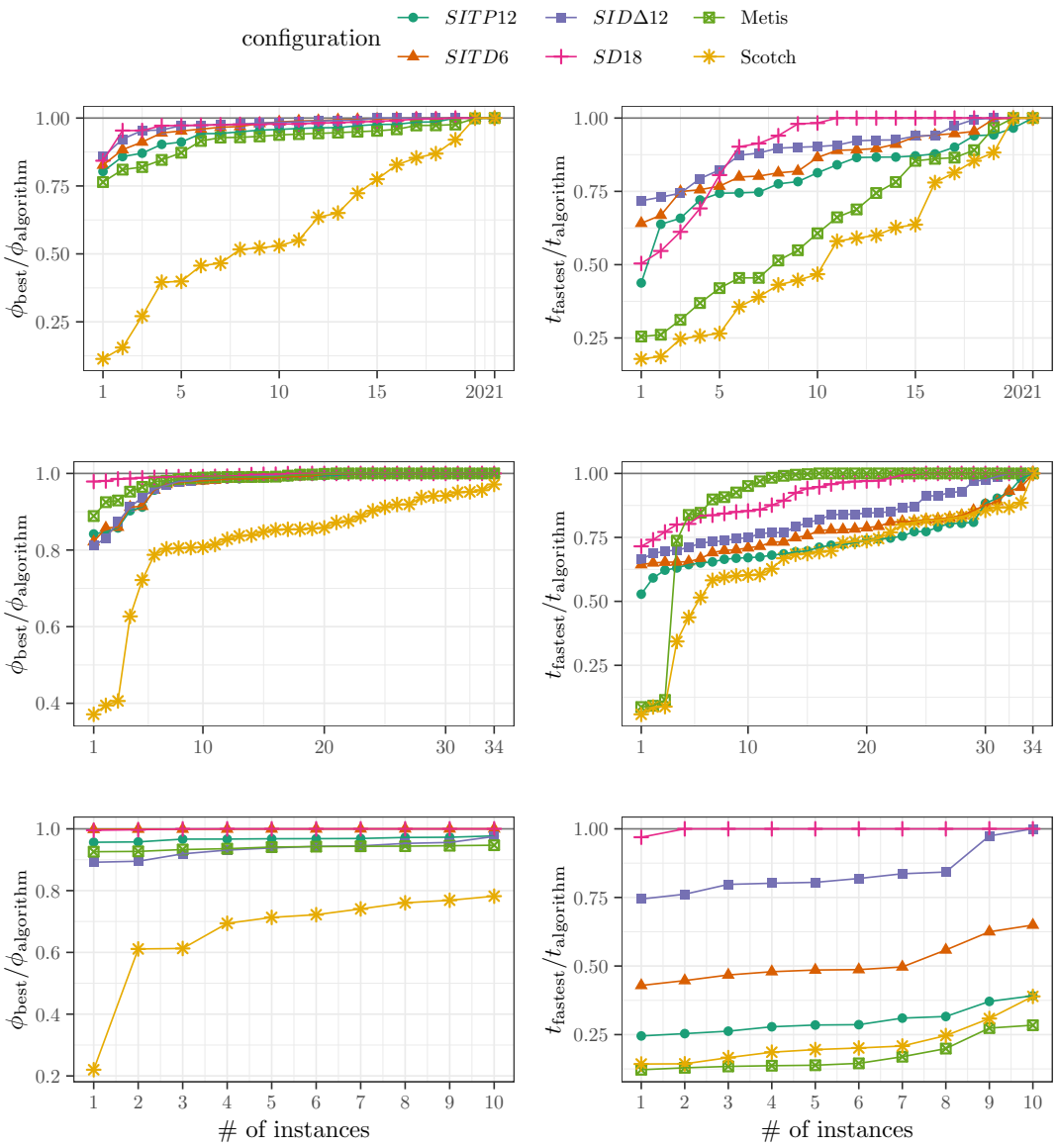

Figure 2: Performance plots for number of non-zeros (left) and running time (right) for different graph classes, from top to bottom: social graphs, meshes and road networks.

number of non-zeros is reduced by all of the configurations. The highest number of non-zeros we observe is 21% higher than that of Metis on the graph `eu-2005` using configuration $SITP12$.

For this graph class, the largest kernel has 96% of the nodes of the original graph and is obtained by configuration $SITP12$ for instance `p2p-Gnutella`. The smallest kernel has 25% of the nodes and is obtained by all configurations for instance `email-EuAll`.

Compared to Scotch and averaged over the social networks our algorithm is between 1.8 and 2.2 times faster than Scotch and produces orderings with an improvement between 2.13 and 2.23 in terms of the number of non-zeros.

