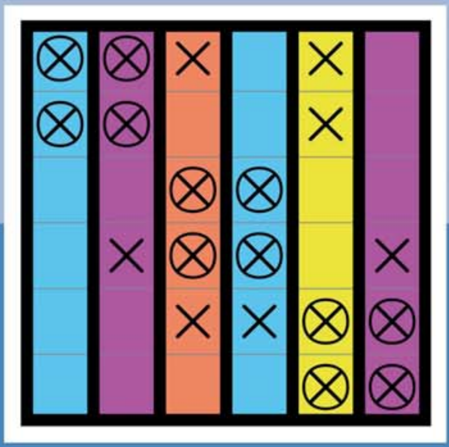
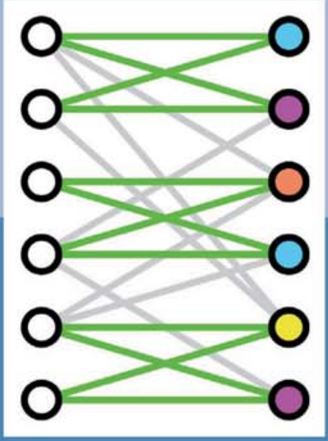
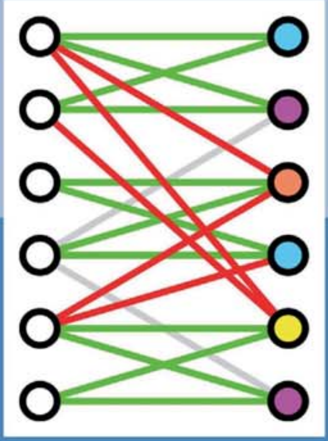
Full and partial Jacobian computation via graph coloring: Algorithms and applications

Michael Lülfesmann







Cuvillier Verlag Göttingen
Internationaler wissenschaftlicher Fachverlag

Full and partial Jacobian computation via graph coloring: Algorithms and applications

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

Full and partial Jacobian computation via graph coloring: Algorithms and applications

Der Fakultät für Mathematik, Informatik und Naturwissenschaften
der RWTH Aachen University vorgelegte Dissertation zur Erlangung des
akademischen Grades eines Doktors der Naturwissenschaften

vorgelegt von

Diplom-Informatiker Michael Lülfesmann
aus Bielefeld

Berichter: apl. Professor Dr.-Ing. H. Martin Bücker
Professor Dr. Rob H. Bisseling
Professor Dr. Berthold Vöcking

Tag der mündlichen Prüfung: 5. April 2012

Diese Dissertation ist auf den Internetseiten der Hochschulbibliothek online verfügbar.

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

Bibliografische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.d-nb.de abrufbar.
1. Aufl. - Göttingen: Cuvillier, 2012
Zugl.: (RWTH) Aachen, Univ., Diss., 2012

978-3-95404-101-5

D 82 (Diss. RWTH Aachen University, 2012)

© CUVILLIER VERLAG, Göttingen 2012
Nonnenstieg 8, 37075 Göttingen
Telefon: 0551-54724-0
Telefax: 0551-54724-21
www.cuvillier.de

Alle Rechte vorbehalten. Ohne ausdrückliche Genehmigung des Verlages ist es nicht gestattet, das Buch oder Teile daraus auf fotomechanischem Weg (Fotokopie, Mikrokopie) zu vervielfältigen.
1. Auflage, 2012
Gedruckt auf säurefreiem Papier

978-3-95404-101-5

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

Abstract

Simulations and optimizations are carried out to investigate real-world problems in science and engineering. For instance, solving systems of linear equations with sparse Jacobian matrices is mandatory when using a Newton-type algorithm. The sparsity of Jacobian matrices is exploited and only a subset of the nonzero elements is determined to successfully reduce the usage of the restricting resources—memory and computational effort. This reduction is crucial to investigate real-world problems.

The determination of all nonzero elements is denoted as full Jacobian computation, opposed to the partial Jacobian computation where only a subset of the nonzero elements is computed. Reducing the computational effort to determine nonzero elements with automatic differentiation is modeled by graph coloring problems. Beside the bipartite graph model for general Jacobian matrices, regular Cartesian grids are a graph class arising from stencil-based computations. In this thesis, graph coloring algorithms for full and partial Jacobian computation are introduced, for both representations. Furthermore, for regular grids, the presented algorithms even result in minimal colorings. Thereafter, several classes of Jacobian matrices are considered to assess which coloring method should be employed for which class.

Iterative solvers for systems of linear equations are matrix-free and require solely access to (transposed) Jacobian matrix-vector products. These products are efficiently provided by automatic differentiation without storing the nonzero elements of the Jacobian matrix. However, when using standard preconditioning techniques to speed up the solution of the linear systems, the access to these nonzero elements is necessary. In this thesis, the preconditioning technique is restricted to a subset of the Jacobian elements which are determined using partial Jacobian computation. The bipartite graph model is employed to determine a coloring but also to carry out the symbolic factorization for the preconditioning. An initial set of nonzero elements is given; further nonzero elements are chosen without exceeding the computational effort and the available memory. A classification for these nonzero elements is introduced. Strategies and algorithms to select these elements are given.

Finally, the coloring algorithms as well as the combination of preconditioning and partial Jacobian computation are applied to several applications from science and engineering. It is shown that the demands of memory and computational effort are successfully reduced.

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

Zusammenfassung

Simulationen und Optimierungen werden genutzt, um anwendungsnahe Fragestellungen in den Natur- und Ingenieurwissenschaften zu untersuchen. Das Lösen von linearen Gleichungssystemen mit dünnbesetzten Jacobi-Matrizen ist zum Beispiel für das Newton-Verfahren zwingend erforderlich. Speicherverbrauch und Berechnungsaufwand werden durch das Ausnutzen der Dünnbesetztheit dieser Matrizen und die Berechnung einer Teilmenge der Nichtnullelemente verringert. Diese Reduktion ist für die Untersuchung von anwendungsnahen Fragestellungen entscheidend.

Das Bestimmen aller Nichtnullelemente einer Jacobi-Matrix wird als vollständige Berechnung bezeichnet. Bei der partiellen Berechnung hingegen wird nur eine Teilmenge der Nichtnullelemente bestimmt. Die Nichtnullelemente werden mit Hilfe des automatischen Differenzierens berechnet. Das Verringern des Berechnungsaufwands wird als Graphfärbungsproblem modelliert. Neben dem bipartiten Graphmodell für Jacobi-Matrizen mit beliebiger Struktur werden auch reguläre kartesische Gitter betrachtet. In dieser Arbeit werden Graphfärbungsalgorithmen sowohl für die vollständige als auch für die partielle Berechnung von Jacobi-Matrizen eingeführt. Für die regulären Gitter sind die Färbungen sogar minimal. Abschließend wird für verschiedene Matrixklassen das Färbungsverfahren, das zur geringsten Farbanzahl führt, gesucht.

Zum Lösen linearer Gleichungssysteme mit Hilfe eines iterativen Verfahrens ist die Berechnung von (transponierten) Jacobi-Matrix-Vektor-Produkten ausreichend. Eine explizite Aufstellung der Jacobi-Matrizen ist nicht erforderlich. Durch das automatische Differenzieren werden Jacobi-Matrix-Vektor-Produkte effizient zur Verfügung gestellt, wobei das Speichern der Nichtnullelemente der entsprechenden Jacobi-Matrix nicht erforderlich ist. Der Zugriff auf die Nichtnullelemente ist jedoch notwendig, um den Lösungsprozess mit Hilfe von Standardtechniken der Vorkonditionierung zu beschleunigen. In dieser Arbeit werden die Vorkonditionierer aufgrund einer Teilmenge der Nichtnullelemente bestimmt. Das bipartite Graphmodell wird nicht nur verwendet, um eine Färbung zu berechnen, sondern auch, um das symbolische Faktorisieren für die Vorkonditionierung durchzuführen. Nachdem eine anfängliche Teilmenge von Nichtnullelementen gegeben ist, werden weitere Nichtnullelemente ausgewählt, ohne dass der Berechnungsaufwand und der verfügbare Speicher überschritten werden. Nach der Klassifizierung der Nichtnullelemente werden Auswahlstrategien und entsprechende Algorithmen eingeführt.

Sowohl die Färbungsalgorithmen als auch die Kombination von Vorkonditionierung und partieller Berechnung von Jacobi-Matrizen werden in Anwendungen der Natur- und Ingenieurwissenschaften angewendet. Dadurch wird gezeigt, dass die Anforderungen an den Speicher und den Berechnungsaufwand erfolgreich verringert werden.

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

Acknowledgements

My deepest gratitude is to my advisor, apl. Professor Dr.-Ing. H. Martin Bücker. Starting with an introductory course in high-performance computing many years ago, he introduced me to the world of science and guided me till finishing my thesis.

I would like to acknowledge the reviewers of this thesis, Professor Dr. Rob H. Bisseling and Professor Dr. Berthold Vöcking, for carefully reading and commenting on my thesis.

At the Institute of Scientific Computing at the RWTH Aachen University I really appreciated the stimulating research environment. I would like to express thanks to Professor Christian H. Bischof, Ph.D., the former Institute's head, for his advises how to manage research and projects. Special thanks go to my former colleagues Dr. Oliver Fortmeier, Dr. Monika Petera, Dr. Arno Rasch, Dr. Andre Vehreschild, Johannes Willkomm, and Dr. Andreas Wolf. Beside valuable discussions, I could benefit from their different skills and experiences.

Professor Ken-ichi Kawarabayashi, Ph.D. of the National Institute of Informatics, Tokyo, Japan granted me kind hospitality during my research visit in 2010. Many thanks to him. He drew my attention to work on exact graph coloring algorithms.

While using parallel computing resources and parallelization techniques, the support of the members of the HPC Team of Center for Computing and Communication (CCC) at the RWTH Aachen University was very helpful. In particular, I would like to express special thanks to Dirk Schmidl. I am also thankful to the students, Alexandru Calotoiu, Mircean Gherzan, Daniel Hering, and Simon Lessenich, for their collaboration. Special thanks to the proofreaders, Dr. Oliver Fortmeier, Dirk Schmidl, Christian Terboven, and Johannes Willkomm, and their comments on different parts of my thesis. For the case studies, I really appreciated the support with simulation programs given by some researchers from the RWTH Aachen University.

Christina supported me throughout the process of writing my thesis. Most notably, she distracted me to regain my strength. Thank you.

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

Contents

1 Introduction **1**

2 Exploiting sparsity in Jacobian computation **5**
2.1 Full Jacobian computation 5
2.2 Partial Jacobian computation 7
2.3 Graph representation 9
2.4 Grid representation 12

3 Colorings for full and partial Jacobian computation **17**
3.1 Exact sub-exponential coloring algorithm for regular grids 17
3.1.1 Full Jacobian computation 17
3.1.2 Partial Jacobian computation 30
3.2 Size-independent exact coloring algorithm for regular grids 35
3.3 Two-sided coloring algorithm for general graphs 41
3.4 Are two-sided colorings better than one-sided colorings? 49
3.4.1 Main diagonal 49
3.4.2 Block diagonal 51
3.4.3 Stencils 52

4 Preconditioning using partial Jacobian computation **57**
4.1 Iterative solvers using preconditioning techniques 57
4.2 Preconditioning and automatic differentiation 58
4.3 Motivation: Block diagonal 60
4.4 Choosing elements for preconditioning 62
4.4.1 Initially required elements 65
4.4.2 Potentially required elements 68
4.4.3 Additionally required elements 74
4.4.4 Parallelism and required elements 81
4.5 Results 84

5 Case studies in science and engineering **91**
5.1 Distillation in process engineering 91
5.2 Carbon sequestration in geosciene 93
5.3 Flow field around an airfoil in aeronautical engineering 97
5.4 Blood flow in biomedical engineering 99

6 Concluding summary **103**

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

Bibliography **105**

Appendix **111**
- A.1 Overview of stencils . . . 111
  - A.1.1 Full Jacobian computation . . . 111
  - A.1.2 Partial Jacobian computation . . . 111
- A.2 Colored tiles . . . 112
  - A.2.1 Full Jacobian computation . . . 112
  - A.2.2 Partial Jacobian computation . . . 112
- A.3 Proof of Lemma 3.8 . . . 113

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

1 Introduction

Simulations using partial differential equations (PDE) and PDE-constrained optimizations are employed to attack real-world problems in scientific computing. In both areas, solving systems of linear equations with Jacobian matrices as coefficient matrices is mandatory. To solve these linear systems, sparse Jacobian matrices demand a matrix-free iterative solver with access to (transposed) Jacobian matrix-vector products. Automatic differentiation is a technique to provide not only Jacobian matrices but also these products. The great benefit of these Jacobian matrix-vector products is that there is no need to store the nonzero elements of the Jacobian matrix. If a Jacobian matrix exceeds the main memory, it is essential to use matrix-free iterative solvers. Memory consumption and computational effort are restricting resources in simulation and optimization. In this thesis, the sparsity of Jacobian matrices is exploited to decrease both resources.

Exploiting the sparsity of Jacobian matrices is important to decrease the computational effort. Determining all nonzero elements is denoted as full Jacobian computation, opposed to the partial Jacobian computation where only a subset of the nonzero elements is computed. Determining such a subset is important for employing the preconditioning techniques in the second part of this thesis.

Nonzero elements may be determined row- or column-wise with automatic differentiation. Either rows or columns can be combined to linear combinations to reduce the computational effort. There are special classes of matrices for which the number of linear combinations can be reduced further. Therefore, the nonzero elements can be determined by rows and columns. These rows and columns are combined to linear combinations of rows and linear combinations of columns. The arrow-shaped matrix is a good example for this matrix class. These reductions of linear combinations can be modeled as combinatorial optimization problems, in particular, graph coloring problems. That is, each linear combination is associated to a color. Combining either rows or columns is denoted as one-sided coloring, whereas combining rows and columns is denoted as two-sided coloring.

Reducing the number of colors for full and partial Jacobian computation is studied in the first main part of this thesis. In particular, algorithms to solve the resulting graph coloring problems are developed. Regular Cartesian grids are a graph class occurring in stencil-based computations. A sub-exponential exact coloring algorithm is introduced to determine minimal colorings for full and partial Jacobian computation. This algorithm takes advantage of the grid and stencil properties by employing a divide-and-conquer scheme and separators derived from Lipton-Tarjan separators. Thereafter, since the runtime depends directly on the grid size, a grid size-independent approach with linear time complexity is invented. A small grid is

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

colored and, then, a coloring information—if available—is extracted which is sufficient to color a larger grid of arbitrary size with the smallest number of colors. Although this approach is not universal, it works for all considered stencils. Following one-sided colorings to exploit Jacobian matrices in stencil-based computations, the next topic is two-sided colorings for general graphs. An algorithm for partial Jacobian computation is introduced. Thereafter, several classes of Jacobian matrices are considered to assess whether there is a two-sided coloring with less colors than a minimal one-sided coloring.

For matrix-free iterative solvers, the access to Jacobian matrix-vector products is sufficient without assembling the full matrix. Unfortunately, standard preconditioning techniques expect that all nonzero elements of a Jacobian matrix are available. Using preconditioners increases the convergence behavior and speeds up solving systems of linear equations. If the available memory is a limiting resource, storing all these elements is often impossible. Cullum and Tuma [21] proposed to restrict the input for the preconditioning method to a subset of the nonzero elements of a Jacobian matrix. However, they do not address how to choose this subset. Gebremedhin, Pothen, and Manne [27] introduced the coloring definition for the partial Jacobian computation and mentioned its usage for preconditioning. When a subset of the Jacobian elements—solely for preconditioning purposes—is determined, these nonzero elements may be stored and, afterwards, the preconditioning techniques are restricted to these elements.

Combining preconditioning techniques with the partial Jacobian computation is the topic of the second half of this thesis. It is assumed that the sparsity pattern of the Jacobian matrix is available. This information is employed to determine a preconditioner. A set of nonzero elements is suggested by domain experts and, then, additional nonzero elements are chosen to speed up solving systems of linear equations. When these elements are chosen, the limited memory and the computational effort must be taken into account. To determine the initial nonzero elements, a coloring with a specific number of colors is needed. Additional nonzero elements are chosen without increasing this number of colors and without exceeding the available memory. The nonzero elements are categorized to different classes. Several algorithms are developed to choose nonzero elements. At the end, a couple of additional nonzero elements are omitted to obtain a structure which is beneficial for solving the preconditioned system of linear equations in parallel. The aim is to reduce the degree of dependence between the employed processors.

The main contributions of this thesis are in the fields of coloring algorithms for sparsity exploitation of Jacobian matrices as well as the combination of preconditioning and partial Jacobian computation. In the first part, a sub-exponential exact coloring algorithm and a linear-time algorithm which is independent of the original grid size are introduced for full and partial Jacobian computation in stencil-based computations. For general graphs, a two-sided coloring algorithm for partial Jacobian computation is given. Thereafter, several classes of Jacobian matrices are considered to assess whether there is a two-sided coloring with less colors than a minimal one-sided coloring. In the second part, the combination of preconditioning and the partial

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

Jacobian computation is a completely new approach. It includes the categorization of nonzero elements and choosing specific nonzero elements for preconditioning.

This thesis is structured as follows: Already known graph models associated to sparse Jacobian matrices are described in Chap. 2. These models comprise a bipartite graph model for general Jacobian matrices and a regular Cartesian grid for Jacobian matrices arising from stencil-based computations. Furthermore, the optimization problems concerning the exploitation of Jacobian matrices and a first introduction to graph coloring algorithms is given. Moreover a consistent notation for the thesis is introduced. In Chap. 3, an exact sub-exponential coloring algorithm on regular grids for full and partial Jacobian computation in stencil-based computation is presented. Another new coloring algorithm to compute minimal colorings is presented to reduce the complexity of the previous algorithm. The determination of the coloring information is independent of the original grid size. For general graphs, a coloring algorithm for the partial Jacobian computation is described which is also applicable for the full Jacobian computation. Thereafter, for some matrix classes, the reduction in the number of colors by using two-sided colorings compared to one-sided colorings is evaluated. In the following chapter, a preconditioning technique is combined with the partial Jacobian computation. The preconditioning for solving systems of linear equations is introduced and motivated. Afterwards, the nonzero elements of Jacobian matrices are classified. For every class, algorithms are introduced to determine different subsets of required nonzero elements. At the end, solving preconditioned systems of linear equations in parallel is considered. The usage of the required elements for preconditioning is evaluated by the number of matrix-vector products, number of nonzero elements, and number of colors. In Chap. 5, several applications from science and engineering are considered to show the practical relevance for using the previously described techniques, in particular, the graph coloring and preconditioning. This thesis closes with a concluding summary.

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

2 Exploiting sparsity in Jacobian computation

Before describing the progress in coloring algorithms and partial Jacobian computation for preconditioning, we give a short survey of the state of the art in exploiting the sparsity of Jacobian matrices. After presenting the Jacobian computation for all nonzero elements, we explain how to reduce the computational effort when only a subset of these nonzero elements is determined. Thereafter, these exploitation techniques are modeled as graph coloring problems. Finally, the focus is on Jacobian matrices occurring from discretizing domains with stencil-based methods. In contrast to general graphs, regular grids are employed.

2.1 Full Jacobian computation

Given a program implementing some mathematical function

$$f(x) : \mathbb{R}^n \;\rightarrow\; \mathbb{R}^m, \tag{2.1}$$

the derivative of the vector-valued function f with respect to some vector $x \in \mathbb{R}^n$ in the direction of a vector $s \in \mathbb{R}^n$ is defined by

$$\frac{\partial f}{\partial x} s = \lim_{h \to 0} \frac{f(x + hs) - f(x)}{h}. \tag{2.2}$$

Let $A := \partial f / \partial x$ denote the $m \times n$ Jacobian matrix whose columns are given by

$$A = [a_1 a_2 \cdots a_n].$$

Then, by choosing $s \in \{0,1\}^n$ as a binary vector, any sum of columns a_j can be computed where the jth entry of s is nonzero, i.e.,

$$As = \sum_{j \text{ with } s_j = 1} a_j.$$

Moreover, the product of the Jacobian matrix A and some $n \times p$ *seed matrix* S can be approximated by $p + 1$ evaluations of the function f using divided differencing. Similarly, the forward mode of automatic differentiation is capable of computing that product, $A \cdot S$, without truncation error using $p + 1$ times the time needed to evaluate f. Therefore, p indicates a rough measure of the time needed to compute the Jacobian matrix.

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

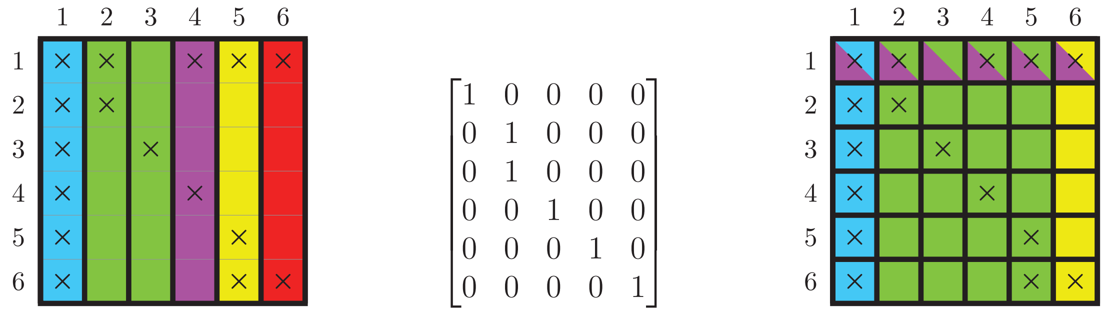

(a) (b) (c)

Figure 2.1: (a) Sparsity pattern of Jacobian matrix A with $p = 5$ column groups ({1}, {2,3}, {4}, {5}, {6}). (b) Seed matrix S corresponding to column groups in (a). (c) Bidirectional partition of A with $p = 4$ groups (row: {1}, column: {1}, {2,. . . ,5}, {6}).

Automatic differentiation (AD) [31, 55] comprises a set of techniques to transform the function f into another function which computes the derivative. The two major modes of AD are the forward mode to compute linear combinations of the columns of a Jacobian matrix and the reverse mode to compute linear combinations of the rows. Using AD in a naïve way in the forward or reverse mode, we need $p = n$ or $p = m$ directional derivatives, respectively.

The sparsity pattern of a Jacobian matrix A can be determined beforehand or is known due to the discretization of the underlying physical model. This information can be used to decrease the number of directional derivatives p by combining several columns to a linear combination without losing values. This technique is called *column compression*. The idea to reduce p—and hence the time to compute all nonzero elements of a sparse Jacobian matrix—consists of partitioning the columns of the Jacobian matrix into groups of those columns whose sum contains all the nonzero elements of the columns in that group [22]. The definition and the corresponding problem for row groups is straightforward. The property characterizing such a column group is introduced in the following definition:

Definition 2.1. Two columns a_i and a_j are *structurally orthogonal* if and only if they do not have any nonzero element in the same row, i.e.,

$$a_i \perp a_j \quad :\Longleftrightarrow \quad \nexists k :\ a_{k,i} \neq 0 \wedge a_{k,j} \neq 0.$$

In the example given in Fig. 2.1(a), the columns a_2 and a_3 are structurally orthogonal since there is no row in which both columns have a nonzero element. So, the sum $a_2 + a_3$ contains all nonzero elements of these two columns. The columns a_1 and a_2 are not structurally orthogonal, so-called *structurally non-orthogonal*, because both have nonzero elements in rows 1 and 2. The combinatorial optimization problem to find a minimal p is formulated as follows:

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

Problem 2.2. *Given a Jacobian matrix A, partition its columns into a minimal number of groups of structurally orthogonal columns. More precisely, find a binary $n \times p$ matrix S such that all nonzero elements of A are contained in the matrix-matrix product $A \cdot S$ and the number of columns p, representing the number of groups, is minimized.*

A solution to that problem, a so-called *unidirectional partition*, may not be unique. For the illustrating example in Fig. 2.1(a), a solution is indicated by using different colors. This unidirectional partition consists of $p = 5$ column groups $\{1\}$, $\{2, 3\}$, $\{4\}$, $\{5\}$, and $\{6\}$. Compared to the naïve way, this is a reduction of one directional derivative. The corresponding seed matrix is given in Fig. 2.1(b).

A further reduction is possible using a combination of rows and columns, a so-called *bidirectional partition*. This is beneficial if there is at least one column and one row which are pretty dense, i.e., there are a lot of nonzero elements in this column or row, respectively. An obvious example for the bidirectional partitioning is the arrow shaped matrix with one full row, one full column, and a full diagonal. The combinatorial optimization problem to find a minimal number of groups is formulated as follows:

Problem 2.3. *Given a Jacobian matrix A, find a binary $p_r \times m$ matrix S_r and a binary $n \times p_c$ matrix S_c such that all nonzero elements of A are contained in the matrix-matrix products $S_r \cdot A$ and $A \cdot S_c$, and the number of rows and columns $p = p_r + p_c$, representing the number of groups, is minimized.*

An illustrating example for the bidirectional partitioning is depicted in Fig. 2.1(c). All nonzero elements in the first row are covered by the row group $\{1\}$. Now, the column groups can be determined without taking the nonzero elements of the first row into account. The columns a_2 and a_4 can be part of the same column group, although the values of the elements $a_{1,2}$ and $a_{1,4}$ are destroyed. The reason is that these elements are determined by the row group $\{1\}$. The bidirectional partition consists of the row group $\{1\}$ and the column groups $\{1\}$, $\{2, \dots, 5\}$, and $\{6\}$. That is, the bidirectional partition consists of $p = p_r + p_c = 1 + 3$ groups. This is a reduction of one group compared to the unidirectional partition.

2.2 Partial Jacobian computation

Rather than computing all nonzero elements of the Jacobian matrix A, only a proper subset of the nonzero elements should be determined. Computing such a set of *required* nonzero elements R is called *partial Jacobian computation* as opposed to *full* Jacobian computation where all nonzero elements are computed. The remaining nonzero elements are called *non-required* elements. Gebremedhin, Manne, and Pothen [27] introduced the rules for the partial Jacobian computation. In [40–42], the assumption was validated that the partial Jacobian computation reduces the number of groups p compared to the full Jacobian computation.

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

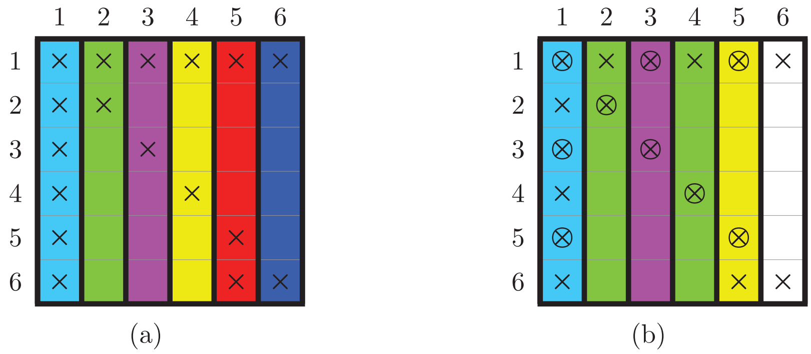


Figure 2.2: (a) Full Jacobian computation for Jacobian matrix with $p = 6$ column groups. (b) Partial Jacobian computation for Jacobian matrix from (a) with $p = 4$ column groups with required elements denoted by symbol $\otimes$ and non-required elements by symbol $\times$.

For the unidirectional partitioning, the property how to combine columns is introduced in the following definition:

Definition 2.4. Two columns a_i and a_j are *partially structurally orthogonal* with respect to the required elements R if and only if they do not have a nonzero element in the same row where at least one nonzero element is required, i.e.,

$$a_i \perp_R a_j \quad :\Longleftrightarrow \quad \nexists k : \ a_{k,i} \neq 0 \wedge a_{k,j} \neq 0 \wedge (a_{k,i} \in R \vee a_{k,j} \in R).$$

We consider the following Jacobian matrix to explain the column compression in partial Jacobian computation: Let the sparsity pattern of the 6×6 Jacobian matrix A be given by an arrow-shaped structure with an additional nonzero element $a_{6,5}$. This pattern is depicted in Fig. 2.2(a). All columns of this Jacobian matrix are structurally non-orthogonal due to row 1 which is full of nonzero elements. Six column groups are needed for the full Jacobian computation. This changes when we define a subset R with the required elements indicated by the symbol $\otimes$ in Fig. 2.2(b). The required elements in the first row are $a_{1,1}$, $a_{1,3}$, and $a_{1,5}$. The non-required elements are indicated by the symbol $\times$. Due to the fact that we are not interested in the non-required elements $a_{1,2}$ and $a_{1,4}$, the columns a_2 and a_4 are partially structurally orthogonal and can be combined into one column group. The column a_6 does not contain a required element and therefore is not part of any column group.

The adaption of the optimization problems 2.2 and 2.3 for the full Jacobian computation to the partial Jacobian computation is straightforward regarding the modified definition of partial structural orthogonality. We skip the definition of the bidirectional partitioning for partial Jacobian computation in this section and refer to the explanation in the next section with the corresponding graph model.

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

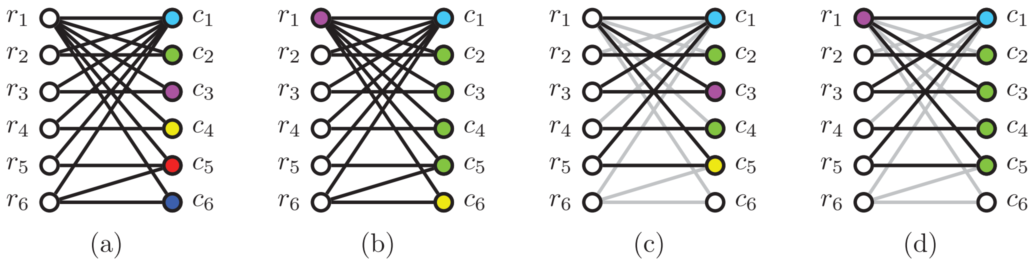


Figure 2.3: (a) Unidirectional partition and (b) bidirectional partition of the bipartite graph associated to Jacobian matrix in Fig. 2.2 for full Jacobian computation. (c) Unidirectional partition and (d) bidirectional partition for partial Jacobian computation with required edges indicated in black and non-required edges in gray.

2.3 Graph representation

Coleman and Moré [18] were the first authors who modeled the computation of sparse Jacobian matrices by graphs. In particular, they introduced the column intersection graph. Since then, various graph models have been used to describe different sparsity-exploiting derivative computations [16, 17, 33–35]. Coleman and Verma [19] introduced a bipartite graph model. In contrast to the other graph models, this model is sufficient for the bidirectional partitioning in full and partial Jacobian computation. Therefore, we mainly consider the bipartite graph throughout this thesis.

The sparsity pattern of a Jacobian matrix A can be represented as an undirected bipartite graph $G = (V_r \uplus V_c, E)$ which consists of m vertices in V_r to represent the rows and n vertices in V_c to represent the columns. The symbol $\uplus$ indicates that the sets V_r and V_c are disjoint. The vertex $r_i \in V_r$ corresponds to row i and the vertex $c_j \in V_c$ to column j. There is an edge $(i, j) \in E$ if and only if a nonzero element $a_{i,j} \neq 0$ exists. The bipartite graph G associated to the matrix in Fig. 2.2(a) consists of 6 vertices in V_r to represent the rows, 6 vertices in V_c to represent the columns, and 17 edges connecting vertices from V_r and V_c representing the nonzero elements. The graph G is depicted in Fig. 2.3.

The unidirectional partitioning of columns can be modeled as a graph coloring problem on the bipartite graph. The following definition is used to partition the column vertices into different groups.

Definition 2.5. Two column vertices c_i and c_j are *structurally orthogonal* if and only if they are not connected by a path of length 2, i.e.,

$$c_i \perp c_j \quad :\Longleftrightarrow \quad \nexists r_k \in V_r : \ (r_k, c_i) \in E \ \wedge \ (r_k, c_j) \in E.$$

Two vertices are *distance-k neighbors*, if they are connected by a path of length k. For column compression, all column vertices have to be colored so that different colors are assigned to any pair of distance-2 neighbors. This rule is given more precisely in the following definition:

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

Definition 2.6 (***Distance-2 coloring***)**.** Let $G = (V_r \uplus V_c, E)$ be a bipartite graph. A mapping $\Phi\colon V_c \to \{1, \dots, p\}$ is a *distance-2 coloring of* G if the following condition holds:

- for every path (c_i, r_k, c_j) with $c_i, c_j \in V_c$ and $r_k \in V_r$, $\Phi(c_i) \neq \Phi(c_j)$.

This coloring is also denoted as *one-sided coloring* due to its restriction to column vertices. An illustration of this definition is given in Fig. 2.3(a) where a group of structurally orthogonal vertices is indicated using the same color. This distance-2 coloring Φ corresponds to the unidirectional partitioning in Fig. 2.2(a). For example, the vertices c_1 and c_2 are colored differently, because they are distance-2 neighbors. Gebremedhin, Manne, and Pothen [27] denote this as a partial distance-2 coloring. The term partial is left out in the following to avoid confusion in connection with the term partial Jacobian computation. The field of one-sided coloring algorithms for full Jacobian computation is well-known. There are several notes [18, 22, 27, 34] describing algorithms and underlying graph models.

Minimizing the number of colors p of the coloring Φ means to minimize the computational effort. The combinatorial optimization problem in terms of the bipartite graph is then as follows:

Problem 2.7. *Given a bipartite graph $G = (V_r \uplus V_c, E)$, partition its column vertices into a minimal number of groups of structurally orthogonal vertices. More precisely, find a distance-2 coloring of V_c such that the number of colors p, representing the number of groups, is minimized.*

The distance-2 coloring and the optimization problem for the row vertices are defined analogously to the column vertices. The bidirectional partitioning can be modeled as a graph coloring problem, too. The rules for such a *two-sided coloring* of the bipartite graph are given in the following definition:

Definition 2.8 (***Star bicoloring*** [27])**.** Let $G = (V_r \uplus V_c, E)$ be a bipartite graph. A mapping $\Phi\colon [V_r \uplus V_c] \to \{0, 1, \dots, p\}$ is a *star bicoloring of* G if the following conditions are met:

1. Vertices in V_c and V_r receive disjoint colors, except for color 0; i.e., for every $r_i \in V_r$ and $c_j \in V_c$, either $\Phi(r_i) \neq \Phi(c_j)$ or $\Phi(r_i) = \Phi(c_j) = 0$.
2. At least one endpoint of every edge receives a nonzero color; i.e., for every $(r_i, c_j) \in E$, $\Phi(r_i) \neq 0$ or $\Phi(c_j) \neq 0$.
3. For every path (u, v, w) with $\Phi(v) = 0$, $\Phi(u) \neq \Phi(w)$.
4. Every path of length three with four vertices uses at least three colors.

Here, the mapping Φ assigns the colors $\{0, 1, \dots, p\}$ to vertices such that certain vertices receive different colors. The “neutral” color zero is distinguished from the remaining colors. A coloring of the form $\Phi(v) = 0$, which assigns the color zero to the vertex v, indicates that no color is actually assigned to this vertex. A star bicoloring according to Def. 2.8 is given in Fig. 2.3(b). This star bicoloring corresponds to the bidirectional partitioning given in Fig. 2.1(c). The optimization problem for the star bicoloring is then as follows:

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

Algorithm 2.1: Determine a distance-2 coloring of a bipartite graph G when restricted to E_R.

```
function D2ColoringRestricted($G = (V_r \uplus V_c, E), E_R$)
    coloring $\Phi \leftarrow [-1 \ldots -1]$; forbiddenColors $\leftarrow [0 \ldots 0]$
    foreach $c_i \in V_c$ with $\exists r_k : (r_k, c_i) \in E_R$ do
        foreach $c_j \in N_2(c_i, G, E_R)$ with $\Phi(c_j) > 0$ do
            forbiddenColor[$\Phi(c_j)$] $\leftarrow i$
        $\Phi(c_i) \leftarrow \min\{k > 0\colon$ forbiddenColor[$k$] $\neq i\}$
    return $\Phi$
```

Problem 2.9. *Given a bipartite graph $G = (V_r \uplus V_c, E)$, find a star bicoloring of V_c and V_r such that the number of colors $p = p_c + p_r$, representing the number of column and row groups, is minimized.*

Following [27,41,42], we adapt the Definitions 2.6 and 2.8 to definitions for the partial Jacobian computation on bipartite graphs. Therefore, an additional edge set E_R for the *required edges* corresponding to required elements is necessary. We note that the full Jacobian computation is a special case of the partial Jacobian computation when $E_R = E$ holds for the following Definitions 2.10 and 2.12. The following definition is concerned with the unidirectional partitioning by columns.

Definition 2.10 (*Restricted distance-2 coloring* [27]). Let $G = (V_r \uplus V_c, E)$ be a bipartite graph and let $E_R \subseteq E$ denote the set of required edges. A mapping $\Phi\colon V_c \to \{0, 1, \ldots, p\}$ is a *distance-2 coloring of G when restricted to E_R* if the following conditions hold for every edge $(r_i, c_j) \in E_R$, where $r_i \in V_r$ and $c_j \in V_c$:

1. $\Phi(c_j) \neq 0$, and
2. for every path (c_k, r_i, c_j) with $c_k \in V_c$, $\Phi(c_k) \neq \Phi(c_j)$.

