universität wien

MASTERARBEIT / MASTER'S THESIS

Titel der Masterarbeit / Title of the Master‘s Thesis

„Reduced Nested Dissection for Fill Reducing Node Orderings“

verfasst von / submitted by

Wolfgang Martin Ost, BSc ETH MSc ETH

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of

Master of Science (MSc)

Wien, 2019 / Vienna 2019

Studienkennzahl lt. Studienblatt / degree programme code as it appears on the student record sheet:	A 066 910
Studienrichtung lt. Studienblatt / degree programme as it appears on the student record sheet:	Computational Science
Betreut von / Supervisor:	Univ.-Prof. Dr. Monika Henzinger
Mitbetreut von / Co-Supervisor:	Dipl.-Math. Dipl.-Inform. Dr. Christian Schulz Prof. Darren Strash, Hamilton College

web-version, modified titlepage

Abstract

When factorizing sparse matrices, non-zeros can be introduced. These non-zeros are called fill-in. If the fill-in is large, factorization can become prohibitively expensive in terms of storage and computation time. A permutation of a matrix can reduce the fill-in and make factorization feasible. The minimum fill-in problem is to find a permutation that minimizes the fill-in. It is commonly solved using a graph representation of the matrix.

We introduce reduction rules for the minimum fill-in problem and apply them in a nested dissection algorithm we call reduced nested dissection. Reducing the graphs reduces the time to compute node separators, which speeds up the nested dissection algorithm.

We evaluate the performance of reduced nested dissection on a set of social networks, citation networks and web graphs. Our reductions initially reduce the graphs to approximately half their size. They improve the number of non-zeros by 2.5% and the operation count of factorization by 5% over nested dissection without reductions. The running time is reduced by 45% on average, with the best improvement at 95.6%.

We also introduce an algorithm for the minimum fill-in problem based on graph clustering, intended to allow for faster computation of node orderings compared to nested dissection. Orderings from this algorithm lead to orders of magnitudes more non-zeros compared to nested dissection, and the running time is reduced only slightly over reduced nested dissection.

Zusammenfassung

Bei der Faktorisierung von dünn besetzten Matrizen entstehen oft neue Nicht-Nullen. Diese neuen Elemente werden als Fill-In bezeichnet. Ist der Fill-In groß, kann die Faktorisierung im Hinblick auf Speicherbedarf und Laufzeit teuer werden. Eine Permutation der Matrix kann den Fill-In reduzieren und die Faktorisierung ermöglichen. Das Problem, eine Permutation zu finden, die den Fill-In minimiert, wird üblicherweise mit einem graphtheoretischen Ansatz gelöst.

Wir entwickeln einen Algorithmus auf der Basis von Nested Dissection, genannt Reduced Nested Dissection. Mit Hilfe von Reduktionsregeln verkleinern wir die Graphen und verringern damit die Zeit, die benötigt wird, um Trenner zu finden. Damit wird der Nested Dissection Algorithmus signifikant beschleunigt.

Wir evaluieren Reduced Nested Dissection anhand von sozialen Netzwerken und ähnlichen Graphen. Mit unseren Reduktionen verringert sich die Anzahl an Nicht-Nullen in den Matrixfaktoren um 2.5% und die Anzahl an Operationen in der Faktorisierung um 5%. Die Laufzeit verbessert sich im Schnitt um 45%, mit einer maximalen Verbesserung von 95.6%.

Weiterhin führen wir einen Algorithmus für das Fill-In Problem ein, der auf Clustering von Graphen aufbaut. Das Ziel ist, Permutationen in geringerer Zeit als mit Reduced Nested Dissection zu erhalten. Dieser Algorithmus führt zu Permutationen mit Anzahl an Nicht-Nullen und Operationen die um Größenordnungen über Ergebnissen von Nested Dissection liegen. Er ist nur wenig schneller als Reduced Nested Dissection.

Acknowledgment

I am thankful to Dr. Christian Schulz and Prof. Darren Strash for their guidance and valuable insight. Our weekly meetings were always helpful.

The thesis would not have been possible without Prof. Monika Henzinger's official supervision.

Thanks also go to my friends who were a great source of motivation and moral support over the last two years.

I am grateful to my parents for supporting me through all my studies and enabling me to pursue this second Master's degree. I would not be where I am without their continued patience and support.

Contents

Chapter 1

Introduction

Solving sparse linear systems of equations

$$Ax = b \tag{1.0.1}$$

is a fundamental task in scientific computing. Such equations arise in a variety of applications, such as computational fluid dynamics, structural engineering, economic modeling and circuit simulation [16].

Sparse linear systems can be solved by direct methods [17, 23]. Such methods decompose the matrix A into factors that simplify the solution of the system. The drawback is, that such factors can become dense, i.e., they have many more non-zeros than the original matrix [17, 23, 41]. Then, solving the system is 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, this fill-in can be significantly reduced, leading to sparse factors [17, 23, 41].

For symmetric positive definite matrices[1] we can reorder rows and columns by a symmetric permutation $PAP^\top$ [23, 41]. The minimum fill-in problem is to find a permutation matrix P, such that the number of non-zeros introduced during factorization is minimized.

This problem can be solved using a graph theoretic approach introduced by Porter [37] and Rose [41]. A symmetric matrix $A = (a_{ij})_{i,j=1}^n$ is represented by an undirected graph G, where nodes represent the rows and columns of A. Nodes i, j in G are connected by an edge if the matrix element $a_{ij} \neq 0$. An elimination step in A is reflected in G by removing the node corresponding to the eliminated column and connecting its neighborhood to form a clique. These added edges provide an upper bound to the number of non-zeros introduced in an elimination step. By minimizing this bound, we obtain the desired permutation matrix P. Figure 1.1 illustrates this model of elimination.

[1]which can be factored by Cholesky factorization [23]

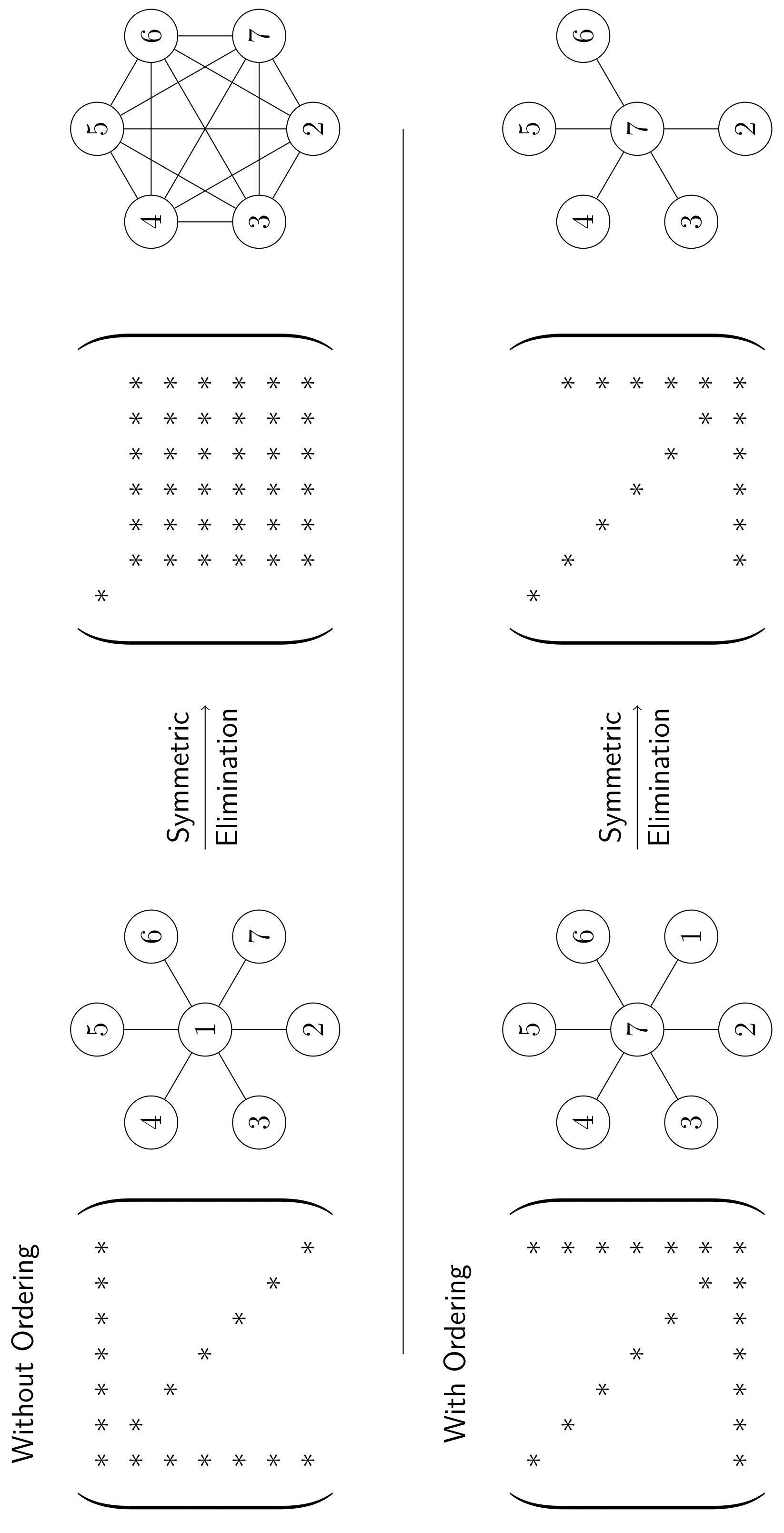


Figure 1.1: A matrix and the corresponding graph before and after a step of symmetric Gaussian elimination. Above: unordered matrix, below: permuted matrix. Note that permuting the first and seventh column corresponds to exchanging nodes 1 and 7. Without the permutation the matrix is filled in completely. With the permutation no fill-in is generated during the factorization.

The minimum fill-in problem is NP-complete [53], so heuristics such as the minimum degree algorithm [41, 50] and nested dissection [18] are commonly used. The minimum degree algorithm is a greedy scheme, eliminating the minimum degree node at every step. Nested dissection computes a node separator and orders the subgraphs recursively.

1.1 Our Contribution

We introduce reduction rules to reduce the graph size, allowing for faster computation of fill-reducing orderings, while maintaining or even improving their quality. We extend nested dissection by applying these reductions at every level, which reduces the time to compute the node separator. We also describe a new algorithm for the fill-in problem based on graph clustering.

Our reduction rules initially reduce the graphs to approximately half their size. The running time of nested dissection is reduced by approximately 45%. Our reductions reduce the number of non-zeros and operation count by around 2.5% and 5%, respectively.

The clustering based algorithm leads to higher fill-in and operation count than nested dissection. It is faster than nested dissection and often as fast or faster than nested dissection with reductions.

1.2 Structure of the Thesis

The thesis is structured as follows. Chapter 2 introduces the graph theoretic basics and formally defines node orderings and the fill-in problem. Chapter 3 provides an overview over related problems and algorithms. Chapter 4 introduces our algorithm and the reduction rules. Our experiments to evaluate our implementation are described in Chapter 5. Chapter 6 provides an overview over future challenges and concludes the thesis.

Chapter 2

Fundamentals

An *undirected graph* $G = (V, E)$ is a set of *nodes* V connected by a set E of undirected *edges*. The set of nodes adjacent to a node $x \in V$ form its *open neighborhood* $N_G(x) := \{y \in V \mid \{x, y\} \in E\}$. The *closed neighborhood* $N_G[x]$ also contains the node itself: $N_G[x] := N_G(x) \cup \{x\}$. The size of a node's neighborhood is its *degree* $\deg(x) := |N_G(x)|$. For a set of nodes $A \subseteq V$ we define its neighborhood $N_G(A) := (\cup_{x \in A} N_G(x)) \setminus A$. When clear from the context we omit G and write $N(x)$, $N[x]$ and $N(A)$, respectively.

Let $a, b \in V$. Node a *dominates* node b if $N[b] \subseteq N[a]$. If $N[a] = N[b]$, then a and b are *indistinguishable*. Nodes a and b are *twins* if $N(a) = N(b)$.

Given a set of nodes $A \subseteq V$, the set of edges with both endpoints in A is $E(A) := \{\{a, b\} \in E \mid a, b \in A\}$. The set A induces a subgraph $G[A] := (A, E(A))$.

A graph $G = (V, E)$ is *complete* if all nodes are connected by an edge. A *clique* is a set of nodes $C \subseteq V$ that induces a complete graph $G[C]$. An *edge clique cover* is a set of cliques $\mathcal{K}$, such that for every edge $\{x, y\} \in E$, there is a clique $C \in \mathcal{K}$ with $x, y \in C$.

A set of nodes $X \subseteq V$ is an *induced cycle*, if its induced subgraph $G[X]$ is a cycle.

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 *node separator* of a graph $G = (V, E)$ is a set of nodes $S \subset V$ whose removal separates V into disjoint sets V_1 and V_2, such that there are no edges between V_1 and V_2. 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 called a *separation clique*. Usually, we are interested in small separators for which $|V_1| \approx |V_2|$. To this end, we introduce a *balance constraint*: $|V_i| \leq (1+\varepsilon)\lceil |V|/2 \rceil$, for some parameter $\varepsilon \geq 0$.

The *graph partitioning problem* is related to the node separator problem: Find sets $V_1, \ldots, V_k$, such that $V_1 \cup \cdots \cup V_k = V$ and $V_i \cap V_j = \emptyset$ for $i \neq j$. The set of *cut edges* $C = \{\{x,y\} \in E \mid x \in V_i, y \in V_j, i \neq j\}$ is called *edge separator*. A node $x \in V_i$ with a neighbor $y \in V_j,\ i \neq j$ is a *boundary node*. The set of boundary nodes is a node separator. In the graph partitioning problem, the objective is usually to minimize the size of the edge separator. As in the node separator problem, a balance constraint ensures that the partitions are of similar size: $\forall\ i \in \{1, \ldots, k\} : |V_i| \leq (1+\varepsilon)\lceil |V|/k \rceil$, with $\varepsilon \geq 0$.

A *clustering* of a graph is a partitioning where the number of partitions k is unknown and there is no balance constraint. Formally, we can define a cluster as a set of nodes that are connected in some sense. However, there is a wide variety of possible definitions [40].

In a graph $G = (V, E)$, a *matching* is a set of edges $M \subseteq E$ where no two edges in M share a node, i.e., for all $e_i, e_j \in E$ $e_i \cap e_j = \emptyset$. An *independent set* is a set of nodes $I \subseteq V$ where no two nodes in I are connected by an edge, i.e., for all $x, y \in I$ $\{x,y\} \notin E$. A set $K \subseteq V$ is a *vertex cover* if every edge $e \in E$ is incident to at least one node in K.

A *reduction rule* for some problem is a polynomial-time transformation of a graph $G = (V, E)$ into a *reduced graph* $G' = (V', E')$, where $|V'| \leq |V|$ and $|E'| \leq |E|$. A solution to the problem on G' can be mapped to a solution on G. If an optimal solution on G' is mapped to an optimal solution on G, then the reduction rule is *exact*. Otherwise, it is *inexact*.

Often, we use *contractions* to reduce a graph. Contracting an edge $\{x,y\} \in E$ merges the nodes x and y, summing the node weights. Any parallel edges in the transformed graph are replaced by a single edge, again summing the weights. Similarly, we contract a set of nodes $X \subseteq V$ by merging all nodes in X and replacing parallel edges with a single edge. *Uncontracting* an edge or a set of nodes reverts the contraction.

2.1 Symmetric Factorization

Consider a linear system $Ax = b$, where $A \in \mathbb{C}^{n \times n}$ and A is self-adjoint and positive definite. Cholesky factorization decomposes A into a lower triangular matrix L and its transpose $L^\top$, such that $A = LL^\top$. The system $LL^\top x = b$ can then be solved by back and forward substitution. LDL-factorization introduces an additional diagonal factor D and decomposes $A = LDL^\top$. In LDL-factorization the matrix L is unit lower triangular.

To compute the Cholesky factorization we write A as

$$A = \begin{pmatrix} a & w^\top \\ w & B \end{pmatrix}, \tag{2.1.1}$$

where $w \in \mathbb{C}^{n-1}, B \in \mathbb{C}^{(n-1)\times(n-1)}$. Then column and row elimination yields

$$A = \underbrace{\begin{pmatrix} 1 & 0 \\ \frac{w}{a} & I \end{pmatrix}}_{L_1'} \underbrace{\begin{pmatrix} a & 0 \\ 0 & B - \frac{ww^\top}{a} \end{pmatrix}}_{A_1'} \underbrace{\begin{pmatrix} 1 & \frac{w^\top}{a} \\ 0 & I \end{pmatrix}}_{L_1'^\top}, \tag{2.1.2}$$

where I is the identity matrix. The matrix A_1' can be further factorized and we obtain

$$A = \underbrace{\begin{pmatrix} \sqrt{a} & 0 \\ \frac{w}{\sqrt{a}} & I \end{pmatrix}}_{L_1} \underbrace{\begin{pmatrix} 1 & 0 \\ 0 & B - \frac{ww^\top}{a} \end{pmatrix}}_{A_1} \underbrace{\begin{pmatrix} \sqrt{a} & \frac{w^\top}{\sqrt{a}} \\ 0 & I \end{pmatrix}}_{L_1^\top}. \tag{2.1.3}$$

By repeating this process on the submatrix $B - \frac{ww^\top}{\sqrt{a}}$ we obtain a factorization $L_1 L_2 \cdots L_n I L_n^\top \dots L_2^\top L_1^\top$, where $L_1 L_2 \cdots L_n = L$.

By omitting the factorization step in (2.1.3), we obtain an LDL-factorization. Then $A = L_1' L_2' \cdots L_n' D L_n'^\top \cdots L_2'^\top L_1'^\top$ with $L = L_1' L_2' \cdots L_n'$.

2.2 Node Ordering

Let $G = (V, E)$ be a graph with vertices V and edges E.

Definition 2.2.1. The *deficiency* $D_G(x)$ of a node x in a graph G is the set of distinct pairs of nodes in $N(x)$, that are not themselves neighbors:

$$D_G(x) := \{\{a, b\} \mid a, b \in N(x),\ a \neq b,\ a \notin N(b)\}.$$

When clear from the context we omit G and write $D(x)$.