Meshes. On the meshes, the reductions do not yield a speedup except for a few instances. Those instances are chordal graphs (`add20`, `add32`, `memplus`) and stiffness matrices (`bcsstk*`). Chordal graphs are reduced completely by simplicial node reduction. Here, we observe speedups between 6.9 (`add20`) and 11.5 (`memplus`). The stiffness matrices contain many indistinguishable nodes, so the graph size is reduced significantly. After applying simplicial node reduction and indistinguishable node reduction, `bcsstk29` is reduced to 72% of its original size and `bcsstk30` is reduced to 30% in terms of number of nodes. For these `bcsstk30/31/32` we obtain speedups between 1.08 and 1.36 with configuration $SID\Delta$. For `bcsstk29` and `bcsstk33` we do not observe a speedup with this configuration. Note that our reference, Metis without reductions, contracts indistinguishable nodes by default. When indistinguishable nodes are not contracted, our algorithm is more than 20% slower on these instances. On the other instances the reductions do not have a sufficient impact to reduce

Table 1: Avg. speedup $\mathcal{S}$, improvement in num. of non-zeros (nnz) and kernel size n' from simplicial node reduction and degree-2 elimination on the road networks.

Configuration	nnz	$\mathcal{S}$	n'
S	1.04	1.35	0.77
D	1.00	4.69	0.30
SD	1.06	6.03	0.20

Table 2: Top: Geometric means of the improvement in number of non-zeros (nnz) relative to Metis (larger is better) and speedup ($\mathcal{S}$) relative to Metis for different configurations. Bottom: Average number of nodes in the kernel and standard deviation σ (smaller is better).

Testset	Social		Meshes		Road	
Redu.	Number of non-zeros					
	nnz	$\mathcal{S}$	nnz	$\mathcal{S}$	nnz	$\mathcal{S}$
$SITP12$	1.03	1.35	0.99	0.93	1.03	1.79
$SITD6$	1.05	1.44	0.99	0.98	1.06	3.07
$SID\Delta12$	1.06	1.50	0.99	1.05	1.00	5.05
$SD18$	1.06	1.49	1.01	1.16	1.06	6.03
$SD\Delta12$	1.05	1.44	1.01	1.08	1.00	6.37
	Kernel Sizes					
	mean	σ	mean	σ	mean	σ
$SITP12$	0.57	0.23	0.83	0.32	0.37	0.18
$SITD6$	0.58	0.22	0.82	0.32	0.20	0.13
$SID\Delta12$	0.57	0.23	0.82	0.32	0.20	0.13
$SD18$	0.60	0.23	0.90	0.28	0.20	0.13
$SD\Delta12$	0.61	0.23	0.90	0.28	0.20	0.13

running time or number of non-zeros. On 6 instances configuration SD leads to speedups between 1.03 (`cs4`) and 1.18 (`uk`); `finan512` has a speedup of 1.11 with configuration $SID\Delta$. No configuration leads to a speedup greater than 1 on the remaining instances. The improvement in number of non-zeros ranges from 0.96 (`vibrobox`, configuration $SID\Delta$) to 1.07 (`fe_ocean`, configuration $SID\Delta$). The graphs are reduced by no more than 20%, on average (see Table 2).

Scotch is faster than Metis without reductions on a few instances, but slower in general. Its orderings lead to more non-zeros. Compared to Scotch, our algorithm is between 2 and 2.4 times faster and improves the number of non-zeros between 1.26 and 1.3 times.

Road Networks. Applying reductions to road networks leads to high speedups (see Figure 2) and improvements in quality (see Figure 2). The average speedups are between 1.79 and 6.37 (see Table 2). The number of non-zeros is improved between 1.03 and 1.06-fold. Road networks contain many degree-2 nodes, so degree-2 elimination is highly effective. After removing simplicial nodes and degree-2 nodes the `osm` instances retain less than 20% of their nodes; the instances `road_usa` and `road_central` are reduced to around 45% of their original size. Simplicial node reduction on its own yields a speedup of 1.35 and an improvement in number of non-zeros by 4% (see Table 1). Degree-2 elimination without simplicial node reduction does not improve the number of non-zeros, but leads to a 4.69-fold speedup. Reducing the road networks by both simplicial node reduction and degree-2 elimination (configuration SD) yields a 6-fold speedup on average (see Table 2), with the lowest speedup at 3.5 and the highest speedup at 8.2. This is also the highest speedup we observe. The number of non-zeros is improved by 1.06 on average with this configuration. While triangle contraction further improves the running time, it also leads to a larger number of non-zeros.

Configuration $SITP$ results in the lowest speedups, between 1.3 (`road_central`) and 2.2 (`asia.osm`). With configuration $SID\Delta$ the number of non-zeros is increased on 4 of the 10 road networks, however, never by more than 6%. Configuration SD improves the number of non-zeros the most, by up to 1.08 (`great-britain.osm`). On the road networks Scotch is consistently faster than Metis without reductions, but the quality of its orderings is significantly worse. Compared to Scotch, our algorithm is between 1.4 and 5.4 times faster whenever degree-2 elimination or path compression are used, on average. Otherwise, our algorithm is slower. The number of non-zeros is always improved, between 1.6 and 1.7 times.