A bipartite graph with a restricted distance-2 coloring is depicted in Fig. 2.3(c). The required edges are indicated in black and the non-required edges in gray. This restricted distance-2 coloring corresponds to the unidirectional partition of the Jacobian matrix in Fig. 2.2(b). The optimization problem for the restricted distance-2 coloring is as follows:

Problem 2.11. *Given a bipartite graph $G = (V_r \uplus V_c, E)$, partition its column vertices into a minimal number of groups of partially structurally orthogonal vertices. More precisely, find a distance-2 coloring of V_c when restricted to E_R such that the number of colors p, representing the number of groups, is minimized.*

This NP-hard problem [26, 40] is solved by heuristics. A heuristic to compute a restricted distance-2 coloring is given in Alg. 2.1. All vertices c_i which are incident to a required edge obtain a color which is not assigned to a distance-2 neighbor c_j before. The function $N_2(v, G)$ returns the distance-2 neighbors of the vertex v. The function $N_2(v, G, E_r)$ yields the distance-2 neighbors restricted to E_R. The index of

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

vertex c_i is stored in the entry of the array forbiddenColors which corresponds to the color of a distance-2 neighbor c_j of vertex c_i, $\Phi(c_j)$. When all distance-2 neighbors have been visited, the heuristic assigns the smallest valid color to vertex c_i.

In contrast to the previous definition addressing partial Jacobian computation for the unidirectional partitioning, the following definition is concerned with the bidirectional partitioning.

Definition 2.12 (***Restricted star bicoloring*** [27])**.** Let $G = (V_r \uplus V_c, E)$ be a bipartite graph and let $E_R \subseteq E$ denote the set of required edges. A mapping $\Phi\colon [V_r \uplus V_c] \to \{0, 1, \ldots, p\}$ is a *star bicoloring of G when restricted to E_R* if the following conditions are met:

1. Vertices in V_c and V_r receive disjoint colors, except for color 0; i.e., for every $r_i \in V_r$ and $c_j \in V_c$, either $\Phi(r_i) \neq \Phi(c_j)$ or $\Phi(r_i) = \Phi(c_j) = 0$.
2. At least one endpoint of an edge in E_R receives a nonzero color; i.e., for every $(r_i, c_j) \in E_R$ with $r_i \in V_r$ and $c_j \in V_c$, $\Phi(r_i) \neq 0$ or $\Phi(c_j) \neq 0$.
3. For every edge $(r_i, c_j) \in E_R$, $r_i, r_\ell \in V_r$, and $c_j, c_k \in V_c$,
    a) if $\Phi(r_i) = 0$, then, for every path (c_k, r_i, c_j), $\Phi(c_k) \neq \Phi(c_j)$;
    b) if $\Phi(c_j) = 0$, then, for every path (r_i, c_j, r_ℓ), $\Phi(r_i) \neq \Phi(r_\ell)$;
    c) if $\Phi(r_i) \neq 0$ and $\Phi(c_j) \neq 0$, then for every path (c_k, r_i, c_j, r_ℓ), either $\Phi(c_k) \neq \Phi(c_j)$ or $\Phi(r_i) \neq \Phi(r_\ell)$.

The corresponding minimization problem in terms of the bipartite graph is given as follows:

Problem 2.13. *Given a bipartite graph $G = (V_r \uplus V_c, E)$, find a star bicoloring of G when restricted to E_R such that the number of colors $p = p_c + p_r$, representing the number of column and row groups, is minimized.*

The vertices of a bipartite graph can be colored in different vertex orderings. These orderings have a deep impact on the result. That is, due to the greedy characteristic of the algorithm, differently ordered vertices result in a different number of colors. Certain preorderings are often better than the natural ordering $v_1, \ldots, v_n$ in terms of number of colors. Examples for preordering algorithms are: largest-first ordering (LFO) [62], smallest-last ordering [44], and incident-degree ordering (IDO) [10]. These orderings are carried out before applying the coloring heuristic. Furthermore, there are dynamic orderings which are determined during the coloring. That is, a vertex is immediately colored after its selection. Then, the next vertex is chosen and colored. For example, the dynamic largest-first ordering (DLFO) [28] is such an ordering method.

2.4 Grid representation

In the previous sections, the graph coloring problems are described for general Jacobian matrices. In this section, we focus on a special case of the graphs, regular

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

Cartesian grids. These grids can be employed together with stencil-based methods to discretize partial differential equations and, in addition, to approximate the partial derivatives. Surveys about several representations for exploiting the sparse Jacobian computation in this context are given in [13, 14]. The function

$$f : \mathbb{R}^{MN} \longrightarrow \mathbb{R}^{MN}, \tag{2.3}$$

which is a special case of (2.1), is computed by a stencil operation on a regular $M \times N$ grid. That is, the value of a quantity on a grid point is updated by the weighted values of the quantity on neighboring grid points. We consider only neighbors in space rather than in time. The neighborship relation for a grid point (m, n) is defined by the stencil $\mathcal{N}(m, n)$, the set of all neighboring grid points whose values influence the new value at (m, n). We assume that the update of a grid point involves its old value so that $(m, n) \in \mathcal{N}(m, n)$. The grid point (m, n) is called the *center* of the stencil $\mathcal{N}(m, n)$. An example for a five-point stencil is

$$\mathcal{N}_{\text{5pt}}(m, n) = \{(m + 1, n), (m - 1, n), (m, n), (m, n + 1), (m, n - 1)\} \tag{2.4}$$

for any center (m, n) that is not located on the boundary of the grid. That is, the neighbors of the center in $\mathcal{N}_{\text{5pt}}$ are immediately adjacent in the north, south, west, and east directions. A center on the boundary has less neighbors in $\mathcal{N}_{\text{5pt}}$. To avoid excessive case-by-case analyses of grid points whose neighborship intersects with the boundary, we informally use (2.4) to denote the neighborhood relationship for all points (m, n) with $1 \le m \le M$ and $1 \le n \le N$. The five-point stencil $\mathcal{N}_{\text{5pt}}$ is illustrated in Fig. 2.4(a). To this end, let $\psi(m, n)$ denote the one-dimensional index used for the grid point (m, n) in a certain numbering scheme. We assume a natural ordering where the grid points are numbered starting from left to right and from bottom to top:

$$\psi(m, n) = m + (n - 1)M.$$

The numbering scheme for the five-point stencil $\mathcal{N}_{\text{5pt}}$, for example, is

$$\psi(1, 1) = 1,\ \psi(2, 1) = 2,\ \ldots,\ \psi(M, 1) = M,\ \psi(1, 2) = M + 1,\ \ldots,\ \psi(M, N) = MN.$$

This one-dimensional numbering is indicated in Fig. 2.4(b). This figure illustrates the nine-point stencil defined by

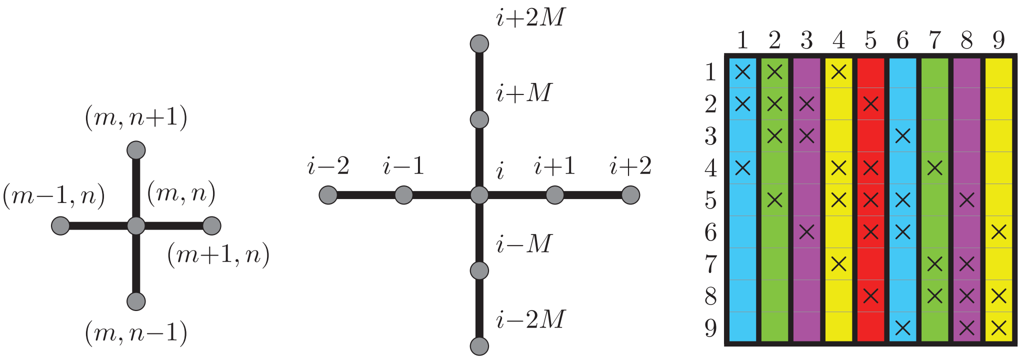
$$\mathcal{N}_{\text{9pt}}(m, n) = \mathcal{N}_{\text{5pt}}(m, n) \cup \{(m + 2, n), (m - 2, n), (m, n + 2), (m, n - 2)\}.$$

Given a regular $M \times N$ grid and a stencil $\mathcal{N}(m, n)$ describing the neighborhood relationship for all grid points $1 \le m \le M$ and $1 \le n \le N$, the sparsity pattern of the $MN \times MN$ Jacobian matrix A defined by the function f in (2.3) is determined and given as follows. A nonzero Jacobian element is characterized by

$$a_{i,j} \ne 0 \quad \Longleftrightarrow \quad i = \psi(m, n), j = \psi(k, l), \text{ and } (k, l) \in \mathcal{N}(m, n).$$

An example of a nonzero pattern is illustrated in Fig. 2.4(c) for the five-point stencil $\mathcal{N}_{\text{5pt}}$ on a 3×3 grid. The row 5 and column 5 are associated to the grid point $(2, 2)$

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.



	1	2	3	4	5	6	7	8	9
1	×	×		×					
2	×	×	×		×				
3		×	×			×			
4	×			×	×		×		
5		×		×	×	×		×	
6			×		×	×			×
7				×			×	×	
8					×		×	×	×
9						×		×	×

(a) (b) (c)

Figure 2.4: (a) Five-point stencil $\mathcal{N}_{5\text{pt}}$, (b) nine-point stencil $\mathcal{N}_{9\text{pt}}$ for a regular $M \times N$ grid. (c) Nonzero pattern of Jacobian matrix resulting from the five-point stencil $\mathcal{N}_{5\text{pt}}$ using a natural ordering of grid points on a 3×3 grid. Background padding indicates $p = 5$ groups of structurally orthogonal columns.

with $5 = \psi(2, 2)$. In this example, the only non-boundary grid point $(2, 2)$ of this small grid induces five nonzero elements in row $\psi(2, 2) = 5$ of the nonzero pattern. For every grid point $j = \psi(k, l)$, $(k, l) \in \mathcal{N}_{5\text{pt}}(2, 2)$, there is a nonzero element $a_{5,j}$ in row 5. Hence, the nonzero elements are $a_{5,2}$, $a_{5,4}$, $a_{5,5}$, $a_{5,6}$, and $a_{5,8}$. Further stencils are depicted in Appendix A.1.1.

Rather than considering Jacobian matrices, we now focus on the underlying regular $M \times N$ grid and some general stencil $\mathcal{N}(m, n)$. The combinatorial problem can then be reformulated in terms of the underlying grid. Since a grid point corresponds to a row/column of the Jacobian matrix, we get the following definition that characterizes the property needed to partition the grid points into groups.

Definition 2.14. Two grid points (i, j) and (k, l) are *structurally orthogonal* if and only if their stencils do not have a grid point in common, i.e.,

$$\begin{aligned} (i,j) \perp (k,l) \quad &:\Longleftrightarrow \quad \nexists (m,n): \ (i,j) \in \mathcal{N}(m,n) \ \wedge \ (k,l) \in \mathcal{N}(m,n) \\ &\Longleftrightarrow \quad \mathcal{N}(i,j) \cap \mathcal{N}(k,l) = \emptyset. \end{aligned}$$

To illustrate this definition, we resume the example of the five-point stencil $\mathcal{N}_{5\text{pt}}$. The centers of all stencils depicted in Fig. 2.5(a) are structurally orthogonal. A group of structurally orthogonal center grid points is called a *cover*. In general, there are grid points that are not structurally orthogonal so that multiple covers are needed to contain all grid points. Therefore, the corresponding combinatorial optimization problem is given as follows:

Problem 2.15. *Given a grid, partition its grid points into a minimal number of groups of structurally orthogonal grid points. More precisely, find a sequence of covers containing all grid points such that stencils within a cover do not overlap and the number of covers p, representing the number of groups, is minimized.*

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

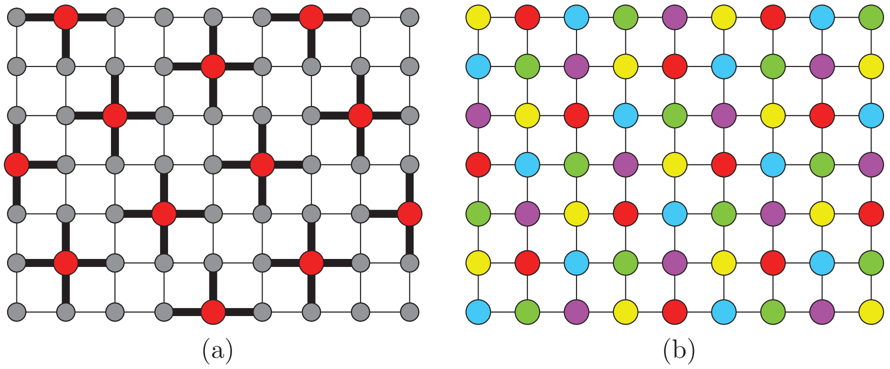


Figure 2.5: (a) Cover corresponding to a group of structurally orthogonal center points for the five-point stencil $\mathcal{N}_{5\text{pt}}$ on a 9×7 grid. (b) Sequence of covers obtained from using $p = 5$ covers of the form given in (a).

In Fig. 2.5(b), a solution is shown that was constructed by taking $p = 5$ covers of the form shown in Fig. 2.5(a) and shifting them so as to contain all grid points. The different covers are depicted in that figure using different background padding. Since the stencil involves five grid points, there is no solution with $p < 5$. Hence, the sequence of covers shown in Fig. 2.5(b) is indeed an optimal solution to the combinatorial optimization problem. For this five-point stencil, the open literature [29, 46, 48] gives the explicit formula to construct the sequence of covers given in Fig. 2.5(b).

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

3 Colorings for full and partial Jacobian computation

In Sect. 3.1, a sub-exponential exact coloring algorithm for stencil-based computations is introduced to determine minimal colorings for partial and full Jacobian computation. This algorithm takes advantage of the grid and stencil properties. For the full Jacobian computation, this algorithm is given in [43]. Since the runtime depends directly on the grid size, a grid size-independent approach with linear complexity is introduced in Sect. 3.2. The minimal colorings in the first two sections are distance-2 colorings in terms of the bipartite graph. After considering these one-sided colorings to exploit the sparsity pattern of Jacobian matrices in stencil-based computations, we move on to two-sided colorings for general graphs and introduce an algorithm for partial Jacobian computation in Sect. 3.3. In the last section, we consider several classes of Jacobian matrices to assess if there is a two-sided coloring with less colors than a minimal one-sided coloring.

3.1 Exact sub-exponential coloring algorithm for regular grids

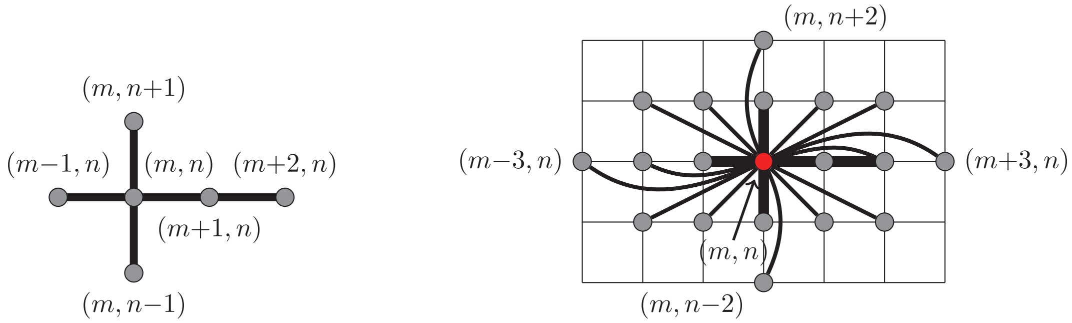
3.1.1 Full Jacobian computation

The optimization problem 2.15 given in terms of the grid can also be formulated as a coloring problem [13,14]. That is, all grid points belonging to a cover receive the same color. Unfortunately, the distance-1 coloring of the grid, i.e., all adjacent grid points are colored differently, is not sufficient to address the structural non-orthogonality of a center to the non-directly connected grid points. To demonstrate this relationship, we transform the optimization problem to a distance-1 coloring problem on a graph. The vertices represent the grid points and there is an edge between the grid points (m,n) and (i,j) if and only if (m,n) and (i,j) are structurally non-orthogonal. Due to Def. 2.14, between the grid point (m,n) and its structurally non-orthogonal grid points (i,j), the edges

$$\{((m,n),(i,j)) \mid \forall (i,j) : (m,n) \not\perp (i,j)\}$$

must be added. In order to clarify this relationship, we consider the center (m,n) of the six-point stencil $\mathcal{N}_{6\text{pt}}(m,n)$ in Fig. 3.1(b). The grid point (m,n) is not only center of $\mathcal{N}_{6\text{pt}}(m,n)$, but also part of the stencils $\mathcal{N}_{6\text{pt}}(m,n-1)$, $\mathcal{N}_{6\text{pt}}(m-2,n)$,

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.



(a) (b)

Figure 3.1: (a) Six-point stencil $\mathcal{N}_{6\mathrm{pt}}$. (b) Six-point stencil $\mathcal{N}_{6\mathrm{pt}}(m,n)$, highlighted with bold edges, with all structurally non-orthogonal grid points to center (m,n).

$\mathcal{N}_{6\mathrm{pt}}(m-1,n)$, $\mathcal{N}_{6\mathrm{pt}}(m+1,n)$, and $\mathcal{N}_{6\mathrm{pt}}(m,n+1)$. Therefore, there is an edge from (m,n) to each grid point $(i,j) \in \mathcal{N}_{6\mathrm{pt}}(m,n) \cup \mathcal{N}_{6\mathrm{pt}}(m,n-1) \cup \mathcal{N}_{6\mathrm{pt}}(m-2,n) \cup \mathcal{N}_{6\mathrm{pt}}(m-1,n) \cup \mathcal{N}_{6\mathrm{pt}}(m+1,n) \cup \mathcal{N}_{6\mathrm{pt}}(m,n+1)$. The original grid is planar, i.e., it can be drawn so that no edges cross each other. The introduced graph is usually not planar.

The distance-1 coloring problem for general graphs is NP-hard [26]. Using a coloring algorithm implementing an exhaustive search yields an optimal solution indeed, but is barely applicable for graphs occurring in real-world applications. Therefore, in practice, this problem is often tackled using linear-time greedy heuristics. The result can be determined in reasonable time, but the number of colors ranges between the optimal solution, the chromatic number χ, and the maximum degree of the graph plus one, $\Delta + 1$, depending on the ordering in which the vertices are colored [24, 62]. A comparison of different orderings together with a greedy heuristic for regular grids is given in [13, 14]: For various combinations of grid size and stencil, the colorings are never optimal, and the number of colors can be up to twice the chromatic number.

The planar separator theorem and some applications are introduced by Lipton and Tarjan [38, 39]. A planar separator, also called vertex separator, is a set of vertices whose removal splits a planar graph into two non-connected components, also denoted as subgraphs. The idea for using this theorem is to recursively separate the graph into subgraphs of sufficiently small size. For these subgraphs, the given problem can be exactly solved, e.g., using the exhaustive search, with reasonable computational effort. For a planar graph with n vertices, the theorem of Lipton and Tarjan states that the two separated subgraphs contain at most $2/3n$ vertices each. Furthermore, the separator consists of at most $2\sqrt{2}\sqrt{n}$ vertices and can be found in $\mathcal{O}(n)$ time. This separator is denoted as $(2\sqrt{2}\sqrt{n}, 2/3)$-separator. Vertex separators are known from various applications like nested dissection or independent set computations. Lipton and Tarjan [39] state that this theorem can be used to solve the distance-1 graph coloring problem on planar graphs.

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

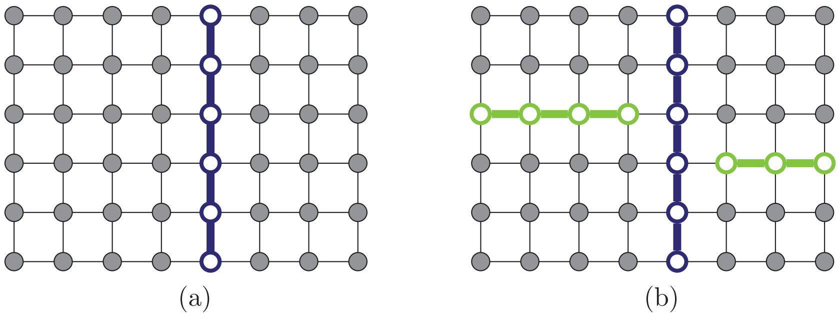


Figure 3.2: (a) Original 8 × 6 grid is divided into two subgrids using the separator $\{(5,1),(5,2),(5,3),(5,4),(5,5),(5,6)\}$. (b) The subgrids from (a) are divided once more into four smallest subgrids, each containing at most $n_0 = 12$ grid points, using the separators $\{(6,3),(7,3),(8,3)\}$ and $\{(1,4),(2,4),(3,4),(4,4)\}$.

The planar separator theorem was originally introduced solely for planar graphs, including regular grids in two dimensions. Later, the theorem was extended to certain classes of non-planar graphs [1, 37] and non-planar regular grids G in one or more dimensions [25]. The upper bound for the size of a separator in d-dimensional, regular grids G with n grid points and $d \geq 2$ is given in [47] by

$$O(n^{(d-1)/d}),$$

i.e., $\sqrt{n}$ for $d = 2$ and $n^{2/3}$ for $d = 3$. Thus, a separator can at most consist of the grid points of a complete row or column for two-dimensional regular grids. In three dimensions, the vertices of a separator can at most form a plane in one of the three directions. To divide the grid in two rather equal-sized subgrids we use a separator whose number of grid points is the given upper bound. In two dimensions, a separator that consists of all grid points of a column is denoted as *column separator*, and a separator that consists of all grid points of a row is denoted as *row separator*.

An example for separators in a two-dimensional grid is given in Fig. 3.2. The set of grid points $\{(5,1),(5,2),(5,3),(5,4),(5,5),(5,6)\}$ forms the separator on the first level in Fig. 3.2(a). This column separator contains six grid points and is smaller than any row separator which would contain eight grid points. This column separator divides the original grid into the subgrids $G[A]$ and $G[B]$ which are induced by the grid point sets A and B. The subgrid $G[A]$, a rectangular area, is spanned by the grid point $(1,1)$ in the south west corner and the grid point $(4,6)$ in the north east corner, and the subgrid $G[B]$ is spanned by the grid points $(6,1)$ and $(8,6)$. As a result of the planar separator theorem, a hierarchy of separators can be built by recursively applying this theorem to both subgrids $G[A]$ and $G[B]$ arising on the previous level. On the second level in our example in Fig. 3.2(b), the subgrids $G[A]$ and $G[B]$ are split using the separators $S = \{(1,4),(2,4),(3,4),(4,4)\}$ and $S = \{(6,3),(7,3),(8,3)\}$, respectively. If a subgrid contains less than a previously specified number of vertices, n_0, we stop applying the separator scheme.

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

Algorithm 3.1: Compute a minimal coloring of regular grid G with n grid points and stencil $\mathcal{N}$ using separators. The set $C_{S_{\text{all}}}$ contains colorings of already colored grid points in separators on higher levels and is initialized as the empty set. The function colorExhS is given in Alg. 3.3.

```
1  function colorVSep($G$, $\mathcal{N}$, $C_{S_{\text{all}}}$)
2      if $n < n_0$ then
3          return colorExhS($G$, $\mathcal{N}$, 'single') not violating coloring $C_{S_{all}}$   ▷ First coloring
4      else
5          Find an $(n^{(d-1)/d} \cdot k, 1/2)$-separator $S$ so that $G = G[A \uplus S \uplus B]$
6          foreach $c_S \in$ colorExhS($G[S]$, $\mathcal{N}$, 'all') not violating coloring $C_{S_{all}}$ do
7              $c_A \leftarrow$ colorVSep($G[A]$, $\mathcal{N}$, $C_{S_{\text{all}}} \cup c_S$)
8              $c_B \leftarrow$ colorVSep($G[B]$, $\mathcal{N}$, $C_{S_{\text{all}}} \cup c_S$)
9              if $c_A \neq \emptyset$ and $c_B \neq \emptyset$ then
10                 return $(c_A, c_S, c_B)$                                     ▷ First coloring
```

Exact coloring algorithm using vertex separators

To solve the optimization problem 2.15 on regular grids with n grid points, a sub-exponential coloring algorithm using a divide-and-conquer method is introduced. Sub-exponential time complexity is used in the sense that our algorithm grows slower than every function $f(n) = c^n$, $\forall c = \text{const}$, i.e., our algorithm is in $c^{o(n)}$. Although this algorithm is mainly described for two dimensions, it can also be applied to grids in higher dimensions. Nevertheless, the time complexity is given for d-dimensional grids. This algorithm is based on the planar separator theorem and given as function colorVSep in Alg. 3.1. The input parameters are a grid G with n grid points, a stencil $\mathcal{N}$, and the set $C_{S_{\text{all}}}$ which stores the colorings for the already colored grid points in the separators on the higher levels. Thus, the latter parameter is initialized with $C_{S_{\text{all}}} = \emptyset$. The result is a coloring which represents a partition of the grid points. At first, the function determines an $(n^{(d-1)/d} \cdot k, 1/2)$-separator in line 5 using the neighborhood relation given by the stencil $\mathcal{N}$. The separator contains $n^{(d-1)/d} \cdot k$ grid points, where $n^{(d-1)/d}$ is the number of grid points to cut the grid and k is the maximum of the height and width between the center and its farthest structurally non-orthogonal grid point. The width between the center (m, n) and the grid point (i, j) is $|m - i|$ and the height between (m, n) and (i, j) is $|n - j|$. Hence, the largest k of a center (m, n) to another structurally non-orthogonal grid point (i, j) is

$$k = \max_{\substack{(i,j) \\ (i,j) \not\perp (m,n)}} \{|m - i|, |n - j|\}. \tag{3.1}$$

An illustrating example is the six-point stencil $\mathcal{N}_{\text{6pt}}(m, n)$ in Fig. 3.1(b). The maximal width between two structurally non-orthogonal grid points, e.g., grid points (m, n) and $(m+3, n)$, is three, and maximal height between two structurally non-orthogonal grid points, e.g., (m, n) and $(m, n-2)$, is two. So, the maximum of height and width is $k = 3$.

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

Lemma 3.1. *Given a regular, d-dimensional grid G with n grid points, a stencil whose largest distance of its center to a structurally non-orthogonal grid point is k, and an $(n^{(d-1)/d} \cdot k, 1/2)$-separator which splits the grid G into the subgrids $G[A]$ and $G[B]$, a grid point in one of the subgrids is structurally orthogonal to all grid points in the other subgrid.*

Proof. We assume that a grid point (m, n) in one of the subgrids is structurally non-orthogonal to a grid point in the other subgrid. The distance between the center (m, n) and all structurally non-orthogonal grid points is at most k. Due to the choice of our separator, its width is k. That is, in two dimensions, the separator consists of k rows or columns of the grid. Hence, the distance between two grid points in the different subgrids is at least $k + 1$. This is a contradiction. □

Due to this lemma, on each level, the colorings for both subgrids can be independently computed in the function colorVSep. First, the separator is colored. Then, for every minimal coloring of the separator (line 6), we recursively call this function for the subgrids $G[A]$ and $G[B]$. The recursive calls also include the employed stencil $\mathcal{N}$ and the colorings for the already colored grid points in the separators on the higher levels, $C_{S_{\text{all}}}$. These already colored grid points, which are stored in $C_{S_{\text{all}}}$, must be considered on the lower levels of the hierarchy to be sure to obtain valid colorings. If the number of grid points in a subgrid falls below the threshold n_0, a minimal coloring is determined using the exhaustive search. If the minimal colorings for $G[A]$ and $G[B]$ are not empty (line 9), both colorings form a valid coloring for $G[A \uplus S \uplus B]$ together with the coloring for the separator S. This combined coloring is valid, because the colorings of the already colored grid points in the separators were taken into account. If there is no valid coloring, the function proceeds to the next minimal coloring for the separator. Furthermore, to ensure that a minimal coloring is found, the algorithm tries to determine a minimal coloring with $p = 1$ color. If there is no such coloring, the algorithm looks for a minimal coloring with $p = 2$ colors. This step is repeated as long as a minimal coloring with p colors is determined. Finally, the result for the original grid G, and also every subgrid, is a minimal coloring.

There are at most $p^{n^{(d-1)/d} \cdot k}$ minimal colorings for the $(n^{(d-1)/d} \cdot k, 1/2)$-separator where p states the minimal number of colors for grid G. For every minimal coloring, there are recursive calls for the subgrids $G[A]$ and $G[B]$. Both subgrids contain at most $n/2$ grid points; and the separator can be determined with an explicit formula in $\mathcal{O}(1)$. The original grid is much larger than the smallest subgrids. Therefore, the runtime for the exhaustive search is negligible; thus, the complexity is $\mathcal{O}(1)$. Hence, the theoretical complexity of Alg. 3.1 is governed by the recursion:

$$t(n) = \begin{cases} \mathcal{O}(1) & n < n_0 \\ \mathcal{O}(1) + 2 \cdot p^{n^{(d-1)/d} \cdot k} \cdot t(n/2) & n \geq n_0 \end{cases}$$

As argued in [38] by using an inductive proof, the overall complexity is $\mathcal{O}(p^{n^{(d-1)/d} \cdot k})$.

Our implementation of the coloring algorithm, depicted in Alg. 3.2, is slightly different from Alg. 3.1. To better exploit the structure of the hierarchy and to be able

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

Algorithm 3.2: Compute all minimal colorings of regular grid G with n grid points and stencil $\mathcal{N}$ using separators.

```
1  function colorVSepAll($G$,$\mathcal{N}$)
2      if $n < n_0$ then
3          return colorExhS($G$,$\mathcal{N}$, 'all')                      ▷ All colorings
4      else
5          $C \leftarrow \emptyset$
6          Find an $(n^{(d-1)/d}, 1/2)$-separator $S$ so that $G = G[A \uplus S \uplus B]$
7          $C_A \leftarrow$ colorVSepAll($G[A]$,$\mathcal{N}$)
8          $C_B \leftarrow$ colorVSepAll($G[B]$,$\mathcal{N}$)
9          $C_S \leftarrow$ colorExhS($G[S]$,$\mathcal{N}$, 'all')
10         foreach $c_A \in C_A, c_S \in C_S, c_B \in C_B$ do
11             if $(c_A, c_S, c_B)$ is valid coloring then
12                 $C \leftarrow C \cup (c_A, c_S, c_B)$
13         return $C$                                                    ▷ All colorings
```

to improve the runtime, we proceed as follows: The separator is only of size $n^{(d-1)/d}$. That is, there are grid points in one subgrid which are structurally non-orthogonal to grid points in the other subgrid. This separator is a factor k smaller than the separator used in Alg. 3.1. Thus, it is less expensive to compute colorings for this separator with the exhaustive search. Instead of coloring the grid points in the separator first, we start determining all minimal colorings for the smallest subgrids without considering structurally non-orthogonal grid points in other subgrids. Hence, we must check for all minimal colorings $c_A \in C_A$ and $c_B \in C_B$ of $G[A]$ and $G[B]$, respectively, as well as the minimal colorings $c_S \in C_S$ of $G[S]$ whether the colorings c_A, c_B, and c_S form a valid coloring for $G[A \uplus S \uplus B]$ (lines 10 and 11). Each valid coloring is added to set C. At the end, all encountered minimal colorings C for the original grid G are returned.

As already mentioned, the coloring algorithm is not only applicable for two-dimensional regular Cartesian grids, but also for regular Cartesian grids in higher dimensions. Further grids are given in the Bravais' lattice classification, in particular, the hexagonal grid in two dimensions as well as the face-centered cubic and body-centered cubic in three dimensions. Our coloring algorithm could probably be modified to compute hexagonal tilings [32] for these grids.

Implementation

After presenting the scheme of our divide-and-conquer coloring algorithm using separators and its theoretical complexity, in the following, we describe the implementation of colorVSepAll in more detail and explain further improvements to reduce the runtime for practical reasons. First, we discuss the three phases of the algorithm—divide, color smallest subgrids, and conquer. In the second part, we explain techniques which are implemented in the algorithm to reduce the runtime and the memory consumption.

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

Algorithm 3.3: Compute a minimal coloring Φ with p colors of a regular grid G with n grid points and stencil $\mathcal{N}$ in a given ordering $\psi_1, \psi_2, \ldots, \psi_n$ using the exhaustive search. Using mode='single', the first minimal coloring is returned, otherwise all minimal colorings. Initialize $\psi_i = \psi_1$ and coloring $\Phi = [0 \ldots 0]$ if not given.

```
function colorExhS($G, \mathcal{N}, p, \text{mode}, \psi_i, \Phi$)
    foreach $c \in \{1, \ldots, p\} \setminus \{\Phi(\psi_j) : \psi_j \in \mathcal{N}(\psi_i)\}$ do
        $C \leftarrow \emptyset$
        if $i < n$ then
            $\Phi(\psi_i) \leftarrow c$
            $C \leftarrow C \cup$ colorExhS($G, \mathcal{N}, p, \text{mode}, \psi_{i+1}, \Phi$)
            if mode = 'single' and $C \neq \emptyset$ then return $C$                ▷ First coloring
        else
            $C \leftarrow C \cup \Phi$
            if mode = 'single' then return $C$                                      ▷ First coloring
    $\Phi(\psi_i) \leftarrow 0$
    return $C$
```

Divide: Starting with the original grid, the algorithm builds up a grid hierarchy by recursively splitting every (sub-)grid G into the subgrids $G[A]$ and $G[B]$ using a separator S. As described in the previous section, this separator can be easily determined due to the regular grid structure. Therefore, the number of grid points in the horizontal (rows) and the vertical (columns) dimension are compared. Then either the grid points in a row or a column are chosen as separator depending on which one contains fewer grid points. Afterwards, the row or column separator in the middle of the grid is selected to get two rather equal-sized subgrids $G[A]$ and $G[B]$. If the number of grid points in a subgrid falls below the threshold n_0, we stop splitting that subgrid and continue with coloring the subgrids on the lowest level of the hierarchy.

Color subgrids on the lowest level: The exhaustive search for coloring the grid points of the subgrids on the lowest level is implemented by a recursive depth-first search illustrated in Alg. 3.3. At the beginning, we execute the coloring function colorExhS restricted to p colors where p is the lower bound given by the number of grid points belonging to the stencil. If no valid coloring is found, this function is started once more with an additional color, i.e., $p = p + 1$. This step is repeated until a valid coloring is determined. Hence, we have a new number of colors p.

The function colorExhS—not using the mode 'single'—colors the grid points of grid G using stencil $\mathcal{N}$ in a given ordering ψ starting at the first grid point ψ_1. The coloring is initialized with zeros. It assigns the first color in the range $\{1, \ldots, p\}$ to the grid point ψ_1 and recursively proceeds to the next grid point ψ_2. This grid point ψ_2 is colored with a color c which is not already assigned to any structurally non-orthogonal grid point. This step is repeated for the grid points in the ordering ψ. The process

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

stops if either there is no valid color c in $\{1, \dots, p\}$ left or all grid points are colored: In the first case, the function moves back to the previous grid point ψ_{i-1}. If there is another valid color c, the function proceeds once again to grid point ψ_i. Otherwise, the function goes back in the ordering and uncolors all grid points as long as there is a grid point $\psi_j, j < i$, for which another valid color c can be chosen. The function chooses this color and repeats the coloring step for the next grid point ψ_{j+1} in the ordering and so on. In the second case, where a valid coloring is found, the function stores this minimal coloring in C and goes backwards to the first grid point ψ_j which can be colored with another c and continues with coloring the next grid point ψ_{j+1}. Thus, all minimal colorings are enumerated and a minimal coloring is guaranteed to be found.

Conquer: The algorithm combines the minimal colorings $c_A \in C_A$ and $c_A \in C_A$ of subgrids $G[A]$ and $G[B]$, respectively, together with minimal colorings for the separator S to determine the minimal colorings for $G[A \uplus S \uplus B]$ on every level of the hierarchy. The function colorVSepAll starts at the smallest subgrids and yields all minimal colorings for the original grid G.

Valid colorings of $G[A]$ and $G[B]$ may exclude each other if structurally non-orthogonal grid points in $G[A]$ and $G[B]$ have the same color. The colorings for $G[A]$, $G[B]$, and $G[S]$ have already been determined before the check for validity occurs in lines 10–11 in Alg. 3.2. We improve the implementation to decrease the computational effort: Therefore, before determining colorings for the separator, the function checks if the colorings c_A and c_B can be combined—without considering the grid points in the separator. If two colorings do not exclude each other, there is a chance to find a valid coloring c_S for the grid points of the separator to obtain a coloring of the grid $G[A \uplus S \uplus B]$. To get those colorings, the exhaustive search is used to compute all minimal colorings for the grid points in the separator. All grid points in $G[A]$ and $G[B]$ which are structurally non-orthogonal to grid points in the separator are taken into account. These grid points are located near the separator depending on the underlying stencil $\mathcal{N}$. On the highest level, solely one minimal coloring for the original grid G is required. Therefore, the function terminates after determining the first minimal coloring while combining the colorings of $G[A]$, $G[B]$, and $G[S]$.

The check for all pairs of colorings of $G[A]$ and $G[B]$ and the coloring of the separator S can proceed independently. Since we are concerned with shared-memory architectures, we employ OpenMP to distribute the corresponding computational work among multiple threads. Recall the function colorVSepAll depicted in Alg. 3.2 where the validity check of the colorings is schematically given in lines 10–11. The computational effort for the tasks differs: On the one hand, the check can be immediately stopped after detecting two structurally non-orthogonal grid points with the same color. On the other hand, all structurally non-orthogonal grid points of the subgrids $G[A]$, $G[B]$ and $G[S]$ located in different subgrids are checked if their colorings form a minimal coloring for $G[A \uplus S \uplus B]$. Therefore, we choose the dynamic scheduling strategy of OpenMP. After all OpenMP threads have finished this foreach loop, the algorithm moves on to the next higher level of the hierarchy.

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

Next, we explain the techniques which result in major improvements, decreasing the computational effort and the memory usage:

Coloring table In the separator hierarchy, two or more subgrids with the same height and width can occur. The minimal colorings for these subgrids are identical. Rather than computing these colorings several times, the algorithm stores them in the coloring table and repeatedly reuses them. Therefore, before splitting a subgrid using a separator or coloring a subgrid on the lowest level, the algorithm checks whether the minimal colorings for this subgrid size are already stored in the coloring table. In this case, these colorings are reused. Otherwise, we descend to the next level in the hierarchy by separating the current subgrid once more or color the subgrid on the lowest level by the exhaustive search. If the colorings for a subgrid are not required anymore, those colorings are removed from the coloring table to free memory.