Definition 2.2.2. Eliminating a node x from a graph $G = (V, E)$ results in the *elimination graph* G_x:

$$G_x := (V \setminus \{x\}, E_x),$$

where $E_x := E(V \setminus \{x\}) \cup D_G(x)$.

The elimination graph is obtained by removing x and its incident edges from G, and connecting the neighbors of x to form a clique.

The elimination graph obtained by eliminating a set of nodes $X \subseteq V$ is defined as

$$G_X := (\ldots((G_{x_1})_{x_2})\ldots)_{x_m}, \tag{2.2.1}$$

where $x_i \in X$, $i = 1, \ldots, m$.

Definition 2.2.3. A node ordering of a graph $G = (V, E)$ with $n = |V|$ is a bijection $\sigma : \{1, 2, \ldots, n\} \to V$.

An ordering σ of a graph G defines a sequence of elimination graphs $G^{(1)}G^{(2)}\ldots G^{(n)}$, where

$$G^{(i)} := \begin{cases} (G^{(i-1)})_{\sigma(i)} & \text{if } i = 1, \ldots, n \\ G & \text{if } i = 0. \end{cases} \tag{2.2.2}$$

In $G^{(n)}$, all nodes have been eliminated, i.e., $G^{(n)} = (\emptyset, \emptyset)$.

The *fill-in* associated with an ordering is defined as

$$\phi(G, \sigma) := \sum_{i=1}^{n} |D_{G^{(i-1)}}(\sigma(i))|. \tag{2.2.3}$$

We denote the minimum fill-in ordering of a graph G by

$$\Sigma(G) = \arg\min_{\sigma}\{\phi(G, \sigma)\}, \tag{2.2.4}$$

with the minimum fill-in $\Phi(G) = \phi(G, \Sigma(G))$. Note that

$$\Phi(G) \geq \Phi(G^{(1)}) \geq \ldots \geq \Phi(G^{(n-1)}). \tag{2.2.5}$$

An ordering σ of a graph G generates a triangulation of G

$$T(\sigma) := \bigcup_{i=1}^{n} D_{G^{(i-1)}}(\sigma(i)). \tag{2.2.6}$$

A minimum fill-in ordering $\Sigma(G)$ generates a *minimum triangulation* $T(\Sigma(G))$, where $\Phi(G) = |T(\Sigma(G))|$ [36]. A triangulation T is a *minimal triangulation* if no proper subset of T is also a triangulation. An ordering that generates a minimal triangulation is *minimal.* If G is triangulated, then it has a perfect elimination order, i.e., $\Phi(G) = 0$.

We use the following notation for node orderings:

$$\sigma = x_1 x_2 \cdots x_n \tag{2.2.7}$$

corresponds to

$$\begin{aligned} \sigma(1) &= x_1, \\ \sigma(2) &= x_2, \\ &\vdots \\ \sigma(n) &= x_n. \end{aligned} \tag{2.2.8}$$

We write $x\Sigma(G_x)$ if x is to be eliminated before the nodes in G_x. Whenever a set of nodes $P = \{p_1, p_2, \ldots, p_n\}$ can be eliminated in any order, we use P in the notation instead of $p_1 p_2 \cdots p_n$. For example

$$P\Sigma(G_P) \tag{2.2.9}$$

is an ordering in which the nodes in P are eliminated in any order before the nodes in G_P.

Chapter 3

Related Work

In this chapter, we give an overview over algorithms for the minimum fill-in problem and describe two of them in more detail. Then, we shortly review a common approach to computing separators and discuss reduction rules. Lastly, we describe the label propagation algorithm.

3.1 Minimum Fill-In Orderings

Yannakakis has proven that the problem of finding a minimum fill-in ordering is NP-complete [53]. Exact algorithms have been introduced in the context of non-serial dynamic programming [7, 8], but they are not practical for large matrices due to their exponential running time [41]. For graphs with a perfect elimination order, the problem can be solved in $\mathcal{O}(|V| + |E|)$ time [43].

Tinney and Walker [50] 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 the node with the smallest degree is selected to be eliminated at each step [41]. There have been several improvements to this algorithm, both in its design and implementation [19, 20, 22].

A significant part of the minimum degree algorithm's runtime is spent in updating node degrees. Most of the improvements to the minimum degree algorithm are thus focused on reducing the number of nodes to update [22]. Amestoy et al. [3] introduce 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 [41, 50]: 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 [41].

In 1973, George [18] 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 [21]. 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 [5, 6, 29]. A similar approach based on multisectors instead of bisectors is presented in [6].

In some variants of these algorithms, the notion of indistinguishable nodes is used to build a compressed graph, which speeds up the computation of an ordering [4, 6, 26].

Node orderings from heuristic algorithms provide a basis to find minimal triangulations [25]. This, in turn, makes it possible to locally optimize node orderings [9, 36]. While this does not guarantee that the resulting orderings are minimum, the fill-in can be reduced significantly in some cases [9]. Unlike a minimum triangulation, a minimal triangulation can be found in polynomial time [25].

Node ordering algorithms also have applications in areas where minimum triangulations are of interest, such as in Bayesian networks [31] and computer vision [14].

For asymmetric matrices, reordering columns can lead to a reduced number of non-zeros in the matrix factors [15]. The minimum degree algorithm for symmetric matrices is based on an algorithm proposed by Markowitz [34] for computing the inverse of asymmetric matrices arising in linear programming. A minimum node ordering for the symmetric matrix $A^\top A$ can be used as an ordering for the asymmetric matrix A, but algorithms that do not rely on this product also exist, such as the column approximate minimum degree algorithm introduced by Davis et al. [15].

3.2 The Minimum Degree Algorithm

The minimum degree algorithm (Algorithm 1) is a greedy algorithm to compute a node ordering based on the degree of the nodes [22, 41]. At each step of the algorithm the node with the smallest degree is eliminated from the current elimination graph. Since the degree of a node corresponds to the number of non-zeros in the corresponding matrix column, the degree of an eliminated node gives an estimate of the operation count in the elimination step. Thus, the minimum degree algorithm minimizes the operation

Algorithm 1: The minimum degree algorithm

```
input : An undirected graph $G = (V, E)$
output: An ordering $\sigma$

MinDegree($G$)
1 $i \leftarrow 1$
2 $G^{(0)} \leftarrow G$
3 while $i \leq |V|$ do
4     $x \leftarrow \arg\min_{a \in V} \deg_{G^{(i-1)}}(a)$
5     $G^{(i)} \leftarrow G_x^{(i-1)}$
6     $\sigma(i) \leftarrow x$
```

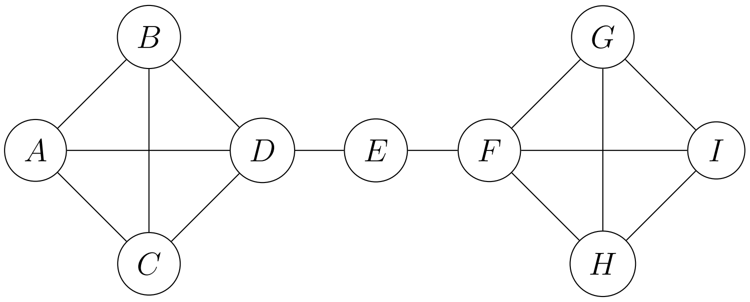


Figure 3.1: For this graph, the minimum degree ordering is not optimal. Eliminating in alphabetic order would incur no fill-in, but the minimum degree algorithm eliminates E first.

count of the factorization in each elimination step. It also tends to minimize the fill-in, since the degree of a node x also gives an upper bound for the fill-in, with $|D(x)| = \mathcal{O}(\deg(x)^2)$.

However, orderings from the minimum degree algorithm are not generally optimal. The graph in Figure 3.1 has a perfect elimination order, and thus, zero fill-in. However, the minimum degree ordering has fill-in, since node E is eliminated first. In fact, the fill-in of a minimum degree ordering can be arbitrarily greater than the minimum fill-in [41]. See Figure 3.2 for an example.

A straightforward implementation of the minimum degree algorithm would explicitly construct the elimination graph in every iteration. Since the fill-in can be arbitrarily greater than the minimum fill-in, such an implementation could require far more memory than is required for the input graph. The generalized element model [19, 22, 48] provides a representation of elimination graphs that solves this problem. In this model, graphs are represented by an edge clique cover. A trivial edge clique cover of a graph $G = (V, E)$ is the set E.

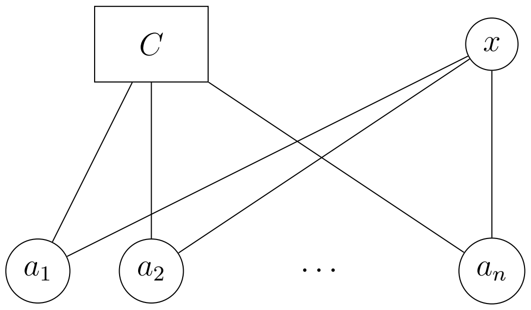


Figure 3.2: A graph for which the minimum degree ordering has fill-in arbitrarily greater than the minimum fill-in, adapted from [41]. C is a clique with $m \geq n$ nodes. Each node a_i is adjacent to each node in C. The minimum degree algorithm eliminates x first, which leads to a fill-in of $\frac{n(n-1)}{2}$. An ordering where all nodes a_i are eliminated first has fill-in m.

Eliminating a node x changes the edge clique cover. Let $\{C_1, \ldots, C_k\}$ be the cliques that contain x. In the edge clique cover of the elimination graph $C_1, \ldots, C_k$ are replaced by the clique $(\bigcup_{i=1}^{k} C_i) \setminus \{x\}$.

Implementations based on this approach have the advantage that the memory required to represent the elimination graphs never exceeds the amount of memory required for the original graph.

There are several techniques to improve the running time of the minimum degree algorithm [22]. They focus mostly on avoiding computing the node degree in the elimination graph. *Mass elimination* and *indistinguishable nodes* make it possible to eliminate multiple neighboring nodes at the same time. *Element absorption* reduces the number of cliques being processed: If there are cliques C_1 and C_2 in the edge clique cover where $C_1 \subseteq C_2$, then C_1 can be discarded. With *multiple elimination* an independent set of minimum degree nodes is eliminated instead of just a single node. Lastly, *incomplete degree update* delays the update of the degree of dominating nodes.

3.3 Nested Dissection

Let $G = (V, E)$ be an undirected graph. Nested dissection (Algorithm 2) computes a node ordering of a graph G by first computing a separator S separating V into subsets V_1 and V_2 and then recursively ordering $G[V_1]$, $G[V_2]$ and $G[S]$. If the number of nodes in the graph is below some recursion limit, they are ordered by some other algorithm, usually the minimum degree algorithm.

For square grids [18] and planar graphs [32] the number of non-zeros introduced by a nested dissection ordering for a graph with n nodes is bounded by $\mathcal{O}(n^2 \log_2 n)$, where n is the number of variables

Algorithm 2: Nested dissection

```
input : An undirected graph $G = (V, E)$
output: An ordering $\sigma$
NestedDissection($G$)
1 if $|G| \geq$ *recursion limit* then
2     $V_1, V_2, S \leftarrow$ Separator($G$)
3     foreach $G'$ in $(G[V_1], G[V_2], G[S])$ do
4         $\sigma' \leftarrow$ NestedDissection($G'$)
5         $\sigma \leftarrow \sigma\sigma'$
6 else
7     $\sigma \leftarrow$ MinDegree($G$)
```

in the linear system. For square grids it can be shown that the triangular factors have at least $\mathcal{O}(n^2 \log_2 n)$ non-zeros [27]. However, no such bound exists for general graphs [21].

On square grids finding the separator is straightforward. On planar graphs it can be found in linear time [33]. In general, though, computing the separator is the most time consuming step in nested dissection. Ordering a reduced version of the graph should improve the running time of the algorithm, without degrading the quality of the ordering.

3.4 Node Separators

Multilevel algorithms are the most common approach to solving the node separator problem on large graphs [10, 29, 44, 46]. These algorithms consist of three phases. First, the input graph is transformed into a coarser graph. After computing a separator on the coarse graph, the solution is transferred back to the input graph and optimized locally.

In the coarsening phase edges are contracted. The edges to contract are selected by computing a maximum weight matching. Nodes connected by an edge in the matching are merged, and any parallel edges are replaced by a single edge. This process is applied repeatedly, leading to coarser and coarser graphs. To obtain a coarse graph that is representative of the input graph, an edge rating function can be applied to guide the contraction.

Once the coarse graph is small enough, an initial separator is computed. A common way is to use the boundary nodes of an edge separator with partitions V_1 and V_2. The set of boundary nodes in V_1 is a separator, so is the set of boundary nodes in V_2. In general, a vertex cover of the graph induced by the cut edges is a separator [39, 46]. This usually results in a better separator than simply selecting the boundary nodes in V_1 or V_2.

When uncontracting the matchings, the node separators are refined by local search at each step.

3.5 Reduction Rules

Reduction rules play an important role in algorithms for NP-hard problems. They were originally used to reduce the running time of brute force algorithms, e.g., for the maximum independent set problem [49].

Reductions are fundamental to algorithms for fixed parameter tractable problems [1]. A problem is fixed parameter tractable with parameter k, if it has an algorithm with running time $\mathcal{O}(f(k)n^c)$, where n is the problem size, c is some constant and $f(k)$ is an arbitrary function. The vertex cover problem is fixed parameter tractable, if stated as a decision problem: is there a vertex cover with at most k nodes?

One approach to solve such problems is kernelization. Here, the graph is reduced by reduction rules until a smaller graph is obtained. If this reduced graph is bounded by some function of the parameter, it is called a kernel. Solving the problem on the kernel is equivalent to solving it on the original graph. For the vertex cover, a kernel can be obtained by removing nodes x with $\deg(x) > k$ and their incident edges, because these nodes are always in a vertex cover of size k [1, 11].

Branch-and-reduce methods offer another approach to NP-hard problems. These algorithms reduce the problem instance using reduction rules, then branch into multiple subinstances that are then solved recursively. Examples for such methods include algorithms for the independent set problem [30, 51, 52], the vertex cover problem [2, 12] and the dominating set problem [28].

We already gave an example for a reduction rule for the vertex cover problem. A further example is the twin reduction used for maximum independent sets [51] and minimum vertex covers [2]: in this context, twins are nodes a, b with $\deg(a) = \deg(b) = 3$ and $N(a) = N(b)$. Depending on $N(\{a, b\})$, the set $\{a, b\} \cup N(\{a, b\})$ can either be removed or contracted. This is a special case of the crown reduction for the vertex cover problem [13].

3.6 Label Propagation

Label propagation is an algorithm introduced by Raghavan et al. [40] to detect clusters in graphs. Here, we only describe our implementation of the algorithm, although other variations exist.

The algorithm begins by assigning a unique label to each node. Then, the labels are updated repeatedly, until they no longer change. In each step of the algorithm the nodes are updated in random order. Each node is assigned the label that a maximum number of its neighbors share. Ties are broken randomly. If a node's label is already shared by a majority of its neighbors, the label is not updated.

Label propagation allows for fast cluster detection. Each iteration takes time $\mathcal{O}(m)$ for a graph with m edges and the algorithm typically converges after a few iterations [40].

Chapter 4

Reduced Nested Dissection

We now introduce our nested dissection algorithm with reductions, which we call reduced nested dissection. The chapter is structured as follows. After outlining the algorithm we describe our exact reduction rules in Section 4.1 and our inexact reduction rules in Section 4.2. Section 4.3 introduces our node ordering algorithm based on graph clustering.

Algorithm 3: Reduced nested dissection

```
input : An undirected graph $G = (V, E)$
output: An ordering $\sigma$

ReducedNestedDissection($G$)
1 $G' \leftarrow$ ReduceGraph($G$)
2 if $|G| \geq$ *recursion limit* then
3     $V_1, V_2, S \leftarrow$ Separator($G$)
4     foreach $G'$ in $(G[V_1], G[V_2], G[S])$ do
5         $\sigma' \leftarrow$ ReducedNestedDissection($G'$)
6         $\sigma \leftarrow \sigma\sigma'$
7 else
8     $\sigma \leftarrow$ MinDegree($G$)
9 $\sigma \leftarrow$ map ordering $\sigma$ from $G'$ to $G$
```

Algorithm 3 outlines our algorithm. We extend nested dissection by transforming the input graph G to a reduced graph G' and then continuing as in nested dissection. After an ordering has been found it is mapped back to the original graph G.

If the transformation of the graph is fast relative to the computation of the separator and the transformed graph G' is significantly smaller than G,

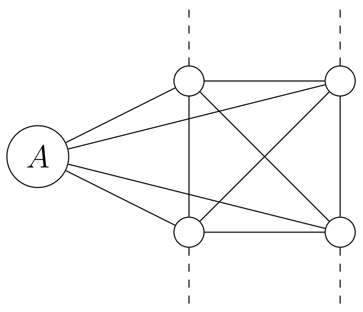


Figure 4.1: A is a simplicial node, since its neighborhood is a clique. The dashed edges lead to some other nodes in the graph.

then reduced nested dissection should be faster than pure nested dissection. In Sections 4.1 and 4.2 we introduce our reduction rules.

4.1 Exact Reductions

A *reduction rule* is a polynomial time transformation from a graph G to a reduced graph G', such that a minimum ordering of G' can be extended to a minimum ordering of G in polynomial time. For a minimum ordering of G', $\Sigma(G')$, there is a corresponding minimum ordering of G, $\Sigma'(G)$.

4.1.1 Simplicial Nodes

Definition 4.1.1. A node x is *simplicial* if its neighborhood $N(x)$ is a clique (see Figure 4.1 for an example).