Using All Reductions. The configuration $SITD\Delta$ uses all reductions. For this configuration degree limit 12 results in the best performance. For all graph classes, using all reductions is no better than using configuration $SID\Delta12$. The kernels obtained by the former are within 1% of the size of the kernels obtained by the latter, on average. This is not sufficient to reduce the running time. On the social networks, the speedup of configuration $SITD\Delta12$ is 1.44, on average, which is lower than the speedup of 1.5 obtained with configuration $SID\Delta12$. On the meshes, the speedup of configuration $SITD\Delta$ is 0.94; on the road networks it is 4.11, on average. The improvement in number of non-zeros does not change by more than 1.5% between the two configurations. Adding triangle contraction to the configuration $SITD$ yields the configuration $SITD\Delta$. With the configuration $SITD\Delta12$ we achieve a speedup of 4.11 on the road networks. The number of non-zeros is increased compared to Metis, the improvement being 0.99. On the social networks, the average speedup does not change and the improvement in number of non-zeros is reduced by less than 1%. On the meshes the number of non-zeros is not changed and the running time is

increased, with the average speedup at 0.94. Adding triangle contraction to configuration $SITD$ does not lead to an improvement in running time or quality. On road nets we get faster running time at the expense of quality.

Comparison with Exact Solutions. In this section we evaluate our algorithm on the 200 instances of the PACE 2017 challenge and compare it against the winning code of the PACE challenge by Kobayashi and Tamaki. We restrict the evaluation to the PACE challenge instances since the exact code could only solve the three chordal instances from the test set used above within a 30 minute time limit. The largest instance in the PACE challenge test set has roughly 30k nodes and 22k edges. We modified the code by Kobayashi and Tamaki to output the time needed to compute the fill-edges. Note that we are interested in node orderings, which can be computed from the fill-edges in linear time. We do not include the time of this postprocessing in the running time of the code since we did not tune the running time of this postprocessing. This means that our speedups are in practice a little bit larger than reported here.

On our machine, the exact solver solved 64 of the public instances and 58 of the hidden instances (122 instances in total) in under 24 hours. Our algorithm computes orderings on all 200 instances in this time limit. In fact, it takes less than a second on all instances. Figure 3 compares the number of fill-edges of our solutions and the exact solution. Nested dissection without reductions yields a minimum fill-in ordering for 3 instances. With reductions, we can solve between 29 instances (with configuration $SITD$) and 34 instances (with configuration $SID\Delta$) to optimality. There are 23 chordal instances that both algorithms can solve, of which only one our algorithm does not solve to optimality.

The reductions also reduce the fill-in of nested dissection orderings on non-chordal graphs. Nested dissection without reduction yields orderings with 151 more fill-edges than the optimum solution, on average. Using configuration SD this is reduced to 142 edges. The remaining three configurations improve the fill-in even further, yielding orderings with 106 more fill-edges than the optimum. On average, with our reductions we have between 29% ($SID\Delta$) and 43% (SD) more fill-edges than the optimum solution; nested dissection without reductions yields 67% more fill-edges.

The performance plot in Figure 3 clearly shows that (reduced) nested dissection is significantly faster than the exact algorithm. With all configurations we obtain a speedup of at least 4 over the exact algorithm; nested dissection without reduction yields a minimum speedup of 2. The low minimum speedup is due to the fact that the exact algorithm tests if the input

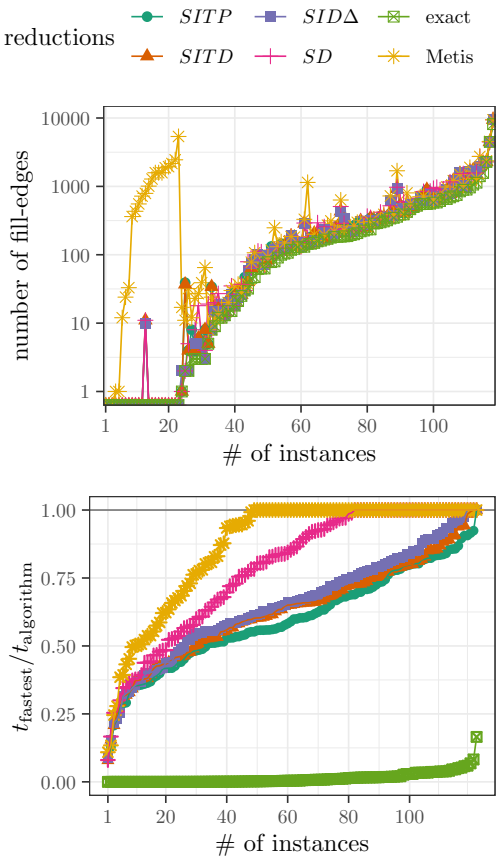

Figure 3: Top: number of fill-edges of orderings computed by our algorithms compared to the optimum fill-in computed by the exact algorithm of Kobayashi and Tamaki submitted to PACE 2017 [17] ordered by size of the optimum solution. The instances are sorted by the value of the exact solution. Bottom: performance plot for running time.