Precoloring Recall from Sect. 2.4 that all ℓ grid points belonging to a stencil are pairwise structurally non-orthogonal and, hence, must be colored differently. For a stencil consisting of ℓ grid points including the center, there exist $\ell!$ different colorings. That is, there are ℓ possibilities to color the first grid point, $\ell-1$ possibilities for the second grid point and so on. All these $\ell!$ colorings are minimal. The precoloring technique works as follows: We color a chosen stencil with ℓ colors and determine all minimal colorings for the remaining uncolored grid points. To compute this subset of the minimal colorings, the exhaustive search in the coloring phase computes all remaining minimal colorings taking the already colored ℓ grid points into account. All other minimal colorings of the grid can be obtained by permuting the ℓ colors assigned to the grid points of the chosen stencil. In Fig. 2.5(b), the stencil $\mathcal{N}_{5\text{pt}}(2,2) = \{(2,1),(1,2),(2,2),(3,2),(2,3)\}$ is an illustrating example for the precoloring technique. These grid points are colored and, afterwards, the exhaustive search colors the remaining uncolored grid points.

The number of minimal colorings for G is $\ell!$ multiplied with the number of minimal colorings of the remaining grid points. The number of minimal colorings of the remaining grid points is independent of how the stencil is colored. When we employ the precoloring technique, instead of determining all colorings by the exhaustive search algorithm and storing them, the number of minimal colorings to compute is reduced by $\ell!$. If the algorithm requires access to all colorings, these colorings are computed by permuting the stored one. We consider the stencil $\mathcal{N}_{5\text{pt}}$ in Fig. 2.5(b) as an illustrating example. This example contains a precolored stencil and one minimal coloring for the remaining grid points. It can be shown that there are two minimal colorings for the remaining grid points. Hence, the number of determined colorings is reduced from $2 \cdot 5!$ to 2. For the stencil $\mathcal{N}_{9\text{pt}}$, we determine four colorings instead of $4 \cdot 9!$ colorings. During the conquer phase, every minimal coloring is recovered by permuting the computed colorings. It is enough to check all colorings of $G[A]$ (without permutations) against all colorings of $G[B]$ including all permutations on every level of the hierarchy. This principle is an extension of the precoloring technique to all subgrids which are not on the lowest level of the hierarchy. Using this technique, we reduce the computational effort in the coloring phase and the overall memory consumption as well.

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

Stencil	Algorithm	t	N								
			9	19	39	79	159	2,559	5,119	10,239	20,479
$\mathcal{N}_{5\text{pt}}$	colorExhS		ϵ	165	$\gg$	$\gg$	$\gg$	$\gg$	$\gg$	$\gg$	$\gg$
	colorVSepAll	1	ϵ	ϵ	ϵ	ϵ	ϵ	ϵ	2	6	23
		2	ϵ	ϵ	ϵ	ϵ	ϵ	ϵ	2	6	16
		4	ϵ	ϵ	ϵ	ϵ	ϵ	ϵ	1	7	24
		8	ϵ	ϵ	ϵ	ϵ	ϵ	ϵ	1	4	47
		16	ϵ	ϵ	ϵ	ϵ	ϵ	ϵ	2	8	55
$\mathcal{N}_{6\text{pt}}$	colorExhS		ϵ	638	$\gg$	$\gg$	$\gg$	$\gg$	$\gg$	$\gg$	$\gg$
	colorVSepAll	1	ϵ	ϵ	ϵ	ϵ	1	18	38	81	167
		2	ϵ	ϵ	ϵ	ϵ	ϵ	12	25	53	124
		4	ϵ	ϵ	ϵ	ϵ	ϵ	6	15	29	63
		8	ϵ	ϵ	ϵ	ϵ	ϵ	4	9	19	52
		16	ϵ	ϵ	ϵ	ϵ	ϵ	2	8	17	56
$\mathcal{N}_{9\text{pt}}$	colorExhS		1	$\gg$	$\gg$	$\gg$	$\gg$	$\gg$	$\gg$	$\gg$	$\gg$
	colorVSepAll	1	1	215	756	1,543	2,810	32,036	62,961	$\gg$	$\gg$
		2	1	144	492	960	1,698	17,976	35,086	71,733	$\gg$
		4	1	74	247	489	847	9,061	17,807	36,017	$\gg$
		8	1	39	126	245	430	4,620	8,953	18,677	83,780
		16	1	24	67	125	217	2,255	4,406	19,659	68,358

Table 3.1: Runtime in seconds to compute one minimal coloring using the exhaustive search and the algorithm colorVSepAll for different $N \times N$ grids using t threads. The smallest subgrids are of size 9×9 ($n_0 = 81$). Runtime below 1s is denoted by ϵ, exceeding 24h (86,400s) by $\gg$.

Results

After describing the sub-exponential algorithm colorVSepAll, we evaluate this coloring algorithm. Therefore, we compare colorVSepAll with the exhaustive search coloring algorithm colorExhS and the standard CPR (coloring) heuristic [22]. The coloring heuristic D2ColoringRestricted in Alg. 2.1 can be applied to the bipartite graph associated with Jacobian matrix resulting from the stencil-based computation. At first, we compare the runtime of both algorithms, colorVSepAll and colorExhS, determining minimal colorings and explain the major impact factors which influence the runtime. Thereafter, we look at the algorithms colorVSepAll and D2ColoringRestricted using several (grid point) orderings to address the runtime and the number of additional colors required by D2ColoringRestricted. For the evaluations, we choose three different two-dimensional stencils: the five-point stencil $\mathcal{N}_{5\text{pt}}$ from Fig. 2.4(a), the six-point stencil $\mathcal{N}_{6\text{pt}}$ from Fig. 3.1(a), and the nine-point stencil $\mathcal{N}_{9\text{pt}}$ from Fig. 2.4(b).

The algorithms are implemented in the programming language C++ and use the Boost Graph Library. The OpenMP model is used for the shared-memory parallelization. All computations are carried out on an AMD Barcelona cluster node with 4 processors, 16 cores, and 32GB main memory provided by the Center for Computing and Communication of the RWTH Aachen University. Thus, it does not make sense to employ more than 16 threads for colorVSepAll.

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

In Table 3.1, we compare the algorithms colorExhS and colorVSepAll for two-dimensional $N \times N$ grids using the stencils mentioned above. Recall that the algorithm colorVSepAll employs t threads. In contrast, the algorithm colorExhS is a serial implementation with $t = 1$. The grid size is varied from $N = 9$ to $N = 20{,}479$ by doubling the number of grid points in the horizontal and the vertical direction, i.e., quadrupling the number of grid points, and adding an additional row and column as separator. The number of grid points varies from 81 to 420 million. In our setting, all smallest subgrids considered in the coloring phase of colorVSepAll are 9×9 grids. As an illustrating example we consider a grid with $N = 39$. The colorings for this grid are combined from the colorings of two 19×39 grids and the separator in between. The colorings for the 19×39 grid are combined from the colorings of two 19×19 grids and the separator, and so on. Instead of computing these colorings several times, the algorithm determines the colorings for all subgrids of the same size only once; otherwise the colorings for the 19×39 grid would be computed twice, for the 19×19 grid four times, for the 9×19 grid eight times, and for the 9×9 grid sixteen times. For all three stencils, colorVSepAll has always a lower runtime than colorExhS. For stencil $\mathcal{N}_{5\text{pt}}$, colorExhS needs 165s to compute a coloring for a 19×19 grid and more than 24h for a 39×39 grid. The algorithm colorVSepAll colors a $20{,}479 \times 20{,}479$ grid in 23s using only 1 thread. In comparison, colorExhS takes more time to color a 19×19 grid. This trend can be recognized for the stencils $\mathcal{N}_{6\text{pt}}$ and $\mathcal{N}_{9\text{pt}}$ as well. For the latter stencil the runtimes for colorExhS and colorVSepAll are both much larger than for the stencils $\mathcal{N}_{5\text{pt}}$ and $\mathcal{N}_{6\text{pt}}$. The phenomenon is discussed two paragraphs later.

As described in the previous section, the algorithm colorVSepAll is designed for parallel computing using OpenMP. All computations of this algorithm are also performed with 2, 4, 8, and 16 threads, as shown in Table 3.1. Due to the short runtime of colorVSepAll for $\mathcal{N}_{5\text{pt}}$, even for the largest considered grid, we do not benefit from using more than two threads. The number of colorings and the neighborship of the centers are too small for dividing the conquer phase to several threads. The runtime grows by increasing the number of OpenMP threads due to the synchronization and scheduling overhead. For stencil $\mathcal{N}_{6\text{pt}}$, the runtime is decreased from 167s to 56s using 16 threads for the $20{,}479 \times 20{,}479$ grid, i.e., a speedup of 167s/56s=2.98 is observed. The runtime for the stencil $\mathcal{N}_{9\text{pt}}$ is quite different. Using only 1 thread, it is not possible to compute a minimal coloring for $N \times N$ grids with $N = 10{,}239$ and $N = 20{,}479$ in 24h. For determining a minimal coloring with colorVSepAll using 16 threads, the runtime can be reduced to 19,659s and 68,358s, respectively. For stencil $\mathcal{N}_{9\text{pt}}$ and the $5{,}119 \times 5{,}119$ grid, the largest considered grid with a runtime smaller than 24h using 1 thread, we obtain a speedup of 62,961s/4,406s=14.29 using 16 threads.

Why is there such a big difference between the runtime of stencil $\mathcal{N}_{9\text{pt}}$ and the other stencils? There are two factors with a strong impact on the runtime of the algorithm colorVSepAll: First, the number of grid points in the neighborship of a center. This number has a direct influence on how many times a coloring must be permuted during the conquer phase. There are 5! permutations for stencil $\mathcal{N}_{5\text{pt}}$

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

Stencil	Algorithm	*N*								
		9	20	40	80	160	2,560	5,120	10,240	20,480
$\mathcal{N}_{5\text{pt}}$	colorExhS	ϵ	527	$\gg$	$\gg$	$\gg$	$\gg$	$\gg$	$\gg$	$\gg$
	colorVSepAll	ϵ	ϵ	ϵ	ϵ	ϵ	3	8	27	71
$\mathcal{N}_{6\text{pt}}$	colorExhS	ϵ	2570	$\gg$	$\gg$	$\gg$	$\gg$	$\gg$	$\gg$	$\gg$
	colorVSepAll	ϵ	ϵ	1	3	6	86	170	357	742
$\mathcal{N}_{9\text{pt}}$	colorExhS	1	$\gg$	$\gg$	$\gg$	$\gg$	$\gg$	$\gg$	$\gg$	$\gg$
	colorVSepAll	1	538	3,338	8,274	14,736	$\gg$	$\gg$	$\gg$	$\gg$

Table 3.2: Runtime in seconds to compute one minimal coloring using the exhaustive search and the algorithm colorVSepAll for different $N \times N$ grids using 1 thread. The smallest subgrids are of size 9×9 ($n_0 = 81$). Runtime below 1s is denoted by ϵ, exceeding 24h (86,400s) by $\gg$.

and 9! for stencil $\mathcal{N}_{9\text{pt}}$; the difference is a factor of 3,024. Furthermore, the size of the neighborship determines how many different structurally non-orthogonal grid points must be considered while coloring a center. The number of structurally non-orthogonal grid points for the centers of our considered stencils are:

$$\begin{array}{ccc} \mathcal{N}_{5\text{pt}} & \mathcal{N}_{6\text{pt}} & \mathcal{N}_{9\text{pt}} \\ \hline 11 & 19 & 31 \end{array}.$$

Second, the structure of the stencil has an impact on the runtime. The number of different colorings (without permutations) depends directly on this structure. There are two colorings for stencil $\mathcal{N}_{5\text{pt}}$ and four colorings for stencil $\mathcal{N}_{9\text{pt}}$, i.e., the number of combinations is at least doubled.

In Table 3.1, the grid size is chosen such that the structure is exploited by reusing each computed minimal coloring at least two times, usually more than two times. In Table 3.2, the grid size is varied by adding only one additional column and row, i.e., for all instances, the $N \times N$ grid is extended to an $(N + 1) \times (N + 1)$ grid. This small variation leads to a grid which cannot be divided in two equal-sized subgrids, but only in mostly equal-sized subgrids. The runtime of colorVSepAll to compute a minimal coloring for those grids with slightly different sizes is significant higher. An explanation is the number of subgrids which must be colored. Recall from Table 3.1 that the runtime is 23s for $\mathcal{N}_{5\text{pt}}$ and 167s for $\mathcal{N}_{6\text{pt}}$ on the $N \times N$ grid for $N = 20{,}479$. The corresponding runtimes increase to 71s and 742s, respectively, on the $(N+1)\times(N+1)$ grid. For those grids and also for the smaller grids the difference in runtime is roughly a factor of 4 to 5 comparing the grids with size N and $N + 1$. However, the runtime for the largest grid is still smaller than determining a minimal coloring for the 19×19 or 20×20 grid using colorExhS. For stencil $\mathcal{N}_{9\text{pt}}$, the factor is slightly higher due to the bigger neighborship of each center. Computing a minimal coloring for the $2{,}560 \times 2{,}560$ grid takes longer than 24h. Here, we compute the colorings only with 1 thread. However, by using parallelization, we expect comparable reductions of runtime as in Table 3.1.

After demonstrating that colorVSepAll is faster than colorExhS for all instances, we want to compare colorVSepAll with D2ColoringRestricted us-

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

Stencil	χ	Ordering		N							
				19	39	79	159	2,559	5,119	10,239	20,479
$\mathcal{N}_{5\text{pt}}$	5	NO	p	7	7	7	7	7	7	7	7
			s	ϵ	ϵ	ϵ	ϵ	6	25	101	411
		LFO	p	7	7	7	7	7	7	7	7
			s	ϵ	ϵ	ϵ	ϵ	15	64	334	1,204
		IDO	p	6	6	6	6	?	?	?	?
			s	ϵ	ϵ	ϵ	2	$\gg$	$\gg$	$\gg$	$\gg$
$\mathcal{N}_{6\text{pt}}$	7	NO	p	9	10	10	10	10	10	10	10
			s	ϵ	ϵ	ϵ	ϵ	9	37	148	603
		LFO	p	10	10	10	10	10	10	10	10
			s	ϵ	ϵ	ϵ	ϵ	22	87	349	1,616
		IDO	p	10	10	10	10	?	?	?	?
			s	ϵ	ϵ	ϵ	3	$\gg$	$\gg$	$\gg$	$\gg$
$\mathcal{N}_{9\text{pt}}$	10	NO	p	15	16	17	17	17	17	17	17
			s	ϵ	ϵ	ϵ	ϵ	15	62	244	1,007
		LFO	p	16	16	17	17	17	17	17	17
			s	ϵ	ϵ	ϵ	ϵ	33	135	539	2,400
		IDO	p	14	14	14	14	?	?	?	?
			s	ϵ	ϵ	ϵ	3	$\gg$	$\gg$	$\gg$	$\gg$

Table 3.3: Runtime in seconds, s, and number of colors, p, for stencils $\mathcal{N}_{5\text{pt}}, \mathcal{N}_{6\text{pt}}, \mathcal{N}_{9\text{pt}}$ and $N \times N$ grids using coloring heuristic (D2ColoringRestricted) with different preorderings. For the exact coloring algorithms, p is the chromatic number χ. Runtime below 1s is denoted by ϵ, exceeding 24h (86,400s) by $\gg$.

ing different orderings in terms of number of colors and runtime. Remember that both exact algorithms yield a minimal coloring whose number of colors is the chromatic number χ. For all three stencils, this number and the results for the heuristic D2ColoringRestricted are given in Table 3.3. For every stencil, the first row contains the number of colors computed by D2ColoringRestricted using the natural ordering (NO) and the second row the corresponding runtime. The following rows are obtained by D2ColoringRestricted with two different preordering algorithms: largest-first ordering (LFO) [62] and incident-degree ordering (IDO) [10]. A more extensive discussion about using D2ColoringRestricted for grids is given in [13, 14]. Furthermore, D2ColoringRestricted is evaluated using several preorderings. First, the colorings determined by D2ColoringRestricted—independent of the chosen preordering—are never minimal in our illustrating examples. For stencil $\mathcal{N}_{5\text{pt}}$, at least 20% more colors are required ($\chi = 5$ compared to $p = 6$), for $\mathcal{N}_{6\text{pt}}$ at least 28.6%, and for $\mathcal{N}_{9\text{pt}}$ at least 40%. In the worst case, 70% more colors are needed for $\mathcal{N}_{9\text{pt}}$ ($\chi = 10$ compared to $p = 17$). Second, for stencils $\mathcal{N}_{5\text{pt}}$ and $\mathcal{N}_{6\text{pt}}$, colorVSepAll is almost always faster than the heuristic D2ColoringRestricted—independent of the used ordering. For the stencil $\mathcal{N}_{9\text{pt}}$, the algorithm D2ColoringRestricted is much faster using the natural ordering or the LFO preordering. Using the IDO preordering, colorVSepAll is only faster for larger grid sizes. A coloring for the 2,559 × 2,559 grid and larger grids

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

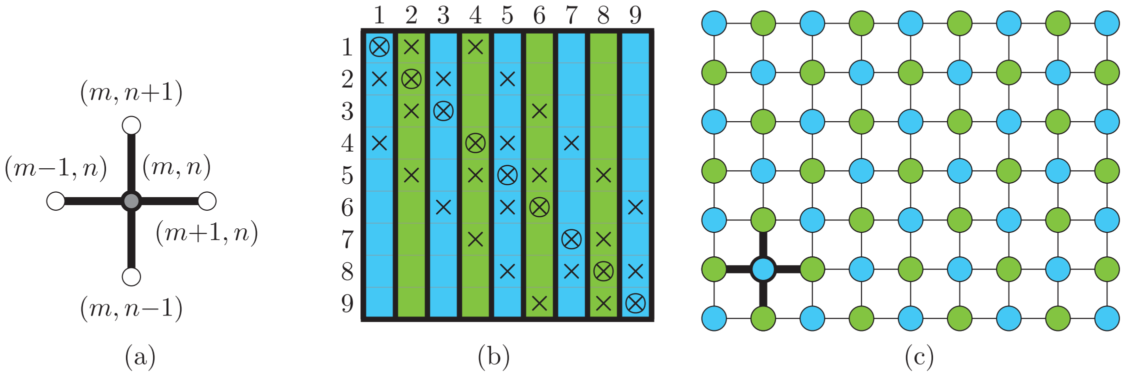


Figure 3.3: (a) Stencil $\mathcal{N}_{5,1\text{pt}}(m,n)$ is a combination of $\mathcal{N}_{5\text{pt}}(m,n)$ and $\mathcal{N}_{1\text{pt}}(m,n)$. (b) Sparsity pattern of Jacobian matrix with $p = 2$ groups of partially structurally orthogonal columns resulting from $\mathcal{N}_{5,1\text{pt}}$ using a natural ordering of grid points on a 3×3 grid. (c) Sequence of covers obtained from using $p = 2$ covers each corresponding to a group of partially structurally orthogonal grid points for $\mathcal{N}_{5,1\text{pt}}$ on a 9×7 grid.

cannot be determined in 24h using the heuristic with IDO. In general, the heuristic D2ColoringRestricted yields the best colorings with the IDO preordering. However, the runtime is worse than using the natural ordering or LFO, and the number of colors are still worse than χ.

In summary, the algorithm colorVSepAll computes exact solutions without the need for an explicit formula. The runtime is significantly shorter compared to the exhaustive search. Although the runtime of our algorithm can be shorter or longer than the heuristic depending on the structure of the stencil, the solution of the greedy heuristic in terms of the colors p is always worse, sometimes dramatically.

3.1.2 Partial Jacobian computation

The stencil-based Jacobian computation is transferred to the principle of determining a subset of the nonzero elements of a Jacobian matrix. Rather than determining arbitrary nonzero elements of the Jacobian matrix, we introduce an additional stencil $\mathcal{N}_{\text{req}}$ which specifies the required elements. This stencil has the same center as the original stencil $\mathcal{N}_{\text{org}}$, and its grid points are a subset of the original stencil. The stencil $\mathcal{N}_{\text{req}}$ specifies the required elements of the Jacobian matrix, and the stencil $\mathcal{N}_{\text{org}}$ specifies the original vicinity. For such stencil combinations, the algorithm colorVSepAll (Alg. 3.2) is adapted to compute minimal colorings.

An illustrating example is the stencil combination of the original stencil $\mathcal{N}_{\text{org}} = \mathcal{N}_{5\text{pt}}(m,n)$ and the stencil $\mathcal{N}_{\text{req}} = \mathcal{N}_{1\text{pt}}(m,n) = \{(m,n)\}$. That is, we are only interested in the value of the center and not the values of the grid points in the neighborship. This stencil combination, which is depicted in Fig. 3.3(a), is denoted as stencil $\mathcal{N}_{5,1\text{pt}}(m,n)$. All grid points belong to stencil $\mathcal{N}_{5\text{pt}}$, and the grid point indicated

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

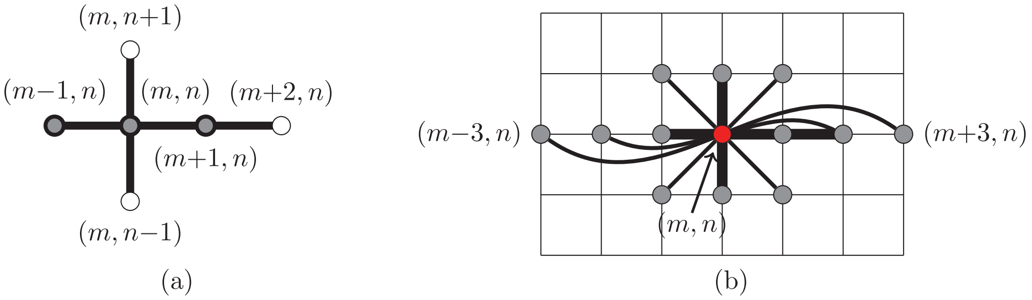


Figure 3.4: (a) Stencil $\mathcal{N}_{6,3\mathrm{pt}}(m,n)$. (b) Stencil $\mathcal{N}_{6,3\mathrm{pt}}(m,n)$ with partially structurally non-orthogonal grid points to center (m,n).

in gray is part of the stencil $\mathcal{N}_{1\mathrm{pt}}$. We resume the example in Fig. 2.4(c). The sparsity pattern of the Jacobian matrix results from the five-point stencil $\mathcal{N}_{5\mathrm{pt}}$ on a 3×3 grid. The sparsity pattern of the Jacobian matrix defined by the stencil $\mathcal{N}_{5,1\mathrm{pt}}$ is depicted in Fig. 3.3(b). Therefore, the required elements $\otimes$ are the main diagonal elements specified by stencil $\mathcal{N}_{1\mathrm{pt}}$, the remaining elements are the non-required elements $\times$. Instead of $p = 5$ column groups for the full Jacobian computation, $p = 2$ column groups are sufficient to determine the required nonzero elements. A 9×7 grid with a minimal coloring for the stencil $\mathcal{N}_{5,1\mathrm{pt}}$ is depicted in Fig. 3.3(c).

After explaining the partial Jacobian computation in terms of the grid, we describe how to modify the implementation for the full Jacobian computation to apply it to the partial one. The main difference is switching from considering structurally orthogonal grid points to partially structurally orthogonal grid points. There is the following dependency between these grid points:

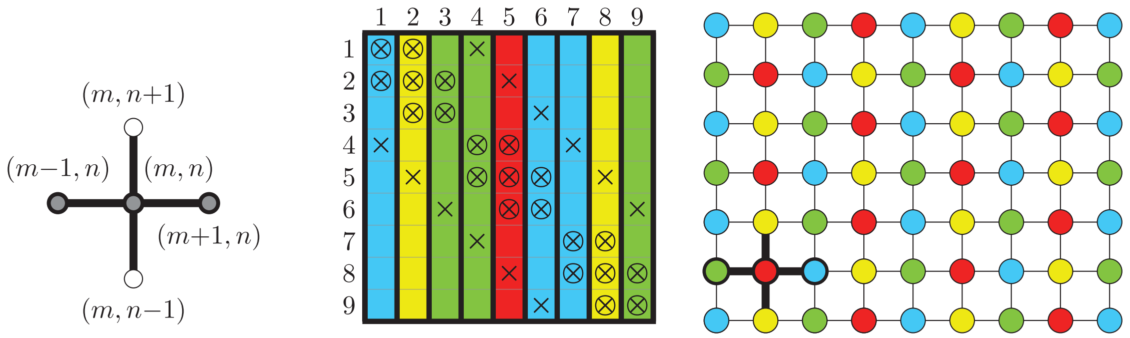
Definition 3.2. Two grid points (m,n) and (i,j) are *partially structurally orthogonal* if and only if the original stencil $\mathcal{N}_{\mathrm{org}}(m,n)$ and the stencil $\mathcal{N}_{\mathrm{req}}(i,j)$ as well as the stencils $\mathcal{N}_{\mathrm{req}}(m,n)$ and $\mathcal{N}_{\mathrm{org}}(i,j)$ do not have a grid point in common, i.e.,

$$(m,n) \perp_R (i,j) \quad :\Longleftrightarrow \quad \mathcal{N}_{\mathrm{org}}(m,n) \cap \mathcal{N}_{\mathrm{req}}(i,j) = \emptyset \wedge \mathcal{N}_{\mathrm{req}}(m,n) \cap \mathcal{N}_{\mathrm{org}}(i,j) = \emptyset.$$

This definition usually causes less partially structurally non-orthogonal grid points to every center compared to the full Jacobian computation. Recall that the center (m,n) of stencil $\mathcal{N}_{6\mathrm{pt}}(m,n)$ is structurally non-orthogonal to 18 grid points. These grid points are depicted in Fig. 3.1(b). The partially structurally non-orthogonal grid points for the stencil $\mathcal{N}_{6,3\mathrm{pt}}(m,n)$ are given in Fig. 3.4. The three-point stencil $\mathcal{N}_{3\mathrm{pt}}$, i.e., the center (m,n) and its two neighbors in the horizontal $(m,n-1)$ and $(m,n+1)$, specifies the required elements. There are only 12 partially structurally non-orthogonal grid points to a center.

We have to employ the modified neighborship relation to compute a coloring for the partial Jacobian computation with the function colorVSepAll (Alg. 3.2). Therefore, the grid G remains unchanged, but as stencil $\mathcal{N}$ a stencil combination is taken into account. Recall that the grid points of the stencil $\mathcal{N}_{\mathrm{req}}$ are a proper subset of the grid points of the stencil $\mathcal{N}_{\mathrm{org}}$ defining the original vicinity. Due to the

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.



(a) (b) (c)

Figure 3.5: (a) Stencil $\mathcal{N}_{5,3\text{pt}}(m,n)$. (b) Sparsity pattern of Jacobian matrix with $p = 4$ groups of partially structurally orthogonal columns resulting from stencil $\mathcal{N}_{5,3\text{pt}}$ using a natural ordering of grid points on a 3×3 grid. (c) Sequence of covers obtained from using $p = 4$ covers each corresponding to a group of partially structurally orthogonal grid points for $\mathcal{N}_{5,3\text{pt}}$ on a 9×7 grid.

smaller number of partially structurally non-orthogonal grid points, we usually need fewer colors to determine the required elements of the Jacobian matrix instead of all nonzero elements.

The precoloring technique must be modified to be consistent with the partial structural orthogonality. Recall that all grid points of a stencil are pairwise structurally non-orthogonal for full Jacobian computation. For partial Jacobian computation, the grid points of the original stencil $\mathcal{N}_{\text{org}}$ are in general not partially structurally non-orthogonal. In contrast, the grid points of the stencil $\mathcal{N}_{\text{req}}$ are pairwise partially structurally non-orthogonal. A minimal coloring for the grid points of the stencil $\mathcal{N}_{5,3\text{pt}}(2,2)$ is given in Fig. 3.5(c) to clarify the dependencies between the grid points: The grid points in stencil $\mathcal{N}_{\text{req}} = \{(1,2),(2,2),(3,2)\}$ are pairwise colored differently. The grid points $(2,1)$ and $(2,3)$ are also partially structurally non-orthogonal to the grid points in the $\mathcal{N}_{\text{req}}$. Thus, these grid points are differently colored from the grid points in $\mathcal{N}_{\text{req}}$. Otherwise, the values of the corresponding required elements would be lost. Due to Def. 3.2, the grid point $(2,1)$ is partially structurally orthogonal to grid point $(2,3)$. Therefore, both grid points are colored with the same color. Regarding the partial structural orthogonality, we obtain this precoloring by applying the algorithm colorExhS solely to the grid points of one stencil.

Results

Before considering the runtime of colorExhS (Alg. 3.2) and colorVSepAll (Alg. 3.3), we evaluate the smallest number of colors for several stencil combinations, which are depicted in Appendix A.1.2. These results are given in Table 3.4. We use the five-point stencil $\mathcal{N}_{5\text{pt}}$, the six-point stencil $\mathcal{N}_{6\text{pt}}$, and the nine-point stencil $\mathcal{N}_{9\text{pt}}$

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

$\mathcal{N}_{\text{org}}$			$\mathcal{N}_{\text{req}}$		
	$\mathcal{N}_{\text{1pt}}$	$\mathcal{N}_{\text{3pt}}$	$\mathcal{N}_{\text{5pt}}$	$\mathcal{N}_{\text{6pt}}$	$\mathcal{N}_{\text{9pt}}$
$\mathcal{N}_{\text{5pt}}$	2	4	5		
$\mathcal{N}_{\text{6pt}}$	3	4	7	7	
$\mathcal{N}_{\text{9pt}}$	3	6	8	10	10

Table 3.4: Number of minimal colors $p = \chi$ for partial Jacobian computation using stencil combinations of original stencil $\mathcal{N}_{\text{org}}$ and stencil $\mathcal{N}_{\text{req}}$.

as original stencils $\mathcal{N}_{\text{org}}$. The one-point stencil $\mathcal{N}_{\text{1pt}}$, the three-point stencil $\mathcal{N}_{\text{3pt}}$, and the five-point stencil $\mathcal{N}_{\text{5pt}}$ are employed as stencil $\mathcal{N}_{\text{req}}$. The number of colors is reduced for all combinations with $|\mathcal{N}_{\text{req}}| < |\mathcal{N}_{\text{org}}|$—except for stencils $\mathcal{N}_{\text{6,5pt}}$ and $\mathcal{N}_{\text{9,6pt}}$. A larger difference between $|\mathcal{N}_{\text{req}}|$ and $|\mathcal{N}_{\text{org}}|$ causes a smaller number of colors. Taking $\mathcal{N}_{\text{5,1pt}}$ instead of $\mathcal{N}_{\text{5pt}}$, we reduce the number of colors from $p = 5$ to $p = 2$; for $\mathcal{N}_{\text{5,3pt}}$, the number of colors is reduced to $p = 4$. Using the stencils $\mathcal{N}_{\text{9,1pt}}$ or $\mathcal{N}_{\text{9,3pt}}$, $p = 3$ colors or $p = 6$ colors, respectively, instead of $p = 10$ colors for $\mathcal{N}_{\text{9pt}}$ are needed. If the difference between the number of grid points of the original stencil $\mathcal{N}_{\text{org}}$ and the stencil $\mathcal{N}_{\text{req}}$ is too small, e.g., stencil $\mathcal{N}_{\text{6,5pt}}$, the number of colors is not reduced compared to the full Jacobian computation. There is no difference between computing the stencil combinations $\mathcal{N}_{\text{5,5pt}}$, $\mathcal{N}_{\text{6,6pt}}$, and $\mathcal{N}_{\text{9,9pt}}$ with partial Jacobian computation and the stencils $\mathcal{N}_{\text{5pt}}$, $\mathcal{N}_{\text{6pt}}$, and $\mathcal{N}_{\text{9pt}}$ with full Jacobian computation. Hence, the full Jacobian computation is a special case of the partial Jacobian computation.

The algorithms colorExhS and colorVSepAll are evaluated for two-dimensional $N \times N$ grids using various stencil combinations. The grid size is varied—as for the full Jacobian computation—from $N = 9$ to $N = 20{,}479$ by doubling the number of grid points in the horizontal and the vertical direction. The runtimes are given in Table 3.5. Opposed to the results for the full Jacobian computation, interpreting these runtimes is more challenging. For stencils $\mathcal{N}_{\text{5,1pt}}$, $\mathcal{N}_{\text{5,3pt}}$, $\mathcal{N}_{\text{6,3pt}}$, and $\mathcal{N}_{\text{9,1pt}}$, the algorithm colorVSepAll is significantly faster than the exhaustive search. The runtime is at most 42s for an $N \times N$ grid with $N = 20{,}479$. Using the algorithm colorExhS to color a grid exceeds 24h starting with $N = 79$ for $\mathcal{N}_{\text{5,3pt}}$ and $N = 2{,}559$ for the other three stencil combinations. A comparison of the runtime for $\mathcal{N}_{\text{6,1pt}}$ is impossible, because the memory usage of colorVSepAll exceeds the available main memory of 32GB for each grid with $N \geq 39$. For stencil $\mathcal{N}_{\text{9,3pt}}$ and grid size $N = 19$, our algorithm colorVSepAll requires only 9s compared to the exhaustive search using 470s. For $\mathcal{N}_{\text{9,5pt}}$, the algorithm colorExhS is faster than colorVSepAll for small grid sizes. However, the sub-exponential complexity of colorVSepAll ensures that there is a smallest grid size for which this algorithm outperforms colorExhS. Next, we compare the algorithm colorVSepAll for full and partial Jacobian computation. When using the five-point stencil $\mathcal{N}_{\text{5pt}}$ as original stencil and the stencils $\mathcal{N}_{\text{1pt}}$ and $\mathcal{N}_{\text{3pt}}$ as stencil $\mathcal{N}_{\text{req}}$, the runtimes are comparable to the runtime for full the Jacobian computation with stencil $\mathcal{N}_{\text{5pt}}$. For the $20{,}479 \times 20{,}479$ grid, the runtime for the stencil $\mathcal{N}_{\text{5pt}}$ is 23s (cf. Table 3.1) compared to the runtimes for $\mathcal{N}_{\text{5,1pt}}$ and $\mathcal{N}_{\text{5,3pt}}$ with 9s and 11s, respectively. The runtimes for

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

Stencil	Algorithm	N								
		9	19	39	79	159	2,559	5,119	10,239	20,479
$\mathcal{N}_{5,1\text{pt}}$	colorExhS	ϵ	ϵ	ϵ	ϵ	ϵ	$\gg$	$\gg$	$\gg$	$\gg$
	colorVSepAll	ϵ	ϵ	ϵ	ϵ	ϵ	ϵ	ϵ	3	9
$\mathcal{N}_{5,3\text{pt}}$	colorExhS	ϵ	ϵ	8,729	$\gg$	$\gg$	$\gg$	$\gg$	$\gg$	$\gg$
	colorVSepAll	ϵ	ϵ	ϵ	ϵ	ϵ	ϵ	2	3	11
$\mathcal{N}_{6,1\text{pt}}$	colorExhS	ϵ	ϵ	ϵ	ϵ	1	$\gg$	$\gg$	$\gg$	$\gg$
	colorVSepAll	ϵ	ϵ	–	–	–	–	–	–	–
$\mathcal{N}_{6,3\text{pt}}$	colorExhS	ϵ	ϵ	ϵ	ϵ	2	$\gg$	$\gg$	$\gg$	$\gg$
	colorVSepAll	ϵ	ϵ	ϵ	ϵ	ϵ	ϵ	ϵ	4	42
$\mathcal{N}_{9,1\text{pt}}$	colorExhS	ϵ	ϵ	ϵ	ϵ	1	$\gg$	$\gg$	$\gg$	$\gg$
	colorVSepAll	ϵ	ϵ	ϵ	ϵ	ϵ	ϵ	ϵ	3	13
$\mathcal{N}_{9,3\text{pt}}$	colorExhS	ϵ	470	$\gg$	$\gg$	$\gg$	$\gg$	$\gg$	$\gg$	$\gg$
	colorVSepAll	ϵ	9	$\gg$	$\gg$	$\gg$	$\gg$	$\gg$	$\gg$	$\gg$
$\mathcal{N}_{9,5\text{pt}}$	colorExhS	ϵ	8,517	$\gg$	$\gg$	$\gg$	$\gg$	$\gg$	$\gg$	$\gg$
	colorVSepAll	ϵ	$\gg$	$\gg$	$\gg$	$\gg$	$\gg$	$\gg$	$\gg$	$\gg$

Table 3.5: Runtime in seconds to compute one minimal coloring using the exhaustive search and the vertex separator algorithm for different $N \times N$ grids using 1 thread. The smallest subgrid is of size 9×9 ($n_0 = 81$). Runtime below 1s is denoted by ϵ, exceeding 24h (86,400s) by $\gg$. Memory usage exceeding 32GB is denoted by –.

stencil $\mathcal{N}_{6\text{pt}}$ and $\mathcal{N}_{6,3\text{pt}}$ are in the same order with 167s to 42s. The runtime for stencil combination $\mathcal{N}_{9,1\text{pt}}$ is reduced to 13s compared to the stencil $\mathcal{N}_{9\text{pt}}$ using 1 thread where the runtime exceeds 24h for a grid with $N = 20{,}479$. For stencils $\mathcal{N}_{9,3\text{pt}}$ and $\mathcal{N}_{9,5\text{pt}}$, as counterpart, minimal colorings can only be determined for very small grid sizes in less than 24h using colorVSepAll, i.e., $\mathcal{N}_{9,3\text{pt}}$ with $N \leq 19$ and $\mathcal{N}_{9,5\text{pt}}$ with $N = 9$. Using the stencil $\mathcal{N}_{9\text{pt}}$ for full Jacobian computation in comparison, the runtimes of colorVSepAll are less than 756s for an $N \times N$ grid with $N \leq 39$. In summary, restricting the original stencil $\mathcal{N}_{\text{org}} = \mathcal{N}_{9\text{pt}}$ by the stencil $\mathcal{N}_{\text{req}} = \mathcal{N}_{1\text{pt}}$, the runtime for the partial Jacobian computation is significantly reduced compared to the full Jacobian computation. For stencils $\mathcal{N}_{5,1\text{pt}}$, $\mathcal{N}_{5,3\text{pt}}$, and $\mathcal{N}_{6,3\text{pt}}$, at most half of the runtime for the full Jacobian computation is needed. Finally, the runtime of colorVSepAll is reasonably faster than using the exhaustive search for stencils $\mathcal{N}_{5,1\text{pt}}$, $\mathcal{N}_{5,3\text{pt}}$, $\mathcal{N}_{6,3\text{pt}}$, and $\mathcal{N}_{9,1\text{pt}}$.