Theorem 4.1.2. *No new edges are added during elimination of a simplicial node x.*

Proof. Since $N(x)$ is a clique, $D(x) = \emptyset$. By definition of the elimination graph (see Definition 2.2.2), no new edges are added when eliminating x. □

Theorem 4.1.3. *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.*

Proof. From Definition 4.1.1 it follows that $D(x) = \emptyset$. The fill-in associated with eliminating x first is $\phi(G, x\Sigma(G_x)) = |D(x)| + \Phi(G_x) = \Phi(G_x)$. From (2.2.5) it follows that $\phi(G, x\Sigma(G_x)) = \Phi(G)$. □

This allows us to eliminate all simplicial nodes first by the following procedure:

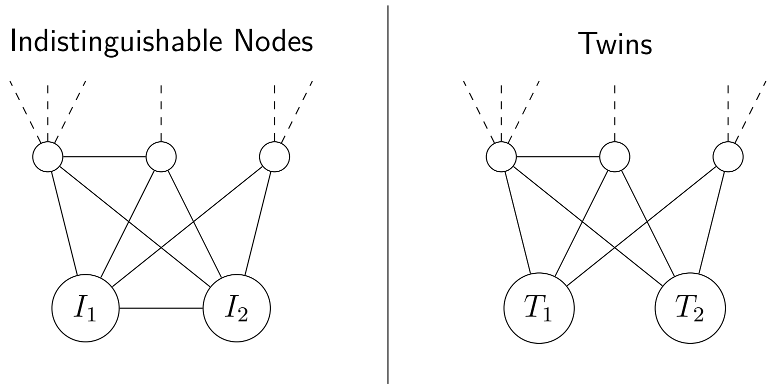


Figure 4.2: Examples for indistinguishable nodes and twins. Nodes I_1 and I_2 are indistinguishable, since they are neighbors and connected to all unlabeled nodes by an edge, i.e., $N[I_1] = N[I_2]$. Nodes T_1 and T_2 are twins, since they are both connected to all unlabeled nodes, but not to each other. $N(T_1) = N(T_2)$.

1. Find any simplicial node x in $G = (V, E)$.
2. Eliminate x from G and place it next in the node ordering.
3. If the elimination graph G_x has simplicial nodes, 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 [41, 42].

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.

4.1.2 Indistinguishable Nodes

Definition 4.1.4. Two nodes a and b are *indistinguishable* if $N[a] = N[b]$.

Figure 4.2 shows an example for indistinguishable nodes. We now show that such nodes can be eliminated together: if a and b are indistinguishable nodes, then there exists a minimum fill-in ordering $x_1 \cdots x_i a b x_{i+1} \cdots x_\ell$.

Lemma 4.1.5. *If a, b are indistinguishable nodes in a graph G, then a and b are indistinguishable in any elimination graph G_x for $x \notin \{a, b\}$.*

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

If a node x with $x \notin N(a)$ and $x \notin N(b)$ is eliminated from G, the neighborhoods of a and b do not change, since $a, b \notin N(x)$. In the elimination graph $N_{G_x}[a] = N_{G_x}[b]$. Thus, a and b are indistinguishable in G_x. □

Lemma 4.1.6. *Let a, b be indistinguishable nodes in a graph $G = (V, E)$. If $a\Sigma(G_a)$ is a minimum ordering of G, then $ab\Sigma((G_a)_b)$ is also minimum ordering of G. $\phi(G, a\Sigma(G_a)) = \phi(G, ab\Sigma((G_a)_b)) = \Phi(G)$.*

Proof. Since $N_G[a] = N_G[b]$, $N_{G_a}[b] = N_G(a)$. Due to the elimination process, $N_{G_a}(b)$ is a clique. Thus, b is simplicial in G_a. With Theorem 4.1.3, $b\Sigma((G_a)_b)$ is a minimum ordering of G_a. Thus, if $a\Sigma(G_a)$ is a minimum ordering of G, $ab\Sigma((G_a)_b)$ is a also minimum ordering of G. □

Theorem 4.1.7. *Let $G = (V, E)$ be a graph with a set of nodes $A \subseteq V$, where $\forall\ a_i, a_j \in A,\ N[a_i] = N[a_j]$. 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 $\phi(G, \sigma') = \Phi(G)$.*

Proof. Lemma 4.1.5 implies that all pairs of nodes in A are indistinguishable in all graphs in the elimination sequence. There is a graph $G^{(m)}$ in the elimination sequence with a minimum ordering $a\Sigma(G_a^{(m)})$, $a \in A$. By Lemma 4.1.6, $a_1 \cdots a_k\Sigma(G_A^{(m)})$ is also a minimum ordering of $G^{(m)}$. In fact, any $A\Sigma(G_A^{(m)})$ is a minimum ordering of $G^{(m)}$. Thus, G has a minimum ordering of the form of σ'. □

Theorem 4.1.7 implies that indistinguishable nodes can be treated as one node: if a node is removed, all its indistinguishable nodes can be removed next. To obtain a reduced graph G', we contract a set of indistinguishable nodes S in G to one node x.

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 [19, 20, 22]. In implementations of the minimum degree algorithm based on the quotient graph model, indistinguishable nodes are represented by a single node, similar to Reduction 2. This reduction is also used in other variants of nested dissection and the minimum degree algorithm [4, 26].

4.1.3 Twins

Definition 4.1.8. Two nodes a and b are *twins* if $N(a) = N(b)$.

Figure 4.2 shows an example of twins. Similar to indistinguishable nodes, twins can be eliminated together.

Theorem 4.1.9. *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 $\phi(G, \sigma') = \Phi(G)$.*

Proof. If a node $x \in N(a) = N(b)$, 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 4.1.7 holds.

If a node $x \notin N(a) \cup \{a, b\}$ is eliminated, the neighborhoods of nodes a and b do not change, i.e., $N_{G_x}[a] = N_G[a]$ and $N_{G_x}[b] = N_G[b]$. Thus, a and b are twins in G_x.

If a is eliminated, $N_{G_a}(b)$ is a clique in the elimination graph G_a and b is simplicial in G_a. With Theorem 4.1.3, $b\Sigma((G_a)_b)$ is a minimum ordering of G_a and $ab\Sigma((G_a)_b)$ is a minimum ordering of 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 in G' is different to the deficiency of the corresponding node 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.

4.1.4 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 triangulations and minimum fill-in orderings. Corollary 1 and Proposition 2 from [41] are central to our proof and we restate them here.

Lemma 4.1.10 (Corollary 1 from [41])**.** *Let $G = (V, E)$ be a graph with separation clique S with components $C_1, C_2, \dots, C_k$. 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 with $e = \{x, y\} \in T$ with $x \in C_j$ and $y \in S$.*

Lemma 4.1.11 (Proposition 2 from [41])**.** *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 4.1.12. *Let $G = (V, E)$ be a graph with a node $a \in V$ where $\deg(a) = 2$, $N(a) \notin E$ and $\{a\}$ is not a separation clique. Then, $a\Sigma(G_a)$ is a minimum ordering of G.*

To prove Lemma 4.1.12 we establish that there exists a minimum triangulation that does not contain an edge to such a node a.

Lemma 4.1.13. *Let G and a be as in Lemma 4.1.12. There exists a minimum triangulation $\hat{T}$ of G, with $N(a) \in \hat{T}$ and $\{a, x\} \notin \hat{T}$ for all $x \in V$.*

Proof. Let $\mathcal{C} = \{C_1, \dots, C_n\}$ 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. Due to the assumptions on a, $N(a) \subset C_i$ for all i.

By Lemma 4.1.11, for all $C_i \in \mathcal{C}$, there exists a minimum triangulation T_i with $N(a) \in T_i$. Thus, there exists a minimum triangulation $\hat{T}$ of G with $N(a) \in \hat{T}$.

$N(a)$ is a separation clique with components $\{a\}$ and $V \setminus (\{a\} \cup N(a))$ in the triangulated graph $\hat{G} = (V, E \cup \hat{T})$. By Lemma 4.1.10 there exists no edge $\{a, x\} \in \hat{T}$.

This implies $N(a) \in \hat{T}$, $\{a, x\} \notin \hat{T}$ and $\hat{T}$ is minimum. □

Proof of Lemma 4.1.12. With Lemma 4.1.13 there exists a minimum triangulation $\hat{T}$ of G with $N(a) \in \hat{T}$ 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 $N(a)$ to the elimination graph. □

With these results we can now prove our original statement.

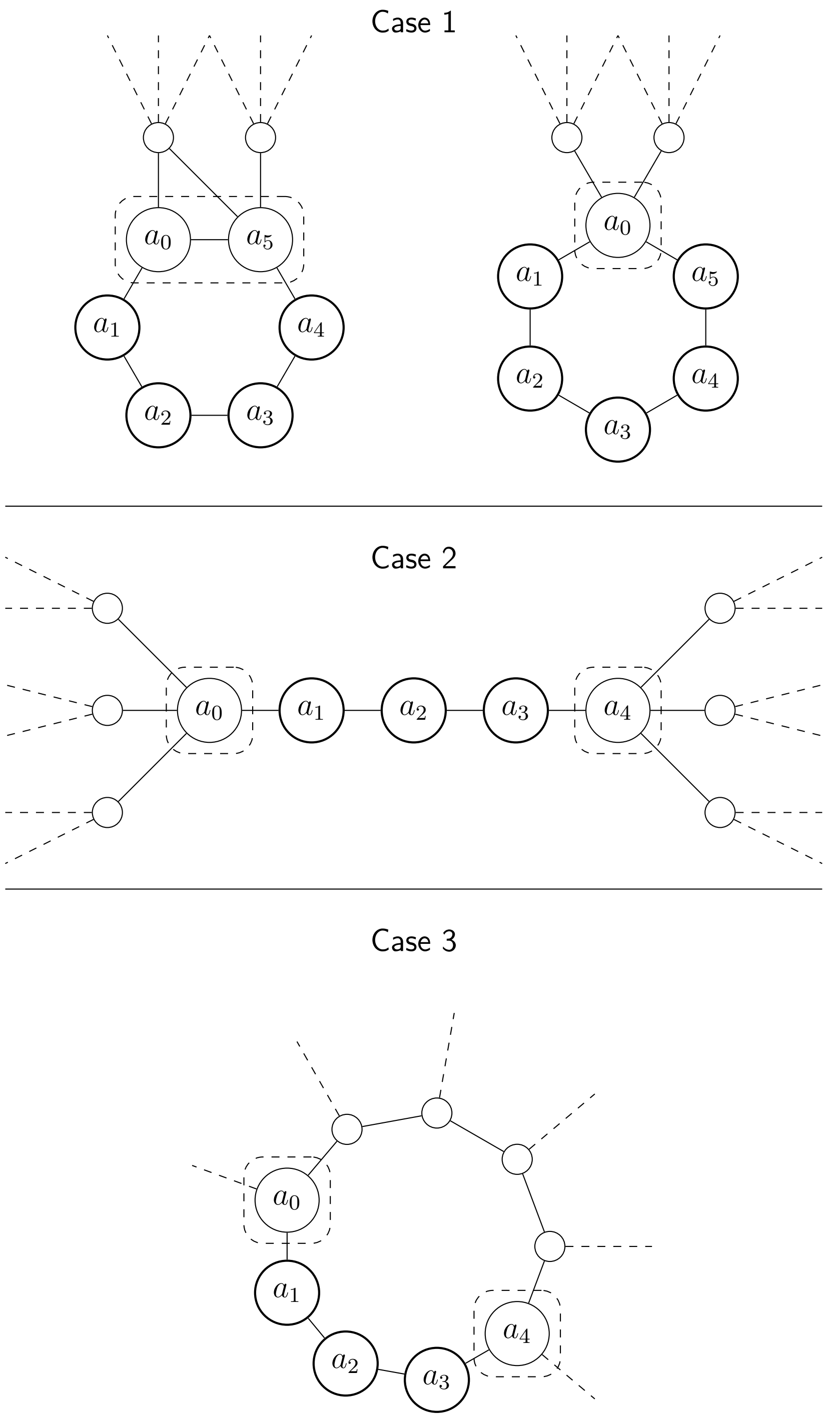


Figure 4.3: Examples for the three cases in the proof of Theorem 4.1.14. Thick nodes are nodes in P. $N(P)$ is marked with dashed rectangles. Dashed edges lead to some other nodes in the graph.

Theorem 4.1.14. *Let $G = (V, E)$ and $P = \{a_1, \dots, a_k\} \subseteq V$ such that $G[P]$ is a path graph and $\forall\ a \in P\ \deg(a) = 2$. Let $N(P) = \{a_0, a_{k+1}\}$ and $N(a_i) = \{a_{i-1}, a_{i+1}\}$, $i = 1, \dots, k$. There exists an ordering*

$$\sigma' = x_1 \cdots x_i a_1 \cdots a_k x_{i+1} \cdots x_\ell,$$

where $V \setminus P = \{x_1, \dots, x_\ell\}$, such that $\phi(G, \sigma') = \Phi(G)$.

Proof. G can be decomposed into non-disjoint graphs $G' := G[V \setminus P]$ and $G'' := G[P \cup N(P)]$, such that

$$G = G' \cup G''. \tag{4.1.1}$$

We distinguish three cases (see Figure 4.3 for examples):

Case 1: If $a_0 = a_{k+1}$ or $a_0 \in N(a_{k+1})$, then G'' is a cycle and $N(P)$ 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 4.1.10, $T' \cup T''$ is a minimum triangulation of G. Since any ordering of G'' generates a minimum triangulation of G'' (by Lemma 4.1.11), $P\Sigma(G''_P)$ 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 4.1.10, 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 $G^{(m_1)}_{a_1}$. Repeating this argument for all $a_i \in P$ proves that $P\Sigma(G^{(m_1)}_P)$ 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 $N(P)$ are not separation cliques, then any $a \in P$ satisfies the conditions in Lemma 4.1.12. In G_a, $\{a_0\}$, $\{a_{k+1}\}$ and $N(P)$ 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'. □

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.

4.2 Inexact Reductions

An *inexact reduction rule* is a transformation from a graph G to a reduced graph G', where minimum ordering of G' is not guaranteed to correspond to a minimum ordering of G. A minimum ordering of G', $\Sigma(G')$, corresponds to an ordering of G, $\sigma'(G)$, where $\phi(G, \sigma'(G)) \geq \Phi(G)$.

While such reductions do not guarantee a minimum ordering, they are useful in reducing the graph size and thus the running time of the reduced nested dissection. In practice, they have little effect on the quality of the node orderings.

4.2.1 Degree-2 Elimination

In a graph without simplicial nodes, the minimum degree is 2. These nodes would be eliminated first by the minimum degree algorithm, so we eliminate them from the graph.

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 reduction is applied recursively until no nodes with degree 2 are left.

Just as with the minimum degree algorithm, degree-2 elimination can lead to a non-minimum ordering (see Figure 4.4). In some cases, degree-2 elimination can lead to a worse ordering than the minimum degree algorithm. Consider node D in Figure 4.4. When A, B and C are eliminated, D has

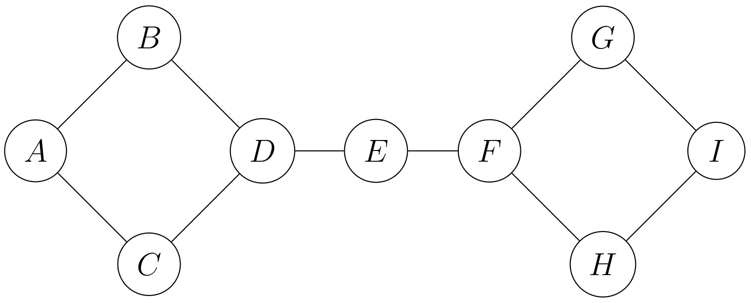


Figure 4.4: Applying degree-2 elimination to this graph can lead to a non-minimum ordering: the order $[A, B, C, D, E, \dots]$ has lower fill-in than the order $[E, A, B, C, D, \dots]$, but both are possible orderings from degree-2 elimination.

degree one and D and E can be eliminated without fill-in. However, degree-2 elimination eliminates E before D, leading to a higher fill-in.

The proof of Theorem 4.1.14 has interesting implications on the exactness of degree-2 elimination.

Corollary 4.2.1. *Let $G = (V, E)$ be a graph. If $x \in V$ is part of any cycle $C \subseteq V$ and $\deg(x) = 2$, $x\Sigma(G_x)$ is a minimum ordering of G.*

Proof. Node x is part of a cycle and thus not a separation clique. Either case 1 or 3 of Theorem 4.1.14 holds, which implies that $x\Sigma(G_x)$ is a minimum ordering of G. □

Corollary 4.2.2. *Let $G = (V, E)$ be a graph. Let $\{x\} \subset V$ be a separation clique and $\deg(x) = 2$. Let Σ be a minimum fill-in ordering and $G^{(m)}$ be the graph in the corresponding elimination sequence from which x is eliminated. x is simplicial in $G^{(m)}$.*

Proof. Since x is a separation clique, Case 2 of Theorem 4.1.14 holds and thus, x is simplicial in $G^{(m)}$. □

Corollaries 4.2.1 and 4.2.2 imply that degree-2 elimination is exact when degree-2 nodes that are part of a cycle are eliminated. In some graphs, degree-2 elimination can therefore be exact.

If all nodes with degree 2 that are not separators can be detected efficiently, we can introduce an exact reduction that only eliminates nodes with degree 2.

4.2.2 Triangle Contraction

Theorem 4.2.3. *Let $G = (V, E)$ be a graph, where $\forall\, x \in V$ $\deg(x) \geq 3$. Let $a, b \in V$ be two neighboring nodes (i.e., $\{a, b\} \in E$), with $\deg(a) = \deg(b) = 3$*

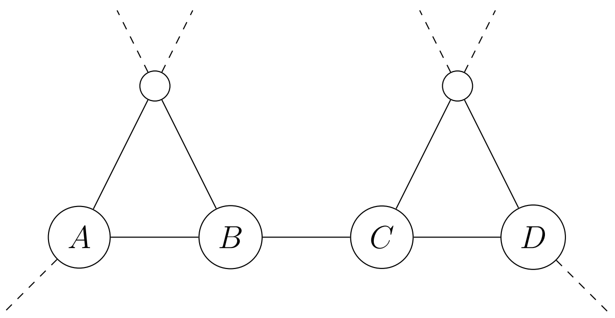


Figure 4.5: Nodes A, B and C, D satisfy the condition in Theorem 4.2.3. If A is eliminated in the minimum degree algorithm, $\deg(B) = 3$ in the elimination graph, so B could be eliminated next. However, if C is eliminated first, $\deg(B) = 4$ in the elimination graph and the condition does no longer hold for A and B.

and $|N(a) \cap N(b)| \geq 1$*. Nodes* a *and* b *can be eliminated first by the minimum degree algorithm.*

Proof. Since all nodes in G have at least degree 3, a can be eliminated first by the minimum degree algorithm. There are two cases:

Case 1: If $|N(a) \cap N(b)| = 1$, then $\deg_{G_a}(b) = 3$, so b can be eliminated after a.