graph is chordal, which our algorithm does not do. Taking into account only the non-chordal instances, the minimum speedup is 13 with reductions and 12 without reductions. On average over all instances, the speedup over the exact algorithm is between 267 (configuration $SITP$) and 344 (configuration SD). For the non-chordal instances, the speedup over the exact algorithm is between 499 (configuration $SITP$) and 663 (configuration SD). No configuration speeds up nested dissection: the lowest speedup is 0.72 with configuration $SITP$ and the highest speedup is 0.95 with configuration SD.

7 Conclusion

By applying data reduction rules exhaustively we obtain improved quality *and* at the same time large improvements in running time on a variety of instances. This directly translates to improvements for typical applications. Overall, we arrive at a system that outperforms the state-of-the-art significantly.

On road networks we obtain orderings with lower fill-in six times faster than nested dissection alone. As orderings of such networks are used in preprocessing of shortest path algorithm like customizable contraction hierarchies, we believe that the additional reductions presented here can yield a significant speed up in the preprocessing time of such algorithms [18, 29].

We have so far not explored the use of these reduction rules in combinations with other algorithms for the minimum fill-in problem. However, the rules presented here are mostly independent of the underlying algorithm. In particular, eliminating simplicial nodes whenever possible appears to be very effective in reducing running time without harming the quality of the resulting ordering. Our implementation is part of the KaHIP framework, available at github.com/KaHIP/KaHIP.

References

[1] T. Akiba, Y. Iwata, Y. Sameshima, N. Mizuno, and Y. Yano. Cut tree construction from massive graphs. In *16th Intl. Conf. on Data Mining, ICDM 2016*, pages 775–780, 2016. doi:10.1109/ICDM.2016.0089.

[2] P. Amestoy, T. Davis, and I. Duff. An approximate minimum degree ordering algorithm. *SIAM J. Matrix Anal. Appl.*, 17(4):886–905, 1996. doi:10.1137/S0895479894278952.

[3] C. Ashcraft. Compressed graphs and the minimum degree algorithm. *SIAM J. Sci. Comput.*, 16(6): 1404–1411, 1995. doi:10.1137/0916081.

[4] C. Ashcraft and J. W. H. Liu. Generalized nested dissection: Some recent progress. In J. G. Lewis, editor, *Proceedings of the Fifth SIAM Conference on Applied Linear Algebra*, pages 130–134. SIAM, 1994.

[5] C. Ashcraft and J. W. H. Liu. Robust ordering of sparse matrices using multisection. *SIAM J. Matrix Anal. Appl.*, 19(3):816–832, 1998. doi:10.1137/S0895479896299081.

[6] D. Bader, A. Kappes, H. Meyerhenke, P. Sanders, C. Schulz, and D. Wagner. Benchmarking for Graph Clustering and Partitioning. In *Encyclopedia of Social Network Analysis and Mining*. Springer, 2014. doi:10.1007/978-1-4939-7131-2_23.

[7] D. Bader, H. Meyerhenke, P. Sanders, and D. Wagner, editors. *Proc. of the 10th DIMACS Impl. Challenge*, Cont. Mathematics, 2012. AMS. doi:10.1090/conm/588.

[8] U. Bertele and F. Brioschi. Contribution to non-serial dynamic programming. *J. Math. Anal. Appl.*, 28(2):313–325, 1969. doi:10.1016/0022-247X(69)90030-4.

[9] U. Bertele and F. Brioschi. A new algorithm for the solution of the secondary optimization problem in non-serial dynamic programming. *J. Math. Anal. Appl.*, 27(3):565–574, 1969. doi:10.1016/0022-247X(69)90137-1.

[10] C. Bichot and P. Siarry, editors. *Graph Partitioning*. Wiley, 2011.

[11] H. L. Bodlaender, P. Heggernes, and Y. Villanger. Faster parameterized algorithms for minimum fill-in. *Algorithmica*, 61(4):817–838, 2011. doi:10.1007/s00453-010-9421-1.

[12] A. Buluç, H. Meyerhenke, I. Safro, P. Sanders, and C. Schulz. Recent advances in graph partitioning. In L. Kliemann and P. Sanders, editors, *Algorithm Engineering: Selected Results and Surveys*, pages 117–158. Springer, 2016. doi:10.1007/978-3-319-49487-6_4.