The reason for the huge differences in the runtime are not only the number of partially structurally non-orthogonal grid points, but, above all, the number of minimal colorings of the subgrids. All combinations of these colorings are processed by colorVSepAll in lines 10–11 in Alg. 3.2. Thus, the higher the number of colorings, the higher is the runtime. The number of minimal colorings—without permutations—for the grids on the lowest levels of the hierarchy—9×9, 9×19, and 19×19—are given in Table 3.6. For the stencils $\mathcal{N}_{5,1\text{pt}}$, $\mathcal{N}_{5,3\text{pt}}$, $\mathcal{N}_{6,3\text{pt}}$, and $\mathcal{N}_{9,1\text{pt}}$ the number of colorings on these levels are reduced compared to the original stencil for the full Jacobian computation. For the other stencils with runtimes exceeding 24h, there are

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

$M \times N$	Stencil									
	$\mathcal{N}_{5\text{pt}}$	$\mathcal{N}_{5,1\text{pt}}$	$\mathcal{N}_{5,3\text{pt}}$	$\mathcal{N}_{6\text{pt}}$	$\mathcal{N}_{6,1\text{pt}}$	$\mathcal{N}_{6,3\text{pt}}$	$\mathcal{N}_{9\text{pt}}$	$\mathcal{N}_{9,1\text{pt}}$	$\mathcal{N}_{9,3\text{pt}}$	$\mathcal{N}_{9,5\text{pt}}$
9×9	2	1	1	2	64	1	4	1	32	186,368
9×19	2	1	1	2	16,384	1	4	1	68	–
19×19	2	1	1	2	16,384	1	4	1	8192	–

Table 3.6: Number of minimal colorings (without permutations) using algorithm colorVSepAll for different $M \times N$ grids. Symbol – denotes unknown number of colorings due to runtime exceeding 24h (86,400s).

a lot of different colorings. For stencil $\mathcal{N}_{9,3\text{pt}}$, there are 32 minimal colorings for the 9×9 grid, 68 colorings for the 9×19 grid, and 8,192 colorings for the 19×19 grid. Thus, there are a lot of combinations to check in colorVSepAll. For the stencils $\mathcal{N}_{6,1\text{pt}}$ and $\mathcal{N}_{9,5\text{pt}}$, there are also a lot of minimal colorings. Comparing the number of minimal colorings to the runtimes in Table 3.5 uncovers the deep impact of the number of colorings on the runtime: A higher number of minimal colorings leads to an increased runtime. For the stencils $\mathcal{N}_{5,1\text{pt}}$, $\mathcal{N}_{5,3\text{pt}}$, $\mathcal{N}_{6,3\text{pt}}$, and $\mathcal{N}_{9,1\text{pt}}$, the number of minimal colorings is only one on every level in the hierarchy, and the runtime is at most 42s for a grid up to $N = 20{,}479$. For the other stencils where the number of colorings is significant higher, the algorithm cannot compute a minimal coloring for most considered grid sizes in 24h.

In summary, using stencil combinations for partial Jacobian computation, the runtime is much faster than using the original stencil for full Jacobian computation or is slower. Although the algorithm colorExhS seems to be better than the algorithm colorVSepAll for stencil $\mathcal{N}_{9,5\text{pt}}$, there will be a smallest grid size for which colorVSepAll outperforms colorExhS due to the sub-exponential complexity. Using colorVSepAll, the huge number of minimal colorings arising on all levels of the hierarchy leads to high runtimes exceeding 24h. To overcome this issue, a new size-independent method to compute a coloring for a grid of arbitrary size is introduced in the next section.

3.2 Size-independent exact coloring algorithm for regular grids

The algorithms colorVSepAll (Alg. 3.2) and colorExhS (Alg. 3.3) compute minimal (restricted) colorings for regular grids depending on the (partial) structural orthogonality of the given stencil. Unfortunately, the runtime depends—beside the stencil itself—directly on the grid size. Explicit solutions to color regular grids are known for various special stencils. Investigating a formula for every single stencil is time consuming and, above all, complex in more than two dimensions. By observing the colorings in Fig. 2.5(b), Fig. 3.3(c), and Fig. 3.5(c), we realize that these colorings have a quite regular, repeating structure. Given a small-sized grid G and a minimal coloring Φ, we search for a subgrid, a so-called tile, whose coloring information is

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

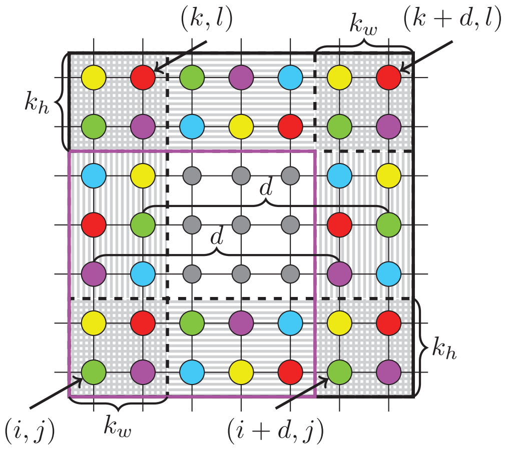


Figure 3.6: The 7×7 subgrid is a detail of an $M \times N$ grid with a minimal coloring Φ for the five-point stencil $\mathcal{N}_{5\text{pt}}$. The rectangular regions at the border are indicated by black boxes with background shading and the 5×5 tile with coloring information by a magenta box.

sufficient to color a larger grid of arbitrary size. Therefore, this colored tile is placed several times next to each other. Thus, a regular grid of arbitrary size can be colored in linear time. Currently, we cannot show that this approach works in general for all stencils. In the following, our approach is explained for two-dimensional grids, but it is also applicable for grids in higher dimensions. In three dimensions, the result is a rectangular cuboid instead of a tile. To avoid excessive distinguishing between full and partial Jacobian computation, without loss of generality, our approach is only explained for full Jacobian computation.

For a given grid G with a minimal coloring Φ, we look for a subgrid with four rectangular regions. These regions are at the left, right, bottom, and top inside the subgrid. The regions on the opposite sides, i.e., left and right as well as bottom and top, are of the same height and width. An illustrating example is given in Fig. 3.6. The regions are indicated with background shadings in gray. The maximum width or height between a center (m,n) and a structurally non-orthogonal grid point (i,j) is given in (3.1) without taking the orientation into account. Here, we distinguish between the width in the horizontal direction and the height in the vertical direction

$$k_w = \max_{\substack{(i,j) \\ (i,j) \not\perp (m,n)}} \{|m-i|\} \quad \text{and} \quad k_h = \max_{\substack{(i,j) \\ (i,j) \not\perp (m,n)}} \{|n-j|\}.$$

The rectangular regions at the left and right have a width of k_w and the rectangular regions at the bottom and top have a height of k_h. The regions are so high or wide, depending on the orientation, that two structurally non-orthogonal grid points cannot be on the different sides outside such a rectangular region.

The region at the left is spanned by the grid points (i,j) and (k,l) and at the right by the grid points $(i+d,j)$ and $(k+d,l)$ for some distance d. This distance is independent from the width k_w. Two grid points, one in each region, form a pair if

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

both grid points are in the same row and the distance d between these grid points is the same as for all other pairs of grid points. Each pair of grid points is colored identically, i.e.,

$$\Phi(\psi(i,j)) = \Phi(\psi(i+d,j)), \ldots, \Phi(\psi(k,l)) = \Phi(\psi(k+d,l)).$$

Considering the rectangular regions at the bottom and top is analogous except for the orientation. The relation between two regions is introduced in the following definition:

Definition 3.3. Two rectangular regions are *consistent* if both have the same height and width and the corresponding grid points are colored identically, i.e.,

- for the left and right regions, $\Phi(\psi(i,j)) = \Phi(\psi(i+d,j))$, and
- for the bottom and top regions, $\Phi(\psi(i,j)) = \Phi(\psi(i,j+d))$.

We look for consistent left and right regions as well as consistent bottom and top regions. If we detect four pairwise consistent regions, we obtain a tile with coloring information. An illustrating example for such a tile is emphasized by the magenta box in Fig. 3.6.

The four-point stencil $\mathcal{N}_{4\text{pt}}$ in Fig. 3.7 is studied to illustrate the idea of our approach. All structurally non-orthogonal grid points to the center (m,n) are depicted in Fig. 3.7(b). The horizontal width is $k_w = 1$ and the vertical height is $k_h = 2$. A 3×4 subgrid with pairwise consistent rectangular regions is indicated by a black box with a solid line in Fig. 3.7(c). This subgrid is spanned by the grid point $(1,2)$ in the south west corner and by the grid point $(3,5)$ in the north east corner. The rectangular region at the left with width k_w is spanned by the grid points $(1,2)$ and $(1,5)$ in the south west and north east. The region at the right is spanned by the grid points $(3,2)$ and $(3,5)$. The regions at the left and right are consistent, i.e., $\Phi(\psi(1,2)) = \Phi(\psi(3,2))$, $\Phi(\psi(1,3)) = \Phi(\psi(3,3))$, $\Phi(\psi(1,4)) = \Phi(\psi(3,4))$, $\Phi(\psi(1,5)) = \Phi(\psi(3,5))$. The regions at the bottom and top—both of height k_h—are spanned by the grid points $(1,2)$ and $(3,3)$ as well as the grid points $(1,4)$ and $(3,5)$. These regions are also consistent, i.e., $\Phi(\psi(1,2)) = \Phi(\psi(1,4))$, $\Phi(\psi(2,2)) = \Phi(\psi(2,4))$, $\ldots$, $\Phi(\psi(3,3)) = \Phi(\psi(3,5))$. Finally, the right and top rectangular regions of this subgrid are pruned to get the so-called colored tile. Afterwards, the coloring information can be used to color a grid of arbitrary size. An example for a 9×5 grid using the 2×2 colored tile from Fig. 3.7(c) is depicted in Fig. 3.7(d). This tile is repeatedly placed next to each other in the horizontal and vertical direction. Another way to think about coloring a larger grid is to use the resulting subgrid and place it on the grid with overlapping the left and right region as well as the bottom and top region. That is, the right region of a subgrid is overlapped with the left region of the next subgrid in horizontal direction and the top region of a subgrid with the bottom region of the above subgrid. This resulting coloring is minimal, because the given coloring Φ is minimal and no extra color is added.

In the following lemma, it is shown that the coloring from the previous example and other colorings created using colored tiles are valid.

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

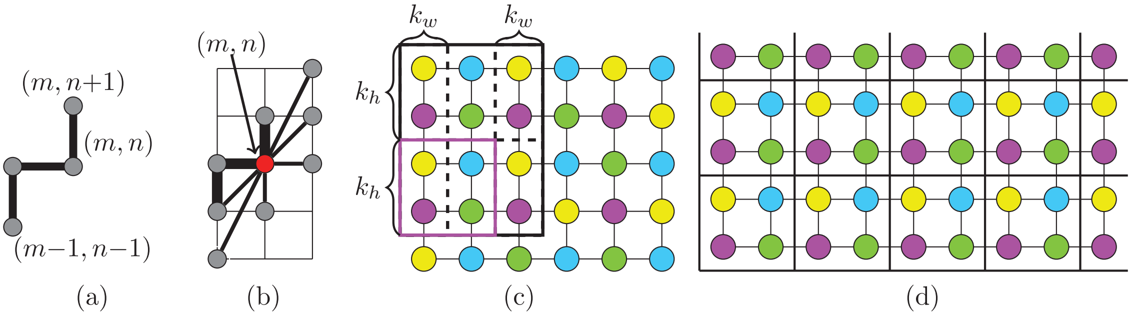


Figure 3.7: (a) Four-point stencil $\mathcal{N}_{4\text{pt}}$. (b) Stencil $\mathcal{N}_{4\text{pt}}(m, n)$ with structurally non-orthogonal grid points to center (m, n). (c) Colored tile, indicated by a magenta box, found in a coloring using $p = 4$ colors each corresponding to a group of structurally orthogonal center points for $\mathcal{N}_{4\text{pt}}$ on a 6×5 grid. (d) Sequence of covers obtained by repeatedly placing the colored tile from (b) on a 9×5 grid.

Lemma 3.4. *Given a regular grid G containing a colored tile with the smallest number of colors, this colored tile can be repeatedly placed next to each other in the horizontal and vertical direction. The result is a minimal coloring Φ' for a grid G' of arbitrary size.*

Proof. Instead of considering the colored tile, we look at the original subgrid whose rectangular regions at the right and top are not pruned. We distinguish between two different kinds of grid points in the subgrid: grid points which are part of the rectangular regions and grid points which are not part of the rectangular regions. It might be that there are no grid points of the second kind. We show for both kinds of grid points that the resulting coloring Φ' is valid.

First, the grid points which are not part of the rectangular regions are considered. To each grid point, there is no structurally non-orthogonal grid point outside the subgrid, because the regions with width k_w or height k_h are chosen in an appropriate way. Thus, each segment of the coloring Φ'—corresponding to the colored subgrid—is valid, because the given coloring Φ is minimal.

Second, we consider the grid points which are contained in the rectangular regions and their structurally non-orthogonal grid points. In the coloring Φ', such a grid point is part of the regions in the overlap of two (or four) subgrids. That is, each grid point and its structurally non-orthogonal grid points are already in the given minimal coloring Φ. Thus, the coloring Φ' cannot be invalid.

In summary, due to the width and height of the rectangular regions, the tile can be placed next to each other in the horizontal and vertical direction, and the corresponding coloring Φ' is valid and minimal. □

The algorithm GETTILE takes an already computed minimal coloring Φ for a grid G and determines a subgrid, if available, whose coloring can be used to color a

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

Algorithm 3.4: Determine a tile restricted by grid point (i, j) and grid point $(k - k_w, l - k_h)$ for a given $M \times N$ grid G, stencil $\mathcal{N}$ and minimal (restricted) coloring Φ (Def. 2.6 or 2.10). The stencil $\mathcal{N}$ states width k_w and height k_h.

```
function getTile($G, \mathcal{N}, \Phi$)
    foreach $(i, j) \in G$ do                                                    ▷ south west corner
        foreach $(k, l) \in G$ with $k - i + 1 \geq 2k_w$ and $l - j + 1 \geq 2k_h$ do    ▷ ne corner
            if $\Phi(\psi(i + h, v)) = \Phi(\psi(k - k_w + h, v)), \forall h, \forall v\colon 0 \leq h \leq k_w, j \leq v \leq l$ and
              $\Phi(\psi(h, j + v)) = \Phi(\psi(h, l - k_h + v)), \forall h, \forall v\colon i \leq h \leq k, 0 \leq v \leq k_h$ then
                return $(i, j), (k - k_w, l - k_h)$                                   ▷ tile found
    return $(0, 0), (0, 0)$                                                     ▷ no tile found
```

larger grid by placing the colored tiles next to each other. This algorithm is given in Alg. 3.4. The algorithm searches for a subgrid where the rectangular regions at the left and right as well as at the bottom and top are consistent. Therefore, the algorithm compares subgrids which are spanned by the grid points (i, j) and (k, l). Due to the conditions $k - i + 1 \geq 2k_w$ and $l - j + 1 \geq 2k_h$, these subgrids are large enough to cover the four rectangular regions. The colorings of the regions at the left and right as well as at the bottom and top are compared if these regions are consistent. Therefore, all pairs of grid points with the same distance are checked. The distance in the horizontal direction is $k - i - k_w + 1$ and in the vertical direction $l - j - k_h + 1$. If all compared grid points are colored identically, an appropriate subgrid is found. The regions at the right and top of this subgrid are pruned. The result is the tile restricted by the grid points (i, j) and $(k - k_w, l - k_h)$. This tile is possibly not the smallest tile available. Furthermore, this approach does not guarantee to determine a tile. It depends on the given coloring Φ if a tile exists.

Lemma 3.5. *Given a regular grid G with n grid points and a minimal (restricted) coloring Φ, the algorithm* getTile *determines the size of a colored tile, if available, in $\mathcal{O}(n^3)$.*

Proof. We assume that a solution, a colored tile, exists. The algorithm getTile iterates over all pairs of grid points (i, j) and (k, l). These grid points span the subgrids to be checked. Each subgrid is spanned by the grid point (i, j) in the south west direction and by the grid point (k, l) in the north east direction. The horizontal and vertical distance between both grid points, $2k_w$ and $2k_h$, is big enough to cover the four rectangular grids at the border. The algorithm checks if the regions at the bottom and top as well as at the left and right are consistent. If there exists such a subgrid, the corresponding tile is the result of the algorithm.

The algorithm getTile iterates over all pairs of grid points (i, j) and (k, l). For each subgrid spanned by these grid points, the grid points at equal distance in the regions belonging together must be checked. Therefore, at most $2 \cdot n$ grid points must be visited. Thus, the complexity is $\mathcal{O}(n^3)$. □

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

When the colored $Q \times R$ tile is determined, the color given by the explicit formula

$$(((i-1) \bmod Q) + 1, ((j-1) \bmod R) + 1) \tag{3.2}$$

can be assigned to the grid point (i, j) of a grid with arbitrary size. To color the grid in Fig. 3.7(d), the coloring information for the tile in Fig. 3.7(c) is used. For example, if the grid point $(2, 4)$ should be colored, the color

$$\Phi(\psi(((2-1) \bmod 2) + 1, ((4-1) \bmod 2) + 1)) = \Phi(\psi(2, 2))$$

is assigned.

We cannot prove that our approach works for all stencils or stencil combinations. If the algorithm does not return a tile with coloring information, there are two possibilities to continue: On the one hand, another minimal coloring for grid G with the same grid size can be employed. On the other hand, the grid size can be increased and a minimal coloring for this grid can be used as input for getTile. If we do not obtain a tile with coloring information by employing one or both possibilities, the last change is to use the algorithm colorVSepAll (Alg. 3.2) to compute a minimal coloring.

Results

Our approach is evaluated by considering several stencils for the full Jacobian computation and stencil combinations for the partial Jacobian computation on two-dimensional $N \times N$ grids. We start with considering a 9×9 grid. For this grid, a minimal (restricted) coloring is computed by the algorithm colorVSepAll. This coloring is the input for the algorithm getTile. This algorithm immediately stops if a $(Q + k_w) \times (R + k_h)$ subgrid—and the corresponding $Q \times R$ tile—is found. If such a subgrid does not exist, we switch to the 19×19 grid and a suitable minimal coloring.

For each considered stencil, the minimal coloring for the 9×9 or 19×19 grid is sufficient to obtain a tile. That is, for every stencil or stencil combination, a colored tile is found. The results are given in Table 3.7. For stencil $\mathcal{N}_{4\text{pt}}$, a 2×2 tile in the colored 9×9 grid is found. The width and height for this stencil are $k_w = 1$ and $k_h = 2$. The coloring of the 9×9 grid with colorVSepAll using 1 thread together with the determination of the tile takes less than one second. For stencil $\mathcal{N}_{9\text{pt}}$, there is no tile in the coloring of the 9×9 grid. Thus, a 19×19 grid is colored and a 10×10 tile is found in 214 seconds. Most tiles are computed in less than 1s including the determination of the colorings with colorVSepAll using 1 thread. For most stencils—in particular, stencils $\mathcal{N}_{4\text{pt}}$, $\mathcal{N}_{5\text{pt}}$, $\mathcal{N}_{5,1\text{pt}}$, $\mathcal{N}_{5,3\text{pt}}$, $\mathcal{N}_{6,1\text{pt}}$, $\mathcal{N}_{6,3\text{pt}}$, and $\mathcal{N}_{9,1\text{pt}}$—the minimal coloring of the 9×9 grid contains a tile. For the remaining three stencils, the minimal coloring of the 19×19 grid is appropriate to obtain a tile. Recall from the previous section that the runtime for computing a minimal coloring with the algorithm colorVSepAll depends on the grid size. In some cases, no coloring could be determined in 24h, still for small grid sizes. Using the approach introduced in this section, we obtain the

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

Stencil	k_w	k_h	N	$Q \times R$	s
$\mathcal{N}_{4\text{pt}}$	1	2	9	2×2	ϵ
$\mathcal{N}_{5\text{pt}}$	2	2	9	5×5	ϵ
$\mathcal{N}_{6\text{pt}}$	3	2	19	7×7	ϵ
$\mathcal{N}_{9\text{pt}}$	4	4	19	10×10	214

(a)

Stencil	k_w	k_h	N	$Q \times R$	s
$\mathcal{N}_{5,1\text{pt}}$	1	1	9	2×2	ϵ
$\mathcal{N}_{5,3\text{pt}}$	2	1	9	4×2	ϵ
$\mathcal{N}_{6,1\text{pt}}$	3	2	9	3×2	ϵ
$\mathcal{N}_{6,3\text{pt}}$	3	2	9	4×2	ϵ
$\mathcal{N}_{9,1\text{pt}}$	2	2	9	3×3	ϵ
$\mathcal{N}_{9,3\text{pt}}$	3	2	19	4×3	9

(b)

Table 3.7: Overall runtime s in seconds to determine a colored $Q \times R$ tile on an $N \times N$ grid with $N = 9$ or $N = 19$. The minimal coloring Φ of an $N \times N$ grid is computed by colorVSepAll. Runtime below 1s is denoted by ϵ.

colorings in at most 214s and are able to color a grid point of an arbitrarily sized grid using a colored tile and equation (3.2). The corresponding colored tiles are depicted in Appendix A.2.

Although we are not able to prove that this approach can be used in general, it works for all considered stencils and stencil combinations. This open issue could be considered for further work.

3.3 Two-sided coloring algorithm for general graphs

After introducing coloring algorithms exploiting the sparsity pattern of Jacobian matrices occurring in stencil-based computations, we move on to coloring heuristics for bipartite graphs associated to general Jacobian matrices. Given a bipartite graph $G = (V_r \uplus V_c, E)$ corresponding to a Jacobian matrix and the set of required elements E_R for the partial Jacobian computation, we introduce a coloring algorithm which determines a star bicoloring of G when restricted to E_R. Recall that the star bicoloring is a special case of the restricted star bicoloring where every edge is a required edge, $E_R = E$. Therefore, an explicit version of the star bicoloring algorithm for the full Jacobian computation is omitted. A star bicoloring is also denoted as two-sided coloring, because the row and column vertices of a bipartite graph are colored.

There are several papers concerning two-sided colorings for the full Jacobian computation [19, 27, 33]. Although the coloring definition Def. 2.12 for the partial Jacobian computation and the idea for an algorithm arises in [27], the first algorithm to compute restricted star bicolorings was given in [40] and extended in [15]. The partial Jacobian computation in general and, in particular, the implementation of this algorithm was first evaluated in [40, 41] by choosing the nonzero elements on the main diagonal as required elements. It was demonstrated that determining only the main diagonal elements, instead of all nonzero elements, reduces the number of colors p significantly. Calotoiu investigated further modifications of the implementation and vertex orderings in [15]. He evaluated the partial Jacobian computation for

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

several patterns, including the block diagonal with 1000×1000 blocks. In the meantime, Calotoiu and I made further progress with trying different vertex orderings and evaluated different techniques for star bicoloring algorithms in a joint work. In a more recent work, the two-sided coloring was given as an integer linear programming (ILP) formulation [30]. Due to the NP-hardness of this optimization problem, solving this ILP formulation exactly is limited to small matrices. One algorithm variant for the restricted star bicoloring from the joint work with Calotoiu is given in this section. Going beyond this joint work, block diagonals with different block sizes are considered in the results section.

Our coloring heuristic comprises several techniques introduced in other two-sided coloring approaches for full Jacobian computation [19, 27, 33]. These techniques are adapted to obtain a coloring heuristic for the restricted star bicoloring. We show that there are matrices for which a star bicoloring is better than a distance-2 coloring. Furthermore, our implementation is compared to the up-to-date software package ColPack [28] to show that the results are comparable in the number of colors for the full Jacobian computation. Thereafter, we continue with the partial Jacobian computation and compute restricted star bicolorings to obtain the nonzero elements of block diagonals with different block sizes. We consider matrices and patterns employed in the next chapter to show the benefit of the partial Jacobian computation.

For star bicolorings, with reference to condition 2 of Def. 2.12, solely a subset of the vertices must be colored with nonzero colors. The remaining vertices receive the "neutral" color zero. That is, the corresponding rows and columns are not determined and thus do not affect the computational effort. The vertices which must be colored with nonzero colors are chosen. This selection process can be considered as explicitly or implicitly determining a vertex cover or its complement, the independent set [27]. The minimum vertex cover problem as well as the maximum independent set problem are NP-hard [26]. Rather than computing an (almost) minimum vertex cover, it is more important to compute a vertex cover so that the contained vertices can be colored with a number of colors as small as possible. The star bicoloring scheme given in [27] computes a vertex cover first and colors the selected vertices afterwards. In contrast, the algorithm introduced in [33] computes a vertex cover implicitly and colors a vertex immediately after being selected. Our algorithm comprises the implicit choice of the vertex cover [33] combined with a weighting function for the row and column vertices [20]. The vertices are colored by employing the coloring algorithm proposed for the full Jacobian computation [27] adapted to the partial Jacobian computation.

The algorithm StarBicoloringRestricted, depicted in Alg. 3.5, chooses the vertices one after another from the vertex set $V_r \uplus V_c$ regarding the required edges E_R. In line 5, the next vertex is chosen by the function getNextVertex which is given in Alg. 3.6. A weighting factor for preferring row over column vertices or the other way around is used. This factor ρ is employed while comparing the maximum degrees of the sets of row and column vertices of the bipartite subgraph induced by the required edges E_R, i.e., $\Delta(V_r, G[E_R])$ and $\Delta(V_c, G[E_R])$. Depending on the maximum degrees and ρ, the algorithm chooses a row vertex v_r or column vertex v_c with the

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

Algorithm 3.5: Determine a star bicoloring of G when restricted to E_R

```
1  function StarBicoloringRestricted($G = (V_r \uplus V_c, E), E_R, \rho$)
2      forbiddenColors ← $[0 \dots 0]$; coloring $\Phi \leftarrow [-1 \dots -1]$
3      $E'_R \leftarrow E_R$
4      while $E'_R \neq \emptyset$ do
5          $v \leftarrow$ getNextVertex($G, E'_R, \rho$)              ▷ Dynamic selection on subgraph
6          $E'_R \leftarrow E'_R \setminus \{(v,w) \in E'_R \colon w \in N_1(v, G[E'_R])\}$
7          foreach $w \in N_1(v, G)$ do
8              if $\Phi(w) \leq 0$ then
9                  foreach $x \in N_1(w, G)$ with $\Phi(x) > 0$ do
10                     if $(v,w) \in E_R$ or $(w,x) \in E_R$ then
11                         forbiddenColors[$\Phi(x)$] ← $v$
12             else
13                 foreach $x \in N_1(w, G[E_R])$ with $\Phi(x) > 0$ do
14                     foreach $y \in N_1(x, G)$ with $\Phi(y) > 0$ and $y \neq w$ do
15                         if $\Phi(w) = \Phi(y)$ then
16                             forbiddenColors[$\Phi(x)$] ← $v$
17         $\Phi(v) \leftarrow \min\{j > 0 \colon$ forbiddenColors$[j] \neq v\}$
18     foreach $v_c \in V_c$ with $\Phi(v_c) > 0$ do $\Phi(v_c) \leftarrow \Phi(v_c) + \max\{\Phi(v_r) \colon v_r \in V_r\}$
19     foreach $v \in V_r \uplus V_c$ with $\Phi(v) = -1$ do $\Phi(v) \leftarrow 0$
20     return $\Phi$
```

highest degree, i.e., $d(v_r, G[E_R])$ or $d(v_c, G[E_R])$. In [20], the weighting factor is hard coded with $\rho = 2$. A more general suggestion for weighting the vertices is given in [27]. The resulting vertex ordering influences the number of colors obtained by the algorithm. An additional edge set E'_R is introduced to memorize for which required edges no incident vertex is colored yet. After choosing a vertex v, the algorithm removes the incident edges in E'_R and determines the colors which are not allowed. The smallest nonzero color allowed is then assigned to the vertex v. Therefore, for every vertex v the distance-1, -2 and -3 neighbors—$w = N_1(v, G)$, $x = N_2(v, G)$, and $y = N_3(v, G)$—regarding the required edges are visited in lines 7–16. Each color that cannot be assigned to the vertex v, due to restrictions from Def. 2.12, is marked as forbidden in lines 11 and 16. Therefore, the index of vertex v is stored in the entry of the array forbiddenColors corresponding to the color of the distance-2 neighbor x of vertex v. In line 17, the smallest color not marked as forbidden is assigned to the vertex v. The nonzero colors for the row and column vertices are separated by adding the highest color assigned to a row vertex to the nonzero color of each column vertex in line 18. At the end, the algorithm assigns the color zero to all vertices not yet colored.

Lemma 3.6. *Given a bipartite graph $G = (V_r \uplus V_c, E)$, the required edges E_R, and the parameter ρ, the algorithm* StarBicoloringRestricted *computes a star bicoloring Φ of G when restricted to E_R in $\mathcal{O}(|V_r \uplus V_c| \cdot \Delta^3(V_r \uplus V_c))$.*

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

Algorithm 3.6: Determine a vertex of $G[E_R]$ with the maximum degree of the row and column vertices weighted by ρ.

```
procedure getNextVertex($G = (V_r \uplus V_c, E), E_R, \rho$)
    if $\Delta(V_r, G[E_R]) > \rho \cdot \Delta(V_c, G[E_R])$ then
        return $v_r \in V_r : d(v_r, G[E_R]) = \Delta(V_r, G[E_R])$
    else
        return $v_c \in V_c : d(v_c, G[E_R]) = \Delta(V_c, G[E_R])$
```

Proof. The first condition in Def. 2.12 requires separate colors for row and column vertices—except color zero. This is automatically enforced by adding the highest color assigned to a row vertex to the nonzero colors of the column vertices in line 18. Hence, the colors for the row and column vertices are distinguishable. During the coloring phase the row and column vertices can be colored identically, because no nonzero color assigned to a row vertex is ever compared to a nonzero color assigned to a column vertex.

As long as the edge set E'_R is not empty, the next vertex v is chosen and colored. After selecting the vertex v, the algorithm removes all incident edges to v in E'_R. Thus, for all edges in E'_R at least one incident vertex is colored with a nonzero color; and the condition 2 in Def. 2.12 is not violated.

For each path (v, w, x) with $\Phi(w) = 0$ and $(v, w) \in E_R$ or $(w, x) \in E_R$, the vertices v and x must be colored differently, i.e., $\Phi(v) \neq \Phi(x)$. Depending on whether the vertex v is a row or column vertex, i.e., $v \in V_r$ or $v \in V_c$, either the condition 3a or the condition 3b is applicable. In lines 8–11, the already assigned nonzero colors to distance-2 neighbors x of vertex v are forbidden if the vertex w is either colored with the color zero or not yet colored. Thus, condition 3a and condition 3b cannot be violated.

In consequence of the condition 3c, for each path (v, w, x, y) with $(w, x) \in E_R$, $\Phi(w) > 0$, and $\Phi(x) > 0$, the vertices v and x or the vertices w and y must be colored differently, i.e., $\Phi(v) \neq \Phi(x)$ or $\Phi(w) \neq \Phi(y)$. These paths are considered in lines 13–16. If the vertices w, x, and y are already colored with nonzero colors and the vertices w and y are colored identically, i.e., $\Phi(w) = \Phi(y)$, the color assigned to vertex x is forbidden for vertex v. There is no need to check paths where the vertex y is colored with the color zero, i.e., $\Phi(y) = 0$, because vertex v can be assigned an arbitrary nonzero color due to $\Phi(w) > 0$. Thus, the condition 3c is valid.

At most all vertices in $V_r \uplus V_c$ are visited in line 5. For every vertex v all $N_2(v, G)$-paths (v, w, x) and $N_3(v, G)$-paths (v, w, x, y) are considered. Thus, the complexity of this algorithm is $\mathcal{O}(|V_r \uplus V_c| \cdot \Delta^3(V_r \uplus V_c))$. □

Results

In this section, we carry out three comparisons to assess the star bicoloring and especially our algorithm StarBicoloringRestricted. First, distance-2 colorings determined by D2ColoringRestricted (Alg. 2.1) are compared to star bicol-

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

Matrix	n	NNZ	D	$\lfloor n/32 \rfloor$	$\lfloor n/16 \rfloor$	$\lfloor n/8 \rfloor$	$\Delta(V_r)$	$\Delta(V_c)$
685_bus	685	3,249	0.69	85	42	21	13	13
crystm01	4,875	105,339	0.44	609	304	152	27	27
hor_131	434	4,182	2.22	54	27	13	32	26
memplus	17,758	99,147	0.03	2219	1109	554	353	353
msc00726	726	34,518	6.55	90	45	22	88	88
nos3	960	15,844	1.72	120	60	30	18	18
nos7	729	4,617	0.87	91	45	22	7	7
orsirr_2	886	5,970	0.76	110	55	27	14	14
poisson3Da	13,514	352,762	0.19	1689	844	422	110	110

Table 3.8: Properties of the chosen $n \times n$ matrices from the University of Florida Sparse Matrix Collection [23]: number of columns n, number of nonzero elements NNZ, density (in %) D (NNZ$/n^2 \cdot 100$), and block sizes $\lfloor n/32 \rfloor$, $\lfloor n/16 \rfloor$, $\lfloor n/8 \rfloor$ as well as maximum degree of row vertices $\Delta(V_r)$ and column vertices $\Delta(V_c)$ of the associated bipartite graph.

orings computed by StarBicoloringRestricted. When the full Jacobian computation is considered, the required elements for both algorithms are the nonzero elements of the Jacobian matrix. Second, we compare the results computed by our algorithm StarBicoloringRestricted with the results computed by the recent software package ColPack [28]. We show that our algorithm is comparable in the number of colors. Finally, for the partial Jacobian computation, we compare restricted distance-2 colorings and restricted star bicolorings for the nonzero elements of block diagonals with different block sizes. The chosen matrices are taken from the University of Florida Sparse Matrix Collection [23]. These matrices are also used in the next chapter to consider partial Jacobian computation for preconditioning. The main properties of the matrices, which are the number of columns, the number of nonzero elements, and the density, are given in Table 3.8. Further properties are the block sizes $k = \lfloor n/32 \rfloor$, $k = \lfloor n/16 \rfloor$, and $k = \lfloor n/8 \rfloor$ as well as the maximum degree of the row vertices $\Delta(V_r)$ and the column vertices $\Delta(V_c)$ of the associated bipartite graph $G = (V_r \uplus V_c, E)$.

Besides using a star bicoloring or a restricted star bicoloring, the orderings in which the vertices are colored have a significant impact on the result. Due to the greedy characteristic of the algorithm, these orderings result in a different number of colors. This topic goes beyond the scope of this thesis. More details for full Jacobian computation and references to other articles are presented in [27]. For the partial Jacobian computation, in particular, for the main diagonal elements, the vertex orderings are addressed in [40]. In theory, a minimal distance-2 coloring cannot need fewer colors than a minimal star bicoloring, because every distance-2 coloring is also a star bicoloring. If a distance-2 coloring is better than a star bicoloring, another vertex ordering for the two-sided coloring algorithm should be employed.

Before evaluating the coloring heuristics in terms of the number of colors, we discuss the choice of the parameter ρ for the algorithm StarBicoloringRestricted. This parameter has an impact on the vertex ordering and thus on the number of colors. For

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

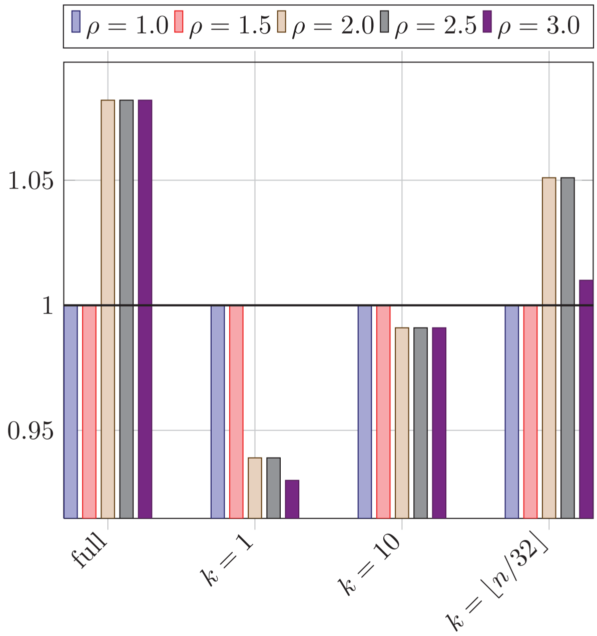


Figure 3.8: Accumulated number of colors determined by the coloring heuristic StarBicoloringRestricted for varied parameter ρ normalized to $\rho = 1.0$. The test matrices are the same as in Table 3.9.

the full Jacobian computation, $E_R = E$, and for the partial Jacobian computation with the required elements in the block diagonals, in particular, blkDiag(A, k) with $k = 1$, $k = 10$, and $k = \lfloor n/32 \rfloor$, (restricted) star bicolorings are computed by StarBicoloringRestricted. The parameter ρ is varied from 1.0 to 3.0 in steps of 0.5. For the test matrices, the number of colors for different required elements and choices of ρ are determined but these values are not presented in detail. Instead, the numbers of colors for all matrices are accumulated for each combination of required elements and ρ and afterwards normalized to $\rho = 1.0$ for the same required elements. These results are given in Fig. 3.8. The parameter ρ has been assessed in [40] for the full Jacobian computation with different test matrices. Although the parameter $\rho = 3.0$ has been the best choice in that work, we cannot identify that this choice is beneficial for the considered required elements and test matrices in this thesis. If we consider the full and partial Jacobian computation with $k = \lfloor n/32 \rfloor$ for the given matrices, the choices of $\rho = 1.0$ and $\rho = 1.5$ lead to fewer colors than the remaining values. For the partial Jacobian computation with $k = 1$ and $k = 10$, the choices of $\rho = 1.0$ and $\rho = 1.5$ result in more colors than the remaining choices of ρ. Thus, for every selection of the required elements and even for each matrix, an individual choice of ρ can be beneficial. To avoid excessive distinguishing between different choices for parameter ρ, we set the parameter to $\rho = 1.5$ for the rest of this thesis. In general, we cannot suggest a specific parameter ρ for all matrices. Therefore, the default value for parameter ρ is given as $\rho = 1.0$ if there is no specific ρ evaluated for the considered matrix.