Case 2: If $|N(a) \cap N(b)| = 2$, then a and b are indistinguishable.

$\square$

We can reduce the graph by contracting neighboring nodes of degree 3 if they share at least one neighbor. However, this only leads to a minimum ordering if the degree of the nodes does not change during the elimination process. If a node $x \in N(a), x \notin N(b)$ is eliminated before a or b and $\deg(a) = 4$ in the elimination graph, the ordering will no longer be minimum.

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

4.3 Node Ordering with Graph Clustering

Our node ordering algorithm based on graph clustering is outlined in Algorithm 4. We first cluster the input graph by label propagation. Then,

Algorithm 4: Node Ordering with Graph Clustering

```
   input : An undirected graph $G = (V, E)$
   output: An ordering $\sigma$

ClusteredOrdering($G$)
   // Find clusters in the graph
 1 $C \leftarrow$ LabelPropagation($G$)
   // Order the contracted graph
 2 $G' \leftarrow$ contract clusters $C$ in $G$
 3 $\sigma' \leftarrow$ ReducedNestedDissection($G'$)
   // Order the subgraphs
 4 foreach cluster $c_i$ in $C$, in order determined by $\sigma'$ do
 5     $\sigma \leftarrow \sigma$ ReducedNestedDissection($G[c_i]$)
```

we contract the clusters and order the resulting graph by reduced nested dissection. Each cluster is also ordered individually by reduced nested dissection. The orderings of the clusters are then arranged in the order of the contracted graph to yield the ordering of the input graph.

With this algorithm we aim to obtain orderings in shorter time than with (reduced) nested dissection. Ordering the graph after contracting clusters should be faster than ordering the uncontracted graph. The subgraphs induced by the clusters will be relatively small and may often fall under the recursion limit. Since the nodes in these clusters are highly connected, simplicial node and indistinguishable node reduction should be very effective here.

This algorithm works under the following assumption: in a minimum fill-in ordering we expect clusters to be ordered together, i.e., they would form a block in the corresponding permuted matrix. Thus, we expect that ordering clusters independently will lead to a good ordering. By ordering the contracted graph we also take the connections between clusters into account.

Chapter 5

Experimental Evaluation

We now describe our experimental evaluation of our algorithms. Section 5.1 outlines our implementation of the reduced nested dissection algorithm. In Section 5.2 we describe our test instances, method of evaluation and setup. Section 5.3 discusses the results of our experiments.

5.1 Implementation Details

We implemented reduced nested dissection in C++ within version 2.10 of the KaHIP graph partitioning framework [45].

Our implementation of the minimum degree algorithm is based on the generalized element model, but does not make use of the running time improvements described in Section 3.2.

To apply simplicial node reduction (Reduction 1), we iterate through nodes in order by non-decreasing degree. To test if node x is simplicial, we iterate through the neighbors $y \in N(x)$. If $\deg(y) < \deg(x)$, x is not simplicial. If $|N(y) \cap N(x)| = \deg(x) - 1$, we move on to the next neighbor, otherwise, x is not 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.

Indistinguishable node and twin reductions (Reductions 2 and 3) are similar in their implementation. To detect all indistinguishable nodes, we first compute a hash of the closed neighborhood of each node x_i as

$$h_c(x_i) = \sum_{y_j \in N[x_i]} j. \tag{5.1.1}$$

We then sort the hashes and iterate through the list. Whenever we find two nodes x, y with equal hashes, we test if $\deg(x) = \deg(y)$ and $N[x] = N[y]$.

We use the same process to detect twins, but use the open neighborhood instead of the closed neighborhood. The sets of twins and indistinguishable nodes are contracted. When mapping the ordering from the reduced graph to the input graph, we simply replace the contracted node by the corresponding set of twins or indistinguishable nodes.

For path compression (Reduction 4) we detect paths starting from a node with degree 2 by recursively adding neighbors with degree 2. These paths are contracted. When mapping the ordering from the reduced graph to the input graph, we order the paths from the end whose neighbor has been eliminated first.

To apply degree-2 elimination (Inexact Reduction 1), we first detect paths as in path compression. We build the reduced graph by copying nodes that are not eliminated. When adding the edges, we connect to the neighborhood of a path instead of to the path itself. Parallel edges are merged into one. The eliminated nodes are ordered arbitrarily.

We detect set of nodes A to be contracted in triangle contraction (Inexact Reduction 2) by the following procedure:

1. Let x be some node with $\deg(x) = 3$. Add x to A.
2. If x has a neighbor y with $\deg(y) = 3$ and $|N(x) \cap N(y)| \geq 1$, add x and y to A. Let $a \in (N(x) \cap N(y))$.
3. Let $z \in N(y), z \notin A$. If $\deg(z) = 3$ and $a \in N(z)$, add z to A. Otherwise, stop.
4. Repeat step 3 with the neighbors of z.

In the ordering of the input graph, nodes in A are ordered as they are added to A.

When contracting nodes the degree of the nodes and their neighbors change. To account for this, we do not use the node degree in the minimum degree algorithm, but an adjusted degree based on the node weights of the neighborhood:

$$\deg_{\text{adj}}(x) = \sum_{y \in N(x)} w_y + (w_x - 1) + c_x, \tag{5.1.2}$$

where w_x is the weight of node x. A node with weight m represents m nodes in the input graph. Thus, summing up the node weights of neighboring nodes yields the size of the neighborhood in the original graph. $w_x - 1$ counts the edges to the indistinguishable neighbors of node x. When contracting twins, counting these edges overestimates the degree, so we introduce a contraction

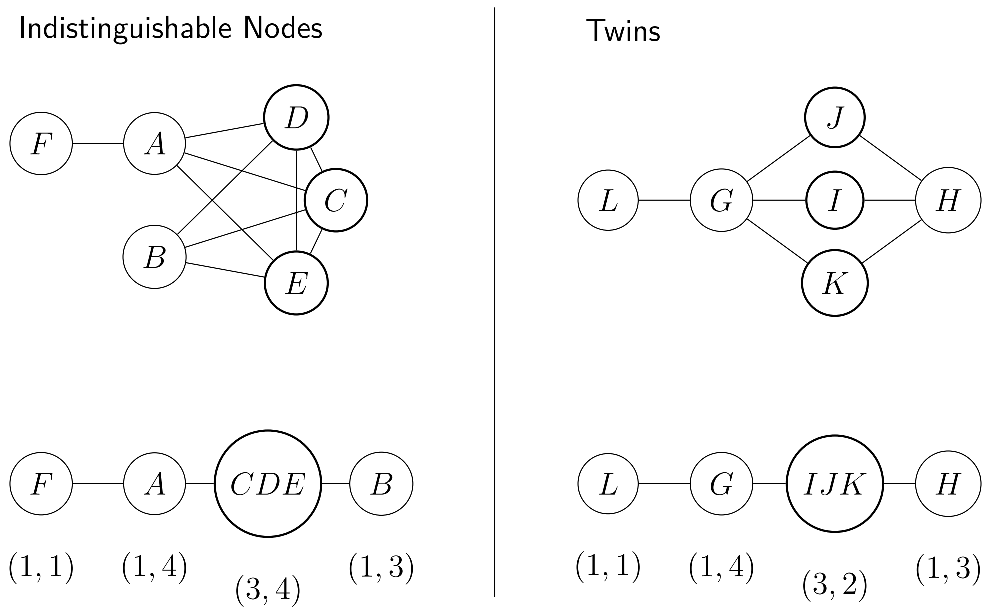


Figure 5.1: Examples for the adjusted degree for indistinguishable nodes and twins. Above: uncontracted graphs, below: contracted graphs. The nodes in the contracted graphs are labeled with the node weight and the adjusted degree. Nodes C, D and E are indistinguishable. Nodes I, J and K are twins. In the case of indistinguishable nodes $c_{CDE} = 0$ and $\deg_{\text{adj}}(CDE) = w_A + w_B + (w_{CDE} - 1) + c_{CDE} = 1+1+(3-1)+0 = 4$. In the case of twins $c_{IJK} = -(w_{IJK} - 1)$ and $\deg_{\text{adj}}(IJK) = w_G + w_H + (w_{IJK} - 1) + c_{IJK} = 2$.

offset c_x. For a node x that represents indistinguishable nodes $c_x = 0$. If x represents a set of twins, then $c_x = -(w_x - 1)$. Figure 5.1 gives an example of the adjusted degree for indistinguishable nodes and twins.

5.2 Experimental Setup

We evaluate our algorithm on undirected graphs from [35]. These graphs include social networks, citation networks and web graphs. Table 5.1 lists their basic properties. We also use a subset of graphs from Walshaw's graph partitioning archive [47].

We evaluate our orderings with the `gotst`-program from the software package Scotch (version 6.0.6) [38]. This program performs a Cholesky factorization and reports the number of non-zeros in the matrix factors and the operation count of the factorization. We compare our node orderings against orderings from Metis (version 5.1.0) [29].

Graph	Number of Nodes	Number of Edges
`amazon-2008`	735 323	3 523 472
`as-22july06`	22 963	4 846
`as-skitter`	554 930	5 797 663
`citationCiteseer`	268 495	1 156 647
`cnr-2000`	325 557	2 738 969
`coAuthorsCiteseer`	227 320	814 134
`coAuthorsDBLP`	299 067	977 676
`coPapersCiteseer`	434 102	16 036 720
`coPapersDBLP`	540 486	15 245 729
`email-EuAll`	16 805	60 260
`enron`	69 244	254 449
`eu-2005`	862 664	16 138 468
`in-2004`	1 382 908	13 591 473
`loc-brightkite_edges`	56 739	212 945
`loc-gowalla_edges`	196 591	950 327
`p2p-Gnutella04`	6 405	29 215
`PGPgiantcompo`	10 680	24 316
`soc-Slashdot0902`	28 550	379 445
`web-Google`	356 648	2 093 324
`wiki-Talk`	232 314	1 458 806
`wordassociation-2011`	10 617	63 788

Table 5.1: Properties of the social networks from [35].

Graph	Number of Nodes	Number of Edges
3elt	4 720	13 722
4elt	15 606	45 878
add20	2 395	7 462
add32	4 960	9 462
bcsstk29	13 992	302 748
bcsstk30	28 924	1 007 284
bcsstk31	35 588	572 914
bcsstk33	8 738	291 583
crack	10 240	30 380
cs4	22 499	43 858
cti	16 840	48 232
data	2 851	15 093
fe_4elt2	11 143	32 818
fe_pwt	36 519	144 794
fe_sphere	16 386	49 152
memplus	17 758	54 196
uk	4 824	6 837
vibrobox	12 328	165 250
whitaker3	9 800	28 989
wing_nodal	10 937	75 488

Table 5.2: Properties of the graphs from Walshaw's benchmarking archive [47].

Reduction	Abbreviation
Simplicial Node Reduction	0
Indistinguishable Node Reduction	1
Twin Reduction	2
Path Compression	3
Degree-2 Elimination	4
Triangle Contraction	5

Table 5.3: Reductions and their abbreviations in this text.

We compiled our implementation with version 6.3.0 of `g++`, with the optimization level set to `-O3`.

All running times we report were measured on a machine with two Intel Xeon E7-8867 v3 processors (16 cores, 2.5 GHz) and 1000 GB RAM. We run our code sequentially on a single core.

5.3 Experimental Results

Here, we describe the results of our experiments. First, we evaluate the effect of reductions on the quality of nested dissection orderings and on the running time of nested dissection. In Sections 5.3.2 and 5.3.3 we study how the choice of recursion limit and imbalance influences quality and running time.

In all these experiments we apply the reductions once in the chosen order on each recursion level. The reduced graph we obtain in this way may be further reduced by repeating the reductions. We can apply them exhaustively, i.e., until the graph can no longer be reduced. In Section 5.3.4 we evaluate how exhaustive application of reductions affects the quality of the orderings.

Lastly, in Section 5.3.5 we present the results from our clustering based node ordering algorithm.

Unless otherwise noted, we use the *ecosocial* preconfiguration of KaHIP to compute node separators, with the imbalance set to 20%. The default recursion limit is set to 120 nodes, which is also the default in Metis.

Our implementation reads the order of reductions as a list of numbers. In this section, we use the same notation. To give an example, 0 3 4 would refer to an order of reductions where simplicial node reduction is applied first, followed by path compression and degree-2 elimination. See Table 5.3 for reference.

5.3.1 Combinations of Reductions

Here, we compare reduced nested dissection with nested dissection without reductions in terms of the quality of its orderings and in terms of its running time. For each of the graphs listed in Table 5.1 we ran reduced nested dissection with different combination of reductions, as listed below. We also compare our results against orderings obtained from Metis.

We use the following combinations:

0 1: Simplicial node and indistinguishable node reduction.

0 1 2: Simplicial node, indistinguishable node and twin reduction.

0 3: Simplicial node reduction and path compression.

0 4: Simplicial node reduction and degree-2 elimination.

0 4 5: Simplicial node reduction, degree-2 elimination and triangle contraction. Note that triangle contraction requires degree-2 elimination.

0 1 2 3 4: All reductions except triangle contraction. Note that combining path compression and degree-2 elimination in this way is not necessarily useful, since the compressed paths are eliminated by degree-2 elimination.

0 1 2 3 5: All reductions except degree-2 elimination. Note that this combination is not expected to perform well, since triangle contraction requires degree-2 elimination.

0 1 2 4 5: All reductions except path compression.

0 1 3 4 5: All reductions except twin reduction.

0 2 3 4 5: All reductions except indistinguishable node reduction.

0 1 2 3: Only exact reductions.

There are two reasons for always starting with simplicial node reduction: first, simplicial node reduction finds a perfect elimination order if a graph is triangulated. Second, some of the reductions only work if simplicial nodes are removed (see Sections 4.2.1 and 4.2.2). Degree-2 elimination assumes that there are no nodes of degree 1, which are removed by simplicial node reduction. Triangle contraction can only be applied in combination with degree-2 elimination, and thus also requires simplicial node reduction.

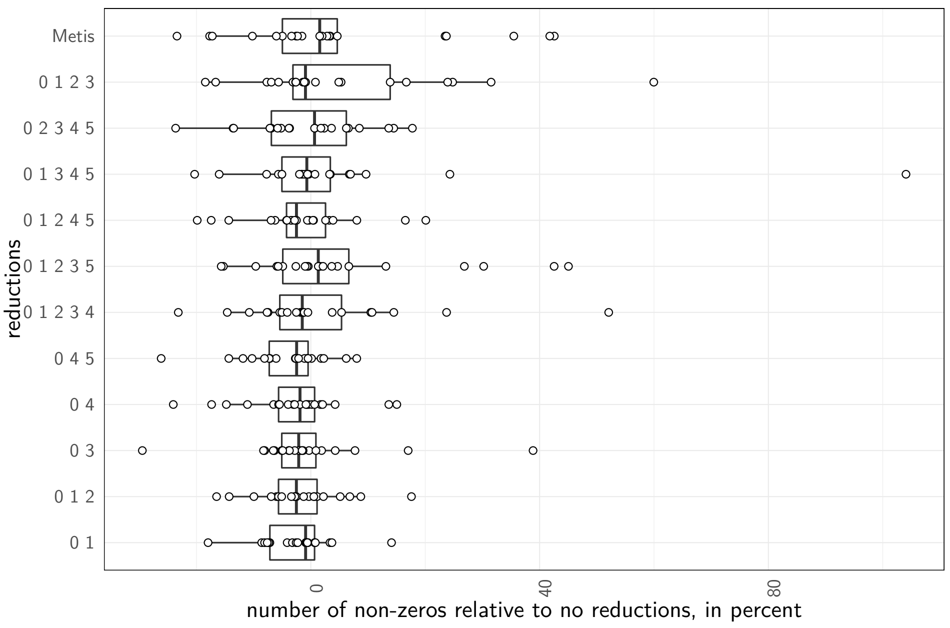
Figure 5.2 (on Page 40) shows the improvement in the number of non-zeros over nested dissection. In terms of the median, the combination 0 4 5, 0 1 2 and 0 1 2 4 5 lead to the best results, with an improvement of $\sim 2.5\%$. The combination 0 1 2 3 5 leads to worse results than nested dissection without reductions, as expected. In terms of operation count, the combination 0 3 performs best, with a median improvement close to 5% (see Figure 5.3). It appears that in general the reductions influence the operation count more than the number of non-zeros.

These results are summarized in Table 5.4. It should be noted that exact reductions do not guarantee node orderings with lower number of non-zeros or operation count. Since both nested dissection and the minimum degree algorithm are heuristics, this is not unexpected. The minimum degree algorithm in particular is very sensitive to the initial order of the matrix [22], which a reduction can change significantly.

Figure 5.4 shows the improvement in running time over nested dissection without reductions. All reduction orders improve the running time of nested dissection. In only nine cases is reduced nested dissection slower than nested dissection. The largest improvement is gained for the graph `coPapersCiteseer` and the reduction combination 0 1 3 4 5 with 95.6%. For this graph, nested dissection without reductions takes 17 175 seconds; reduced nested dissection takes 751.4 seconds. For the combination 0 4 5 the running time is improved by 36.4% in the median. For the combination 0 1 2 it is improved by 44%.

Table 5.5 lists the graph sizes after application of reduction rules before any separators are computed. Simplicial node reduction has by far the greatest impact, on average reducing the graph to 57% of its original size. Degree-2 elimination, twin reduction and indistinguishable node reduction reduce the graph size by an additional 6%–13%. Path compression and triangle contraction only lead to small reductions. Overall, the graphs are reduced to approximately half their original size. Table 5.6 shows the average reduction in graph size over all applications of the reduction rules. Path compression and triangle contraction contribute more to reducing the graphs in later stages of the algorithm. Simplicial node reduction and degree-2 elimination reduce the graphs the most.