[13] M. Cygan, F. V. Fomin, L. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.

[14] J. Dahlum, S. Lamm, P. Sanders, C. Schulz, D. Strash, and R. F. Werneck. Accelerating local search for the maximum independent set problem. In *Intl. Symp. on Experimental Algorithms*, pages 118–133. Springer, 2016. doi:10.1007/978-3-319-38851-9_9.

[15] T. A. Davis and Y. Hu. The university of florida sparse matrix collection. *ACM Trans. Math. Softw.*, 38(1):1–25, 2011. doi:10.1145/2049662.2049663.

[16] T. A. Davis, S. Rajamanickam, and W. M. Sid-Lakhdar. A survey of direct methods for sparse linear systems. *Acta Numer.*, 25:383—-566, 2016. doi:10.1017/S0962492916000076.

[17] H. Dell, C. Komusiewicz, N. Talmon, and M. Weller. The PACE 2017 Parameterized Algorithms and Computational Experiments Challenge: The Second Iteration. In *12th International Symposium on Parameterized and Exact Computation (IPEC 2017)*, volume 89 of *LIPIcs*, pages 30:1–30:12, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.IPEC.2017.30.

[18] J. Dibbelt, B. Strasser, and D. Wagner. Customizable contraction hierarchies. *ACM Journal of Experimental Algorithmics*, 21(1):1.5:1–1.5:49, 2016. doi:10.1145/2886843.

[19] R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Spinger, 1999. doi:10.1007/978-1-4612-0515-9.

[20] F. V. Fomin and Y. Villanger. Subexponential parameterized algorithm for minimum fill-in. *SIAM Journal on Computing*, 42(6):2197–2216, 2013. doi:10.1137/11085390X.

[21] F. V. Fomin, I. Todinca, and Y. Villanger. Large induced subgraphs via triangulations and CMSO. *SIAM Journal on Computing*, 44(1):54–87, 2015. doi:10.1137/140964801.

[22] A. George. Nested dissection of a regular finite element mesh. *SIAM J. Numer. Anal.*, 10(2):345–363, 1973. doi:10.1137/0710032.

[23] A. George and J. W. H. Liu. A quotient graph model for symmetric factorization. In I. S. Duff and G. W. Stewart, editors, *Sparse Matrix Proceedings 1978*, pages 154–175. SIAM, 1978.

[24] A. George and J. W. H. Liu. An automatic nested dissection algorithm for irregular finite element problems. *SIAM J. Numer. Anal.*, 15(5):1053–1069, 1978. doi:10.1137/0715069.

[25] A. George and J. W. H. Liu. A fast implementation of the minimum degree algorithm using quotient graphs. *ACM Trans. Math. Softw.*, 6(3):337–358, 1980. doi:10.1145/355900.355906.

[26] A. George and J. W. H. Liu. The evolution of the minimum degree ordering algorithm. *SIAM Rev.*, 31(1):1–19, 1989. doi:10.1137/1031001.

[27] A. George and J. W. Liu. The evolution of the minimum degree ordering algorithm. *SIAM Review*, 31(1):1–19, 1989. doi:10.1137/1031001.

[28] G. H. Golub and C. F. Van Loan. *Matrix Computations*. Johns Hopkins Studies in the Mathematical Sciences. Johns Hopkins University Press, 2013.

[29] L. Gottesbüren, M. Hamann, T. N. Uhl, and D. Wagner. Faster and better nested dissection orders for customizable contraction hierarchies. *Algorithms*, 12(9):196, 2019. doi:10.3390/a12090196.

[30] B. Hendrickson and E. Rothberg. Improving the run time and quality of nested dissection ordering. *SIAM J. Sci. Comput.*, 20(2):468–489, 1998. doi:10.1137/S1064827596300656.

[31] M. Henzinger, A. Noe, C. Schulz, and D. Strash. Practical minimum cut algorithms. In *Proc. of the 20th Workshop on Algorithm Engineering and Experiments, ALENEX*, pages 48–61, 2018. doi:10.1137/1.9781611975055.5.

[32] M. Henzinger, A. Noe, and C. Schulz. Shared-memory exact minimum cuts. In *International Parallel and Distributed Processing Symposium, IPDPS*, pages 13–22. IEEE, 2019. doi:10.1109/IPDPS.2019.00013.

[33] M. Henzinger, A. Noe, and C. Schulz. Shared-memory branch-and-reduce for multiterminal cuts. In *Proceedings of the Twenty-First Workshop on Algorithm Engineering and Experiments, ALENEX 2020*, pages 42–55. SIAM, 2020. doi:10.1137/1.9781611976007.4.