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

	p_{d2}		p_{sb}	
Matrix	NO	ORD	SBRes	ColPack
685_bus	14	13	16	13
crystm01	27	27	34	27
hor_131	34	32	26	32
memplus	353	353	89	205
msc00726	160	135	152	135
nos3	18	18	30	18
nos7	12	12	11	11
orsirr_2	18	14	16	17
poisson3Da	112	110	79	78

Table 3.9: Number of colors p_{d2} to determine the Jacobian matrix computed by function D2ColoringRestricted for natural ordering (NO) and smallest number among orderings (ORD) NO, LFO, and IDO; and p_{sb} computed by function StarBicoloringRestricted (SBRes) with $\rho = 1.5$ and the minimum of several two-sided coloring algorithms included in ColPack.

The number of colors p_{d2} computed by function D2ColoringRestricted, the number of color p_{sb} computed by StarBicoloringRestricted with $\rho = 1.5$, and the number of color p_{sb} computed by ColPack are compared. For the algorithm D2ColoringRestricted different number of colors p_{d2} are given in Table 3.9: the number of colors computed using the natural ordering (NO) and the smallest number of colors determined using the orderings NO, largest-first ordering (LFO), and incidence-degree ordering (IDO). The function StarBicoloringRestricted does not use an explicit ordering, but an ordering by employing getNextVertex. For comparing the results of StarBicoloringRestricted to ColPack, different two-sided coloring algorithms included in ColPack are employed: ImplicitCoveringStarBicoloring, ExplicitCoveringStarBicoloring, ExplicitCoveringModifiedStarBicoloring, and ImplicitCoveringGreedyStarBicoloring. Furthermore, also different vertex orderings are assessed: NO, LFO, dynamic largest first-ordering (DLFO), smallest-last ordering (SLO), and random ordering. For every matrix, the smallest number of colors for all combinations of coloring algorithms and vertex orderings is given. That is, a lot of different coloring and ordering strategies of ColPack are compared to one coloring and one ordering strategy implemented in our algorithm StarBicoloringRestricted.

For the matrices memplus, poisson3Da, hor_131, and nos7, a computed star bicoloring is better than the distance-2 colorings. The distance-2 colorings are determined by using the vertex orderings NO, LFO, and IDO. For the matrices poisson3Da and memplus, the number of colors are respectively reduced by 28% and 75%. Thus, significant savings in the number of colors are possible by using a star bicoloring algorithm. We compare the two-sided coloring algorithm StarBicoloringRestricted and the algorithms included in ColPack. For the matrices memplus, hor_131, and orsirr_2, our algorithm is better; for the matrix nos7, there is no difference; and for the remaining matrices, at least one algorithm in ColPack computes smaller numbers of

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

Matrix	$k=1$		$k=10$		$k=\lfloor n/32 \rfloor$	
	p_{d2}	p_{sb}	p_{d2}	p_{sb}	p_{d2}	p_{sb}
685_bus	6	7	12	11	12	13
crystm01	8	9	13	14	22	24
hor_131	15	15	27	18	28	19
memplus	22	23	37	40	146	88
msc00726	25	24	61	56	82	73
nos3	10	10	12	21	18	21
nos7	4	4	10	10	11	10
orsirr_2	5	6	12	12	14	14
poisson3Da	16	17	28	31	42	68

Table 3.10: Number of colors p_{d2} and p_{sb} to determine the required elements, specified by blkDiag(A, k), computed by function D2ColoringRestricted and function StarBicoloringRestricted with $\rho = 1.5$.

colors. Recall that, depending on the vertex ordering, the result of a star bicoloring algorithm can also be a distance-2 coloring. In summary, there are matrices for which two-sided coloring algorithms determine a better result than the distance-2 algorithm. In practice, there are also matrices, for which the distance-2 algorithm results in the smaller number of colors. Currently the best option is to try different vertex orderings and coloring algorithms before computing a Jacobian matrix.

After showing that the star bicoloring can be beneficial for the full Jacobian computation and that our algorithm is comparable to the two-sided coloring algorithms of ColPack, we consider required elements included in the $k \times k$ blocks of the block diagonal of $n \times n$ Jacobian matrices A. These elements are specified by blkDiag(A, k). By determining the colorings for different block sizes—$k = 1$, $k = 10$, and $k = \lfloor n/32 \rfloor$—and employing solely the natural ordering, there are some matrices for which the number of colors of the restricted star bicoloring is smaller than the number of colors of the restricted distance-2 coloring. The results are given in Table 3.10. For $k = 1$, the star bicoloring is better than the distance-2 coloring only for the matrix msc00726. The difference is one color (25 to 24) or 4%. For $k = 10$, there are three matrices with a better star bicoloring. The reduction for 685_bus as well as msc00726 is by 8% and for hor_131 by 33%. For $k = \lfloor n/32 \rfloor$, there are four matrices with a better star bicoloring: memplus with a reduction by 40%, hor_131 by 32%, msc00726 by 11%, and nos7 by 9%. With a larger block size k, it is more likely that the star bicoloring is better than the distance-2 coloring. Overall, the reductions are between 4% and 40%. That is, there are combinations of matrices and block sizes for which a star bicoloring should be chosen.

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

3.4 Are two-sided colorings better than one-sided colorings?

In this section, we evaluate for different matrix structures if these may benefit from (restricted) star bicolorings compared to minimal (restricted) distance-2 colorings measured in the number of colors. In the first part, partial Jacobian computation for general Jacobian matrices is considered where a block diagonal specifies the required nonzero elements. We investigate if the star bicoloring reduces the number of colors compared to a distance-2 coloring. The special case of the main diagonal is considered in more detail. Keep in mind that in this section exact colorings are considered instead of non-minimal results computed by greedy coloring heuristics. In the second part, we look at colorings concerning regular grids for full and partial Jacobian computation. First, we assess if a minimal star bicoloring needs less colors than a minimal distance-2 coloring for full Jacobian computation. Thereafter, we study this also for the partial Jacobian computation.

3.4.1 Main diagonal

For partial Jacobian computation, the numbers of colors for minimal distance-2 colorings—rows and columns—and for a minimal star bicoloring are compared when restricted to the main diagonal. The bipartite graph $G = (V_r \uplus V_c, E)$ with $|V_r| = |V_c|$ represents the square Jacobian matrix. Suppose that there are no zero elements in the main diagonal. We show that a minimal star bicoloring needs the same number of colors as a minimal distance-2 coloring, $\chi_{\mathrm{d2}} = \chi_{\mathrm{sb}}$, when restricted to the diagonal elements, i.e., $E_D = \{(r_i, c_i) \in E \colon 1 \leq i \leq |V_r|\}$. For the computation of $E_R = E_D$, let χ_c and χ_r denote the smallest numbers of colors for distance-2 colorings of G on V_c and V_r, respectively. In the following theorem, we prove that $\chi_c = \chi_r$ holds.

Theorem 3.7. *The mapping $\Phi_r \colon V_r \to \{0, \dots, p_{d2}\}$ is a distance-2 coloring of G when restricted to E_D, iff $\Phi_c \colon V_c \to \{0, \dots, p_{d2}\}$ is a distance-2 coloring of G when restricted to E_D with $\Phi_r(r_i) = \Phi_c(c_i)$, $r_i \in V_r$, $c_i \in V_c$, and $1 \leq i \leq |V_r|$.*

Proof. We show that the vertices $c_i, c_j \in V_c$ are distance-2 neighbors, iff a vertex $r_i \in V_r$ is a distance-2 neighbor of a vertex $r_j \in V_r$. If c_i and c_j are distance-2 neighbors, there is a path (c_i, r_j, c_j) with $(r_j, c_j) \in E_D$ or a path (c_i, r_i, c_j) with $(r_i, c_i) \in E_D$. The three possible path combinations are given in Fig. 3.9. From Def. 2.10 follows that the distance-2 neighbors c_i and c_j are colored differently, i.e., $\Phi_c(c_i) \neq \Phi_c(c_j)$.

Because of the existence of the edges (r_j, c_i) or (r_i, c_j) and the diagonal elements in E_D, there are the paths (r_i, c_i, r_j) and (r_i, c_j, r_j), respectively. Thus, the vertices r_i and r_j are also distance-2 neighbors and $\Phi_r(r_i) \neq \Phi_r(r_j)$ follows directly. □

As a result, the distance-2 coloring of G on V_r when restricted to E_D can also be applied as a distance-2 coloring of G on V_c when restricted to E_D. Thus, there is no difference in the minimal number of colors for both colorings, i.e., $\chi_c = \chi_r$.

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

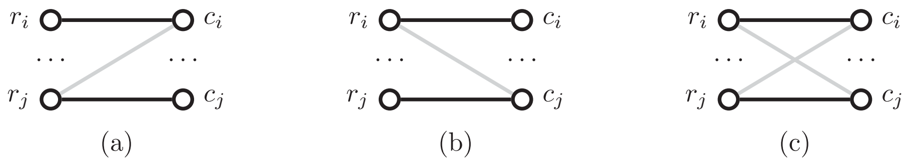


Figure 3.9: Distance-2 neighbors $r_i, r_j \in V_r$ as well as $c_i, c_j \in V_c$ connected by paths (a) (r_i, c_i, r_j) or (c_i, r_j, c_j), (b) (r_i, c_j, r_j) or (c_i, r_i, c_j), and (c) (r_i, c_i, r_j) and (r_i, c_j, r_j) or (c_i, r_j, c_j) and (c_i, r_i, c_j).

For a minimal distance-2 coloring of G—either on V_c or on V_r—when restricted to E_D, let χ_{d2} denote the smallest number of colors. For a minimal star bicoloring of G when restricted to E_D, let χ_{sb} denote the number of colors. Let p_{d2} and p_{sb} denote the numbers of colors for distance-2 colorings of G and for star bicolorings of G, respectively. In the following lemmas, we prove that $\chi_{\text{sb}} \geq p_{\text{d2}}$ and $\chi_{\text{d2}} \geq p_{\text{sb}}$. Using these results, the relation $\chi_{\text{d2}} = \chi_{\text{sb}}$ is shown.

Lemma 3.8. *Given a mapping $\Phi_{sb} \colon [V_r \uplus V_c] \to \{0, 1, \ldots, \chi_{sb}\}$ which is a mimimal star bicoloring of G when restricted to E_D with the smallest number of colors χ_{sb}, there is a mapping $\Phi_c \colon V_c \to \{0, 1, \ldots, p_{d2}\}$ which is a distance-2 coloring of G when restricted to E_D with the number of colors p_{d2} not exceeding χ_{sb}. That is, the property $\chi_{sb} \geq p_{d2}$ holds.*

The proof for the Lem. 3.8 is given in Appendix A.3.

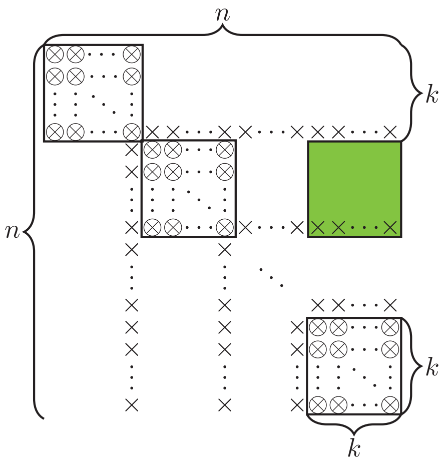
Lemma 3.9. *Let the mapping $\Phi_c \colon V_c \to \{0, 1, \ldots, \chi_{d2}\}$ be a minimal distance-2 coloring of G when restricted to E_D with the smallest number of colors χ_{d2}. This mapping is also a star bicoloring $\Phi_{sb} \colon [V_r \uplus V_c] \to \{0, 1, \ldots, p_{sb} = \chi_{d2}\}$ of G when restricted to E_D with $\Phi_{sb}(r_i) = 0$ and $\Phi_{sb}(c_i) = \Phi_c(c_i), 1 \leq i \leq |V_r|$. That is, the property $\chi_{d2} \geq p_{sb}$ holds.*

Proof. The row vertices r_i are colored with the color zero, i.e., $\Phi_{\text{sb}}(r_i) = 0, 1 \leq i \leq |V_r|$. Thus, the conditions 1, 3b, and 3c of Def. 2.12 cannot be violated. The condition 2 of Def. 2.12 holds due to condition 1 of Def. 2.10. As a consequence, solely the condition 3a is a candidate for a violation. This condition is identical to condition 2 in Def. 2.10. Thus, every minimal restricted distance-2 coloring is also a restricted star bicoloring. □

Theorem 3.10. *From Lem. 3.8 with $\chi_{sb} \geq p_{d2} \geq \chi_{d2}$ and Lem. 3.9 with $\chi_{d2} \geq p_{sb} \geq \chi_{sb}$, the statement $\chi_{sb} = \chi_{d2}$ follows directly, i.e., the smallest number of colors is the same for a minimal star bicoloring and a minimal distance-2 coloring both restricted to the edges corresponding to the diagonal elements of a Jacobian matrix.*

Thus, the number of colors to determine the diagonal nonzero elements of the Jacobian matrix is the same for a minimal restricted distance-2 coloring and a minimal restricted star bicoloring.

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.



(a)

k	χ_{d2}	p_{sb}
2	1000	1000
4	1000	502
5	1000	403
8	1000	256
10	1000	208

(b)

Figure 3.10: (a) Sparsity pattern of $n \times n$ Jacobian matrix A with $k \times k$ diagonal blocks which are completely filled with required elements $\otimes$. The non-required elements are indicated by symbol $\times$. (b) Table with (smallest) number of colors χ_{d2} and p_{sb} for A with $n = 1000$ and varying k.

3.4.2 Block diagonal

With the Theorem 3.10 in mind, one could assume that there is no conceptual difference in the number of colors between the block diagonal and, its special case, the main diagonal. We show by a counterexample that this assumption is wrong. Our example is an $n \times n$ Jacobian matrix with $k \times k$ blocks placed next to each other on the main diagonal. We assume that these $k \times k$ blocks are completely filled with required elements. In each kth row and column, non-required elements are placed to the right and below of the blocks. We suppose that n/k is even. An illustrating example is given in Fig. 3.10(a). Solely blocks with $k \geq 2$ are considered in the following, because the special case, $k = 1$, has already been covered in this section. First, the minimal number of colors of the restricted distance-2 coloring, χ_{d2}, is given. Afterwards, the number of colors of a restricted star bicoloring, p_{sb}, is described.

We explain the number of colors for a minimal distance-2 coloring when restricted to the block diagonal elements by considering the structure of the Jacobian matrix A instead of a corresponding graph. The colors are composed as follows: Consider two columns with required elements in the same $k \times k$ block. These columns cannot be combined to a column group, because both have required elements in more than one row. Therefore, for each block k colors are required. In addition, the non-required elements outside the blocks avoid the combination of columns with required elements in different blocks. While combining two columns, the non-required elements in the intersecting region of two blocks would sum up required elements inside the blocks. Let us consider two blocks on the diagonal in Fig. 3.10(a) with the intersecting region indicated by a green background padding. In the row $2k$, there are non-required elements from column $2k + 1$ to n. If a column a_i with required elements in the

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

second block and a column a_j with required elements in the last block are combined to a column group, the required element $a_{2k,i}$ in column a_i and the non-required element $a_{2k,j}$ in column a_j would be summed up. Thus, there is no pair of columns which can be combined to a column group and the smallest number of colors for a distance-2 coloring is the number of columns, $\chi_{\text{d2}} = n$. Combining rows to groups, instead, is analogous due to the symmetric structure of the matrix.

There is a star bicoloring when restricted to the block diagonal elements which needs $p_{\text{sb}} = 2 \cdot n/k + k - 2$ colors: The rows $k, 2k, \ldots, (n/k - 1) \cdot k$, which contain non-required elements, are determined row-wise without combining them to a row group. Therefore, $n/k - 1$ colors are required. With this prerequisite, the columns $1, k+1, 2k+1, \ldots, (n/k-1) \cdot k + 1$ can be combined to a column group and the columns $2, k+2, 2k+2, \ldots, (n/k-1) \cdot k + 2$ to another column group. Thus, all columns, except the last column in each block, can be combined to $k - 1$ column groups. The required elements in the last column of each block are left over. These columns must be separately determined. Otherwise, required elements in two of these columns would be summed up. To determine these required elements, n/k column groups are needed. In summary, $n/k - 1$ row groups and $(k-1) + (n/k)$ columns groups are required and, hence, we need $p_{\text{sb}} = 2 \cdot n/k + k - 2$ colors.

Rather than using $\chi_{\text{d2}} = n$ colors for determining the required elements, a restricted star bicoloring reduces the number of colors to $p_{\text{sb}} = 2 \cdot n/k + k - 2$. By increasing the block size k, the number of colors for the restricted star bicoloring is reduced. We consider the number of colors for an $n \times n$ Jacobian matrix and vary the block size k to illustrate this behavior: For $k = 2$, the restricted star bicoloring needs as much colors as the restricted distance-2 coloring; and for $k \geq 3$, the restricted star bicoloring needs fewer colors, because the factor n/k is reduced. For a 1000×1000 Jacobian matrix, the number of colors χ_{d2} and p_{sb} are given in Table 3.10(b) for block size k from 2 to 10 if n/k is even. In summary, the star bicoloring needs less colors than the minimal distance-2 coloring for block size $k \geq 3$.

3.4.3 Stencils

Considering stencil-based computations, we assess whether a (restricted) star bicoloring potentially reduces the number of colors compared to a minimal (restricted) distance-2 coloring. Lower bounds for the number of colors needed by the (restricted) star bicoloring are introduced. These bounds are compared to the chromatic numbers of minimal (restricted) distance-2 colorings, χ_{d2}. The gap between the lower bounds and χ_{d2} indicates whether a minimal (restricted) star bicoloring potentially reduces the number of colors. First, a lower bound for minimal colorings for the full Jacobian computation is considered. Thereafter, we move on to the partial Jacobian computation.

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

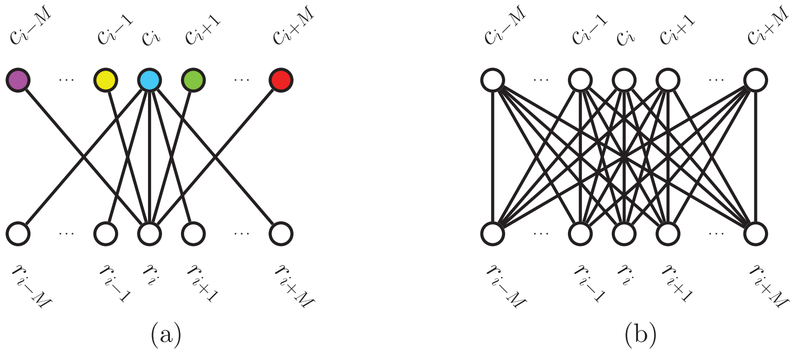


Figure 3.11: Bipartite subgraphs induced by the grid points of the five-point stencil $\mathcal{N}_{5\text{pt}}$ with edges incident to (a) the center $i = \psi(m, n)$ and (b) to all grid points. The numbering scheme of the one-dimensional index ψ is the natural ordering.

Full Jacobian computation

The bipartite graph $G = (V_r \uplus V_c, E)$ is employed for stencil-based computations on $M \times N$ grids [13, 14] to address star bicolorings. To this end, we once more use a numbering scheme to map a two-dimensional index of a grid point to a one-dimensional index. To every grid point (m, n) with $i = \psi(m, n)$, a row vertex r_i and a column vertex c_i are associated leading to the vertex sets

$$V_r = \{r_i \mid 1 \leq i \leq MN\} \quad \text{and} \quad V_c = \{c_i \mid 1 \leq i \leq MN\}.$$

There are the edges $(r_i, c_j) \in E$ and $(r_j, c_i) \in E$ if the grid point $j = \psi(k, l)$ belongs to the stencil with center $i = \psi(m, n)$, i.e.,

$$(r_i, c_j), (r_j, c_i) \in E \iff i = \psi(m, n), j = \psi(k, l), \text{ and } (k, l) \in \mathcal{N}(m, n). \tag{3.3}$$

The five-point stencil $\mathcal{N}_{5\text{pt}}$ is used as an illustrating example for the minimal number of colors. The bipartite subgraph induced by the grid points of this stencil is depicted in Fig. 3.11(a). The divide-and-conquer algorithm colorVSepAll using separators computes a minimal distance-2 coloring with the smallest number of colors, χ_{d2}. We examine whether there is a star bicoloring with a smaller number of colors. Therefore, another bipartite graph is considered as illustrating example. The vertices correspond to grid points which belong to a stencil whose center is not at the boundary of the grid. For each pair of distance-2 neighbors the connecting edges are added. This graph is given in Fig. 3.11(b).

Five colors are required for the minimal distance-2 coloring, because the grid points are pairwise structurally non-orthogonal. That is, all column vertices are pairwise distance-2 neighbors and all row vertices are pairwise distance-2 neighbors. The row vertices as well as the column vertices must be colored differently depending on whether the vertices in V_r or V_c are colored. A minimal coloring for the column vertices is depicted in Fig. 3.11(a). Due to the condition 2 of the star bicoloring in

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

Def. 2.8, every edge must be incident to a vertex colored with a nonzero color, i.e., either $\Phi(r_j) \neq 0$ or $\Phi(c_j) \neq 0, \forall j \in \{i-M, i-1, i, i+1, i+M\}$. Recall that all column vertices are distance-2 neighbors and all row vertices are distance-2 neighbors. The vertices can be colored as follows:

- Assign nonzero colors to the column vertices and the color zero to the row vertices. This coloring is not only a star bicoloring, but also a distance-2 coloring on the column vertices.
- Assign nonzero colors to the row vertices and the color zero to the column vertices. This coloring is also a distance-2 coloring on the row vertices.
- Assign nonzero colors to the vertices r_i and c_i and, in addition, assign nonzero colors either to vertex r_j or c_j for the remaining indices $j \in \{i-M, i-1, i+1, i+M\}$. There are five distance-2 neighbors, either row or column vertices, and a vertex in the opposite set which must be colored with a different color. That is, nonzero colors are assigned to at least six vertices. Hence, six colors are needed.

There is no further star bicoloring due to the following reason: It is not possible to color the vertices by assigning either the color zero to the vertex r_i and to a vertex $c_j, j \in \{i-M, i-1, i+1, i+M\}$ or the color zero to the vertex c_i and a vertex $r_j, j \in \{i-M, i-1, i+1, i+M\}$, because at least one edge would not be incident to a vertex colored with a nonzero color. Thus, the only possibility to color the row and column vertices with five nonzero colors is a minimal distance-2 coloring which is also a minimal star bicoloring.

For the stencil $\mathcal{N}_{\text{5pt}}$, the number of colors and the number of grid points are identical. We generalize this observation for arbitrary stencil. Recall that the center is part of the neighborship relation, i.e., $(m,n) \in \mathcal{N}(m,n)$.

Lemma 3.11. *Given a regular grid G and a stencil $\mathcal{N}$, the number of colors of a minimal star bicoloring is at least the number of grid points of the stencil, $|\mathcal{N}|$.*

Proof. We consider the bipartite graph associated to the given stencil $\mathcal{N}$. At least one incident vertex of every edge $(r_i, c_i), 1 \leq i \leq |V_r|$, must be colored with a nonzero color. As a consequence of Def. 2.14, the grid points in stencil $\mathcal{N}$ are pairwise structurally non-orthogonal. That is, due to (3.3), if the vertices c_j and c_k are distance-2 neighbors of vertex c_i, the vertex c_k is also a distance-2 neighbor of vertex c_j. The corresponding row vertices r_i, r_j, and r_k are also pairwise distance-2 neighbors. If all column vertices as well as all row vertices in the subgraph corresponding to a stencil are pairwise distance-2 neighbors and the edges $(r_i, c_i) \in E, 1 \leq i \leq |V_r|$ exist, at least $|\mathcal{N}|$ grid points must be colored with different nonzero colors. □

The coloring behavior for the six-point stencil $\mathcal{N}_{\text{6pt}}$ and the nine-point stencil $\mathcal{N}_{\text{9pt}}$ is quite similar. However, as already described in [14, 29], a minimal distance-2 coloring consists of seven or ten colors, respectively. For both stencils, there is one extra color compared to the number of grid points. All remaining, not colored vertices obtain this seventh or tenth color. Here, there is a difference of one color between the

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

	$\mathcal{N}_{5,1\mathrm{pt}}$	$\mathcal{N}_{5,3\mathrm{pt}}$	$\mathcal{N}_{6,1\mathrm{pt}}$	$\mathcal{N}_{6,3\mathrm{pt}}$	$\mathcal{N}_{9,1\mathrm{pt}}$	$\mathcal{N}_{9,3\mathrm{pt}}$	$\mathcal{N}_{9,5\mathrm{pt}}$
$\lvert\mathcal{N}_{\mathrm{req}}\rvert$	1	3	1	3	1	3	5
χ_{d2}	2	4	3	4	3	6	8
χ_{sb}	2	4	3	4	3	–	–

Table 3.11: Lower bound $\lvert\mathcal{N}_{\mathrm{req}}\rvert$ for a minimal star bicoloring; the number of colors χ_{d2} for a minimal distance-2 coloring and χ_{sb} for a minimal star bicoloring on an $M \times N$ grid. Symbol – denotes unknown number of colorings due to a runtime exceeding 5 days.

minimal coloring and the number of grid points. However, it is most unlikely that this extra color can be exploited using a star bicoloring.

We conclude that the star bicoloring cannot be beneficial if the number of colors of the minimal distance-2 coloring is the same as the number of grid points in the stencil. Hence, the distance-2 coloring is a minimal coloring for the stencil $\mathcal{N}_{5\mathrm{pt}}$. If there is a gap between the chromatic number χ_{d2} for the distance-2 coloring and the lower bound with the number of grid points, there can be a better minimal star bicoloring. For the stencils $\mathcal{N}_{6\mathrm{pt}}$ and $\mathcal{N}_{9\mathrm{pt}}$, where the gap is only one color, a reduction by the star bicoloring is most unlikely. Consideration of further stencils in detail is beyond the scope of this thesis.

Partial Jacobian computation

For the partial Jacobian computation, combinations of an original stencil $\mathcal{N}_{\mathrm{org}}$ and a stencil $\mathcal{N}_{\mathrm{req}}$ are considered. Recall that the stencil $\mathcal{N}_{\mathrm{req}}$ specifies the required elements of the Jacobian matrix. Minimal distance-2 colorings for stencil combinations can be computed by the algorithm colorVSepAll. In the following, we study the lower bound for the minimal number of colors for restricted star bicolorings.

In contrast to the full Jacobian computation, the grid points in stencil $\mathcal{N}_{\mathrm{org}}$ are not necessarily pairwise partially structurally non-orthogonal. Therefore, the number of grid points of stencil $\mathcal{N}_{\mathrm{org}}$, $\lvert\mathcal{N}_{\mathrm{org}}\rvert$, is not a lower bound for a minimal star bicoloring. Instead, we consider the stencil $\mathcal{N}_{\mathrm{req}}$. The grid points of such a stencil are pairwise partially structurally non-orthogonal. Therefore, the number of the grid points, $\lvert\mathcal{N}_{\mathrm{req}}\rvert$, is a lower bound. The lower bound for a star bicoloring is the number of grid points of the stencil $\mathcal{N}_{\mathrm{req}}$ because of Lem. 3.11.

By taking some of the stencil combinations from Sect. 3.1.2, we evaluate whether star bicolorings are better than distance-2 colorings. For these combinations, the lower bound $\lvert\mathcal{N}_{\mathrm{req}}\rvert$ and the number of colors for a minimal distance-2 coloring are given in Table 3.11. There is a smallest width m and a smallest height n so that the minimal number of colors p is identical for all $M \times N$ grids with $M \geq m$ and $N \geq n$. For the stencils $\mathcal{N}_{5,1\mathrm{pt}}$, $\mathcal{N}_{6,1\mathrm{pt}}$, and $\mathcal{N}_{9,1\mathrm{pt}}$, the required elements in the Jacobian matrix are the nonzero elements on the main diagonal. We proved in Theorem 3.10 that the minimal restricted star bicoloring is never better than the minimal restricted distance-2 coloring for this special case. For the stencil combinations with stencil $\mathcal{N}_{\mathrm{req}} =$

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

$\mathcal{N}_{\text{3pt}}$ or $\mathcal{N}_{\text{req}} = \mathcal{N}_{\text{5pt}}$, the gap between the lower bound and the number of colors of a minimal distance-2 coloring is at most three. We use an exhaustive search algorithm to compute minimal star bicolorings. For $\mathcal{N}_{5,\text{3pt}}$ and $\mathcal{N}_{6,\text{3pt}}$, there is no benefit in using a star bicoloring. For $\mathcal{N}_{9,\text{3pt}}$ and $\mathcal{N}_{9,\text{5pt}}$, the runtime of the exhaustive search exceeds several days. Thus, we do not obtain minimal star bicolorings and have no indication whether a two-sided coloring is better.

In summary, the lower bound is the number of grid points in the stencil $\mathcal{N}_{\text{req}}$. For the chosen test matrices, there is no minimal star bicoloring for the considered stencil combinations with a smaller number of colors than a minimal distance-2 coloring. The higher the difference in the number of grid points between $\mathcal{N}_{\text{org}}$ and $\mathcal{N}_{\text{req}}$, a better minimal star bicoloring is more likely.

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

4 Preconditioning using partial Jacobian computation

This chapter addresses the combination of preconditioning techniques for iterative solvers and the partial Jacobian computation. In Sect. 4.1, iterative solvers and preconditioning techniques are described. These techniques are then discussed in the context of automatic differentiation in Sect. 4.2. Using the block diagonal elements of a Jacobian matrix for preconditioning is motivated in Sect. 4.3. Due to limitations of the memory the full Jacobian matrix cannot be computed. Thus, only the sparsity pattern of the Jacobian matrix is known. This pattern is employed to determine a preconditioner. A subset of nonzero elements of the Jacobian matrix is selected, the so-called initially required elements. Next, further nonzero elements are chosen to speed up solving preconditioned systems of linear equations. The potentially required elements enlarge the initially required elements and are determinable with the same coloring. The additionally required elements are a subset of the potentially required elements which do not cause any fill-in element while obtaining the preconditioner. The classification of these nonzero elements is introduced in Sect. 4.4. While these nonzero elements are chosen, the computational effort and the limited memory are taken into account. Algorithms to determine these elements are introduced. The Sect. 4.4 is closed by restricting these elements so that it is beneficial for solving systems of linear equations in parallel. The results of this chapter are summarized in the last section.

4.1 Iterative solvers using preconditioning techniques

In the field of scientific computing, solving systems of linear equations

$$A \cdot x = b \tag{4.1}$$

is an essential ingredient, where the coefficient matrix $A \in \mathbb{R}^{n \times n}$ is a large, regular and sparse Jacobian matrix, the vector $b \in \mathbb{R}^n$ denotes the right-hand side and the vector $x \in \mathbb{R}^n$ is the vector of unknowns. Such systems occur, for example, during the numerical solution of non-linear systems. Rather than computing an exact solution of the sparse linear system (4.1) with a direct solver, in this thesis an iterative nonsymmetric solver is selected to compute an approximative solution of the system. Compared to the direct solver, the iterative solver often uses less operations and memory. Examples for iterative solvers are the generalized minimal residual method (GMRES) [58] and the biconjugate gradients stabilized method (Bi-CGSTAB) [61].

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

In contrast to direct solvers, iterative solvers are matrix-free. Hence, these solvers do not require the full Jacobian matrix, but the Jacobian matrix-vector product $A{\cdot}v$ and the transposed Jacobian matrix-vector product $A^T \cdot v$ are sufficient. Compared to the full Jacobian computation, computing these products with automatic differentiation requires less computational effort and memory because the Jacobian matrix is never explicitly assembled and the number of colors is $p = 1$.

Preconditioning [4, 57] is a technique to speed up solving a system of linear equations, i.e., the preconditioner, a matrix $M \in \mathbb{R}^{n \times n}$, is used to increase the convergence rate of the iterative solver. Furthermore, without preconditioning some ill-conditioned iterative systems may diverge. For the (left) preconditioning, both sides of the system are multiplied on the left by a matrix M^{-1} yielding the system

$$M^{-1}\, A \cdot x = M^{-1}\, b.$$

The preconditioner M is an approximation of the Jacobian matrix A. There are two extreme cases: First, the identity matrix I_n is chosen as preconditioner, i.e., $M = I_n$. In this case, there is no computational effort to determine the matrix M. However, the preconditioned linear system is identical to the original system. Second, the full Jacobian matrix A is chosen as preconditioner, i.e., $M = A$. Here, the computational effort to invert M is very high. Nevertheless, the preconditioned system is solved after the first iteration step. In practice, the preconditioner M is something in between I_n and A, i.e., the choice is a trade-off between increasing the convergence rate of the iterative solver and the computational effort to solve a system with matrix M as coefficient matrix. Most importantly, the preconditioning should not be more computationally expensive than the benefit from an increased convergence rate. The additional computational effort of carrying out the preconditioning arises from

- determining the matrix M,
- solving systems of linear equations with matrix M as coefficient matrix, and/or
- computing matrix-vector products with matrix M.

Hence, it is important that the computational effort to solve these system of linear equations and these matrix-vector products is small. The incomplete LU factorization [45] is an example for a method to determine a preconditioner.

4.2 Preconditioning and automatic differentiation

In numerical simulations, the size of the main memory and also the computational resources are often limiting factors. Therefore, it is often infeasible to store and compute all nonzero elements of a large Jacobian matrix. Instead of the fully assembled Jacobian matrix including all nonzero elements, iterative solvers necessitate only the Jacobian matrix-vector product or the transposed Jacobian matrix-vector product. However, while using preconditioning techniques, a common assumption to determine a preconditioner M is that every element of the Jacobian matrix is accessible. Currently, there is no efficient technique to access nonzero Jacobian elements at specific

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

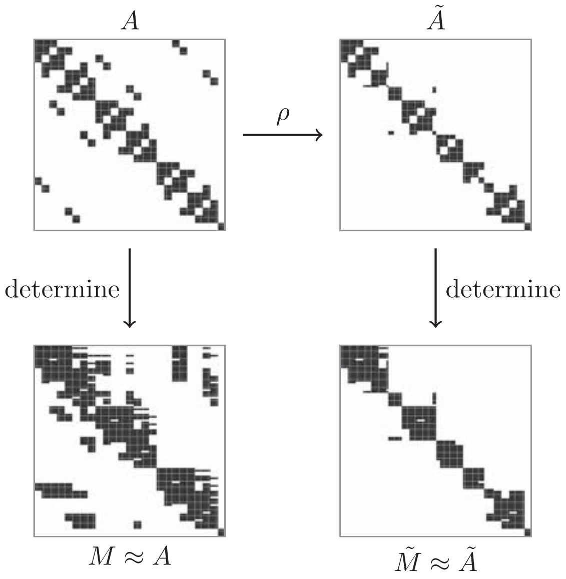


Figure 4.1: Commonly used approach: Determine the preconditioner M with access to all nonzero elements of Jacobian matrix A. New approach: Determine the preconditioner $\tilde{M}$ with access to a subset of A using the sparsification operator ρ.

positions. The efficient access is only provided for (full) rows and columns. The finite differencing enables to estimate all elements of a column. With AD, all elements of a column or a row can be determined. As already described, the nonzero elements of several columns or rows can be obtained as a single linear combination.

Cullum and Tuma [21] proposed a preconditioner M which can be assembled by selecting only a subset of the nonzero elements of a Jacobian matrix A. This selection technique is denoted as *sparsification*. For this purpose, it suffices to solely determine the nonzero elements in this subset. Preliminary work has been presented in [40–42] based on this idea. The partial Jacobian computation is considered in this context; more precisely, a graph coloring is computed to determine only the nonzero elements on the main diagonal of a Jacobian matrix A. In those works, the main diagonal elements are taken as a simple preconditioner, denoted as diagonal scaling. A related approach [59] is the probing which approximates the diagonal elements of the inverse of a sparse matrix A by computing matrix-vector products with A. We do not further consider this approach due to its restriction on the main diagonal elements. Another approach [2] is the computation of a subset of the inverse elements of a sparse matrix. However, in contrast to our approach employing automatic differentiation, the inverse elements are selected and not the nonzero elements of the Jacobian matrix. For this reason, we do not follow this method.