On the social networks, our implementation of nested dissection performs slightly better than Metis. In the median, the number of non-zeros for Metis is 1.56% greater than for our nested dissection without reductions (see Table 5.4). The operation count is even 5.08% greater. For further comparison, we applied reduced nested dissection to the graphs in Table 5.2. Here, we used the *eco* preconfiguration for computing node separators. Table 5.7 reports the number of non-zeros and operation count for reduction orders 0 1 2 and 0 4 5. For eleven graphs, at least one reduction order yields a lower number of non-zeros. In terms of operation count, reduced nested dissection performs better for thirteen graphs. Three graphs stand out: `add20`, `add32` and `memplus`. `add20` and `memplus` are chordal graphs and are reduced completely by simplicial node reduction. For these graphs, we obtain a perfect elimination order. `add32` is reduced to 0.8% of its original size by simplicial node reduction, which improves the quality of the ordering.



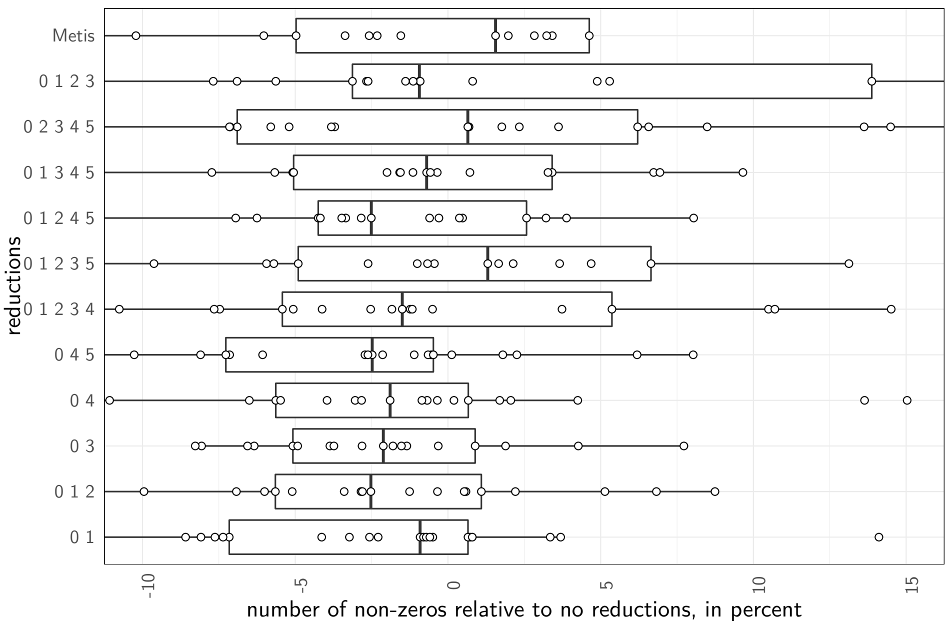
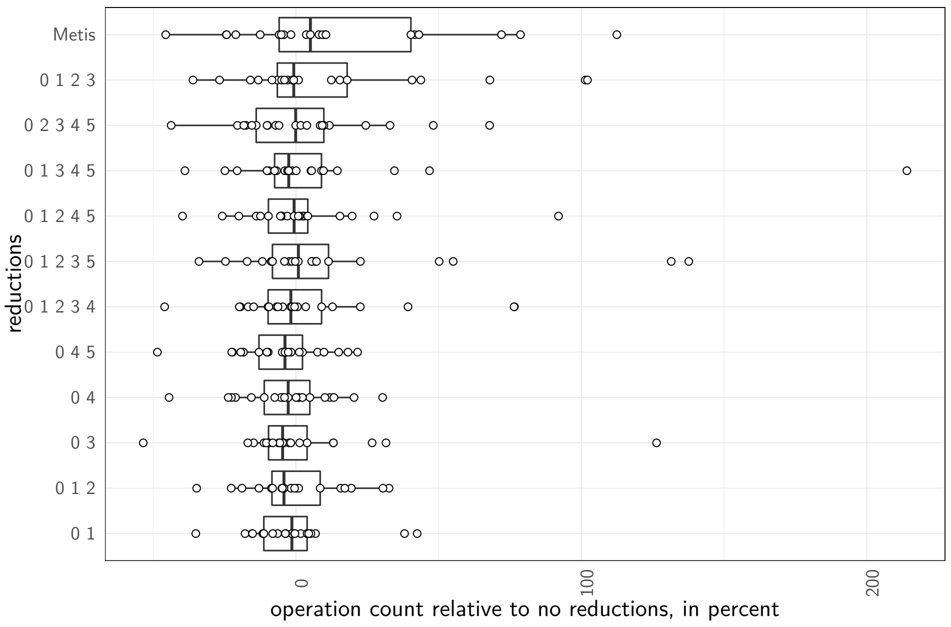


Figure 5.2: Number of non-zeros for different reduction combinations, relative to the number of non-zeros for nested dissection without reductions. The boxes extend from the first to the third quartile, the bar represents the median. Above: all points, below: zoomed plot. Combination 0 2 3 4 5 and 0 1 2 3 5 lead to more non-zeros in the median than nested dissection without reductions. All other combinations reduce the number of non-zeros.



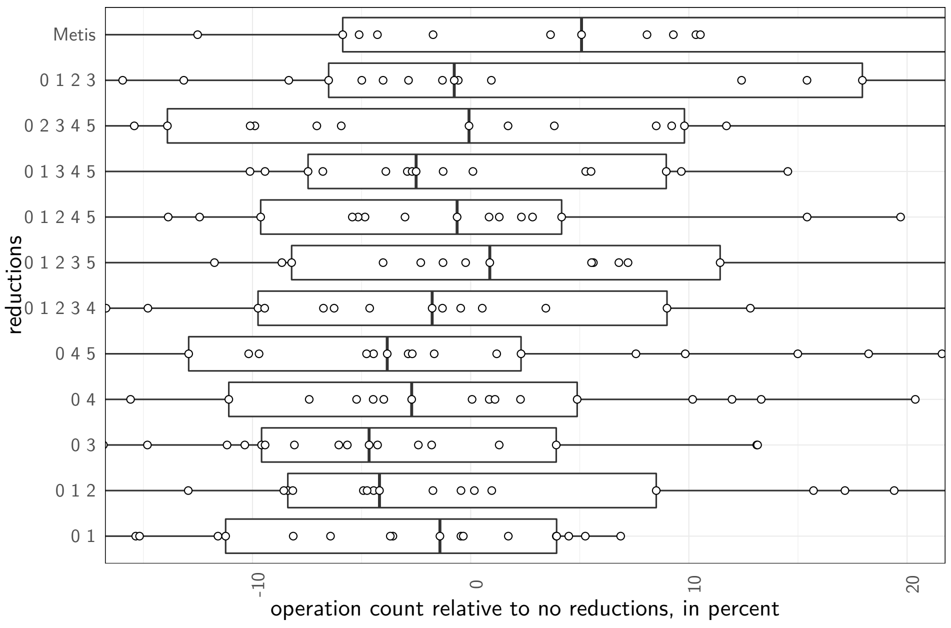


Figure 5.3: Operation count for different reduction combinations, relative to the operation count of nested dissection without reductions. The boxes extend from the first to the third quartile, the bar represents the median. Above: all points, below: zoomed plot. Only combination 0 1 2 3 5 leads to a higher operation count in the median than nested dissection without reductions.

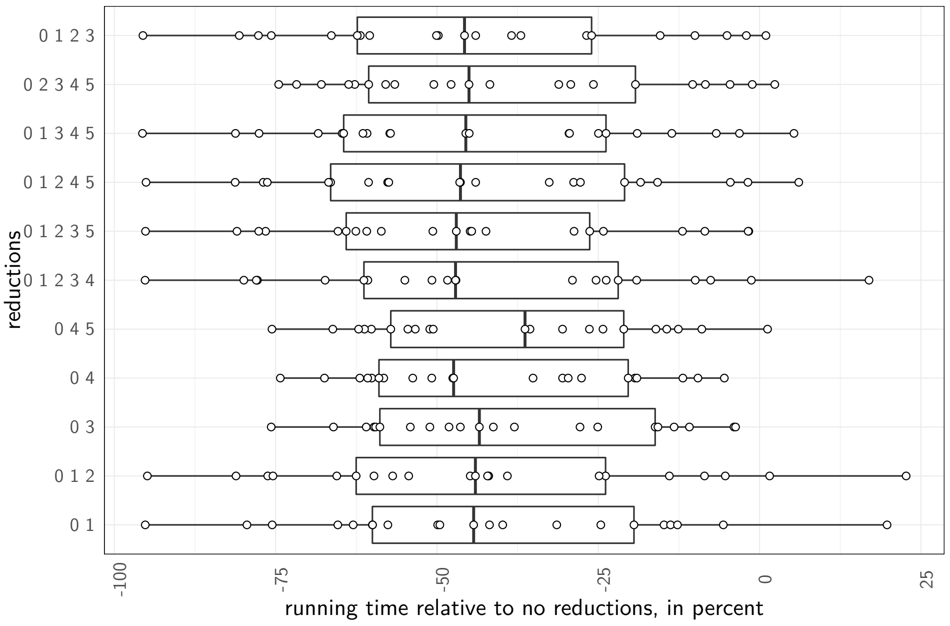


Figure 5.4: Running time for different reduction combinations, relative to the running time of nested dissection without reductions. The boxes extend from the first to the third quartile, the bar represents the median. With all combinations, reduced nested dissection is faster than nested dissection without reductions, except for some graphs. The combination 0 4 yields the best improvement in terms of median.

	Number of Non-Zeros					Operation Count				
Algorithm	Mean	Std. Dev.	Q1	Median	Q3	Mean	Std. Dev.	Q1	Median	Q3
0 1	-2.34	6.24	-7.16	-0.91	0.66	-1.12	16.98	-11.23	-1.41	3.94
0 1 2	-1.66	7.61	-5.65	**-2.52**	1.09	-0.47	16.62	-8.38	-4.18	8.50
0 3	-0.74	12.36	-5.08	-2.12	0.89	2.74	33.04	-9.59	**-4.66**	3.92
0 4	-2.90	9.01	-5.64	-1.89	0.67	-3.19	16.68	-11.09	-2.70	4.88
0 4 5	**-4.07**	7.52	**-7.27**	-2.48	**-0.48**	**-5.15**	16.02	-12.92	-3.83	**2.31**
0 1 2 3 4	1.60	15.44	-5.42	-1.50	5.37	3.98	29.46	-9.75	-1.77	9.00
0 1 2 3 5	5.52	16.98	-4.90	1.30	6.65	15.35	44.72	-8.21	0.88	11.43
0 1 2 4 5	-1.48	9.47	-4.25	-2.51	2.57	2.90	26.57	-9.62	-0.62	4.17
0 1 3 4 5	4.35	24.55	-5.05	-0.70	3.41	9.91	50.23	-7.45	-2.50	8.96
0 2 3 4 5	-0.68	10.10	-6.90	0.65	6.21	2.66	25.05	**-13.90**	-0.08	9.80
0 1 2 3	5.41	17.98	-3.13	-0.94	13.88	13.40	37.71	-6.51	-0.75	17.95
Metis	4.54	18.58	-4.97	1.56	4.63	13.82	38.45	-5.86	5.08	40.29

Table 5.4: Mean, standard deviation (Std. Dev.), first and third quartile (Q1, Q3) and median of the improvement over our implementation of nested dissection without reductions for the graphs in Table 5.1. The number sequences in the first column refer to the reduction combinations. Smaller numbers are better for the mean and the quartiles. The best values in each column are highlighted in bold.

Combination	0	1	2	3	4	5	Total
0 1 2 3	57	94	92	99			50
0 2 3 4 5	57		93	99	90	99	48
0 1 3 4 5	57	94		99	88	99	47
0 1 2 4 5	57	94	92		89	99	45
0 1 2 3 5	57	94	92	99		99	49
0 1 2 3 4	57	94	92	99	90		45
0 4 5	57				87	99	51
0 4	57				87		51
0 3	57			99			57
0 1 2	57	94	92				50
0 1	57	94					54

Table 5.5: Impact of the reductions on the graph size on the first level of recursion for each combination, averaged over the graphs. All values in percent. For each reduction, we list the size of the reduced graph relative to the output of the previous reduction. The last column lists the size of the fully reduced graph relative to the input graph.

Combination	0	1	2	3	4	5
0 1 2 3	63	94	98	94		
0 2 3 4 5	65		98	96	91	97
0 1 3 4 5	66	99		95	91	97
0 1 2 4 5	66	94	98		87	97
0 1 2 3 5	64	94	98	94		98
0 1 2 3 4	66	94	98	96	90	
0 4 5	65				88	96
0 4	65				88	
0 3	63			94		
0 1 2	62	94	98			
0 1	62	94				

Table 5.6: Avergae impact of the reductions on the graph size over all recursion levels, averaged over the graphs. All values in percent. Every time we apply a reduction we compute the relative graph size before and after the application. Here, we list the average of these relative graph sizes over the full execution of the algorithm.

	Number of Non-Zeros			Operation Count		
Graph	0 1 2	0 4 5	Metis	0 1 2	0 4 5	Metis
`3elt`	9.19×10^4	9.32×10^4	$\mathbf{8.93 \times 10^4}$	2.78×10^6	2.88×10^6	$\mathbf{2.60 \times 10^6}$
`4elt`	3.53×10^5	3.51×10^5	$\mathbf{3.47 \times 10^5}$	1.37×10^7	1.35×10^7	$\mathbf{1.33 \times 10^7}$
`add20`	$\mathbf{9.86 \times 10^3}$	$\mathbf{9.86 \times 10^3}$	1.10×10^4	$\mathbf{7.86 \times 10^4}$	$\mathbf{7.86 \times 10^4}$	1.07×10^5
`add32`	$\mathbf{1.44 \times 10^4}$	$\mathbf{1.44 \times 10^4}$	1.51×10^4	$\mathbf{4.34 \times 10^4}$	$\mathbf{4.34 \times 10^4}$	4.90×10^4
`bcsstk29`	1.65×10^6	$\mathbf{1.64 \times 10^6}$	$\mathbf{1.64 \times 10^6}$	3.36×10^8	3.31×10^8	$\mathbf{3.21 \times 10^8}$
`bcsstk30`	4.43×10^6	$\mathbf{4.15 \times 10^6}$	4.35×10^6	1.16×10^9	$\mathbf{1.01 \times 10^9}$	1.11×10^9
`bcsstk31`	4.20×10^6	$\mathbf{4.17 \times 10^6}$	4.31×10^6	1.06×10^9	$\mathbf{1.05 \times 10^9}$	1.13×10^9
`bcsstk33`	2.19×10^6	$\mathbf{2.10 \times 10^6}$	2.16×10^6	8.00×10^8	$\mathbf{7.35 \times 10^8}$	7.70×10^8
`crack`	$\mathbf{1.71 \times 10^5}$	1.72×10^5	1.72×10^5	$\mathbf{6.40 \times 10^6}$	6.59×10^6	6.96×10^6
`cs4`	$\mathbf{1.35 \times 10^6}$	1.37×10^6	1.38×10^6	$\mathbf{3.55 \times 10^8}$	3.57×10^8	3.74×10^8
`cti`	1.48×10^6	$\mathbf{1.45 \times 10^6}$	1.61×10^6	4.07×10^8	$\mathbf{4.01 \times 10^8}$	4.92×10^8
`data`	$\mathbf{8.20 \times 10^4}$	8.50×10^4	8.29×10^4	$\mathbf{3.82 \times 10^6}$	4.21×10^6	3.96×10^6
`fe_4elt2`	2.61×10^5	2.59×10^5	$\mathbf{2.57 \times 10^5}$	1.18×10^7	$\mathbf{1.13 \times 10^7}$	1.16×10^7
`fe_pwt`	1.38×10^6	1.38×10^6	$\mathbf{1.35 \times 10^6}$	1.07×10^8	1.06×10^8	$\mathbf{1.04 \times 10^8}$
`fe_sphere`	6.26×10^5	6.28×10^5	$\mathbf{6.22 \times 10^5}$	$\mathbf{6.31 \times 10^7}$	6.33×10^7	6.49×10^7
`memplus`	$\mathbf{7.20 \times 10^4}$	$\mathbf{7.20 \times 10^4}$	7.83×10^4	$\mathbf{8.17 \times 10^5}$	$\mathbf{8.17 \times 10^5}$	1.63×10^6
`uk`	3.57×10^4	3.74×10^4	$\mathbf{3.48 \times 10^4}$	4.56×10^5	5.03×10^5	$\mathbf{4.36 \times 10^5}$
`vibrobox`	$\mathbf{2.00 \times 10^6}$	$\mathbf{2.00 \times 10^6}$	2.11×10^6	$\mathbf{8.25 \times 10^8}$	8.55×10^8	9.32×10^8
`whitaker3`	2.62×10^5	2.63×10^5	$\mathbf{2.58 \times 10^5}$	1.37×10^7	1.38×10^7	$\mathbf{1.36 \times 10^7}$
`wing_nodal`	1.76×10^6	1.73×10^6	$\mathbf{1.70 \times 10^6}$	6.02×10^8	5.62×10^8	$\mathbf{5.51 \times 10^8}$

Table 5.7: Number of non-zeros and operation count for the graphs from Table 5.2. We compare reduced nested dissection with reduction orders 0 1 2 and 0 4 5 with results from Metis. Lowest values are highlighted in bold.

5.3.2 Effect of the Recursion Limit on Running Time and Quality

To study how the recursion limit influences the quality of the node orderings and the running time of reduced nested dissection, we ordered the graphs in Table 5.1 using reduction orders 0 4 5 and 0 1 2 and varied the recursion limit from 50 to 2000 nodes in steps of 50 nodes. We chose these reduction orders because they perform best (see Section 5.3.1) and because they use different sets of reductions. For this experiment we omitted five graphs because of their large running times up to five hours. This should not affect the results of this study.

As Figures 5.5 and 5.6 show, the number of non-zeros and operation count do not change with the recursion limit in most cases. Where they change, there is no clear trend. For the graphs `amazon-2008` and `web-Google` the effect of the recursion limit appears to be random.

The effect of the recursion limit on the running time of reduced nested dissection is clearer (see Figure 5.7). With increased recursion limit, the running time decreases. With a lower recursion limit, more separators need to be computed. With a higher recursion limit, these computations are replaced by a single application of the minimum degree algorithm, which is most likely significantly faster.

While increasing the recursion limit may improve the running time of nested dissection, at some point this will cancel out the advantages of nested dissection over the minimum degree algorithm. Nested dissection orderings are usually more suitable for parallel factorization than minimum degree orderings [24]. This should be taken into account when choosing the recursion limit: a lower recursion limit might be advantageous, even though computing the node ordering will take more time.