[34] D. Hespe, C. Schulz, and D. Strash. Scalable kernelization for maximum independent sets. In *Proc. of the 20th Workshop on Algorithm Engineering and Experiments, ALENEX*, pages 223–237, 2018. doi:10.1137/1.9781611975055.19.

[35] D. Hespe, S. Lamm, C. Schulz, and D. Strash. WeGotYouCovered: The winning solver from the PACE 2019 Implementation Challenge, vertex cover track. In *2020 Proceedings of the SIAM Workshop on Combinatorial Scientific Computing*, pages 1–11. SIAM, 2020. doi:10.1137/1.9781611976229.1.

[36] H. Kaplan, R. Shamir, and R. E. Tarjan. Tractability of parameterized completion problems on chordal and interval graphs: Minimum fill-in and physical mapping. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pages 780–791, Nov 1994. doi:10.1109/SFCS.1994.365715.

[37] H. Kaplan, R. Shamir, and R. E. Tarjan. Tractability of parameterized completion problems on chordal, strongly chordal, and proper interval graphs. *SIAM Journal on Computing*, 28(5):1906–1922, 1999. doi:10.1137/S0097539796303044.

[38] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Sci. Comput.*, 20(1):359–392, 1998. doi:10.1137/S1064827595287997.

[39] S. Kratsch. Recent developments in kernelization: A survey. *Bulletin of the EATCS*, 113, 2014. URL `http://eatcs.org/beatcs/index.php/beatcs/article/view/285`.

[40] S. Lamm, C. Schulz, D. Strash, R. Williger, and H. Zhang. Exactly solving the maximum weight independent set problem on large real-world graphs. In *Proc. of the 21st Workshop on Algorithm Engineering and Experiments, ALENEX 2019*, pages 144–158, 2019. doi:10.1137/1.9781611975499.12.

[41] D. LaSalle and G. Karypis. Efficient nested dissection for multicore architectures. In *Euro-Par 2015: Parallel Processing*, pages 467–478. Springer, 2015. doi:10.1007/978-3-662-48096-0_36.

[42] J. Leskovec and A. Krevl. SNAP Datasets: Stanford large network dataset collection. `http://snap.stanford.edu/data`, 2014.

[43] H. Meyerhenke, P. Sanders, and C. Schulz. Partitioning complex networks via size-constrained clustering. In J. Gudmundsson and J. Katajainen, editors, *Experimental Algorithms*, pages 351–363. Springer, 2014. doi:10.1007/978-3-319-07959-2_30.

[44] T. Ohtsuki, L. K. Cheung, and T. Fujisawa. Minimal triangulation of a graph and optimal pivoting order in a sparse matrix. *J. Math. Anal. Appl.*, 54(3):622–633, 1976. doi:10.1016/0022-247X(76)90182-7.

[45] W. Ost, C. Schulz, and D. Strash. Engineering data reduction for nested dissection. *CoRR*, abs/2004.11315, 2020. URL `https://arxiv.org/abs/2004.11315`.

[46] F. Pellegrini. Scotch. Version 6.0.6, 2020. URL `https://www.labri.fr/perso/pelegrin/scotch/`.

[47] D. Rose, R. E. Tarjan, and G. S. Lueker. Algorithmic aspects of vertex elimination on graphs. *SIAM J. Comput.*, 5(2):266–283, 1976. doi:10.1137/0205021.

[48] D. J. Rose. Triangulated graphs and the elimination process. *J. Math. Anal. Appl.*, 32:597–609, 1970. doi:10.1016/0022-247X(70)90282-9.

[49] D. J. Rose. A graph-theoretic study of the numerical solution of sparse positive definite systems of linear equations. In R. C. Read, editor, *Graph Theory and Computing*, pages 183–217. Academic Press, 1972. doi:10.1016/B978-1-4832-3187-7.50018-0.

[50] C. Schulz and D. Strash. Graph partitioning: Formulations and applications to big data. In *Encyclopedia of Big Data Technologies*. Springer, 2019. doi:10.1007/978-3-319-63962-8_312-2.

[51] A. J. Soper, C. Walshaw, and M. Cross. A combined evolutionary search and multilevel optimisation approach to graph-partitioning. *J. Global. Optim.*, 29(2):225–241, 2004. doi:10.1023/B:JOGO.0000042115.44455.f3.

[52] D. A. Spielman. Algorithms, graph theory, and the solution of laplacian linear equations. In *Automata, Languages, and Programming - 39th International Colloquium, ICALP*, LNCS, pages 24–26, 2012. doi:10.1007/978-3-642-31585-5_5.

[53] H. Tamaki. Positive-instance driven dynamic programming for treewidth. In *25th European Symposium on Algorithms, ESA'17*, volume 87 of *LIPIcs*, pages 68:1–68:13, 2017. doi:10.4230/LIPIcs.ESA.2017.68.