The difference between the common approach and our approach is illustrated in Fig. 4.1. Assuming that all nonzero elements of Jacobian matrix A are accessible is currently the most common way of determining a preconditioner. Hence, the pre-

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

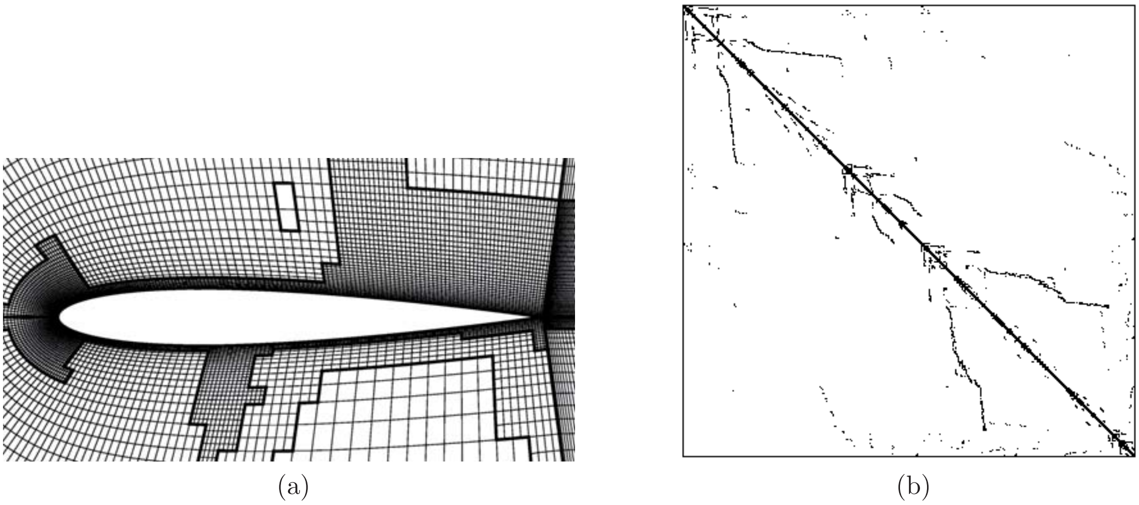

(a) (b)

Figure 4.2: (a) Discretization of a NACA0012 profile in the adaptive flow solver Quadflow [9]. (b) Sparsity pattern of the Jacobian matrix corresponding to (a).

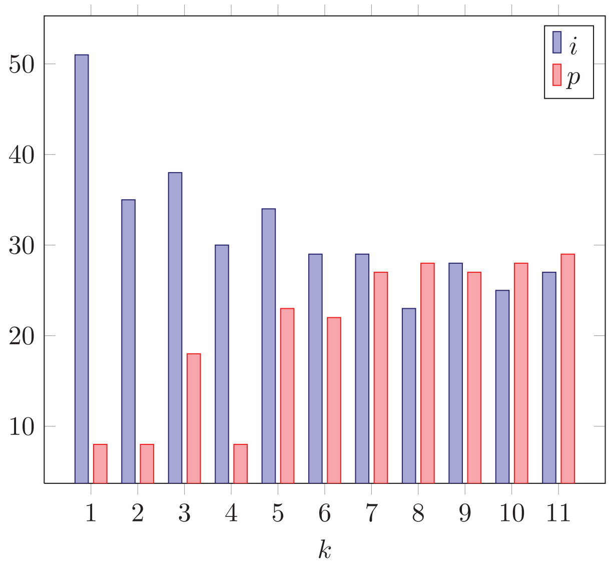
conditioner M is directly approximated from the Jacobian matrix A. Our approach introduces a sparsification operator ρ. This operator selects a subset of the nonzero elements of A and yields a sparsified matrix $\tilde{A}$. The preconditioning techniques determine the preconditioner $\tilde{M}$ from matrix $\tilde{A}$ instead from A. Hence, the preconditioner $\tilde{M}$ contains most likely less nonzero elements than the preconditioner M. Our technique reduces the memory consumption since the number of nonzero elements of $\tilde{A}$ and also of $\tilde{M}$ is decreased. The main questions are how to choose the nonzero elements of the Jacobian matrix $\tilde{A}$ using the sparsification operator ρ and whether the iterative solver better converges employing preconditioner $\tilde{M}$.

4.3 Motivation: Block diagonal

The block diagonal preconditioner is an extension of the diagonal scaling. This preconditioner is generated by accessing $k \times k$ blocks on the diagonal of the Jacobian matrix A. The special case of the diagonal preconditioner, $k = 1$, is evaluated in [41,42]. To assess the block diagonal preconditioner, we vary the block size k and evaluate the number of nonzero elements to be stored, the number of colors to determine these nonzero elements, and the number of iterations needed by the iterative solver. We next consider a non-linear system which arises from a two-dimensional, adaptive discretization around an airfoil, taken from the flow solver Quadflow [9]. The NACA0012 airfoil discretization and the nonzero pattern of the corresponding Jacobian matrix are given in Fig. 4.2.

We want to solve the system of linear equations (4.1) where the coefficient matrix A is a 1600×1600 Jacobian matrix, taken from a Quadflow simulation. Furthermore,

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.



(a)

k	NNZ($\tilde{M}$)	p	i
0	–	–	176
1	1,600	8	51
2	3,200	8	35
3	4,266	18	38
4	6,400	8	30
5	6,400	23	34
6	7,464	22	29
7	8,240	27	29
8	12,800	28	23
9	10,046	27	28
10	11,008	28	25
11	11,694	29	27
12	12,992	28	22
13	13,088	30	25
14	13,864	30	23
1,600	30,720	36	9

(b)

Figure 4.3: Number of iterations i for solving a system of linear equations using GMRES and preconditioner $\tilde{M} = \text{ILU}(\tilde{A}, 0)$ as well as number of colors p determining $\tilde{A}$, specified by blkDiag(A, k), varying the block size k. Number of nonzero elements NNZ$(\tilde{M})$ is included in (b).

the matrix contains 30,720 nonzero elements, and the right-hand side b is the sum of all columns, i.e., the exact solution $x = [1, \ldots, 1]^T$ is known. The iterative solver GMRES is used to solve the system of linear equations in the MATLAB environment. Therefore, we need the Jacobian matrix-vector product $A \cdot v$. As a preconditioner $\tilde{M}$, we apply the incomplete LU factorization with level 0, ILU(0), i.e., no *fill-in elements* occur during the factorization. A fill-in element is a zero element which becomes a nonzero element in the matrices resulting from ILU. In Fig. 4.3(a), the number of colors and the number of iterations of the iterative solver are shown for the block size k varied from $k = 1$ to $k = 11$. The iterative solver stops after iteration i if the relative residual norm is less than a threshold ϵ, i.e.,

$$\frac{||b - Ax_i||_2}{||b||_2} < \epsilon, \quad \epsilon = 10^{-6},$$

where x_i is the solution vector after iteration i. This figure shows that the number of iterations of GMRES tends to be smaller and the number of colors goes up if the block size k increases. In addition to this figure, the number of nonzero elements NNZ$(\tilde{M})$ for $\tilde{M} = \text{ILU}(\tilde{A}, 0)$ and $\tilde{A}$ specified by blkDiag(A, k) are given in Table 4.3(b). In this table, the block size k is varied up to $k = 14$, plus $k = 1{,}600$. The colorings, which are necessary to exploit the structure of matrix $\tilde{A}$, are computed using the partial distance-2 algorithm restricted to the block diagonal. This algorithm is given as D2ColoringRestricted in Alg. 2.1. With increasing block size k, the number

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

of nonzero elements of the preconditioner $\tilde{M}$ increases as well. Due to a possible misalignment of the blocks using different k, the number of nonzero elements is not monotonically increasing (cf. $k = 8$ and $k = 9$). Up to block size $k = 6$, the iteration number i decreases noticeably. In the interval of k between 7 and 14, the number of iterations is between 22 and 29.

At last, two additional cases are considered in Fig. 4.3(b): First, the system of linear equations is solved without preconditioning. The number of iterations is $i = 176$ in comparison to $i = 51$ when selecting the nonzero elements in the main diagonal ($k = 1$) for preconditioning. Second, the system of linear equations is solved using all nonzero elements in A to determine the preconditioner. The number of iterations is reduced to $i = 9$, but the computational effort for $p = 36$ colors and the storage for 30,720 nonzero elements are required. In contrast to this example where the Jacobian matrix contains only 30,720 nonzero elements, Jacobian matrices of real-world applications may be too large to fit into the main memory. As expected, taking all nonzero elements of the Jacobian matrix for the preconditioner is better than the sparsified Jacobian matrix. But, having the computational effort and memory limitations in mind, the approach is superior compared to using no preconditioning at all.

In summary, it seems valuable to pursue this approach, especially if we run out of memory or cannot afford too much computational effort. In the next section, the selection of nonzero elements for preconditioning is introduced more generally.

4.4 Choosing elements for preconditioning

As already mentioned, not all numerical values of the Jacobian matrix are accessible if there is not enough memory to store all nonzero elements. However, we assume that the pattern of this matrix is known due to the problem structure or can be determined with reduced memory consumption compared to storing all nonzero elements. During the sparsification, the nonzero elements which are required for computing the preconditioner M are selected. Furthermore, depending on the method to determine the preconditioner, fill-in elements may occur at positions originally containing a zero element. The crucial issues for choosing the required elements are

- computational effort to compute them,
- availability of memory to store them, and
- obtaining a pattern of them which is beneficial for parallel computing.

The *initially required elements* R_{init} are chosen by the sparsification operator ρ applied to the Jacobian matrix A. More precisely, $R_{\text{init}} \subseteq \text{pat}(A)$ is the set which contains the positions of the initially required elements. Here, $\text{pat}(A)$ is the set containing the positions of all nonzero elements of the Jacobian matrix A. An example for the initial choice are the nonzero elements in the block diagonal. The *potentially required elements* $R_{\text{pot}} \subseteq \text{pat}(A) \setminus R_{\text{init}}$ are nonzero elements of the Jacobian matrix which are not already chosen for preconditioning. The selection of these elements is

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

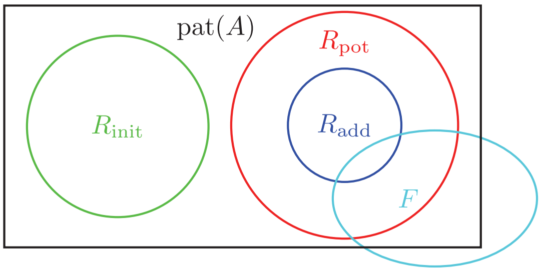


Figure 4.4: Relation between required element sets R_{init}, R_{pot}, and R_{add} and the fill-in set F, induced by R_{init}, concerning the nonzero elements of Jacobian matrix A, pat(A).

Algorithm 4.1: Determine additionally required elements R_{add}

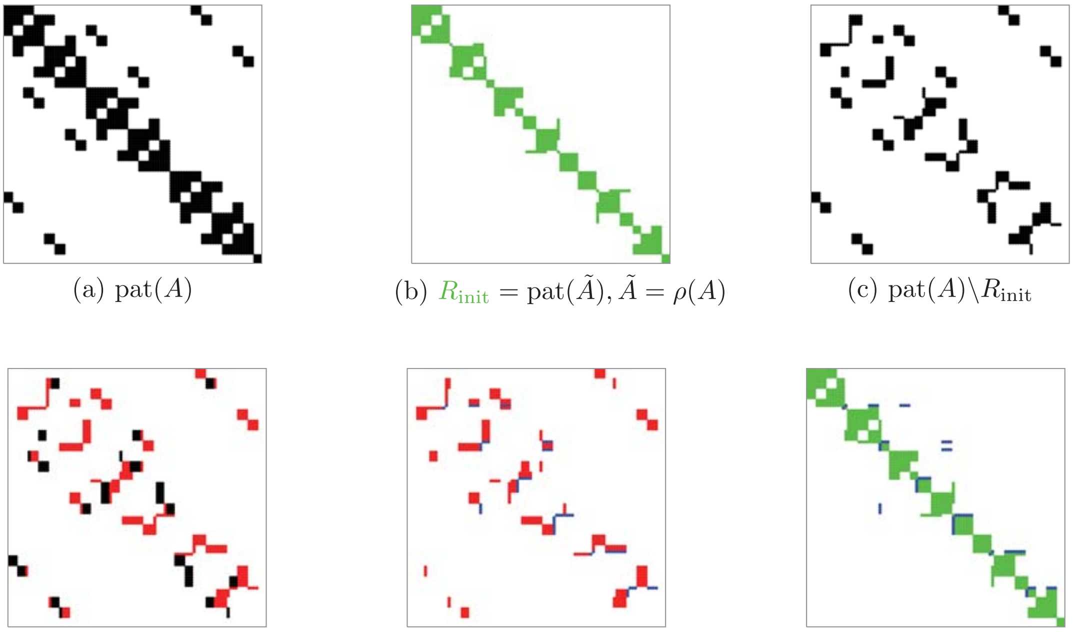
```
require: Sparsity pattern of Jacobian matrix $A$ and level $\ell$ if using ILU
Compute $R_{\text{init}} = \text{pat}(\tilde{A})$ where $\tilde{A} = \rho(A)$
Compute $R_{\text{init}} \uplus F = \text{SILU}(R_{\text{init}}, l)$
Compute coloring $\Phi_{R_{\text{init}}}$
Compute $R_{\text{pot}} \subseteq \text{pat}(A) \backslash R_{\text{init}}$ so that $|\Phi_{R_{\text{init}}}| = |\Phi_{R_{\text{init}} \uplus R_{\text{pot}}}|$
Compute $R_{\text{add}} \subseteq R_{\text{pot}}$ so that $\text{SILU}(R_{\text{init}}, l) \cup R_{\text{add}} = \text{SILU}(R_{\text{init}} \uplus R_{\text{add}}, l)$
```

based on the number of colors required to determine them. The *additionally required elements* $R_{\text{add}} \subseteq R_{\text{pot}}$ are elements which do not lead to any additional fill-in. The set R_{add} forms—together with R_{init}—the set of required elements which are the input for the incomplete LU factorization. The relation among the sets R_{init}, R_{pot}, and R_{add} and the set of fill-in elements F induced by R_{init} is depicted in Fig. 4.4.

In the preceding paragraph, the first part of our new preconditioning approach has been introduced. In this phase, the required elements are selected from the sparsity pattern of the Jacobian matrix without knowing their numerical values. Thereafter, the required elements $R_{\text{init}} \uplus R_{\text{add}}$ are determined by using automatic differentiation, and the preconditioner is computed by carrying out the numerical incomplete LU factorization on these numerical values. In the following, we carry on with gradually describing the determination of the initially, potentially, and additionally required elements, which is schematically given in Alg. 4.1, in more detail:

1. Given the sparsity pattern of a Jacobian matrix A, all nonzero elements of the matrix $\tilde{A} = \rho(A)$ are the initially required elements R_{init} chosen by the sparsification operator ρ. At this point, it is decided which nonzero elements of A are assumed for preconditioning. An illustrating example for the Jacobian matrix A is given in Fig. 4.5(a) and for the matrix $\tilde{A}$ in Fig. 4.5(b). The matrix A is the selection (1:100,1:100) of the matrix given in Fig. 4.2(b). The elements in R_{init} are indicated in green.
2. The initially required elements in R_{init} are the input for the symbolic incomplete LU factorization (SILU) with level ℓ. The result of this factorization is $R_{\text{init}} \uplus F$, i.e., the sets R_{init} and F are disjoint. The set F contains the fill-in elements. The level of the SILU has a strong impact on the number of fill-in elements.

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.



(a) pat(A)

(b) $R_{\text{init}} = \text{pat}(\tilde{A}), \tilde{A} = \rho(A)$

(c) $\text{pat}(A)\backslash R_{\text{init}}$

(d) $R_{\text{pot}} \subseteq \text{pat}(A)\backslash R_{\text{init}}$

(e) $R_{\text{add}} \subseteq R_{\text{pot}}$

(f) $R_{\text{init}} \uplus R_{\text{add}}$

Figure 4.5: Determining the potentially and additionally required elements of the selection (1:100,1:100) of the matrix in Fig. 4.2(b) using ILU(1) and the sparsification is characterized by BLKDIAG$(A, 14)$.

For level $\ell = 0$, the set F is empty and for higher levels it is most likely that fill-in elements occur. Since the nonzero elements in R_{init} form a coefficient matrix of a system of linear equations, the minimal set R_{init} contains at least the diagonal elements.

3. A coloring $\Phi_{R_{\text{init}}}$ for the Jacobian matrix A when restricted to R_{init} is determined by using coloring algorithms corresponding to the graph coloring definitions introduced in Chap. 2. The resulting row and column groups are needed to select the potentially required elements R_{pot} in the following.
4. From the elements in $\text{pat}(A)\backslash R_{\text{init}}$, which are exemplified in Fig. 4.5(c), the potentially required elements R_{pot} are chosen. These elements are nonzero elements of the matrix A which are not already chosen. The elements in R_{pot} are chosen so that the elements in $R_{\text{init}} \uplus R_{\text{pot}}$ can be computed with the same number of colors compared to the elements in R_{init}. In Fig. 4.5(d), the elements in R_{pot} are indicated in red. Selecting extra nonzero elements without increasing the number of colors limits the computational effort of determining these elements with AD.
5. A potentially required element is only added to the additional required elements $R_{\text{add}} \subseteq R_{\text{pot}}$ if this nonzero does not cause an extra fill-in element in addition to the already computed fill-in elements in F. Thus, the result of SILU(R_{init}) combined with the nonzero elements in R_{add} contains nonzero elements at the

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

same positions as the result of SILU($R_{\text{init}} \uplus R_{\text{add}}$). In this step, the number of required elements are restricted due to the limiting memory size. The nonzero elements in R_{add} together with the nonzero elements in R_{init} are respectively indicated in blue and green in Fig. 4.5(f).

After having determined the set R_{add} in the symbolic part, we pass on to the numerical part. Therefore, the numerical values of the nonzero elements $R_{\text{init}} \uplus R_{\text{add}}$ are computed using automatic differentiation where the sparsity pattern is exploited by the coloring $\Phi_{R_{\text{init}}}$ or $\Phi_{R_{\text{init}} \uplus R_{\text{pot}}}$. Afterwards, the numerical incomplete LU factorization (NILU) is performed to obtain a lower triangular matrix L and an upper triangular matrix U for preconditioning. The iterative solver employs the matrices L and U to solve the system of linear equations with coefficient matrix A.

In the following sections, the selection of initially, potentially and additionally required elements is described in more detail. Keep in mind that these required elements are chosen by taking the computational effort and the available memory into account.

4.4.1 Initially required elements

The initially required elements R_{init} result from applying a sparsification operator ρ to the nonzero pattern of a Jacobian matrix A. In this section, these elements are used to compute a preconditioner. The potentially and additionally required elements are not considered. The following two assumptions are made for choosing the sparsification operator ρ. First, the values of the nonzero elements near the main diagonal are more important than nonzero elements farther away. This is motivated by diagonal dominant coefficient matrices. Second, a larger number of nonzero elements usually causes a better preconditioner. In this thesis, the employed sparsification operator ρ, characterized by blkDiag(A, k), yields the nonzero elements in the block diagonal with variable block size k. As a consequence of these assumptions, the number of initially required elements near the diagonal is increased by enlarging the $k \times k$ blocks. Unfortunately, more nonzero elements involve more colors for determining these elements using AD. Hence, the choice of the block size k depends—as already mentioned—on the admitted computational effort, which is measured in the number of colors, and the available memory. In general, the domain specialists have a good understanding of their simulation software and the occurring Jacobian matrices. Thus, they have an intuition which nonzero elements could be important for the preconditioning. Therefore, the specialists can help to find a better sparsification operator than the block diagonal. The target is to obtain a better convergence rate by determining a preconditioner with less or at least the same number of nonzero elements. This topic goes beyond the scope of this thesis and we continue with using the block diagonal.

In the next step, a coloring algorithm computes a coloring Φ when restricted to the initially required elements R_{init}. Two convenient algorithms are the heuristic D2ColoringRestricted in Alg. 2.1 for restricted distance-2 colorings and the

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

heuristic StarBicoloringRestricted in Alg. 3.5 for star bicolorings. Furthermore, the coloring algorithms from Sect. 3.1 can be used for regular grids. The coloring Φ is needed to exploit the sparsity pattern of the Jacobian matrix and to compute the initially required elements as efficiently as possible.

Several matrices from the University of Florida Sparse Matrix Collection [23] are selected to test the approach using the initially required elements R_{init} to determine a preconditioner. The main properties of the matrices, which are the number of columns n, the number of nonzero elements nnz and the density d ($\textsc{nnz}/n^2 \cdot 100$), are given in Table 3.8. There are further properties which are used later on in this chapter: block sizes $k = \lfloor n/32 \rfloor$, $k = \lfloor n/16 \rfloor$, and $k = \lfloor n/8 \rfloor$ as well as maximum degree of the row vertices $\Delta(V_r)$ and the column vertices $\Delta(V_c)$ of the associated bipartite graph $G = (V_r \uplus V_c, E)$.

To meet the practical considerations, we assume that the matrix elements are not directly accessible but in the form of Jacobian matrix-vector products. The general assumption is that the nonzero elements of the coefficient matrices cannot be stored due to storage limitations. In the practical implementations, however, we use the number of nonzero elements as a (generous) upper limit for the storage. The test matrices are employed as coefficient matrices A and the linear system is solved using the iterative solver Bi-CGSTAB with the threshold $\epsilon = 10^{-6}$. The right-hand sides are the sum of all columns because there are no right-hand sides provided by the matrix collection for the selected matrices.

The computational effort is measured in the number of matrix-vector products. Therefore, instead of evaluating the number of iterations which are required to solve the system of linear equations, we consider the number of matrix-vector products. For Bi-CGSTAB, in every iteration two matrix-vector products are carried out. For different block sizes k, the number of matrix-vector products is evaluated. In addition, the number of nonzero elements in R_{init} and the number of colors required to determine these elements with AD are considered. The number of matrix-vector products mvp without preconditioning and using ILU(2) for preconditioning are given in Table 4.1. The ILU preconditioning technique is carried out on several block diagonals with block size $k = 1, 10, \lfloor n/32 \rfloor, \lfloor n/8 \rfloor$. Furthermore, the number of initially required elements R_{init} and occurring fill-in elements F, $\textsc{nnz} = |R_{\text{init}} \uplus F|$ are presented. The number of colors p_{d2} for a restricted distance-2 coloring to determine the initially nonzero elements R_{init} is given.

All systems of linear equations, except matrices crystm01 and nos3, cannot be solved without using at least the diagonal scaling for preconditioning. If a linear solver is not convergent, this is denoted by the symbol – in the column mvp. The diagonal scaling for matrix crystm01 is much better than using no preconditioner. In the last two rows of the table, nv $\varnothing$ and nv, the numbers of matrix-vector products, nonzero elements, and colors are accumulated and normalized. More precisely, in the first row indicated by the symbol nv $\varnothing$, all values mvp, nnz, and p_{d2} are normalized by the values for the main diagonal elements chosen as initially required elements $R_{\text{init}} = \textsc{blkDiag}(A, 1)$. The values of the matrices are accumulated and their average is taken. For block size $k = 10$, the normalized values are $(180/301 + 25/9 +$

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

Matrix A		$k=1$			$k=10$			$k=\lfloor n/32 \rfloor$			$k=\lfloor n/8 \rfloor$		
	MVP	MVP	NNZ	p_{d2}	MVP	NNZ	p_{d2}	MVP	NNZ	p_{d2}	MVP	NNZ	p_{d2}
685_bus	–	301	685	6	180	1,649	11	138	2,351	12	–	–	–
crystm01	81	9	4,875	8	25	11,181	13	15	48,893	22	–	–	–
hor_131	–	210	434	15	101	2,260	29	131	2,468	29	–	–	–
msc00726	–	98	726	25	175	5,280	63	177	8,084	72	132	30,068	110
nos3	394	346	960	10	206	4,942	12	210	5,312	12	–	–	–
nos7	–	170	729	2	148	2,287	10	140	4,017	11	–	–	–
NV $\varnothing$	–	1.00	1.00	1.00	1.18	4.24	2.35	1.00	6.89	2.71	–	–	–
NV	–	1.00	1.00	1.00	0.74	3.28	2.09	0.72	8.46	2.39	–	–	–

Table 4.1: Systems of linear equations are solved without preconditioning and with $\mathrm{ILU}(R_{\mathrm{init}}, 2)$, $R_{\mathrm{init}} = \mathrm{pat}(\tilde{A}) = \textsc{blkDiag}(A, k)$: number of matrix-vector products MVP, number of nonzero elements NNZ $= |R_{\mathrm{init}} \uplus F|$, and number of colors p_{d2} to determine R_{init} using the AD forward mode. Values not available are denoted by –.

$101/210 + 175/98 + 206/346 + 148/170)/6 = 1.18$ for the matrix-vector products and $(1{,}649/685 + 11{,}181/4{,}875 + 2{,}260/434 + 5{,}280/726 + 4{,}942/960 + 2{,}287/729)/6 = 4.24$ for the nonzero elements. If we consider the normalized value of the matrix-vector products 1.18 for $k = 10$, it seems that choosing the block size $k = 10$ is worse compared to $k = 1$. However, with increasing block size, the number of matrix-vector products for the matrices 685_bus, hor_131, nos3, and nos7 becomes smaller. Solely for the remaining two matrices, crystm01 and msc0076, the number of matrix-vector is strongly increased. Thus, one or two matrices can strongly influence the normalized value. We are more interested in the overall number of matrix-vector products. Therefore, we consider an additional normalization without taking the average. In the last row in Table 4.1, indicated by NV, all accumulated values MVP, NNZ, and p_{d2} are normalized by the accumulated values for main diagonal elements chosen as initially required elements $R_{\mathrm{init}} = \textsc{blkDiag}(A, 1)$, i.e., $\textsc{mvp}/\textsc{mvp}(R_{\mathrm{init}})$, $\textsc{nnz}/\textsc{nnz}(R_{\mathrm{init}})$, and $p_{\mathrm{d2}}/p_{\mathrm{d2}}(R_{\mathrm{init}})$ with $R_{\mathrm{init}} = \textsc{blkDiag}(A, 1)$. The values required for the normalization are $\textsc{mvp}(R_{\mathrm{init}}) = 301 + 9 + 210 + 98 + 346 + 170 = 1{,}134$; $\textsc{nnz}(R_{\mathrm{init}}) = 685 + 4{,}875 + 434 + 726 + 960 + 729 = 8{,}409$; and $p_{\mathrm{d2}}(R_{\mathrm{init}}) = 6 + 8 + 15 + 25 + 10 + 2 = 66$. Hence, for block size $k = 1$, the factor is 1.00 for the number of matrix-vector products, the number of nonzero elements, and the number of colors. By increasing the block size from $k = 1$ to $k = \lfloor n/32 \rfloor$, the number of matrix-vector products is reduced to 0.72, i.e., there is a reduction to 72% compared to using the initially required elements for block size $k = 1$. The number of colors increases by a factor of 2.39 and the number of nonzero elements by roughly eight times. Both normalizations are given in the next sections. Nevertheless, we discuss only the normalization without taking the average. For most test matrices, the number of matrix-vector products decreases by using more initially required nonzero elements, e.g., the number of matrix-vector products MVP is reduced from 301 to 138 for matrix 685_bus if the block size is increased from $k = 1$ to $k = \lfloor n/32 \rfloor$. But sometimes the diagonal scaling, $k = 1$, is the best approach among the preconditioners based on more nonzero elements, e.g., for

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

matrix crystm01. If the initially required elements R_{init} are taken from eight large blocks, $k = \lfloor n/8 \rfloor$, these elements and the fill-in elements can solely be stored for the matrix msc00726. For all other matrices, the nonzero elements exceed the assumed memory capacity, the number of nonzero elements of the considered matrix. This is denoted by the symbol – in columns MVP, NNZ, and p_{d2}.

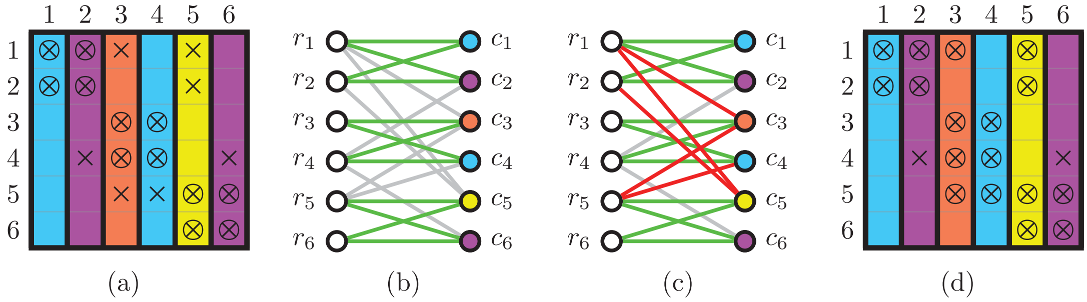
In every iteration of the solver, at least one Jacobian matrix-vector product is evaluated. The system of linear equations is solved using preconditioning techniques. To determine the preconditioner M, a coloring with p colors is needed. Hence, the preconditioning is beneficial if the number of matrix-vector products MVP and the number of colors p, MVP $+ p$, is less than the number of matrix-vector products using no preconditioning. In this thesis, we neglect the time to compute a coloring and assume that this computation is less expensive than evaluating a few Jacobian matrix-vector products. The matrix bus_685 is considered as an illustrating example. Determining the preconditioner $\tilde{M}$ using the initially required elements in blocks of size $k = 1$ requires $p = 6$ colors and MVP $= 301$ matrix-vector products, i.e., a rough measure of the cost is $6 + 301 = 307$. Increasing the block size to $k = 10$, we require $p = 11$ colors and MVP $= 180$ matrix-vector products. Thus, the overall costs, $11 + 180 = 191$, are much lower than 307. The block size $k = \lfloor n/32 \rfloor$ reduces the costs to $12+138 = 150$. Hence, by increasing the block size from $k = 1$ to $k = \lfloor n/32 \rfloor$ the costs are reduced from 307 to 150. However, the number of nonzero elements is increased from NNZ $= 685$ to NNZ $= 2{,}351$.

In Table 4.1, only the number of colors for restricted distance-2 colorings is given. The number of colors for star bicolorings can be found in Table 3.10 in the preceding chapter. In Table 3.10, there are examples for which the star bicoloring heuristic StarBicoloringRestricted saves colors in comparison to the distance-2 coloring heuristic D2ColoringRestricted, e.g., the matrices hor_131 and msc00726.

4.4.2 Potentially required elements

In the previous subsection, the coloring Φ is computed for a given set of initially required elements R_{init}. Using this coloring Φ, there are most likely nonzero elements in $\text{pat}(A) \setminus R_{\text{init}}$ which can be computed in addition to the elements in R_{init} without extra costs, i.e., no extra color is needed. Thus, the aim is to determine more nonzero elements without increasing the number of colors. Hopefully, this approach improves the preconditioner so that the computational effort of solving (4.1)—measured as number of matrix-vector products MVP—is reduced. In summary, more nonzero elements, the potentially required elements R_{pot}, are determined without using an additional color. This approach consists of two parts, which are given in lines 3 and 4 in Alg. 4.1: First, a distance-2 coloring or star bicoloring Φ for the bipartite graph G restricted to the edges in the set $E_{R_{\text{init}}}$ is computed. The edges in $E_{R_{\text{init}}}$ correspond to the initially required elements R_{init} and are denoted as *initially required edges*. Second, the *potentially required edges* of the set $E_{R_{\text{pot}}}$ are detected concerning the coloring Φ, i.e., non-required elements are turned into potentially required elements without making use of an extra color.

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.



(a) (b) (c) (d)

Figure 4.6: (a) Sparsity pattern of Jacobian matrix A with initially required elements $\otimes$ and column groups $\{1,4\}$, $\{2,6\}$, $\{3\}$, and $\{5\}$. (b) Bipartite graph $G(A)$ with initially required edges indicated in green. (c) Bipartite graph $G(A)$ with initially and potentially required edges (in red). (d) Jacobian matrix A with initially and potentially required elements $\otimes$.

Before explaining the computation of R_{pot} in detail, we consider this approach using an illustrating example in Fig. 4.6. The sparsity pattern of a Jacobian matrix with initially required elements and non-required elements is given in Fig. 4.6(a). The required elements are indicated by symbol $\otimes$ and the non-required elements by symbol $\times$. The columns c_1 and c_4 form a column group to determine the initially required elements $a_{1,1}, a_{2,1}, a_{3,4}, a_{4,4} \in R_{\text{init}}$. The column c_4 contains also the non-required element $a_{5,4} \in \text{pat}(A) \backslash R_{\text{init}}$. The corresponding bipartite graph with the initially required edges and the non-required edges is given Fig. 4.6(b). The initially required edges are indicated in green and the non-required edges in gray. The edge (r_5, c_4), which corresponds to the non-required element $a_{5,4}$, is a candidate to become a potentially required edge without violating the given coloring Φ. To verify this, we have to check all paths of length 2 where the edge (r_5, c_4) is included, i.e., the paths (c_4, r_5, c_3), (c_4, r_5, c_5), and (c_4, r_5, c_6). If the outer vertices of the paths are colored differently, e.g., $\Phi(c_4) \neq \Phi(c_3)$, the edge (r_5, c_4) does not violate the coloring Φ. In terms of the matrix, the columns c_1 and c_4 are in the same column group and the non-required element $a_{5,4}$ can be determined because there is no nonzero element at position $(5,1)$. Hence, the edge (r_5, c_4) is added to the set $E_{R_{\text{pot}}}$ in Fig. 4.6(c) and the corresponding element $a_{5,4}$ is added to the set R_{pot} in Fig. 4.6(d). Another column group comprises the columns c_2 and c_6. The non-required elements $a_{4,2}$ and $a_{4,6}$ exclude each other and, hence, neither of both elements can be added to the set R_{pot}.

The algorithm DETPOTREQELEMD2 in Alg. 4.2 determines the potentially required edges $E_{R_{\text{pot}}}$ regarding a distance-2 coloring Φ restricted to $E_{R_{\text{init}}}$. The input parameters for this algorithm are the bipartite graph G, the initially required edges $E_{R_{\text{init}}}$ and the coloring Φ. The function considers paths of length 2, (c_k, r_i, c_j), depending on to which edge sets the edges belong. Therefore, the function iterates over all edges $(r_i, c_j) \in E \backslash E_{R_{\text{init}}}$ with $\Phi(c_j) \neq 0$ and checks for each non-required edge whether it can be added to $E_{R_{\text{pot}}}$. Without loss of generality, the vertex v

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

Algorithm 4.2: Determine potentially required elements for a distance-2 coloring Φ of G when restricted to $E_{R_{\mathrm{init}}}$ (Def. 2.10)

```
function detPotReqElemD2($G = (V_r \uplus V_c, E), E_{R_{\mathrm{init}}}, \Phi$)
    $E_{R_{\mathrm{pot}}} \leftarrow \emptyset$
    foreach $(r_i, c_j) \in E \backslash E_{R_{\mathrm{init}}}$ with $\Phi(c_j) \neq 0$ do                ▷ cond 1
        foreach $c_k \in N_1(r_i, G)$ with $j \neq k$ and $(r_i, c_k) \notin E_{R_{\mathrm{init}}}$ do     ▷ cond 2
            if $\Phi(c_j) = \Phi(c_k)$ then                              ▷ path $(c_k, r_i, c_j)$
                Continue with next edge $(r_i, c_j) \in E \backslash E_{R_{\mathrm{init}}}$
        $E_{R_{\mathrm{pot}}} \leftarrow E_{R_{\mathrm{pot}}} \cup \{(r_i, c_j)\}$                  ▷ valid edge
    return $E_{R_{\mathrm{pot}}}$
```

at the first position of an edge (v, w) in the undirected bipartite graph is located in V_r and vertex w at the second position in V_c. For every vertex r_i, the function visits all neighbors c_k in the bipartite graph, i.e., the algorithm considers all paths (c_k, r_i, c_j). The edge (r_i, c_j) is considered as required edge because this edge should become a potentially required edge. For every distance-2 neighbor c_k of vertex c_j with edge $(r_i, c_k) \notin E_{R_{\mathrm{init}}}$, the property $\Phi(c_j) \neq \Phi(c_k)$ must be checked. This check is only carried out for edges $(r_i, c_k) \notin E_{R_{\mathrm{init}}}$. For the other edges (r_i, c_k), i.e., edges $(r_i, c_k) \in E_{R_{\mathrm{init}}}$, the vertices c_j and c_k are colored differently due to condition 2 in Def. 2.10. If there is a combination of c_j and c_k where the same color is assigned to both vertices, the edge (r_i, c_j) cannot be added to set $E_{R_{\mathrm{pot}}}$. Otherwise, the condition 2 in Def. 2.10 would be violated, i.e., the element corresponding to the edge (r_i, c_j) would be summed up by another nonzero element. Therefore, the algorithm stops considering this edge and continues with the next edge. If the vertex c_j is colored differently to its distance-2 neighbors c_k, the edge (r_i, c_j) is added to $E_{R_{\mathrm{pot}}}$.

Lemma 4.1. *Given a bipartite graph* $G = (V_r \uplus V_c, E)$*, initially required edges* $E_{R_{init}}$*, and a distance-2 coloring* Φ *restricted to* $E_{R_{init}}$*, the maximum number of potentially required edges are computed by the algorithm* detPotReqElemD2 *in* $\mathcal{O}(|E| \cdot \Delta(V_r))$.

Proof. We will prove the correctness by contradiction. Therefore, we assume that the edge $(r_i, c_j) \in R_{\mathrm{pot}}$ determined by the algorithm detPotReqElemD2 violates the coloring Φ. Thus, there must be a path (c_k, r_i, c_j) with $\Phi(c_j) = \Phi(c_k)$. This is impossible because, in this case, the algorithm would not add this edge to $E_{R_{\mathrm{pot}}}$, but it would jump to the next edge.