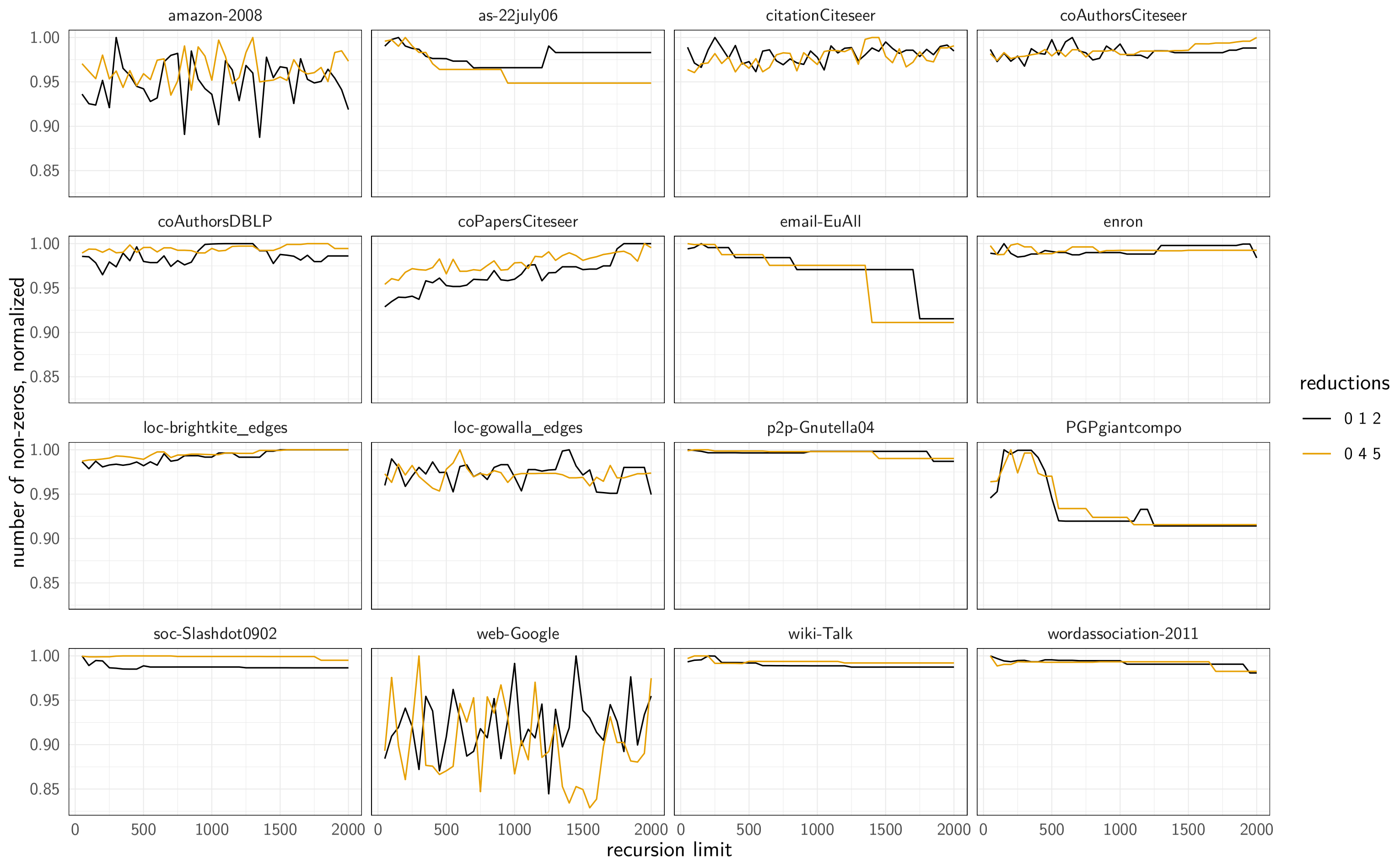


Figure 5.5: Change in number of non-zeros with the recursion limit. For each graph, the number of non-zeros is normalized by the maximum value. Reductions are abbreviated as in Table 5.3. For most of the graphs increasing the recursion limit has no effect on the number of non-zeros. Only `email-EuAll` and `PGPgiantcompo` benefit clearly from a higher limit.

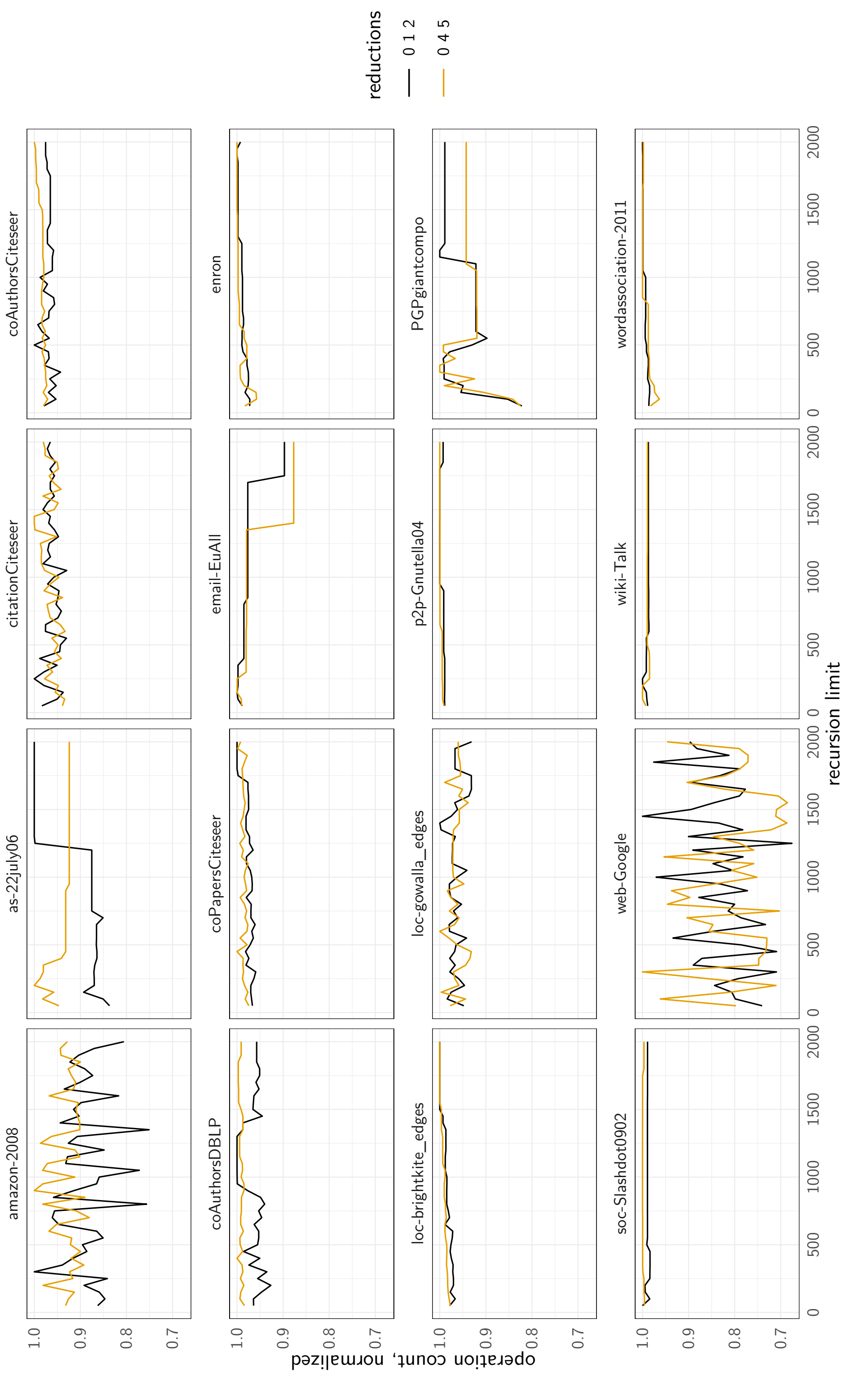


Figure 5.6: Change in operation count of the Cholesky factorization with the recursion limit. For each graph, the operation count is normalized by the maximum value. Reductions are abbreviated as in Table 5.3. As with the number of non-zeros (see Figure 5.5), the recursion limit has no effect on the operation count for most graphs.

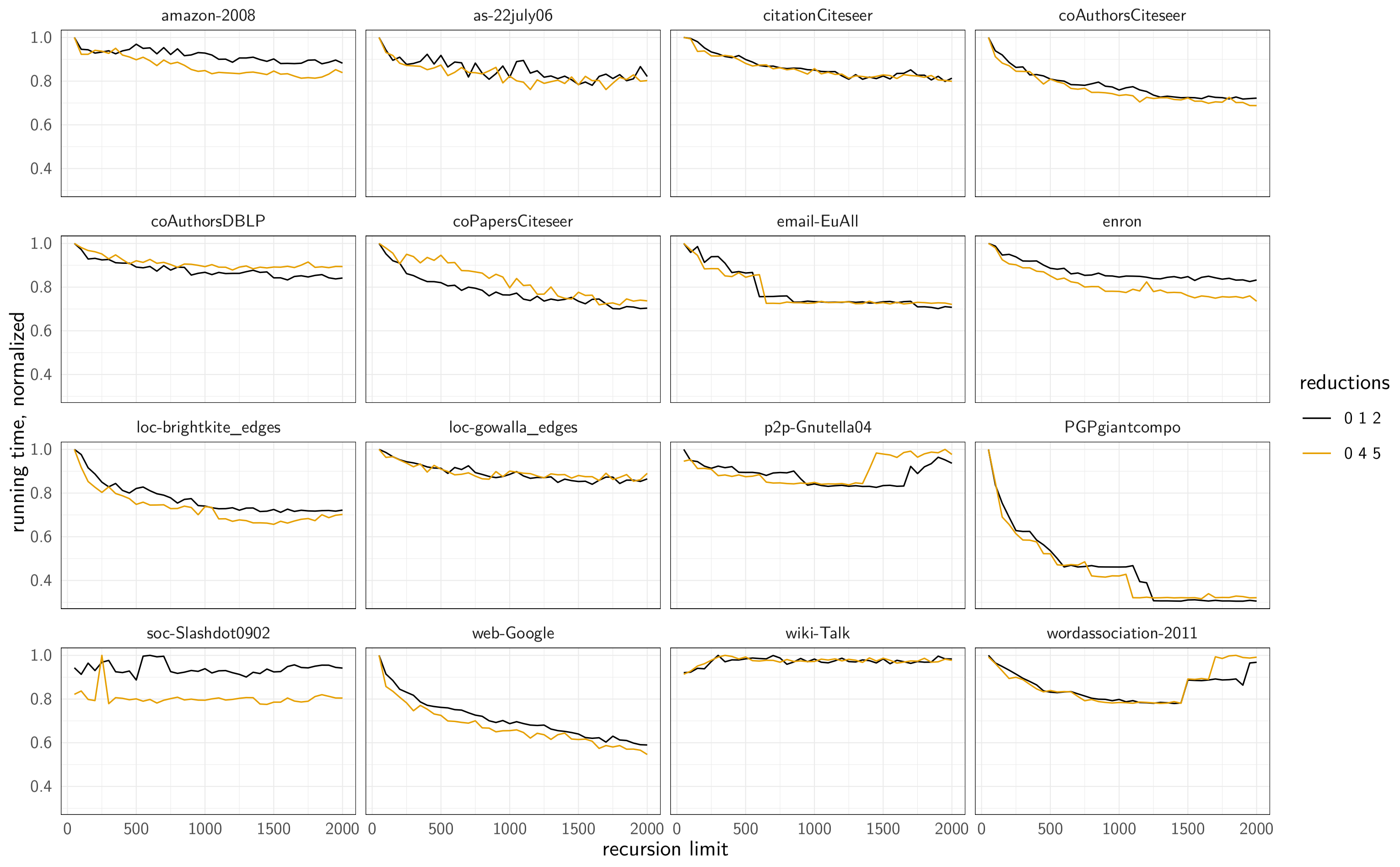


Figure 5.7: Change in running time of reduced nested dissection with the recursion limit. For each graph, the running time is normalized by the maximum value. Reductions are abbreviated as in Table 5.3. For most of the graphs the running time decreases when the recursion limit increases.

5.3.3 Effect of the Imbalance Constraint on Running Time and Quality

To study the influence of the node separator imbalance on the quality of the orderings and the running time of reduced nested dissection, we ordered the graphs in Table 5.1 using reduction orders 0 4 5 and 0 1 2, as in Section 5.3.2. Here, we used the default recursion limit and varied the imbalance parameter ε from 5% to 50% in steps of 5%. Again, we omitted the same graphs.

Figures 5.8 to 5.10 show the change in number of non-zeros, operation count and running time, respectively, with increasing imbalance. Number of non-zeros and operation count both increase with the imbalance. This is unexpected, since a larger imbalance should lead to smaller separators and thus improve the node orderings. For most graphs, increasing the imbalance also leads to increased running time. With increased imbalance, the recursion also becomes more imbalanced and thus deeper. In the worst case, without balance constraint, one of the components of the separator is almost the same size as the input graph. Then, the recursion depth is on the order of the number of nodes in the input graph.

Since imbalance increases the running time, the number of non-zeros and the operation count, a lower imbalance is preferred.

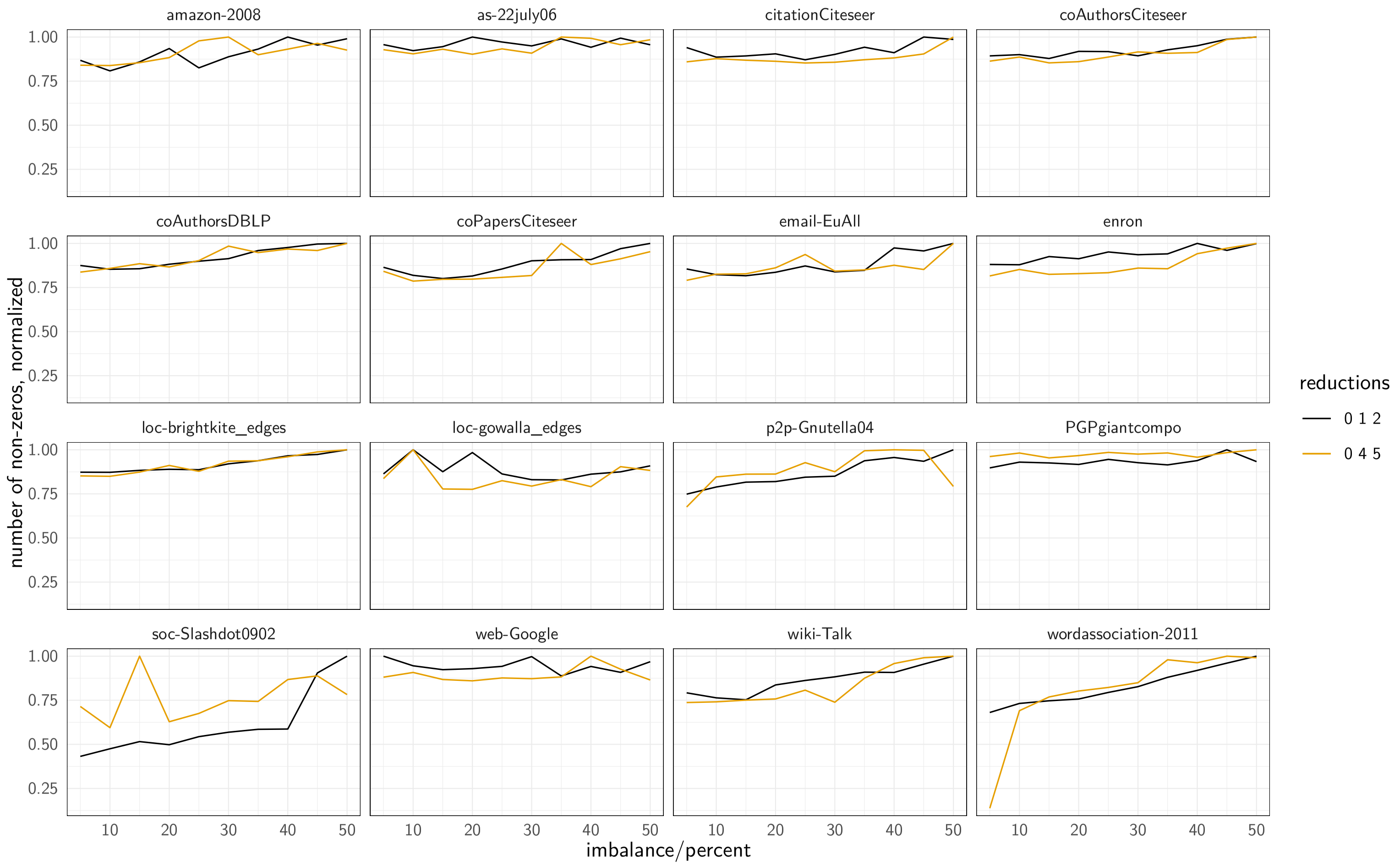


Figure 5.8: Change in number of non-zeros with the imbalance. For each graph, the number of non-zeros is normalized by the maximum value. Reductions are abbreviated as in Table 5.3. Increasing the imbalance increases the number of non-zeros.