[54] W. F. Tinney and J. W. Walker. Direct solutions of sparse network equations by optimally ordered triangular factorization. *Proc. IEEE*, 55(11):1801–1809, 1967. doi:10.1109/PROC.1967.6011.

[55] M. Yannakakis. Computing the minimum fill-in is NP-complete. *SIAM J. Algebraic Discrete Methods*, 2(1):77–79, 1981. doi:10.1137/0602010.

A Proofs Omitted from the Main Text

Proof. [Proof of Theorem 4.1] Since $\Gamma(x)$ is a clique, $D(x) = \emptyset$. The fill-in associated with eliminating x first is $\phi(G, x\Sigma(G_x)) = |D(x)| + \Phi(G_x) = \Phi(G_x)$. From (2.1) it follows that $\phi(G, x\Sigma(G_x)) = \Phi(G)$. ∎

Proof. [Proof of Lemma 4.1] Let $x \in \Gamma(a) \setminus \{b\} = \Gamma(b) \setminus \{a\}$ be eliminated from G. In the elimination graph $\Gamma_{G_x}(a) = (\Gamma(a) \setminus \{x\}) \cup \Gamma(x)$ and $\Gamma_{G_x}(b) = (\Gamma(b) \setminus \{x\}) \cup \Gamma(x)$. Since $a \in \Gamma_{G_x}(b)$ and $b \in \Gamma_{G_x}(a)$, $\Gamma_{G_x}[a] = \Gamma_{G_x}[b]$. Thus, a and b are indistinguishable in G_x.

If a node y with $y \notin \Gamma(a)$ and $y \notin \Gamma(b)$ is eliminated from G, the neighborhoods of a and b do not change, since $a, b \notin \Gamma(y)$. In the elimination graph $\Gamma_{G_y}[a] = \Gamma_{G_y}[b]$. Thus, a and b are indistinguishable in G_y. ∎

Proof. [Proof of Theorem 4.2] Lemma 4.1 implies that all pairs of nodes in A are indistinguishable in all graphs in the elimination sequence. Let $a \in A$ be the node that is eliminated before all other nodes in A. There is a graph $G^{(m)}$ in the elimination sequence with a minimum ordering $a\Sigma(G_a^{(m)})$, $a \in A$. For all $b \in A \setminus \{a\}$ $\Gamma_{G_a^{(m)}}(b)$ is a clique, i.e., these nodes are simplicial after elimination of a. Thus, $A\Sigma(G_A^{(m)})$ is a minimum ordering of $G^{(m)}$ and G has a minimum ordering of the form of σ'. ∎

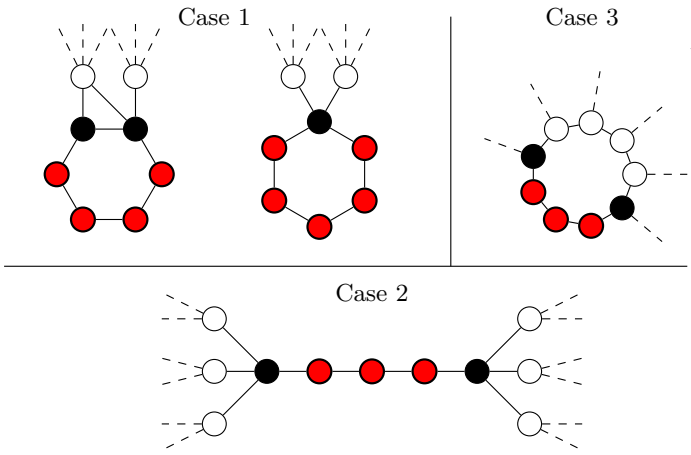

Figure 4: Examples for the three cases in the proof of Theorem 4.4. Red nodes are nodes in P, black nodes are in $\Gamma(P)$. Dashed edges lead to some other nodes in the graph.

Proof. [Proof of Theorem 4.3] If a node $x \in \Gamma(a) = \Gamma(b)$, is eliminated, a and b form a clique in the elimination graph G_x. Thus, a and b are indistinguishable in G_x and Theorem 4.2 holds. If a node $x \notin \Gamma(a) \cup \{a, b\}$ is eliminated, the neighborhoods of nodes a and b do not change, i.e., $\Gamma_{G_x}[a] = \Gamma_G[a]$ and $\Gamma_{G_x}[b] = \Gamma_G[b]$. Thus, a and b are twins in G_x. If a is eliminated, $\Gamma_{G_a}(b)$ is a clique in the elimination graph G_a and b is simplicial in G_a. With Theorem 4.1, $b\Sigma((G_a)_b)$ is a minimum ordering of G_a and $ab\Sigma((G_a)_b)$ is a minimum ordering of G. ∎