Furthermore, we show that the maximum number of potentially required elements is chosen from $E \backslash E_{R_{\mathrm{init}}}$ regarding the coloring Φ. We assume that the edge $(r_i, c_j) \in E_{R_{\mathrm{pot}}}$ excludes another edge $(r_i, c_k) \notin E_{R_{\mathrm{pot}}}$ to become also part of $E_{R_{\mathrm{pot}}}$. To exclude this edge, the condition $\Phi(c_j) = \Phi(c_k)$ must hold. The contradiction is that even edge (r_i, c_j) would not be added to $E_{R_{\mathrm{pot}}}$ if there is any neighbor c_k of vertex r_i which is identically colored to vertex c_j. Thus, if no edge is excluded by another edge, the maximum number of potentially required edges is computed.

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

The algorithm detPotReqElemD2 iterates over all edges (r_i, c_j) and visits the distance-1 neighbors of the vertices r_i. The neighbors can be estimated by the maximum degree-1 in the vertex set V_r. Thus, the complexity is $\mathcal{O}(|E| \cdot \Delta(V_r))$. This complexity is the same as for the heuristic D2ColoringRestricted. $\square$

Continuing the example from above, we assume that the column groups given in Fig. 4.6(a) are the result of the algorithm D2ColoringRestricted, which is given as coloring Φ. The bipartite graph G, the set of initially required edges $E_{R_{\text{init}}}$, and the coloring Φ are the input parameters for the function detPotReqElemD2. This function detects the potentially required edges as follows: There are seven edges which are in $E \backslash E_{R_{\text{init}}}$. Suppose that, at first, the edge $(r_4, c_2) \in E \backslash E_{R_{\text{init}}}$ is considered. The vertex c_6 is adjacent to vertex r_4 and the condition in line 4 holds. The edge (r_4, c_2) cannot be added to $E_{R_{\text{init}}}$ because the vertices c_2 and c_6 are identically colored. Otherwise, condition 2 of Def. 2.10 would be violated. Hence, we jump to the next edge $(r_1, c_3) \in E \backslash E_{R_{\text{init}}}$. The edges (r_1, c_1), (r_1, c_2), and (r_1, c_5) are incident to vertex r_1. The vertex c_3 is differently colored than vertices c_1, c_2, and c_5. Thus, the edge (r_1, c_3) is added to $E_{R_{\text{pot}}}$. We repeat this step for the edges (r_5, c_3), (r_5, c_4), (r_1, c_5), (r_2, c_5), and (r_4, c_6). All these edges, except edge (r_4, c_6), are added to $E_{R_{\text{pot}}}$. This function yields the potentially required edges $E_{R_{\text{pot}}} \subseteq E \backslash E_{R_{\text{init}}}$ as given in Fig. 4.6(c). These edges are indicated in red. The corresponding Jacobian matrix is depicted in Fig. 4.6(d). In this example only two elements are not in the set $E_{R_{\text{init}}} \uplus E_{R_{\text{pot}}}$.

The counterpart for star bicolorings is the algorithm detPotReqElemSB in Alg. 4.3. Its principle is similar to algorithm detPotReqElemD2, but the conditions are modified to the restricted star bicoloring in Def. 2.12. All non-required edges are candidates for the potentially required edges, but, following condition 2 of Def. 2.12, at least an incident vertex must be assigned a nonzero color. Furthermore, the subconditions of condition 3 must be checked. If a condition is not fulfilled, the algorithm skips the considered edge and moves on to check the next possible edge. If no condition is violated, the edge is added to the potentially required edges set $E_{R_{\text{pot}}}$. Proving the correctness follows the same idea as for detPotReqElemD2 in Lem. 4.1. The algorithm visits all edges and the neighbors of the incident vertices. The complexity is $\mathcal{O}(|E| \cdot \Delta^2(V_r \uplus V_c))$.

Results

After describing how the potentially required elements R_{pot} can be determined, we evaluate this approach in terms of the convergence rate. Therefore, the configuration from the last subsection is used again, i.e., the same matrices are taken from the University of Florida Sparse Matrix Collection, the iterative solver Bi-CGSTAB is employed, the right-hand side is the sum of all columns, and the stopping criterion is $\epsilon = 10^{-6}$. The initially required elements R_{init} and the potentially required elements R_{pot} are used to compute the $\text{ILU}(R_{\text{init}} \uplus R_{\text{pot}}, 2)$ preconditioner. That is, fill-in elements are allowed with respect to R_{pot} or, in other words, as many elements

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

Algorithm 4.3: Determine potentially required elements for a star bicoloring Φ of G when restricted to $E_{R_{\text{init}}}$ (Def. 2.12)

```
function detPotReqElemSB($G = (V_r \uplus V_c, E), E_{R_{\text{init}}}, \Phi$)
    $E_{R_{\text{pot}}} \leftarrow \emptyset$
    foreach $(r_i, c_j) \in E \backslash E_{R_{\text{init}}}$ with $\Phi(r_i) \neq 0$ or $\Phi(c_j) \neq 0$ do      ▷ cond 2
        if $\Phi(r_i) = 0$ then                                                                  ▷ cond 3a
            foreach $c_k \in N_1(r_i, G)$ with $j \neq k$ and $(r_i, c_k) \notin E_{R_{\text{init}}}$ do   ▷ path $(c_k, r_i, c_j)$
                if $\Phi(c_j) = \Phi(c_k)$ then
                    Continue with next edge $(r_i, c_j) \in E \backslash E_{R_{\text{init}}}$
        if $\Phi(c_j) = 0$ then                                                                  ▷ cond 3b
            foreach $r_\ell \in N_1(c_j, G)$ with $j \neq \ell$ and $(r_\ell, c_j) \notin E_{R_{\text{init}}}$ do   ▷ path $(r_i, c_j, r_\ell)$
                if $\Phi(r_i) = \Phi(r_\ell)$ then
                    Continue with next edge $(r_i, c_j) \in E \backslash E_{R_{\text{init}}}$
        if $\Phi(r_i) \neq 0$ and $\Phi(c_j) \neq 0$ then                                        ▷ cond 3c
            foreach $c_k \in N_1(r_i, G)$ with $i \neq k$ do
                foreach $r_\ell \in N_1(c_j, G)$ with $j \neq \ell$ do                           ▷ path $(c_k, r_i, c_j, r_\ell)$
                    if $\Phi(c_j) = \Phi(c_k)$ and $\Phi(r_i) = \Phi(r_\ell)$ then
                        Continue with next edge $(r_i, c_j) \in E \backslash E_{R_{\text{init}}}$
        $E_{R_{\text{pot}}} \leftarrow E_{R_{\text{pot}}} \cup \{(r_i, c_j)\}$                   ▷ valid edge
    return $E_{R_{\text{pot}}}$
```

as possible are chosen for a fixed number of colors p. Both described heuristics, detPotReqElemD2 and detPotReqElemSB, are used to compute the potentially required elements R_{pot}.

The results when using $E_{R_{\text{pot}}} =$ detPotReqElemD2 are given in Table 4.2. The number of matrix-vector products mvp and the number of nonzero elements nnz for R_{init} from Table 4.1 are repeated to make the comparison easier. The last row (nv) contains the accumulated values normalized to R_{init} with block size $k = 1$ without taking the average as in Table 4.1. For $k = 10$ and $k = \lfloor n/32 \rfloor$, the number of matrix-vector products is not available for all matrices. In particular, the number of nonzero for R_{init}, R_{pot}, and the fill-in elements exceeds the number of nonzero elements of the matrix, i.e., $|R_{\text{init}} \uplus (F \cup R_{\text{pot}})| > |A|$. Thus, an accumulation is not possible for both the number of matrix-vector products and the number of nonzero elements. Using the potentially required elements in addition to the initially required elements is better for preconditioning: For the normalization without taking the average (row nv), the overall number of matrix-vector products for $k = 1$ can be reduced to 92% (0.92 compared to 1.00). More precisely, the number of matrix-vector products decreases for the matrices 685_bus, hor_131, and nos3. Increasing the block size to $k = 10$ and using the potentially required elements in addition to the initially required elements leads to a reduction in the number of matrix-vector products for the matrices crystm01 and nos3. Increasing the size to $k = \lfloor n/32 \rfloor$, for matrices crystm01 and nos3, the number of matrix-vector products is decreased in comparison to using solely R_{init}. Thus, for

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

Matrix	$k = 1$				$k = 10$				$k = \lfloor n/32 \rfloor$			
	R_{init}		$R_{\text{init}} \uplus R_{\text{pot}}$		R_{init}		$R_{\text{init}} \uplus R_{\text{pot}}$		R_{init}		$R_{\text{init}} \uplus R_{\text{pot}}$	
	MVP	NNZ	MVP	NNZ	MVP	NNZ	MVP	NNZ	MVP	NNZ	MVP	NNZ
685_bus	301	685	193	1,793	180	1,649	–	–	138	2,351	–	–
crystm01	9	4,875	23	8,505	25	11,181	17	25,567	15	48,893	11	85,011
hor_131	210	434	192	1,799	101	2,260	–	–	131	2,468	–	–
msc00726	98	726	164	7,248	175	5,280	–	–	177	8,084	–	–
nos3	346	960	304	2,953	206	4,942	172	6,905	210	5,312	172	6,905
nos7	170	729	170	729	148	2,287	–	–	140	4,017	–	–
NV ∅	1.00	1.00	1.28	3.76	1.18	4.24	–	–	1.00	6.89	–	–
NV	1.00	1.00	0.92	2.74	0.74	3.28	–	–	0.72	8.46	–	–

Table 4.2: Systems of linear equations are solved with ILU(2) using $R_{\text{init}} =$ BLKDIAG(A, k) and $R_{\text{init}} \uplus R_{\text{pot}}$, $R_{\text{pot}} =$ DETPOTREQELEMD2: number of matrix-vector products MVP when using R_{init} and $R_{\text{init}} \uplus R_{\text{pot}}$ as well as number of nonzero elements NNZ for $|R_{\text{init}} \uplus F|$ and $|R_{\text{init}} \uplus (F \cup R_{\text{pot}})|$ including the fill-in elements. Values not available are denoted by –.

Matrix	$k = 1$				$k = 10$				$k = \lfloor n/32 \rfloor$			
	R_{init}		$R_{\text{init}} \uplus R_{\text{pot}}$		R_{init}		$R_{\text{init}} \uplus R_{\text{pot}}$		R_{init}		$R_{\text{init}} \uplus R_{\text{pot}}$	
	MVP	NNZ	MVP	NNZ	MVP	NNZ	MVP	NNZ	MVP	NNZ	MVP	NNZ
685_bus	301	685	213	1,972	180	1,649	–	–	138	2,351	–	–
crystm01	9	4,875	25	9,471	25	11,181	21	30,223	15	48,893	12	93,330
hor_131	210	434	185	1,972	101	2,260	–	–	131	2,468	–	–
msc00726	98	726	190	8,559	175	5,280	–	–	177	8,084	–	–
nos3	346	960	308	3,379	206	4,942	166	11,665	210	5,312	166	11,665
nos7	170	729	170	729	148	2,287	–	–	140	4,017	–	–
NV ∅	1.00	1.00	1.37	4.28	1.18	4.24	–	–	1.00	6.89	–	–
NV	1.00	1.00	0.96	3.10	0.74	3.28	–	–	0.72	8.46	–	–

Table 4.3: Evaluate R_{pot} with algorithm DETPOTREQELEMSB for star bicolorings instead of distance-2 colorings as in Table 4.2.

each matrix and $k = 10$ or $k = \lfloor n/32 \rfloor$—if the values are available—the number of matrix-vector products is reduced by using the potentially required elements R_{pot}. The number of nonzero elements $|R_{\text{init}} \uplus (F \cup R_{\text{pot}})|$ is increased compared to not using the potentially required elements. In the normalized result for $k = 1$ without taking the average, this number is more or less increased threefold with 2.74. For the matrix crystm01 with $k = 10$ and $k = \lfloor n/32 \rfloor$, the difference is roughly factor 2 (25,567/11,181 and 85,011/48,893). In summary, we obtain a better convergence behavior by employing more nonzero elements.

The considered potentially required elements R_{pot} are obtained by using distance-2 colorings. We now switch to star bicolorings and evaluate if there are benefits using the method DETPOTREQELEMSB instead of DETPOTREQELEMD2. The results are given in Table 4.3. For block size $k = 1$, the number of matrix-vector products can be reduced for matrix hor_131 from 192 (Table 4.2) to 185 (Table 4.3) by switching to a star bicoloring. For matrix nos3 with $k = 10$ and $k = \lfloor n/32 \rfloor$, the number

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

of matrix-vector products is always lower than using distance-2 colorings. The algorithm StarBicoloringRestricted determines a coloring with less number of colors compared to D2ColoringRestricted for matrices hor_131 and msc00726 with block sizes $k = 10$ and $k = \lfloor n/32 \rfloor$. These results have already been given in Table 3.10 in the previous chapter. For matrix hor_131, the number of colors can be reduced from $p_{\text{d2}} = 27$ to $p_{\text{sb}} = 18$ for block size $k = 10$ and from $p_{\text{d2}} = 28$ to $p_{\text{sb}} = 19$ for block size $k = \lfloor n/32 \rfloor$. Unfortunately, the number of nonzero elements exceeds the memory; thus, the number of matrix-vector products and the number of nonzero elements cannot be compared. However, we may benefit from the reduced number of colors when considering the additionally required elements in the following section. Using StarBicoloringRestricted instead of D2ColoringRestricted decreases not only the number of colors, but also yields different potentially required elements R_{pot}. Using a star bicoloring for the considered matrices, the number of nonzero elements is always greater than using a distance-2 coloring.

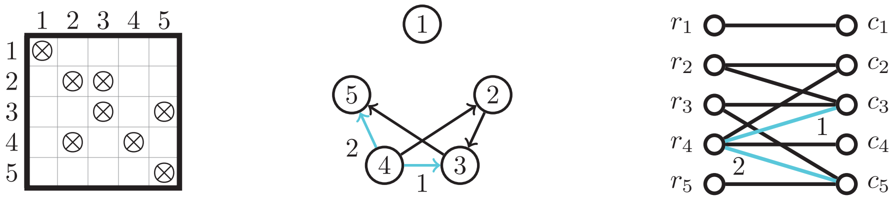
4.4.3 Additionally required elements

After introducing the computation of the potentially required elements R_{pot}, in this subsection, we consider the memory consumption. It is possible to use the elements in R_{pot} for preconditioning. However, a lot of fill-in elements could occur which are not already in the fill-in set F. The memory restriction can be violated by these elements. Only a subset of the elements in R_{pot} is added to the additionally required elements R_{add} based on some rules. These nonzero elements can be used for preconditioning without getting that many fill-in elements compared to R_{pot}. After defining the fill-in elements and integrating this definition into our categorization of nonzero elements, we explain how to choose additionally required elements and evaluate their usage for preconditioning.

Fill-in elements

Hysom and Pothen [36] introduced a graph model for the incomplete LU factorization (ILU). In comparison to the (complete) LU factorization, this incomplete factorization leads to no fill-in elements or solely to a predetermined subset of fill-in elements. The underlying graph is identical to the adjacency graph used for the LU factorization [56]. The so-called *fill path* in the incomplete LU factorization is adapted from the fill path theorem for the LU factorization. A fill path characterizes a possible fill-in edge. Recall that a fill-in element is originally a zero element which becomes a nonzero element during the factorization. A fill path in terms of the adjacency graph of a matrix is a path $(v_i, \ldots, v_k, \ldots, v_j)$ with $k < \min(i, j)$, i.e., the indices of all intermediate vertices are smaller than the indices of the vertices v_i and v_j. A fill-in element occurs at position (i, j) in the matrix if there is a fill path from vertex v_i to vertex v_j and the edge (v_i, v_j) does not exist. The adjacency graph in Fig. 4.7(b) corresponds to the sparsity pattern of a Jacobian matrix A given in Fig. 4.7(a). The edges of the adjacency graph are directed, and there is an edge (v_i, v_j) iff the nonzero

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.



(a) (b) (c)

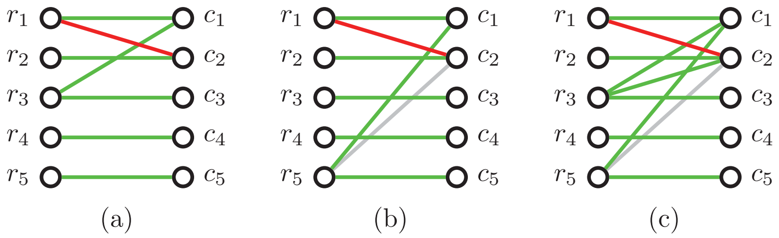
Figure 4.7: (a) Sparsity pattern of Jacobian matrix A. (b) Adjacency graph of A with fill paths (v_4, v_2, v_3), (v_4, v_2, v_3, v_5) and fill-in edges (v_4, v_3), (v_4, v_5). (c) Bipartite graph $G(A)$ with fill paths (r_4, c_2, r_2, c_3), $(r_4, c_2, r_2, c_3, r_3, c_5)$ and fill-in edges (r_4, c_3), (r_4, c_5) indicated by color cyan.

element $a_{i,j}$ exists. We consider two existing fill paths. The fill path (v_4, v_2, v_3) causes the fill-in element $a_{4,3}$ during the factorization due to the ordering specified by the indices 2, 3, and 4. The fill path (v_4, v_2, v_3, v_5) leads to the element $a_{4,5}$. The *fill level* ℓ is an input for the ILU factorization and is used to restrict the number of fill-in edges. The length of a fill path indicates the level of a fill-in element $a_{i,j}$. More precisely, the level of the fill-in element $a_{i,j}$ equals the length of the shortest fill path between the vertices v_i and v_j decreased by one. In Fig. 4.7(b), the fill path (v_4, v_2, v_3) of length 2 is depicted. Thus, the level of the fill-in edge (v_4, v_3) is 1. The second fill path (v_4, v_2, v_3, v_5) is of length 3 and hence the fill-in edge (v_4, v_5) has level 2. During the incomplete LU factorization, solely fill-in elements up to level ℓ are allowed. If level $\ell = 0$ is chosen, no fill-in elements are allowed during the factorization.

For a common graph model throughout this thesis, the definition of the fill path is adapted to the bipartite graph model. The fill path $(r_i, c_k, r_k, c_\ell, r_\ell, \ldots, c_j)$ in the bipartite graph corresponds to the fill path $(v_i, v_k, v_\ell, \ldots, v_j)$ in the adjacency graph. Each fill path starts with a row vertex to define these paths in a consistent way. An intermediate vertex v_k of the fill path in the adjacency graph represents the row k as well as the column k. Every intermediate vertex is doubled to transform the fill path to the (undirected) bipartite graph because there are separate vertices for rows and columns. A path $(r_i, c_k, r_k, c_\ell, r_\ell, \ldots, c_j)$ is a fill path from r_i to c_j, iff all vertices between r_i and c_j have a lower index than i and j. There is a fill-in edge (r_i, c_j) with level ℓ in the bipartite graph, iff there is a shortest fill path of length $2l + 1$ between the vertices r_i and c_j. The fill-in edges are part of the edge set E_F and correspond to the fill-in elements in F. Two fill paths are indicated in Fig. 4.7(c). The fill path (r_4, c_2, r_2, c_3) corresponds to (v_4, v_2, v_3) in Fig. 4.7(b) and the fill path $(r_4, c_2, r_2, c_3, r_3, c_5)$ to (v_4, v_2, v_3, v_5).

We consider a sequence of bipartite graphs to determine the fill-in edges. Therefore, it is sufficient to look at paths of length 3 to determine the fill-in edges. This approach is described for the adjacency graph in [36]. We start with the bipartite graph corresponding to the adjacency graph without fill-in edges. The fill-in edges caused by fill paths of length 3 are added. After this step, there is a modified graph including all fill-in edges of level $\ell = 1$. For every level, we repeat this step on the bi-

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.



(a) (b) (c) (d)

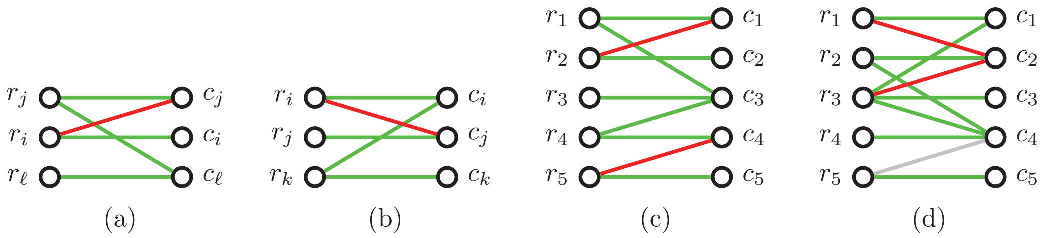
Figure 4.8: Adding edge (r_1, c_2) to $E_{R_{\text{add}}}$ would lead to fill-in edges (a) (r_3, c_2), (b) $(r_5, c_2,)$, and (c) $(r_5, c_2,)$ due to fill path (r_5, c_1, r_1, c_2). (d) Adding edge (r_1, c_2) to $E_{R_{\text{add}}}$ is fine due to both fill paths are closed because edges $(r_3, c_2), (r_5, c_2) \in E_{R_{\text{init}}} \uplus E_F$.

partite graph including the already computed fill-in edges as long as the given level ℓ is reached or no new fill path occurs. We exemplify this procedure with the bipartite graph in Fig. 4.7(c). In the first step, the fill path (r_4, c_2, r_2, c_3) is considered and the fill-in edge (r_4, c_3) is added. This graph is the second graph in the sequence and the starting point to determine the fill-in edges on level 2. Therefore, we consider the fill-in paths introduced by the fill-in edges with level $\ell = 1$. In our example, it is the path (r_4, c_3, r_3, c_5) and the new fill-in edge (r_4, c_5) with level 2 is added. The final graph in the sequence is used to compute the additionally required elements.

Determining the additionally required elements

The incomplete LU factorization is originally defined for nonzero patterns. In contrast, the model introduced in Sect. 4.4 distinguishes between different kinds of nonzero elements: the initially required elements R_{init}, the potentially required elements R_{pot}, the additionally required elements R_{add}, the fill-in elements induced by R_{init}, and the remaining (non-required) nonzero elements. We consider the bipartite graph $G[E_{R_{\text{init}}} \uplus (E_F \cup E_{R_{\text{pot}}})]$ to determine additionally required elements. This graph is induced by the edges $(r_i, c_j) \in E_{R_{\text{init}}} \uplus E_{R_{\text{pot}}}$ and the fill-in edges E_F induced by $E_{R_{\text{init}}}$ already added. The non-required edges $(r_i, c_j) \in E \backslash (E_{R_{\text{init}}} \uplus (E_F \cup E_{R_{\text{pot}}}))$ are not included. We decide for each edge in $E_{R_{\text{pot}}}$ if this edge can be added to $E_{R_{\text{add}}}$ without getting a fill-in element during the factorization. By adding an edge $(r_i, c_j) \in E_{R_{\text{pot}}}$ to the set $E_{R_{\text{add}}}$, a fill path, which does not exist before, can occur. The main question is whether there exists an edge which extends an occurring fill path to become closed, i.e., the first and the last vertex of the path are identical. In graph theory, this is denoted as a cycle. In this situation, no fill-in edge occurs due to the edge which already exists at that position. To illustrate this question, we look at several examples in Fig. 4.8 where the edges in $E_{R_{\text{init}}}$ are colored with green, the edges in $E_{R_{\text{pot}}}$ with red, the fill-in edges in E_F with cyan, and the non-required edges with gray. The non-required edges are not determined and considered during the factorization. Therefore, fill-in edges may arise at their positions. The fill path (r_3, c_1, r_1, c_2) in Fig. 4.8(a) is considered first. The edge $(r_1, c_2) \in E_{R_{\text{pot}}}$ cannot be added to $E_{R_{\text{add}}}$

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.



(a) (b) (c) (d)

Figure 4.9: (a) Fill path (r_i, c_j, r_j, c_ℓ). (b) Fill path (r_k, c_i, r_i, c_j). (c) + (d) Bipartite graphs with fill paths.

since the fill path is not closed and thus results in an extra fill-in element. Otherwise, the fill-in edge (r_3, c_2) would appear. In Fig. 4.8(b), the fill path (r_5, c_1, r_1, c_2) seems to be closed due to the non-required edge (r_5, c_2). But it is not closed because the edge $(r_5, c_2) \in E$ is not part of $E_{R_{\text{init}}} \uplus (E_F \cup E_{R_{\text{pot}}})$. In Fig. 4.8(c), the edge (r_1, c_2) creates two fill paths. The fill path (r_3, c_1, r_1, c_2) does not matter because it is closed due to the edge $(r_3, c_2) \in E_{R_{\text{init}}}$. The second fill path (r_5, c_1, r_1, c_2) causes the fill-in edge (r_5, c_2). In the last illustrating example in Fig. 4.8(d), when the edge $(r_1, c_2) \in E_{R_{\text{pot}}}$ is considered, there occur two fill paths. Both fill paths are closed due to the edges $(r_3, c_2) \in E_{R_{\text{init}}}$ and $(r_5, c_2) \in E_F$. In summary, an edge $(r_i, c_j) \in E_{R_{\text{pot}}}$ can be added to $E_{R_{\text{add}}}$ if this edge does not yield a fill-in element. Thus, this edge can be additionally used for determining a preconditioner.

In the following subsection, a conservative strategy for choosing additionally required elements is introduced. Thereafter, a more sophisticated approach is described in which more information is taken into account. Thus, the decision whether a fill-in occurs can be made more accurately. In principle, one would expect that this approach leads to more edges in $E_{R_{\text{add}}}$.

Conservative approach

In the first approach, for every edge $(r_i, c_j) \in E_{R_{\text{pot}}}$ the fill paths (r_i, c_j, r_j, c_ℓ) or (r_k, c_i, r_i, c_j) are respectively considered depending on the indices i and j. At first, we study the case $i > j$. Therefore, the fill path (r_i, c_j, r_j, c_ℓ), which is given in Fig. 4.9(a), is considered. This fill path exists, if a neighbor c_ℓ of vertex r_j has a larger index, $\ell > j$, and possibly leads to a fill-in edge. Therefore, we do not add (r_i, c_j) to R_{add}. An illustrating example is given in Fig. 4.9(c): Adding the edge (r_2, c_1) to $E_{R_{\text{add}}}$ would generate the fill path (r_2, c_1, r_1, c_3) due to the indices $3 > 1$. If the condition $\ell < j$ holds for all neighbors, there cannot be any fill path. A corresponding example is the path (r_5, c_4, r_4, c_3) in Fig. 4.9(c). This path is not a fill path due to $3 < 4$. The symmetric case $j > i$ with the fill path (r_k, c_i, r_i, c_j) in Fig. 4.9(b) is analogous: If $k > i$ holds, there could occur a fill-in element. Otherwise, $k < i$, this is no fill path; thus, no fill-in edge will occur during the factorization. If the edge (r_i, c_j) does not involve a fill path, this edge can be added to $E_{R_{\text{add}}}$ and the corresponding entry $a_{i,j}$ to R_{add}. An algorithm following this principle is given in Alg. 4.4. This

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

Algorithm 4.4: Determine additionally required elements E_{R_add}

```
1  function detAddReqElem($G = (V_r \uplus V_c, E), E_{R_\text{init}}, E_F, E_{R_\text{pot}}$)
2      $E_{R_\text{add}} = \emptyset$
3      foreach $(r_i, c_j) \in E_{R_\text{pot}}$ do
4          if $i > j$ then                                                    ▷ path $(r_i, c_j, r_j, c_\ell)$
5              if $\exists c_\ell \in N_1(r_j, G[E_{R_\text{init}} \uplus (E_F \cup E_{R_\text{add}})])$ with $\ell > j$ then
6                  Continue with next edge $(r_i, c_j) \in E_{R_\text{pot}}$
7          else if $j > i$ then                                               ▷ path $(r_k, c_i, r_i, c_j)$
8              if $\exists r_k \in N_1(c_i, G[E_{R_\text{init}} \uplus (E_F \cup E_{R_\text{add}})])$ with $k > i$ then
9                  Continue with next edge $(r_i, c_j) \in E_{R_\text{pot}}$
10         $E_{R_\text{add}} = E_{R_\text{add}} \cup \{(r_i, c_j)\}$
11     return $E_{R_\text{add}}$
```

algorithm iterates over all edges (r_i, c_j) in E_{R_pot}. Depending on the relation between the indices i and j, the fill path (r_i, c_j, r_j, c_ℓ) or (r_k, c_i, r_i, c_j) is considered. If a fill path and, possibly, a fill edge occurs, the edge (r_i, c_j) is not added to E_{R_add} and the algorithm jumps to the next edge in E_{R_pot}.

Lemma 4.2. *Given a bipartite graph $G = (V_r \uplus V_c, E)$, the initially required edges $E_{R_{init}}$, the fill-in edges E_F induced by $E_{R_{init}}$, and the potentially required edges $E_{R_{pot}}$, the algorithm* detAddReqElem *computes a set of additionally required edges $E_{R_{add}}$ in $\mathcal{O}(|E| \cdot \Delta(V_r \uplus V_c))$.*

Proof. We will prove the correctness by contradiction. Therefore, the algorithm detAddReqElem adds the edge $(r_i, c_j) \in E_{R_\text{pot}}$ to E_{R_add}. We assume that this edge leads to a fill-in edge during the factorization. In the case $i > j$, the index ℓ must be larger than j to get a fill path (r_i, c_j, r_j, c_ℓ). If this case occurs in line 5, the edge (r_i, c_j) is not added to E_{R_add} and the algorithm jumps to the next edge in E_{R_pot}. In the other case, $j > i$, a fill path (r_k, c_i, r_i, c_j) exists if the index k is larger than index i. Due to line 8 this cannot occur. Thus, there is no edge added to E_{R_add} which would yield a fill-in edge.

The algorithm detAddReqElem iterates over all edges (r_i, c_j) and visits the distance-1 neighbors of either the vertex r_i or the vertex c_j. Therefore, the complexity of the algorithm is $\mathcal{O}(|E| \cdot \Delta(V_r \uplus V_c))$. □

Sophisticated approach

As already mentioned, the previously described strategy is quite conservative. Let us consider the edge $(r_3, c_2) \in E_{R_\text{pot}}$ in Fig. 4.9(d) as an illustrating example. The algorithm detAddReqElem does not add this edge to E_{R_add} because it would lead to the fill path (r_3, c_2, r_2, c_4) and, hence, the fill-in edge (r_3, c_4) could occur. However, since the edge (r_3, c_4) already exists in E_{R_init}, this fill-in edge does not occur. Therefore, before adding an edge (r_i, c_j) to E_{R_add}, we check for all fill paths (r_i, c_j, r_j, c_ℓ), $i > j$, if the edge (r_i, c_ℓ) is already in $E_{R_\text{init}} \uplus E_F$. For the other fill paths (r_k, c_i, r_i, c_j),

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

Algorithm 4.5: Determine additionally required elements $E_{R_{\text{add}}}$

```
function detAddReqElemCycle($G = (V_r \uplus V_c, E), E_{R_{\text{init}}}, E_F, E_{R_{\text{pot}}}$)
    $E_{R_{\text{add}}} = \emptyset$
    do
        foreach $(r_i, c_j) \in E_{R_{\text{pot}}}$ do
            if $i > j$ then
                foreach $c_\ell \in N_1(r_j, G[E_{R_{\text{init}}} \uplus (E_F \cup E_{R_{\text{add}}})])$ with $\ell > j$ do
                    if $(r_i, c_\ell) \notin E_{R_{\text{init}}} \uplus (E_F \cup E_{R_{\text{add}}})$ then    ▷ path $(r_i, c_j, r_j, c_\ell)$
                        Continue with next edge $(r_i, c_j) \in E_{R_{\text{pot}}}$
            else if $j > i$ then
                foreach $r_k \in N_1(c_i, G[E_{R_{\text{init}}} \uplus (E_F \cup E_{R_{\text{add}}})])$ with $k > i$ do
                    if $(r_k, c_j) \notin E_{R_{\text{init}}} \uplus (E_F \cup E_{R_{\text{add}}})$ then    ▷ path $(r_k, c_i, r_i, c_j)$
                        Continue with next edge $(r_i, c_j) \in E_{R_{\text{pot}}}$
            $E_{R_{\text{add}}} = E_{R_{\text{add}}} \cup \{(r_i, c_j)\}$; $E_{R_{\text{pot}}} = E_{R_{\text{pot}}} - \{(r_i, c_j)\}$
    while $|E_{R_{\text{add}}}|$ is increased in last iteration
    return $E_{R_{\text{add}}}$
```

$j > i$, we have to verify whether the edge (r_k, c_j) already exists. If all fill paths are closed, the edge (r_i, c_j) can be added to $E_{R_{\text{add}}}$. This improved approach, which is given as algorithm detAddReqElemCycle in Alg. 4.5, iterates over the edges (r_i, c_j) in $E_{R_{\text{pot}}}$, which can be potentially added to $E_{R_{\text{add}}}$. The paths (r_k, c_i, r_i, c_j) with $j > i$ and $k > i$ or (r_i, c_j, r_j, c_ℓ) with $i > j$ and $\ell > j$ are respectively considered. If the edges (r_k, c_j) or (r_i, c_ℓ) are already in $E_{R_{\text{init}}} \uplus (E_F \cup E_{R_{\text{add}}})$, no new fill-in edges occur. Therefore, the edge (r_i, c_j) can be added to $E_{R_{\text{add}}}$.

The edges which are already in $E_{R_{\text{add}}}$ influence which edges in $E_{R_{\text{pot}}}$ can be added to $E_{R_{\text{add}}}$. Let us consider the edges $(r_1, c_2), (r_3, c_2) \in E_{R_{\text{pot}}}$ as well as the fill paths (r_3, c_1, r_1, c_2) and (r_3, c_2, r_2, c_4) in Fig. 4.9(d). If we check the edge (r_1, c_2) first, this edge cannot be added to $E_{R_{\text{add}}}$ because the fill path (r_3, c_1, r_1, c_2) is not closed. The edge (r_3, c_2) is considered next. The fill path (r_3, c_2, r_2, c_4) is closed by the edge $(r_3, c_4) \in E_{R_{\text{init}}}$. Thus, the edge (r_3, c_2) can be added to $E_{R_{\text{add}}}$. By changing the order in which the edges are checked, the edge (r_3, c_2) is added to $E_{R_{\text{add}}}$ before checking the edge (r_1, c_2). Therefore, the fill path (r_3, c_1, r_1, c_2) is closed and the edge (r_1, c_2) can also be added to $E_{R_{\text{add}}}$. Hence, more additionally required edges can be determined by changing the ordering. Instead of this modification, we again check the remaining edges in $E_{R_{\text{pot}}}$. To increase the number of additionally required edges, the algorithm detAddReqElemCycle is repeated until a fix-point for $E_{R_{\text{add}}}$ is reached, i.e., there is no edge added to $E_{R_{\text{add}}}$ in the last iteration. The correctness of the algorithm is shown in the following lemma:

Lemma 4.3. *Given a bipartite graph $G = (V_r \uplus V_c, E)$, the initially required edges $E_{R_{init}}$, the fill-in edges E_F induced by $E_{R_{init}}$, and the potentially required edges $E_{R_{pot}}$, the algorithm* detAddReqElemCycle *determines a set of additionally required edges $E_{R_{add}}$ in $\mathcal{O}(|E|^2 \cdot \Delta(V_r \uplus V_c))$.*

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

Proof. We will prove the correctness by contradiction. Therefore, the edge $(r_i, c_j) \in E_{R_{\text{pot}}}$ is added to $E_{R_{\text{add}}}$. We assume that this edge leads to a fill-in edge. In the case $i > j$, the index ℓ must be larger than j to have one or more fill paths (r_i, c_j, r_j, c_ℓ). In this case, the algorithm checks for every vertex c_ℓ in line 7 whether the edge (r_i, c_ℓ) is present. If one of these edges does not exist, the edge (r_i, c_j) is not added to $E_{R_{\text{add}}}$ and the algorithm jumps to the next edge in $E_{R_{\text{pot}}}$. The other case, $j > i$, is analogous. Thus, there is no edge added to $E_{R_{\text{add}}}$ which would yield a fill-in edge.

The algorithm detAddReqElemCycle iterates at most $|E_{R_{\text{pot}}}|$ times over all edges $(r_i, c_j) \in E_{R_{\text{pot}}}$ as long as a fix-point for $E_{R_{\text{add}}}$ is reached. In every iteration, it visits the distance-1 neighbors of either the vertex r_i or the vertex c_j. Hence, with $|E_{R_{\text{pot}}}| \leq |E|$, the complexity of the algorithm is $\mathcal{O}(|E|^2 \cdot \Delta(V_r \uplus V_c))$. □

For both algorithms holds: Adding edges to $E_{R_{\text{add}}}$ does not lead to new fill-in edges neither with a level higher nor with a level lower than ℓ.

Results

Instead of evaluating the number of matrix-vector products using the initially and potentially required elements to determine a preconditioner, we substitute the potentially required elements R_{pot} by the additionally required elements R_{add}. Therefore, the number of nonzero elements is decreased due to $R_{\text{add}} \subseteq R_{\text{pot}}$. The configuration from the preceding subsections is adopted to evaluate the number of matrix-vector products with preconditioners based on R_{add}. In Table 4.4, the additionally required elements computed by the algorithms detAddReqElem and detAddReqElemCycle are compared to the results from the last subsection for the block sizes 10 and $\lfloor n/32 \rfloor$. The additionally required elements computed by the algorithms detAddReqElem and detAddReqElemCycle are once more indicated by the symbols $\star$ and $\circ$, respectively.