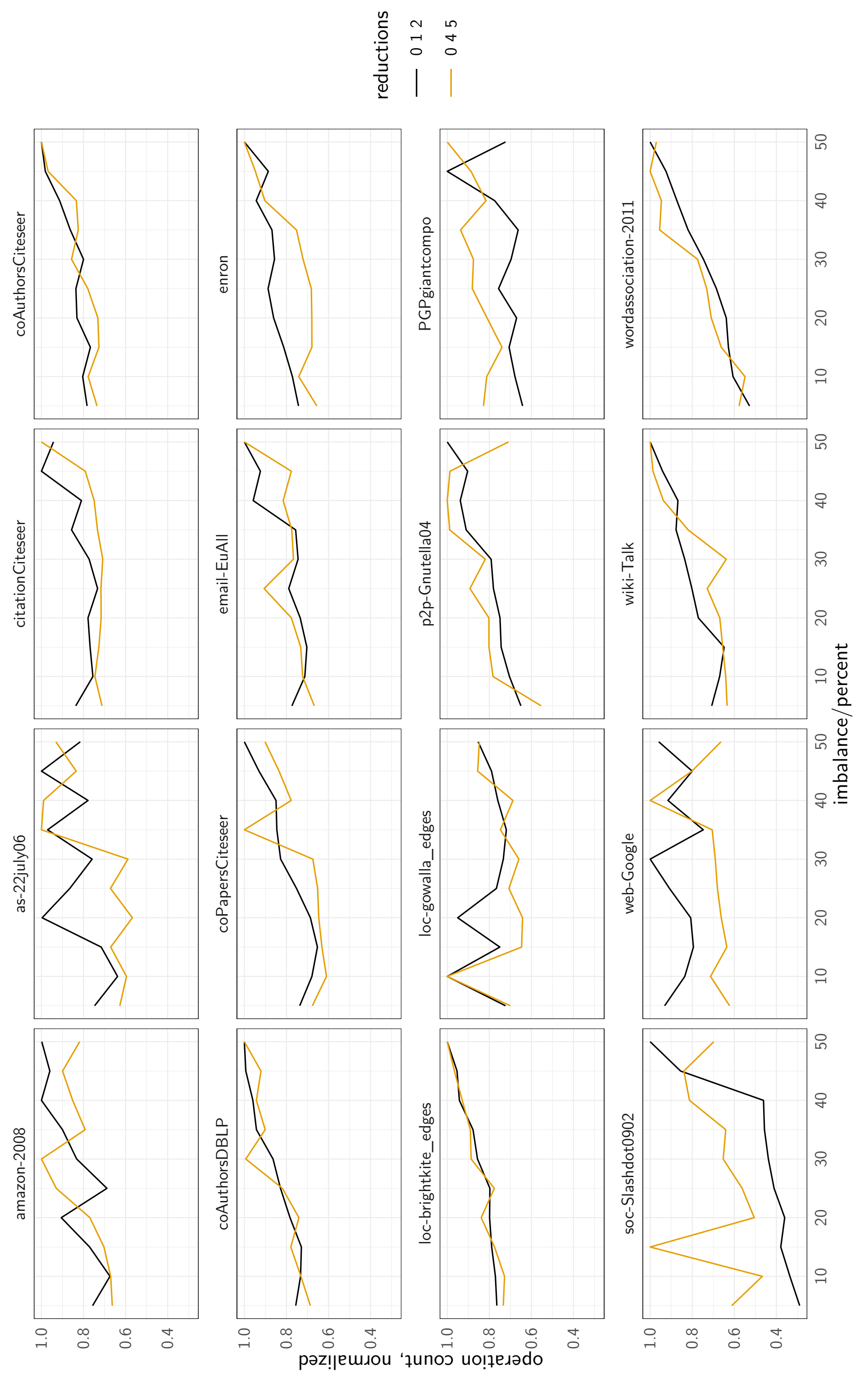


Figure 5.9: Change in operation count of the Cholesky factorization with the imbalance. For each graph, the operation count is normalized by the maximum value. Reductions are abbreviated as in Table 5.3. Increasing the imbalance increases the operation count.

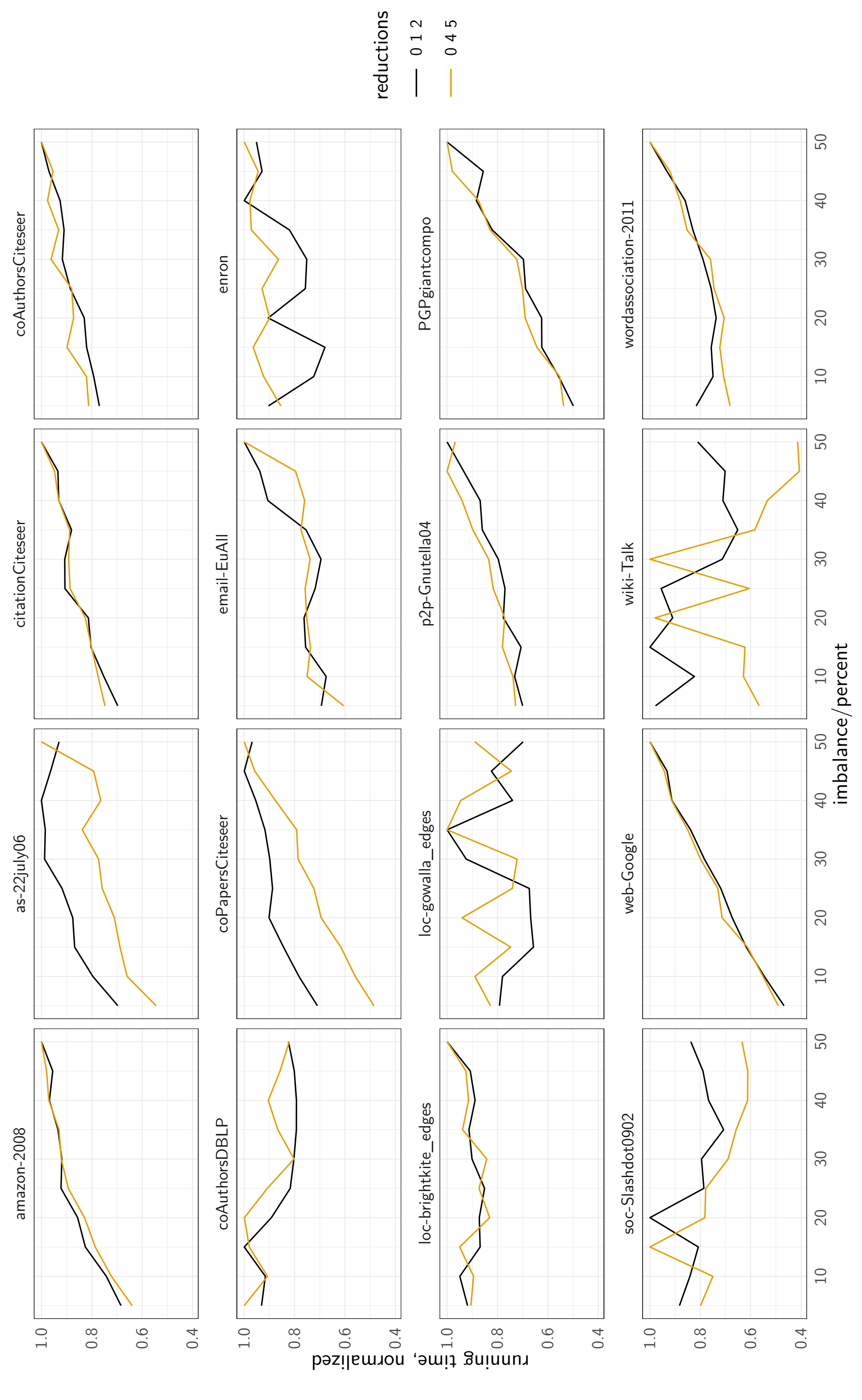


Figure 5.10: Change in running time of reduced nested dissection with the imbalance. For each graph, the running time is normalized by the maximum value. Reductions are abbreviated as in Table 5.3. For most of the graphs, increasing the imbalance also increases the running time. However, some graphs appear to benefit from increased imbalance.

5.3.4 Exhaustive Application of Reductions

Here, we evaluate the influence of exhaustive application of reductions on the number of non-zeros and operation count. We ran reduced nested dissection for reduction combinations 0 1 2 and 0 4 5 with default parameters. To apply reductions exhaustively, we apply the reductions in the specified order. Then, as long as the number of nodes in the graph changes, we repeat the reduction in the same order as before. We repeat this process on every recursion level. Again, we use the same set of graphs as in Sections 5.3.2 and 5.3.3.

The results for combinations 0 1 2 and 0 4 5 are listed in Tables 5.8 and 5.9, respectively. Also listed are the values from the experiments in Section 5.3.1 for the respective reduction combinations. With order 0 1 2 exhaustive application of reductions leads to lower number of non-zeros and lower operation count for the majority of graphs. For order 0 4 5 the result is less clear. It appears that exhaustive reduction does not affect the quality of the ordering in any significant way for this combination. However, applying the reduction combination 0 1 2 exhaustively seems to improve the orderings.

Combination 0 1 2 contains only exact reductions. Combination 0 4 5 also contains inexact reductions. We might conclude, that applying exact reductions exhaustively improves the quality of the node orderings. However, more results are needed to support this hypothesis.

	Number of Non-Zeros		Operation Count	
Graph	Exhaustive	Single	Exhaustive	Single
`amazon-2008`	$\mathbf{2.14 \times 10^9}$	2.38×10^9	$\mathbf{5.50 \times 10^{13}}$	6.66×10^{13}
`as-22july06`	$\mathbf{1.31 \times 10^5}$	1.41×10^5	$\mathbf{6.14 \times 10^6}$	8.83×10^6
`citationCiteseer`	$\mathbf{2.67 \times 10^8}$	2.68×10^8	$\mathbf{2.08 \times 10^{12}}$	2.09×10^{12}
`coAuthorsCiteseer`	$\mathbf{4.00 \times 10^7}$	4.05×10^7	1.64×10^{11}	$\mathbf{1.63 \times 10^{11}}$
`coAuthorsDBLP`	$\mathbf{1.46 \times 10^8}$	1.47×10^8	$\mathbf{1.22 \times 10^{12}}$	1.24×10^{12}
`coPapersCiteseer`	5.00×10^8	$\mathbf{4.53 \times 10^8}$	6.77×10^{12}	$\mathbf{5.48 \times 10^{12}}$
`email-EuAll`	6.64×10^5	$\mathbf{6.62 \times 10^5}$	4.04×10^8	$\mathbf{4.00 \times 10^8}$
`enron`	2.89×10^6	$\mathbf{2.87 \times 10^6}$	$\mathbf{3.20 \times 10^9}$	3.24×10^9
`loc-brightkite_edges`	1.45×10^7	1.45×10^7	$\mathbf{4.01 \times 10^{10}}$	4.06×10^{10}
`loc-gowalla_edges`	1.61×10^8	1.61×10^8	$\mathbf{1.19 \times 10^{12}}$	1.20×10^{12}
`p2p-Gnutella04`	$\mathbf{4.21 \times 10^6}$	4.68×10^6	$\mathbf{7.60 \times 10^9}$	8.94×10^9
`PGPgiantcompo`	7.74×10^4	$\mathbf{7.59 \times 10^4}$	3.80×10^6	$\mathbf{3.52 \times 10^6}$
`soc-Slashdot0902`	5.67×10^7	$\mathbf{5.50 \times 10^7}$	3.82×10^{11}	$\mathbf{3.66 \times 10^{11}}$
`web-Google`	2.79×10^7	$\mathbf{2.69 \times 10^7}$	6.13×10^{10}	$\mathbf{5.10 \times 10^{10}}$
`wiki-Talk`	$\mathbf{9.70 \times 10^7}$	1.00×10^8	$\mathbf{8.37 \times 10^{11}}$	8.76×10^{11}
`wordassociation-2011`	$\mathbf{3.97 \times 10^6}$	3.98×10^6	$\mathbf{6.35 \times 10^9}$	6.41×10^9

Table 5.8: Number of non-zeros and operation count for exhaustive and single application of reductions with combination 0 1 2. Best values are highlighted in bold.

	Number of Non-Zeros		Operation Count	
Graph	Exhaustive	Single	Exhaustive	Single
amazon-2008.graph	2.27×10^9	$\mathbf{2.22 \times 10^9}$	6.15×10^{13}	$\mathbf{5.89 \times 10^{13}}$
as-22july06.graph	1.26×10^5	$\mathbf{1.24 \times 10^5}$	5.84×10^6	$\mathbf{5.44 \times 10^6}$
citationCiteseer.graph	2.65×10^8	2.65×10^8	2.06×10^{12}	$\mathbf{2.05 \times 10^{12}}$
coAuthorsCiteseer.graph	4.01×10^7	$\mathbf{3.91 \times 10^7}$	1.64×10^{11}	$\mathbf{1.55 \times 10^{11}}$
coAuthorsDBLP.graph	1.49×10^8	$\mathbf{1.48 \times 10^8}$	1.29×10^{12}	$\mathbf{1.24 \times 10^{12}}$
coPapersCiteseer.graph	4.56×10^8	$\mathbf{4.44 \times 10^8}$	5.57×10^{12}	$\mathbf{5.25 \times 10^{12}}$
email-EuAll.graph	$\mathbf{6.29 \times 10^5}$	6.61×10^5	$\mathbf{3.75 \times 10^8}$	4.01×10^8
enron.graph	$\mathbf{2.71 \times 10^6}$	2.74×10^6	$\mathbf{2.85 \times 10^9}$	2.86×10^9
loc-brightkite_edges.graph	$\mathbf{1.47 \times 10^7}$	1.52×10^7	$\mathbf{4.15 \times 10^{10}}$	4.45×10^{10}
loc-gowalla_edges.graph	1.60×10^8	$\mathbf{1.36 \times 10^8}$	1.17×10^{12}	$\mathbf{8.90 \times 10^{11}}$
p2p-Gnutella04.graph	4.76×10^6	$\mathbf{4.69 \times 10^6}$	9.15×10^9	$\mathbf{8.97 \times 10^9}$
PGPgiantcompo.graph	7.72×10^4	7.72×10^4	$\mathbf{3.86 \times 10^6}$	3.88×10^6
soc-Slashdot0902.graph	6.57×10^7	$\mathbf{5.57 \times 10^7}$	4.72×10^{11}	$\mathbf{3.74 \times 10^{11}}$
web-Google.graph	2.62×10^7	2.62×10^7	5.02×10^{10}	$\mathbf{5.01 \times 10^{10}}$
wiki-Talk.graph	1.00×10^8	$\mathbf{9.78 \times 10^7}$	8.80×10^{11}	$\mathbf{8.60 \times 10^{11}}$
wordassociation-2011.graph	$\mathbf{4.12 \times 10^6}$	4.18×10^6	$\mathbf{6.84 \times 10^9}$	7.07×10^9

Table 5.9: Number of non-zeros and operation count for exhaustive and single application of reductions with combination 0 4 5. Best values are highlighted in bold.

5.3.5 Node Ordering with Clustering

We ran the clustering based node ordering algorithm with reduction orders 0 1 2 and 0 4 5 and default parameters for the nested dissection. Results for the graphs in Table 5.1 are presented in Tables 5.10 and 5.11. Running times are listed in Table 5.12

The number of non-zeros and operation count of the obtained orderings are higher than those from nested dissection without reductions, in some cases by multiple orders of magnitude. The clustering based algorithm yields better results for the graphs `p2p-Gnutella04`, `soc-Slashdot0902` and `wordassociation-2011`. For five of the six largest graphs `gotst` was not able to compute the factorization, which suggests that the ordering is much worse than one computed with nested dissection.

In terms of running time, the clustering based algorithm is faster than nested dissection without reductions in most cases, and often performs as well as the fastest variant of reduced nested dissection. However, the implementation is inefficient, in that to extract a subgraph corresponding to a cluster we iterate through all nodes and test if they are in the cluster. Optimizing this should bring the running time down further.

Graph	Clustering (0 1 2)	Clustering (0 4 5)	No Reductions
`amazon-2008`	—	—	2.23×10^{9}
`as-22july06`	1.04×10^{6}	9.61×10^{5}	$\mathbf{1.34 \times 10^{5}}$
`as-skitter`	5.85×10^{8}	5.76×10^{8}	$\mathbf{3.65 \times 10^{8}}$
`citationCiteseer`	6.82×10^{9}	6.70×10^{9}	$\mathbf{2.66 \times 10^{8}}$
`cnr-2000`	9.23×10^{8}	9.02×10^{8}	$\mathbf{4.92 \times 10^{6}}$
`coAuthorsCiteseer`	4.23×10^{9}	4.29×10^{9}	$\mathbf{4.16 \times 10^{7}}$
`coAuthorsDBLP`	7.67×10^{9}	7.75×10^{9}	$\mathbf{1.51 \times 10^{8}}$
`coPapersCiteseer`	—	—	5.04×10^{8}
`coPapersDBLP`	—	—	1.67×10^{9}
`email-EuAll`	1.01×10^{6}	1.47×10^{6}	$\mathbf{7.72 \times 10^{5}}$
`enron`	1.89×10^{7}	1.79×10^{7}	$\mathbf{3.06 \times 10^{6}}$
`eu-2005`	—	—	1.89×10^{8}
`in-2004`	—	—	2.26×10^{7}
`loc-brightkite_edges`	2.94×10^{7}	2.93×10^{7}	$\mathbf{1.56 \times 10^{7}}$
`loc-gowalla_edges`	3.76×10^{8}	3.93×10^{8}	$\mathbf{1.37 \times 10^{8}}$
`p2p-Gnutella04`	$\mathbf{4.63 \times 10^{6}}$	4.89×10^{6}	4.81×10^{6}
`PGPgiantcompo`	4.90×10^{6}	5.32×10^{6}	$\mathbf{7.55 \times 10^{4}}$
`soc-Slashdot0902`	$\mathbf{5.56 \times 10^{7}}$	5.66×10^{7}	5.69×10^{7}
`web-Google`	1.50×10^{10}	1.51×10^{10}	$\mathbf{2.47 \times 10^{7}}$
`wiki-Talk`	1.21×10^{9}	1.12×10^{9}	$\mathbf{1.06 \times 10^{8}}$
`wordassociation-2011`	4.32×10^{6}	$\mathbf{4.13 \times 10^{6}}$	4.20×10^{6}

Table 5.10: Number of non-zeros for node ordering with clustering and nested dissection without reductions Lowest values are highlighted in bold. Where values are missing, `gotst` was not able to finish the factorization.