Proof. [Proof of Lemma 4.5] Let $\mathcal{C} = \{C_1, \ldots, C_n\}$ be the set of induced cycles that contain a, i.e., for all i, $a \in C_i$ and $G[C_i]$ is a cycle. Since a has degree two, $\Gamma(a) \subset C_i$ for all i. By Lemma 4.3, for all $C_i \in \mathcal{C}$, there exists a minimum triangulation T_i of $G[C_i]$ with $\Gamma(a) \in T_i$. Thus, there exists a minimum triangulation $\hat{T}$ of G with $\Gamma(a) \in \hat{T}$. $\Gamma(a)$ is a separation clique with components $\{a\}$ and $V \setminus (\{a\} \cup \Gamma(a))$ in the triangulated graph $\hat{G} = (V, E \cup \hat{T})$. By Lemma 4.2 there exists no edge $\{a, x\} \in \hat{T}$. This implies $\Gamma(a) \in \hat{T}$, $\{a, x\} \notin \hat{T}$ and $\hat{T}$ is minimum. ∎

Proof. [Proof of Lemma 4.4] With Lemma 4.5 there exists a minimum triangulation $\hat{T}$ of G with $\Gamma(a) \in \hat{T}$ and $\{a, x\} \notin \hat{T}$. a is simplicial in the triangulated graph $\hat{G} = (V, E \cup \hat{T})$ and $a\Sigma(\hat{G}_a)$ is a minimum ordering of $\hat{G}$. This implies that $a\Sigma(G_a)$ is a minimum ordering of G. Note that eliminating a from G adds the edge $\Gamma(a)$ to the elimination graph. ∎

Proof. [Proof of Theorem 4.4] G can be decomposed into non-disjoint graphs $G' := G[V \setminus P]$ and $G'' := G[P \cup \Gamma(P)]$, such that $G = G' \cup G''$. We distinguish three cases (see Figure 4 for examples):

Case 1: If $a_0 = a_{k+1}$ or $a_0 \in \Gamma(a_{k+1})$, then G'' is a cycle and $\Gamma(P)$ is a separation clique with leaves G' and G''. Let T' be a minimum triangulation of G' and T'' be a minimum triangulation of G''. By Lemma 4.2, $T' \cup T''$ is a minimum triangulation of G. Since any ordering of G'' generates a minimum triangulation of G'' (by Lemma 4.3), $P\Sigma(G''_P)$ is a minimum ordering of G'' and $P\Sigma(G_P)$ is a minimum ordering of G.

Case 2: If $a_0 \neq a_{k+1}$, and $\{a_0\}$ and $\{a_{k+1}\}$ are separation cliques, then all nodes in P are also separation cliques. By Lemma 4.2, there are no edges $\{a_i, a_j\}$, for all $i \neq j$ in a minimum triangulation of G.

Let Σ be any minimum fill-in ordering of G and let $G^{(m)}$ be the graph in the elimination sequence from which $a \in P$ is eliminated. Node a is simplicial in $G^{(m)}$, otherwise $T(\Sigma)$ would not be a minimum triangulation. Since all $a \in P$ are separation cliques and $\deg(a) = 2$ in G, $\deg(a) = 1$ in $G^{(m)}$.

Without loss of generality assume that a_1 is eliminated before all other nodes in P. Let $G^{(m_1)}$ be the graph in the elimination sequence from which a_1 is eliminated. If $\deg(a_1) = 1$ in $G^{(m_1)}$, then $\deg(a_2) = 1$ in $G_{a_1}^{(m_1)}$. Repeating this argument for all $a_i \in P$ proves that $P\Sigma(G_P^{(m_1)})$ is a minimum ordering of $G^{(m_1)}$ and Σ is of the form of σ'.

Case 3: If $\{a_0\}$, $\{a_{k+1}\}$ and $\Gamma(P)$ are not separation cliques, then any $a \in P$ satisfies the conditions in Lemma 4.4. In G_a, $\{a_0\}$, $\{a_{k+1}\}$ and $\Gamma(P)$ are not separation cliques. Repeating the argument for G_a leads to a minimum ordering $P\Sigma(G_P)$.

In Case 1 and Case 3, there exists a minimum ordering $a_1 \cdots a_k x_1 \cdots x_\ell$. In Case 2, there exists a minimum ordering $x_1 \cdots x_i a_1 \cdots a_k x_{i+1} \cdots x_\ell$. Both orderings are of the form of σ'. ∎

Proof. [Proof of Corollary 4.1] Node x is part of a cycle and thus not a separation clique. Either case 1 or 3 of Theorem 4.4 holds, which implies that $x\Sigma(G_x)$ is a minimum ordering of G. ∎

Proof. [Proof of Corollary 4.2] Since x is a separation clique, Case 2 of Theorem 4.4 holds and thus, x is simplicial in $G^{(i)}$. ∎