Graph	Clustering (0 1 2)	Clustering (0 4 5)	No Reduction
`amazon-2008`	—	—	5.76×10^{13}
`as-22july06`	7.25×10^{8}	6.25×10^{8}	$\mathbf{6.66 \times 10^{6}}$
`as-skitter`	8.20×10^{12}	8.04×10^{12}	$\mathbf{4.62 \times 10^{12}}$
`citationCiteseer`	3.44×10^{14}	3.31×10^{14}	$\mathbf{2.09 \times 10^{12}}$
`cnr-2000`	1.37×10^{13}	1.35×10^{13}	$\mathbf{3.44 \times 10^{8}}$
`coAuthorsCiteseer`	1.60×10^{14}	1.64×10^{14}	$\mathbf{1.72 \times 10^{11}}$
`coAuthorsDBLP`	4.04×10^{14}	4.11×10^{14}	$\mathbf{1.29 \times 10^{12}}$
`coPapersCiteseer`	—	—	6.75×10^{12}
`coPapersDBLP`	—	—	3.83×10^{13}
`email-EuAll`	6.72×10^{8}	1.22×10^{9}	$\mathbf{5.18 \times 10^{8}}$
`enron`	5.00×10^{10}	4.23×10^{10}	$\mathbf{3.54 \times 10^{9}}$
`eu-2005`	—	—	6.41×10^{11}
`in-2004`	—	—	3.52×10^{9}
`loc-brightkite_edges`	1.04×10^{11}	1.02×10^{11}	$\mathbf{4.67 \times 10^{10}}$
`loc-gowalla_edges`	4.18×10^{12}	4.80×10^{12}	$\mathbf{9.16 \times 10^{11}}$
`p2p-Gnutella04`	$\mathbf{8.72 \times 10^{9}}$	9.54×10^{9}	9.33×10^{9}
`PGPgiantcompo`	5.25×10^{9}	6.32×10^{9}	$\mathbf{3.53 \times 10^{6}}$
`soc-Slashdot0902`	$\mathbf{3.70 \times 10^{11}}$	3.84×10^{11}	3.84×10^{11}
`web-Google`	1.31×10^{15}	1.30×10^{15}	$\mathbf{4.35 \times 10^{10}}$
`wiki-Talk`	3.20×10^{13}	2.74×10^{13}	$\mathbf{9.58 \times 10^{11}}$
`wordassociation-2011`	7.58×10^{9}	7.06×10^{9}	$\mathbf{6.98 \times 10^{9}}$

Table 5.11: Operation count for node ordering with clustering and nested dissection without reductions Lowest values are highlighted in bold. Where values are missing, `gotst` was not able to finish the factorization.

Graph	Clustering (0 1 2)	Clustering (0 4 5)	No Reductions	Fastest Variant
`amazon-2008`	4 104.19	**4 053.28**	6 120.52	5 214.1
`as-22july06`	32.64	**29.72**	78.95	31.78
`as-skitter`	12 890.6	12 523.7	13 272.8	**11 124.6**
`citationCiteseer`	1 121.34	**1 098.51**	1 851.58	1 275.44
`cnr-2000`	9 591.81	10 693.6	13 133.5	**9 005.64**
`coAuthorsCiteseer`	405.67	517.74	871.01	**193.23**
`coAuthorsDBLP`	906.16	910.87	1 412.14	**446.39**
`coPapersCiteseer`	3 423.01	5 975.66	17 175.1	**751.45**
`coPapersDBLP`	4 708.43	6 083.54	12 928.8	**2 421.95**
`email-EuAll`	45.34	**41.40**	123.71	43.69
`enron`	208.97	201.81	451.55	**185.21**
`eu-2005`	28 716.1	43 877.4	42 342.1	**24 389.1**
`in-2004`	19 700.9	27 154.8	37 903.6	**8 342.76**
`loc-brightkite_edges`	166.13	144.28	287.44	**142.08**
`loc-gowalla_edges`	1 379.75	**966.73**	1 971.31	976.18
`p2p-Gnutella04`	32.05	**29.69**	38.13	30.10
`PGPgiantcompo`	**2.68**	2.99	27.87	9.26
`soc-Slashdot0902`	517.70	554.11	523.23	**493.93**
`web-Google`	1 337.76	1 379.48	2 546.99	**1 253.78**
`wiki-Talk`	2 445.85	**2 291.23**	6 671.44	2 371.81
`wordassociation-2011`	56.38	54.24	76.47	**53.79**

Table 5.12: Running times for node ordering with clustering (first two columns, reduction combination in parentheses), for nested dissection without reductions and for the fastest variant of reduced nested dissection (from the experiments in Section 5.3.1). All running times are in seconds. Lowest running times are highlighted in bold.

Chapter 6

Discussion

6.1 Future Work

Our implementation of the minimum degree algorithm does not make use of the improvements described in Section 3.2. Incorporating these should further improve the running time. However, it is not necessarily clear if they can simply be used together with the reductions or if they need to be adjusted somehow. It might be helpful to associate with each node information on how it is related to the original graph.

So far, we use three exact and two inexact reductions. Corollary 4.2.1 suggests a new exact reduction, where nodes with degree two are eliminated. To implement this requires testing for every node with degree two to see if it is in a cycle. A naïve implementation based on breadth-first search would have a worst case complexity of $\mathcal{O}(n^2)$ for a graph with n nodes. However, this should be faster in practice. For example, all neighbors of a node x with degree two are in a cycle if x is in a cycle. If we can efficiently test if a node with degree two is in a cycle, we can also test if it is not in a cycle. Thus, we can find nodes that are separators and we need to compute fewer separators.

Orderings from our clustering-based algorithm are of low quality. Intuitively, we expect clusters to be ordered together. However, it is not clear how graph clustering and the minimum fill-in problem are connected. There are some possible modifications to this algorithm. First, a different clustering algorithm might improve the node orderings. How we define clusters plays an important role here. However, the clustering algorithm should not be too expensive, otherwise we do not gain the desired improvement in running time over reduced nested dissection. Second, the graph can be reduced before the clustering step. This way we can guarantee that nodes that should be eliminated together do not end up in different clusters. Lastly, it might be beneficial to take connections between the clusters into account when ordering the individual subgraphs.

6.2 Conclusion

In this thesis we introduced exact and inexact reductions for the minimum fill-in problem. We applied them in a nested dissection algorithm, which we call reduced nested dissection. We also introduced an algorithm for the minimum fill-in problem based on graph clustering.

Our reduced nested dissection algorithm is faster than nested dissection without reductions, with a median improvement close to 50%. It also yields orderings with lower number of non-zeros and operation count for the Cholesky factorization. However, applying reductions exhaustively has only a small impact on the quality of the node orderings.

The clustering based algorithm leads to node orderings that are orders of magnitudes worse than those from nested dissection, both in terms of the number of non-zeros and the operation count. However, the running time of our implementation is close to that of reduced nested dissection and can still be improved further.

Bibliography

[1] F. N. Abu Khzam. “Topics in graph algorithms: structural results and algorithmic techniques, with applications”. PhD thesis. 2003. URL: https://trace.tennessee.edu/utk_graddiss/1954/.

[2] T. Akiba and Y. Iwata. “Branch-and-reduce exponential/FPT algorithms in practice: A case study of vertex cover”. In: *Theor. Comput. Sci.* 609 (2016), pp. 211–225. ISSN: 0304–3975. DOI: 10.1016/j.tcs.2015.09.023.

[3] P. Amestoy, T. Davis, and I. Duff. “An approximate minimum degree ordering algorithm”. In: *SIAM J. Matrix Anal. Appl.* 17.4 (1996), pp. 886–905. DOI: 10.1137/S0895479894278952.

[4] C. Ashcraft. “Compressed graphs and the minimum degree algorithm”. In: *SIAM J. Sci. Comput.* 16.6 (1995), pp. 1404–1411. DOI: 10.1137/0916081.

[5] C. Ashcraft and J. W. H. Liu. “Generalized nested dissection: some recent progress”. In: *Proceedings of the Fifth SIAM Conference on Applied Linear Algebra.* Ed. by J. G. Lewis. SIAM Publications, 1994, pp. 130–134.

[6] C. Ashcraft and J. W. H. Liu. “Robust ordering of sparse matrices using multisection”. In: *SIAM J. Matrix Anal. Appl.* 19.3 (1998), pp. 816–832. DOI: 10.1137/S0895479896299081.

[7] U. Bertele and F. Brioschi. “A new algorithm for the solution of the secondary optimization problem in non-serial dynamic programming”. In: *J. Math. Anal. Appl.* 27.3 (1969), pp. 565–574.

[8] U. Bertele and F. Brioschi. “Contribution to nonserial dynamic programming”. In: *J. Math. Anal. Appl.* 28.2 (1969), pp. 313–325.

[9] J. R. Blair, P. Heggernes, and J. A. Telle. “A practical algorithm for making filled graphs minimal”. In: *Theor. Comput. Sci.* 250.1 (2001), pp. 125–141. DOI: 10.1016/S0304-3975(99)00126-7.

[10] A. Buluç, H. Meyerhenke, I. Safro, P. Sanders, and C. Schulz. “Recent advances in graph partitioning”. In: *Algorithm Engineering: Selected Results and Surveys*. Ed. by L. Kliemann and P. Sanders. Springer International Publishing, 2016, pp. 117–158. DOI: 10.1007/978-3-319-49487-6_4.

[11] J. F. Buss and J. Goldsmith. “Nondeterminism within P^*”. In: *SIAM J. Comput.* 22.3 (1993), pp. 560–572. DOI: 10.1137/0222038.

[12] J. Chen, I. A. Kanj, and G. Xia. “Improved upper bounds for vertex cover”. In: *Theor. Comput. Sci.* 411.40 (2010), pp. 3736–3756. DOI: 10.1016/j.tcs.2010.06.026.

[13] B. Chor, M. Fellows, and D. Juedes. “Linear kernels in linear time, or how to save k colors in $\mathcal{O}(n^2)$ steps”. In: *Graph-Theoretic Concepts in Computer Science*. Ed. by J. Hromkovič, M. Nagl, and B. Westfechtel. Berlin, Heidelberg: Springer, 2005, pp. 257–269.

[14] F. R. K. Chung and D. B. Mumford. “Chordal completions of planar graphs”. In: *J. Comb. Theory. B* 62.1 (1994), pp. 96–106. DOI: 10.1006/jctb.1994.1056.

[15] T. A. Davis, J. R. Gilbert, S. I. Larimore, and E. G. Ng. “A column approximate minimum degree ordering algorithm”. In: *ACM Trans. Math. Softw.* 30.3 (2004), pp. 353–376. DOI: 10.1145/1024074.1024079.

[16] T. A. Davis and Y. Hu. “The University of Florida Sparse Matrix Collection”. In: *ACM Trans. Math. Softw.* 38.1 (2011), pp. 1–25. DOI: 10.1145/2049662.2049663.

[17] T. A. Davis, S. Rajamanickam, and W. M. Sid-Lakhdar. “A survey of direct methods for sparse linear systems”. In: *Acta Numer.* 25 (2016), pp. 383–566. DOI: 10.1017/S0962492916000076.

[18] A. George. “Nested dissection of a regular finite element mesh”. In: *SIAM J. Numer. Anal.* 10.2 (1973), pp. 345–363. DOI: 10.1137/0710032.

[19] A. George and J. W. H. Liu. “A fast implementation of the minimum degree algorithm using quotient graphs”. In: *ACM Trans. Math. Softw.* 6.3 (1980), pp. 337–358. DOI: 10.1145/355900.355906.

[20] A. George and J. W. H. Liu. “A quotient graph model for symmetric faetorization”. In: *Sparse Matrix Proceedings 1978*. Ed. by I. S. Duff and G. W. Stewart. SIAM Publications, 1978, pp. 154–175.

[21] A. George and J. W. H. Liu. “An automatic nested dissection algorithm for irregular finite element problems”. In: *SIAM J. Numer. Anal.* 15.5 (1978), pp. 1053–1069. DOI: `10.1137/0715069`.

[22] A. George and J. W. H. Liu. “The evolution of the minimum degree ordering algorithm”. In: *SIAM Rev.* 31.1 (1989), pp. 1–19. DOI: `10.1137/1031001`.

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

[24] M. T. Heath, E. Ng, and B. W. Peyton. “Parallel algorithms for sparse linear systems”. In: *SIAM Rev.* 33.3 (1991), pp. 420–460. DOI: `10.1137/1033099`.

[25] P. Heggernes. “Minimal triangulations of graphs: A survey”. In: *Discrete Math.* 306.3 (2006). Minimal Separation and Minimal Triangulation, pp. 297–317. DOI: `10.1016/j.disc.2005.12.003`.

[26] B. Hendrickson and E. Rothberg. “Improving the run time and quality of nested dissection ordering”. In: *SIAM J. Sci. Comput.* 20.2 (1998), pp. 468–489. DOI: `10.1137/S1064827596300656`.

[27] A. J. Hoffman, M. S. Martin, and D. J. Rose. “Complexity bounds for regular finite difference and finite element grids”. In: *SIAM J. Numer. Anal.* 10.2 (1973), pp. 364–369. DOI: `10.1137/0710033`.

[28] Y. Iwata. “A faster algorithm for dominating set analyzed by the potential method”. In: *Parameterized and Exact Computation*. Ed. by D. Marx and P. Rossmanith. Berlin, Heidelberg: Springer, 2012, pp. 41–54.

[29] G. Karypis and V. Kumar. “A fast and high quality multilevel scheme for partitioning irregular graphs”. In: *SIAM J. Sci. Comput.* 20.1 (1998), pp. 359–392. DOI: `10.1137/S1064827595287997`.

[30] S. Lamm, P. Sanders, C. Schulz, D. Strash, and R. F. Werneck. “Finding near-optimal independent sets at scale”. In: *J. Heuristics* 23.4 (2017), pp. 207–229. DOI: `10.1007/s10732-017-9337-x`.

[31] S. L. Lauritzen and D. J. Spiegelhalter. “Local computations with probabilities on graphical structures and their application to expert systems”. In: *J. Roy. Stat. Soc. B. Met.* 50.2 (1988), pp. 157–194. DOI: `10.1111/j.2517-6161.1988.tb01721.x`.

[32] R. J. Lipton, D. J. Rose, and R. E. Tarjan. “Generalized nested dissection”. In: *SIAM J. Numer. Anal.* 16.2 (1979), pp. 346–358. DOI: 10.1137/0716027.

[33] R. J. Lipton and R. E. Tarjan. “A separator theorem for planar graphs”. In: *SIAM J. Appl. Math.* 36.2 (1979), pp. 177–189. DOI: 10.1137/0136016.

[34] H. M. Markowitz. “The elimination form of the inverse and its application to linear programming”. In: *Manag. Sci.* 3.3 (1957), pp. 255–269. DOI: 10.1287/mnsc.3.3.255.

[35] H. Meyerhenke, P. Sanders, and C. Schulz. “Partitioning complex networks via size-constrained clustering”. In: *Experimental Algorithms*. Ed. by J. Gudmundsson and J. Katajainen. Springer International Publishing, 2014, pp. 351–363.

[36] T. Ohtsuki, L. K. Cheung, and T. Fujisawa. “Minimal triangulation of a graph and optimal pivoting order in a sparse matrix”. In: *J. Math. Anal. Appl.* 54.3 (1976), pp. 622–633. DOI: 10.1016/0022-247X(76)90182-7.

[37] S. Parter. “The use of linear graphs in gauss elimination”. In: *SIAM Rev.* 3.2 (1961), pp. 119–130. DOI: 10.1137/1003021.

[38] F. Pellegrini. *Scotch*. URL: https://www.labri.fr/perso/pelegrin/scotch/ (visited on 03/18/2019).

[39] A. Pothen, H. Simon, and K. Liou. “Partitioning sparse matrices with eigenvectors of graphs”. In: *SIAM J. Matrix Anal. Appl.* 11.3 (1990), pp. 430–452. DOI: 10.1137/0611030.

[40] U. N. Raghavan, R. Albert, and S. Kumara. “Near linear time algorithm to detect community structures in large-scale networks”. In: *Phys. Rev. E* 76 (3 2007), p. 036106. DOI: 10.1103/PhysRevE.76.036106.

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

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

[43] D. Rose, R. E. Tarjan, and G. S. Lueker. “Algorithmic aspects of vertex elimination on graphs”. In: *SIAM J. Comput.* 5.2 (1976), pp. 266–283. DOI: 10.1137/0205021.

[44] P. Sanders and C. Schulz. "Advanced multilevel node separator algorithms". In: *Experimental Algorithms*. Ed. by A. V. Goldberg and A. S. Kulikov. Springer International Publishing, 2016, pp. 294–309.

[45] P. Sanders and C. Schulz. "Think locally, act globally: Highly balanced graph partitioning". In: *Proceedings of the 12th International Symposium on Experimental Algorithms (SEA'13)*. Vol. 7933. LNCS. Springer, 2013, pp. 164–175.

[46] C. Schulz. "High quality graph partitioning". PhD thesis. 2013. DOI: `10.5445/IR/1000035713`.

[47] A. J. Soper, C. Walshaw, and M. Cross. "A combined evolutionary search and multilevel optimisation approach to graph-partitioning". In: *J. Global. Optim.* 29.2 (2004), pp. 225–241. DOI: `10.1023/B:JOGO.0000042115.44455.f3`.

[48] B. Speelpenning. *The generalized element method*. Tech. rep. UIUCDCS-R-78-946. Urbana, IL: Department of Computer Science, University of Illinois at Urbana-Champaign, 1978.

[49] R. E. Tarjan and A. E. Trojanowski. "Finding a maximum independent set". In: *SIAM J. Comput.* 6.3 (1977), pp. 537–546. DOI: `10.1137/0206038`.

[50] W. F. Tinney and J. W. Walker. "Direct solutions of sparse network equations by optimally ordered triangular factorization". In: *Proc. IEEE* 55.11 (1967), pp. 1801–1809. DOI: `10.1109/PROC.1967.6011`.

[51] M. Xiao and H. Nagamochi. "Confining sets and avoiding bottleneck cases: A simple maximum independent set algorithm in degree-3 graphs". In: *Theor. Comput. Sci.* 469 (2013), pp. 92–104. DOI: `10.1016/j.tcs.2012.09.022`.

[52] M. Xiao and H. Nagamochi. "Exact algorithms for maximum independent set". In: *Algorithms and Computation*. Ed. by L. Cai, S.-W. Cheng, and T.-W. Lam. Berlin, Heidelberg: Springer, 2013, pp. 328–338.

[53] M. Yannakakis. "Computing the minimum fill-in is NP-complete". In: *SIAM J. Algebraic Discrete Methods* 2.1 (1981), pp. 77–79. DOI: `10.1137/0602010`.