Both sets, the potentially and additionally required elements, are considered together with the initially required elements and the fill-in elements. Instead of the nonzero elements $R_{\text{init}} \uplus (F \cup R_{\text{pot}})$, it is enough to store the nonzero elements in $R_{\text{init}} \uplus (F \cup R_{\text{add}})$. For the block sizes $k = 10$ and $k = \lfloor n/32 \rfloor$, by employing the potentially required elements, the number of nonzero elements exceeds the available memory for most matrices. The additionally required elements can be stored, instead. Furthermore, for block size $k = \lfloor n/32 \rfloor$, the normalized number of accumulated matrix-vector products without taking the average (nv), $\text{mvp}_\star = 0.62$ is better than the number of matrix-vector products $\text{mvp} = 0.72$ for the initially required elements R_{init} given in Table 4.1. In summary, our target to decrease the number of nonzero elements is fulfilled. The number of matrix-vector products needed with R_{add} exceeds the matrix-vector products needed with R_{pot} due to the reduced number of nonzero elements. Most important, the number of matrix-vector products is reduced when employing $R_{\text{init}} \uplus R_{\text{add}}$ compared to using solely R_{init}.

For block sizes $k = 10$ and $k = \lfloor n/32 \rfloor$, the number of matrix-vector products is reduced for the matrices nos3 and nos7 if the preconditioner is based on

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

Matrix	$k = 10$						$k = \lfloor n/32 \rfloor$					
	$R_{\text{init}} \uplus R_{\text{pot}}$		$R_{\text{init}} \uplus R_{\text{add}\star}$		$R_{\text{init}} \uplus R_{\text{add}\circ}$		$R_{\text{init}} \uplus R_{\text{pot}}$		$R_{\text{init}} \uplus R_{\text{add}\star}$		$R_{\text{init}} \uplus R_{\text{add}\circ}$	
	MVP	NNZ	$\text{MVP}_\star$	$\text{NNZ}_\star$	$\text{MVP}_\circ$	$\text{NNZ}_\circ$	MVP	NNZ	$\text{MVP}_\star$	$\text{NNZ}_\star$	$\text{MVP}_\circ$	$\text{NNZ}_\circ$
685_bus	–	–	185	1,989	181	2,093	–	–	124	2,564	131	2,644
crystm01	17	25,567	22	14,029	22	16,502	11	85,011	14	49,398	14	50,220
hor_131	–	–	77	2,474	85	2,552	–	–	86	2,646	104	2,725
msc00726	–	–	180	6,154	184	6,681	–	–	155	8,622	164	8,951
nos3	172	6,905	206	5,105	198	5,512	172	6,905	201	5,374	194	5,520
nos7	–	–	178	2,500	152	2,515	–	–	122	4,100	116	4,125
NV ∅	–	–	1.15	4.78	1.13	5.12	–	–	0.88	7.18	0.90	7.36
NV	–	–	0.75	3.84	0.72	4.26	–	–	0.62	8.65	0.64	8.82

Table 4.4: Systems of linear equations are solved using $R_{\text{init}} \uplus R_{\text{pot}}$ or $R_{\text{init}} \uplus R_{\text{add}}$ with $R_{\text{init}} = \textsc{blkDiag}(A, k)$ for preconditioning: number of matrix-vector products MVP when using $R_{\text{init}} \uplus R_{\text{pot}}$ and $\text{MVP}_\star$ when using $R_{\text{init}} \uplus R_{\text{add}\star}$, $R_{\text{add}\star} = \textsc{detAddReqElem}$, or $\text{MVP}_\circ$ when using $R_{\text{init}} \uplus R_{\text{add}\circ}$, $R_{\text{add}\circ} = \textsc{detAddReqElemCycle}$; number of nonzeros $\text{NNZ} = |R_{\text{init}} \uplus (F \cup R_{\text{pot}})|$, $\text{NNZ}_\star = |R_{\text{init}} \uplus (F \cup R_{\text{add}\star})|$, and $\text{NNZ}_\circ = |R_{\text{init}} \uplus (F \cup R_{\text{add}\circ})|$. Values not available are denoted by –.

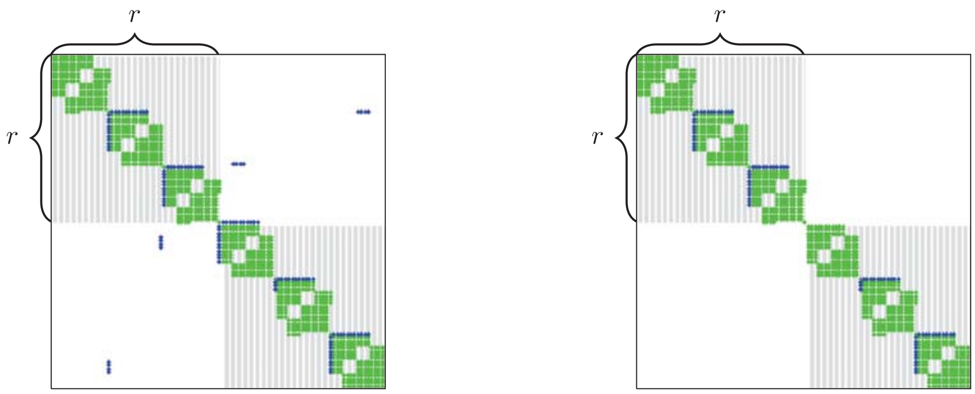
$R_{\text{init}} \uplus R_{\text{add}}$ with R_{add} determined by the algorithm DETADDREQELEMCYCLE instead of DETADDREQELEM. For the matrix 685_bus, it depends on the block size whether the preconditioning based on the additionally required elements selected by DETADDREQELEM or DETADDREQELEMCYCLE leads to a reduced number of matrix-vector products. For matrices hor_131 and msc00726, the additionally required elements determined by DETADDREQELEM leads to a smaller number of matrix-vector products. For matrix crystm01, there is no difference in the number of matrix-vector products between both sets of additionally required elements.

A preconditioner determined by using the potentially required elements R_{pot} together with the initially required elements R_{init} can often not be stored due to memory restrictions. Using the additionally required elements R_{add} instead of R_{pot}, a preconditioner is successfully determined and carried out to increase the convergence rate of the iterative method.

4.4.4 Parallelism and required elements

Today's simulations are often computed on several machines with distributed memory. One of the main challenges is to reduce the communication between the processors. Therefore, the preconditioner should be modified so that the structure can be exploited to solve the preconditioned system of linear equations in parallel. The block structure of the initially required elements employed in the previous sections is beneficial for parallel solving because the blocks and the corresponding parts of the right-hand side can be partitioned and divided between the processors. Afterwards, one or more blocks belong to a processor, and each processor can solve its subsystems of linear equations independently without communicating intermediate results.

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.



(a) $R_{\text{init}} \uplus R_{\text{add}}$ (b) $R_{\text{init}} \uplus R_{\text{add}}^{|r \times r}$

Figure 4.10: Given $R_{\text{init}} = \textsc{blkDiag}(A, k), k = 16$, the set R_{add} is determined using ILU(2): (a) without restrictions and (b) restricted to larger $r \times r$ blocks indicated by gray background padding.

Unfortunately, the additionally required elements R_{add} as well as the induced fill-in elements F are possibly outside of the blocks and destroy this beneficial structure.

In general, the order of the Jacobian matrices is much larger than the number of processors available for solving the systems of linear equations in parallel. Therefore, in addition to the block diagonal with $k \times k$ blocks, another level of $r \times r$ blocks with $r > k$ is introduced. Instead of the $k \times k$ blocks, these larger $r \times r$ blocks are scattered among the processors. In Fig. 4.10, we consider an illustrating example. The $k \times k$ blocks of the block diagonal with $k = 16$ are indicated by the initially required elements colored in green. The $r \times r$ blocks with $r = 48$ are indicated by gray background padding. The additionally required elements are restricted to the larger blocks. That is, solely the larger blocks contain the elements of R_{add} and the fill-in elements induced by R_{init} and R_{add}. Due to the factorization strategy of ILU, there does not occur any fill-in element outside of these blocks. Before selecting an additionally required element, it is checked whether its position is outside of the $r \times r$ blocks. Hence, additionally required elements outside of these blocks are discarded. To determine the additionally required elements, bipartite graphs associated to the $r \times r$ blocks of the Jacobian matrix are the input for the algorithms detPotReqElemD2 (Alg. 4.2) and detAddReqElem (Alg. 4.4).

For solving preconditioned systems of linear equations in parallel, the structure with the larger $r \times r$ blocks is beneficial. However, discarding additionally required elements may involve a worse convergence rate of solving the preconditioned systems by iterative solvers compared to non-discarding any elements. There is no data dependency between the $r \times r$ blocks distributed among different processors. That is, no information must be exchanged while solving a linear system. With this modification, not only this linear system can be solved in parallel, but also the preconditioner $\tilde{M}$ can be constructed in parallel. Therefore, every processor builds its part of $\tilde{M}$.

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

Matrix	$k = 10, r = n$				$k = 10, r = \lfloor n/16 \rfloor$				$k = 10, r = \lfloor n/8 \rfloor$			
	MVP$_\star$	NNZ$_\star$	MVP$_\circ$	NNZ$_\circ$	MVP$_\star$	NNZ$_\star$	MVP$_\circ$	NNZ$_\circ$	MVP$_\star$	NNZ$_\star$	MVP$_\circ$	NNZ$_\circ$
685_bus	185	1,989	181	2,093	156	1,795	152	1,838	155	1,853	159	1,917
crystm01	22	14,029	22	16,502	22	13,722	21	15,904	22	13,962	21	16,356
hor_131	77	2,474	85	2,552	96	2,323	97	2,336	86	2,340	88	2,353
msc00726	180	6,154	184	6,681	188	5,564	174	5,795	182	5,653	160	5,963
nos3	206	5,105	198	5,512	209	5,069	191	5,393	206	5,100	196	5,504
nos7	178	2,500	152	2,515	136	2,407	143	2,417	148	2,430	158	2,443
NV $\varnothing$	1.15	4.78	1.13	5.12	1.12	4.51	1.08	4.71	1.12	4.57	1.07	4.81
NV	0.75	3.84	0.72	4.26	0.71	3.67	0.69	4.01	0.70	3.73	0.69	4.11

Table 4.5: Systems of linear equations are solved using $R_{\text{init}} \uplus R_{\text{add}}$ with $R_{\text{init}} =$ BLKDIAG(A, k) for preconditioning: number of matrix-vector products MVP when using $R_{\text{init}} \uplus R_{\text{add}}$ and number of nonzero elements NNZ $= |R_{\text{init}} \uplus (F \cup R_{\text{add}})|$ with $R_{\text{add}\star}^{|r\times r} =$ DETADDREQELEM and $R_{\text{add}\circ}^{|r\times r} =$ DETADDREQELEMCYCLE both restricted to $r \times r$ blocks.

Results

The described approach is evaluated by solving systems of linear equations. The additionally required elements R_{add} restricted to the $r \times r$ blocks are denoted by the symbol $R_{\text{add}}^{|r\times r}$. These elements are determined by the functions DETADDREQELEM and DETADDREQELEMCYCLE independent of the $r \times r$ blocks. Recall that both functions determine these elements differently. The number of nonzero elements and the number of matrix-vector products for solving system of linear equations are compared to using the additionally required elements R_{add} introduced in Sect. 4.4.3, i.e., without deleting elements in R_{add}. No additionally required element is omitted if there is only one large $r \times r$ block with $r = n$ comprising the whole matrix. For the evaluation, the block size of the $k \times k$ blocks is chosen with $k = 10$. The results for $r = n$ have already been given in Table 4.4 and are repeated in Table 4.5 to ease the comparison. Two particular block sizes for the larger blocks, $r = \lfloor n/16 \rfloor$ and $r = \lfloor n/8 \rfloor$, are considered. These blocks are beneficial if the matrix is stored on 8 or 16 computing nodes with distributed memory.

The number of nonzero elements is reduced by eliminating the additionally required elements outside of the larger $r \times r$ blocks. Furthermore, no additionally required element is located outside of these blocks; thus, no fill-in element can occur outside. If the additionally required elements are determined with the function DETADDREQELEM, the number of nonzero elements is reduced from a factor 3.84—normalized to the initially required elements R_{init} and their induced fill-in elements F—to 3.67 and 3.73 for $r = \lfloor n/16 \rfloor$ and $r = \lfloor n/8 \rfloor$, respectively. The results are depicted in Table 4.5. The factors which are given in row NV are normalized without taking the average. The normalized number of nonzero elements is reduced from 4.26 to 4.01 and 4.11, respectively, when using the function DETADDREQELEMCYCLE. The normalized number of accumulated matrix-vector products decreases from 0.75 to 0.71 or 0.70 by using the initially and additionally

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

required elements $R_{\text{init}} \uplus R_{\text{add}}^{|r\times r}$, $R_{\text{add}}^{|r\times r} = \textsc{detAddReqElem}$, for preconditioning. In other words, the number of matrix-vector products decreases from 75% to 71% or 70%. For $R_{\text{add}}^{|r\times r} = \textsc{detAddReqElemCycle}$ with $r = \lfloor n/16 \rfloor$ and $r = \lfloor n/8 \rfloor$, 69% of the matrix-vector products are needed. For matrix hor_131, the number of nonzero elements is reduced from $\textsc{nnz}_\star = 2{,}474$ for $k = 10, r = n$ to $\textsc{nnz}_\star = 2{,}323$ for $k = 10, r = \lfloor n/16 \rfloor$ and the number of matrix-vector products increases from $\textsc{mvp}_\star = 77$ to $\textsc{mvp}_\star = 96$. There are matrices for which a smaller number of nonzero elements yields a reduced number of matrix-vector products. For matrix 685_bus, for example, a reduction from $\textsc{nnz}_\star = 1{,}989$ for $k = 10, r = n$ to $\textsc{nnz}_\star = 1{,}795$ for $k = 10, r = \lfloor n/16 \rfloor$ yields $\textsc{mvp}_\star = 185$ to $\textsc{mvp}_\star = 156$. Although the reduction of the number of matrix-vector products by discarding of additionally required elements is not expected, it confirms that this approach works and is better than using only the initially required elements.

4.5 Results

After describing the determination of the potentially and additionally required elements for preconditioning, we conclude this chapter with a selection of the results. First, the convergence history is shown for the matrix hor_131, i.e., the relative residual norm for every matrix-vector product of the iterative solver. Second, the convergence history is compared for the matrix memplus employing a distance-2 coloring and a star bicoloring. Third and finally, the most important results are again summarized with respect to the limited computational effort and memory. Therefore, the matrices from the previous sections are employed once more to keep the summary coherent. In particular, we evaluate the relation between the number of nonzero elements $\textsc{nnz}$ and the number of matrix-vector products $\textsc{mvp}$. Although we suppose that the initially required elements R_{init} are given by domain experts, we again start with the nonzero elements in the block diagonal $\textsc{blkDiag}(A, k)$ of the matrices.

In the previous sections, we give the numbers of matrix-vector products needed for solving systems of linear equations without considering the convergence rate. The matrix hor_131 is an illustrating example to show the convergence rates when using no preconditioning and different sets of required elements. These rates are measured in terms of the relative residual norm

$$||b - A \cdot x_{\textsc{mvp}-1}||_2 / ||b - A \cdot x_0||_2$$

and are plotted over the number of matrix-vector products in Fig. 4.11. That is, the relative residual norm is calculated after determining each matrix-vector product $\textsc{mvp}$. Using the iterative solver GMRES without restart, one matrix-vector product is computed in each iteration. We do not mind the memory consumption of the iterative solver. The right-hand side b is once more the sum of all columns. The threshold for the convergence is $\epsilon = 10^{-6}$ and the level for ILU is $\ell = 2$, i.e., we employ ILU(2). The system of linear equations with the matrix A is solved without using preconditioning and by using the initially required elements $R_{\text{init}} = \textsc{blkDiag}(A, 10)$,

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

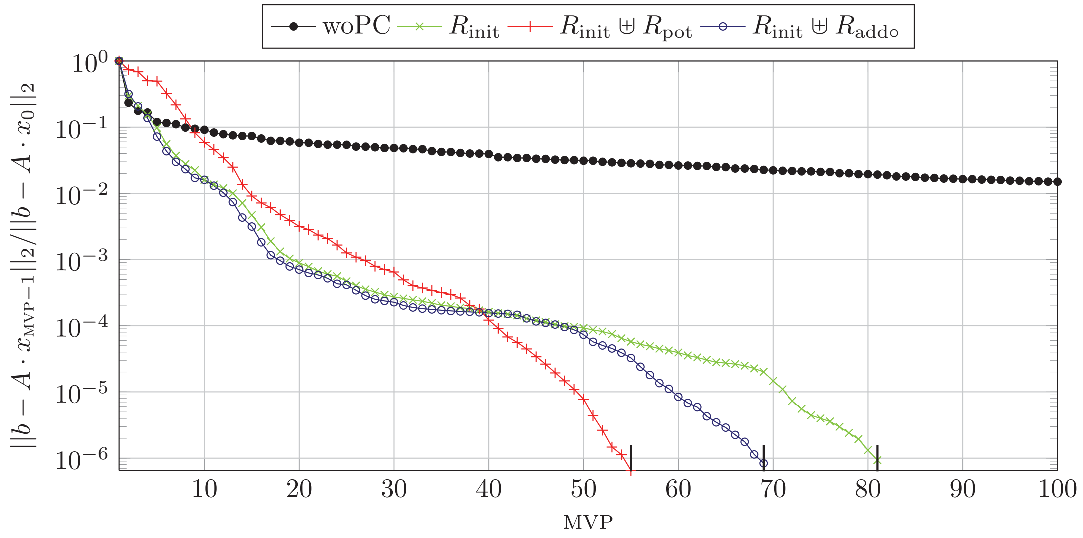


Figure 4.11: Relative residual norms for solving a system of linear equations with coefficient matrix hor_131. The system is solved without preconditioning (woPC) and using R_{init}, $R_{\text{init}} \uplus R_{\text{pot}}$, and $R_{\text{init}} \uplus R_{\text{addo}}$ with $R_{\text{init}} = \textsc{blkDiag}(A, 10)$ for preconditioning.

$R_{\text{init}} \uplus R_{\text{pot}}$, or $R_{\text{init}} \uplus R_{\text{addo}}$ for preconditioning. The sets R_{pot} and R_{addo} are determined by the algorithms DETPOTREQELEMD2 and DETADDREQELEMCYCLE, respectively. The matrix hor_131 contains 4,812 nonzero elements. For this example, to compare the number of nonzero elements and number of matrix-vector products for the potentially required elements R_{pot}, we relax the assumption from the previous sections that we cannot store more than $|A|$ nonzero elements. The number of nonzero elements, already given in Table 4.1 and 4.4, are repeated to have it at a glance together with $|R_{\text{init}} \uplus (F \cup R_{\text{pot}})|$

$\lvert R_{\text{init}} \uplus F\rvert$	$\lvert R_{\text{init}} \uplus (F \cup R_{\text{pot}})\rvert$	$\lvert R_{\text{init}} \uplus (F \cup R_{\text{addo}})\rvert$
2,260	8,943	2,552

The numbers of matrix-vector products are different from the previous sections because of using the iterative solver GMRES instead of Bi-CGSTAB. This substitution shows that preconditioning with different required elements improves the convergence rate for solving systems of linear equations not only for Bi-CGSTAB. The convergence behavior can be identified in Fig. 4.11. The convergence rate for solving the system of linear equations without preconditioning is the slowest. For a converged solution, MVP = 408 matrix-vector products are needed. The plot is cut after MVP = 100 matrix-vector products. The convergence rates for R_{init} and $R_{\text{init}} \uplus R_{\text{addo}}$ are faster than for $R_{\text{init}} \uplus R_{\text{pot}}$ until MVP = 39. Thereafter, the behavior changes completely such that $R_{\text{init}} \uplus R_{\text{pot}}$ induces the fastest convergence rate. Preconditioning with $R_{\text{init}} \cup R_{\text{addo}}$ always outperforms the preconditioning with R_{init}. Employing a preconditioner which is determined by using the initially required elements R_{init}, the

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

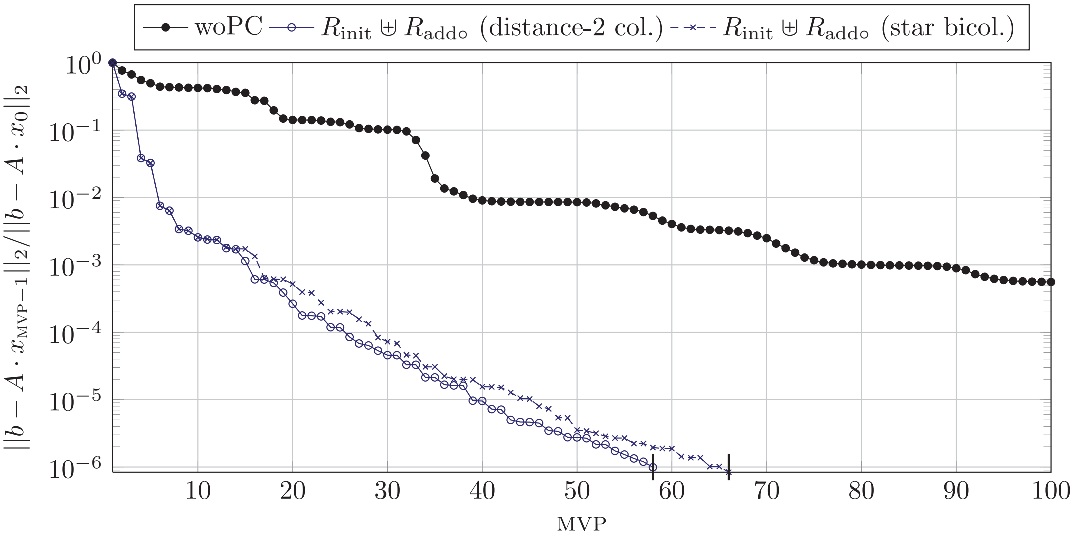


Figure 4.12: Relative residual norms for solving a system of linear equations with matrix memplus. The system is solved without preconditioning (woPC) and using $R_{\text{init}} \uplus R_{\text{addo}}$ with $R_{\text{init}} = \text{BLKDIAG}(A, \lfloor n/32 \rfloor)$ for preconditioning and R_{addo} determined by D2ColoringRestricted and StarBicoloringRestricted.

iterative solver computes a converged solution of the system with MVP $= 81$ matrix-vector products. Only MVP $= 55$ matrix-vector products are needed if the potentially required elements R_{pot} are used in addition. However, the number of nonzero elements increases from 2,260 to 8,943. Using the additionally required elements R_{add} instead, the solution convergences with MVP $= 69$ slower than using R_{pot}, but the number of nonzero elements decreases to 2,552. Nevertheless, it is still faster than using only the initially required elements R_{init}. In summary, the convergence rate up to MVP $= 39$ is similar for the different preconditioners. For MVP > 39, the convergence rates behave as expected, i.e., $R_{\text{init}} \uplus R_{\text{pot}}$ needs less matrix-vector products for convergence than $R_{\text{init}} \uplus R_{\text{addo}}$ which in turn needs less than R_{init}.

Before giving a general summary for the initially, potentially, and additionally required elements, we show that determining additionally required elements can benefit from two-sided colorings compared to one-sided colorings. Therefore, we compare additionally required elements determined with a distance-2 coloring and a star bicoloring for matrix memplus. The configuration for the iterative solver GMRES and the ILU preconditioner are not varied compared to solving the system of linear equations with the coefficient matrix hor_131 in the preceding paragraphs. The only difference is that a given right-hand side from the matrix collection is used. Once more, we relax the assumption that we cannot store more than $|A|$ nonzero elements. In Fig. 4.12, the convergence rates are given when using no preconditioning and when using $R_{\text{init}} \uplus R_{\text{addo}}$ for preconditioning. The elements of $R_{\text{init}} \uplus R_{\text{addo}}$ are determined with both a distance-2 coloring and a star bicoloring. Recall that one of our primary concerns is the computational effort. The computational effort

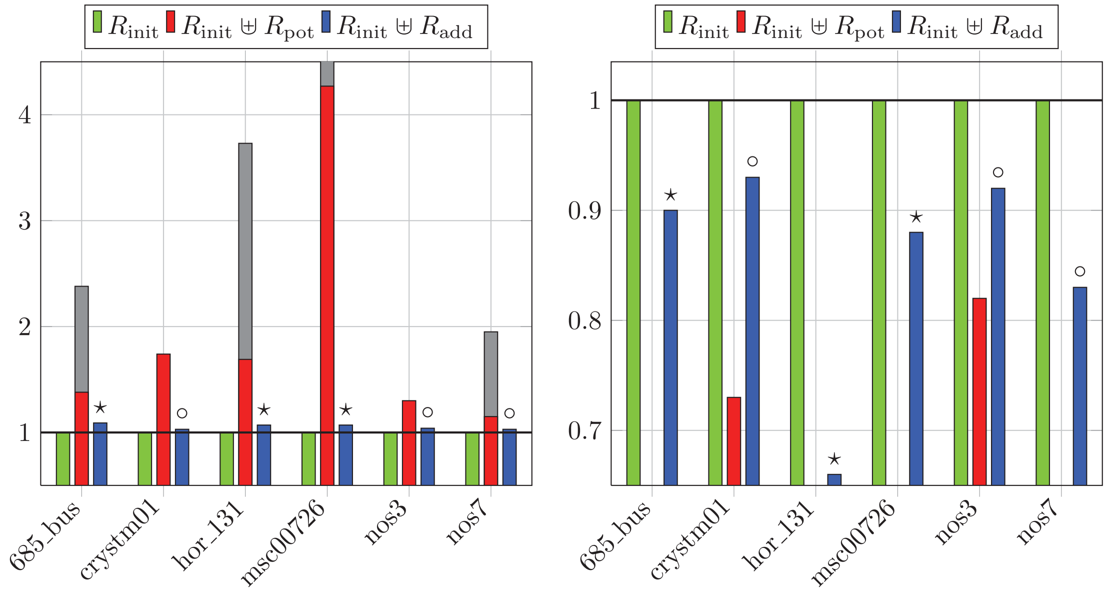
Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

comprises the number of colors to obtain the initially and additionally required elements $R_{\text{init}} \uplus R_{\text{add}\circ}$ and the number of matrix-vector products needed to solve a system of linear equations. Recall that a color respresents a matrix-vector product in automatic differentiation. Thus, increasing the number of colors by one increases the computational costs by one matrix-vector product. Recall from Table 3.10 that the distance-2 coloring determined with algorithm D2ColoringRestricted (Alg. 2.1) comprises $p_{\text{d2}} = 146$ colors and the star bicoloring determined with algorithm StarBicoloringRestricted (Alg. 3.5) comprises only $p_{\text{sb}} = 88$ colors. Although the number of colors are different, the convergence rates are comparable. The number of matrix-vector products is $\textsc{mvp} = 58$ when using the distance-2 coloring and $\textsc{mvp} = 66$ for the star bicoloring. Overall, the computational costs measured in matrix-vector products (cf. Sect. 4.4.1) are $p + \textsc{mvp} = 146 + 58 = 204$ and $88 + 66 = 154$, respectively. In comparison, for solving the system of linear equations without preconditioning, 371 matrix-vector products are needed. For the distance-2 coloring, $\textsc{nnz} = 127{,}994$ nonzero elements are determined and, for the star bicoloring, $\textsc{nnz} = 127{,}985$. Thus, using a star bicoloring to determine $R_{\text{init}} \uplus R_{\text{add}\circ}$ is superior over using a distance-2 coloring in this example.

After describing the convergence behavior for the matrix hor_131 and assessing the benefit of a star bicoloring for the matrix memplus, we evaluate the relation between the initially, potentially, and additionally required elements more generally by using the matrices from the previous sections. Recall from Table 4.1 that for most matrices using a preconditioner is necessary to solve the system of linear equations by Bi-CGSTAB. For the remaining matrices, the number of matrix-vector products is at least reduced compared to using no preconditioner. By increasing the number of nonzero elements, the solution of a system of linear equations, generally, converges faster. Recall that the iterative solver Bi-CGSTAB is used with threshold $\epsilon = 10^{-6}$ and level $\ell = 2$ for ILU to solve the systems of linear equations. We look at the number of nonzero elements and matrix-vector products for block size $k = \lfloor n/32 \rfloor$. These numbers are already given in the previous sections.

The numbers of nonzero elements in Fig. 4.13(a) are normalized to the numbers of nonzero elements occurring while using the initially required elements for preconditioning, $\textsc{nnz}(R_{\text{init}}) = |R_{\text{init}} \uplus F|$. Using potentially required elements, in addition to the initially required elements R_{init}, increases the number of nonzero elements $\textsc{nnz}$, obviously. In this figure, these nonzero elements, $|R_{\text{init}} \uplus (F \cup R_{\text{pot}})|$, are given even if they exceed the number of nonzero elements of the matrix. The amount of nonzero elements which can be stored is indicated by the red part of the bar. The nonzero elements exceeding the memory are indicated by the gray color on top of the red color. In Fig. 4.13(a), the nonzero elements $R_{\text{init}} \uplus (F \cup R_{\text{pot}})$ can be stored only for the matrices crystm01 and nos3. The factor $\textsc{nnz}/\textsc{nnz}(R_{\text{init}}), \textsc{nnz} = |R_{\text{init}} \uplus (F \cup R_{\text{pot}})|$, is between 1.95 and 15.50 compared to the initially required elements and their induced fill-in elements, $\textsc{nnz}(R_{\text{init}})$. The plot is cut when the bars exceed the value 4.5. Using additionally required elements R_{add} in addition to the initially required elements R_{init} yields significantly less nonzero elements for all matrices than using R_{pot}. The factor is between 1.03 and 1.09. The additionally required elements that lead

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.



(a) NNZ/NNZ(R_{init})

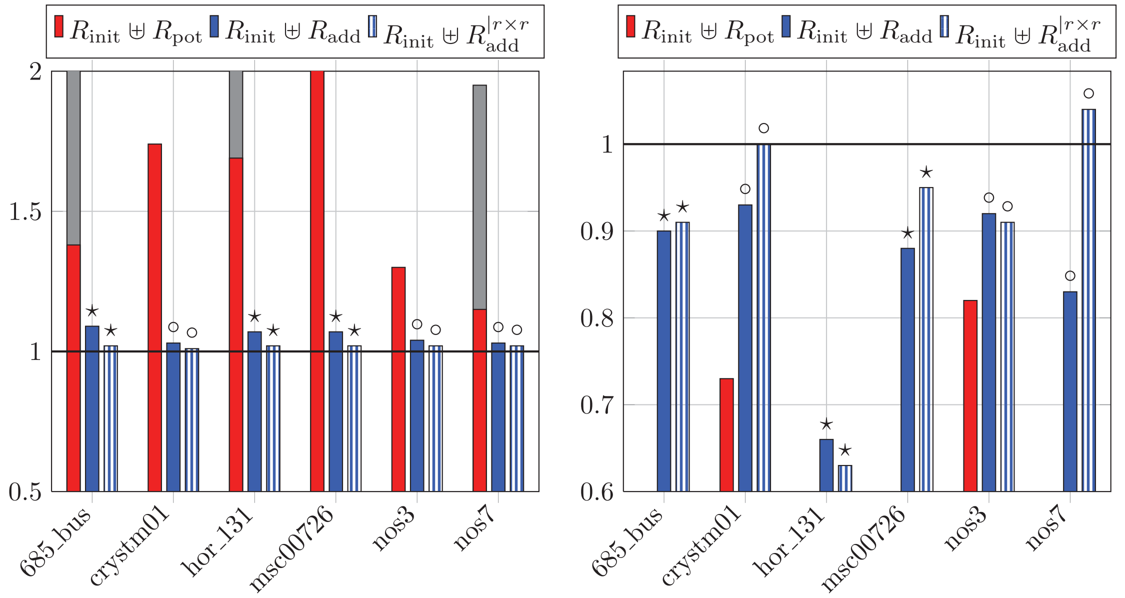
(b) MVP/MVP(R_{init})

Figure 4.13: Systems of linear equations are solved with ILU(2) using $R_{\text{init}} = \text{BLKDIAG}(A, k)$ with $k = \lfloor n/32 \rfloor$, $R_{\text{init}} \uplus R_{\text{pot}}$, and $R_{\text{init}} \uplus R_{\text{add}}$: number of nonzero elements $|R_{\text{init}} \uplus F|$, $|R_{\text{init}} \uplus (F \cup R_{\text{pot}})|$, and $|R_{\text{init}} \uplus (R_{\text{add}} \cup F)|$ normalized to NNZ(R_{init}) as well as number of matrix-vector products MVP normalized to MVP(R_{init}).

to less matrix-vector products are plotted instead of both the additionally required elements determined by DETADDREQELEM and DETADDREQELEMCYCLE. Additionally required elements determined by the algorithms DETADDREQELEM and DETADDREQELEMCYCLE are respectively indicated with the symbols ⋆ and ∘ on top of the corresponding bar.

The number of matrix-vector products needed to solve the systems of linear equations are given in Fig. 4.13(b). For each matrix, these numbers are normalized to the number of matrix-vector products occurring while using the initially required elements R_{init} for preconditioning. The number of matrix-vector products is reduced by using the potentially and additionally required elements together with the initially required elements R_{init} instead of solely using R_{init}. If the nonzero elements $R_{\text{init}} \uplus (F \cup R_{\text{pot}})$ exceed the available memory, the corresponding number of matrix-vector products is not given. For evaluating the number of nonzero elements and the matrix-vector products, the same additionally required elements R_{add} as in Fig. 4.13(a) are considered. The initially and potentially required elements $R_{\text{init}} \uplus R_{\text{pot}}$ can only be stored for the matrices crystm01 and nos3. The number of matrix-vector products increases for these matrices if the additionally required elements R_{add} are used for preconditioning instead of the potentially required elements R_{pot}. Most important, less required elements must be stored. As already stated in the previous

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.



(a) NNZ/NNZ(R_{init})

(b) MVP/MVP(R_{init})

Figure 4.14: Systems of linear equations are solved with ILU(2) using $R_{\text{init}} \uplus R_{\text{pot}}$, where $R_{\text{init}} = \text{BLKDIAG}(A, \lfloor n/32 \rfloor)$, $R_{\text{init}} \uplus R_{\text{add}}$, and $R_{\text{init}} \uplus R_{\text{add}}$ restricted to $r \times r$ blocks, $r = \lfloor n/16 \rfloor$, for parallel computing: number of nonzero elements NNZ including the fill-in elements and number of matrix-vector products MVP both normalized to R_{init}.

sections, the general tendency in Fig. 4.13 is as follows: The higher the number of nonzero elements, the lower is the number of matrix-vector products. For all matrices, employing the additionally required elements reduces the number of matrix-vector products MVP compared to using solely R_{init}.

If systems of linear equations are solved in parallel, the block diagonal structure is beneficial. Determining additionally required elements R_{add} by applying the algorithms DETADDREQELEM or DETADDREQELEMCYCLE to the Jacobian matrix does not preserve this structure. Therefore, another block structure with larger $r \times r$ blocks is introduced in Sect. 4.4.4. There are no nonzero elements outside of these blocks. The required elements are restricted to these larger blocks. Otherwise, these elements could occur at every position where the matrix contains a nonzero element or a fill-in element which is induced while factorizing R_{init}. The number of nonzero elements is given in Fig. 4.14(a). The plot is cut when the bars exceed the value 2.0. The number of matrix-vector products and the number of nonzero elements are again normalized to R_{init}, and the additionally required elements—determined by the same algorithm as in Fig. 4.13—are considered. The employed algorithm is once more indicated by the symbols $\star$ and $\circ$ given on top of the bars. The number of matrix-vector products is shown in Fig. 4.14(b). The numbers are again normalized to R_{init} as in Fig. 4.13.

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

For all matrices except nos7, the number of matrix-vector products, when employing $R_{\text{add}}^{|r\times r}$, is lower or equal compared to using R_{init}. The number of nonzero elements is always lower when using $R_{\text{add}}^{|r\times r}$ compared to employing R_{add}. The matrices can be divided into two classes depending on the number of matrix-vector products: The first class contains the matrices hor_131 and nos3. For these matrices, using the additionally required elements $R_{\text{add}}^{|r\times r}$ leads to less matrix-vector products compared to the additionally required elements R_{add} originally determined. For the remaining matrices, the additionally required elements R_{add} lead to less matrix-vector products. In summary, using the block diagonal structure for solving systems of linear equations in parallel is successfully applied. That is, for most matrices, using the additionally required elements restricted to the $r \times r$ blocks, $R_{\text{add}}^{|r\times r}$, for preconditioning is better than employing only the initially required elements.

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

5 Case studies in science and engineering

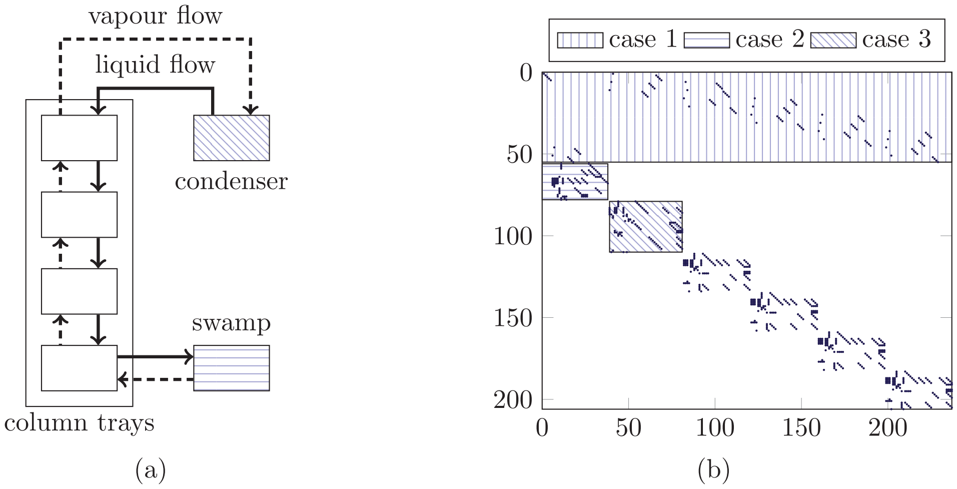
After describing our new coloring algorithms and our approach to combine partial Jacobian computation with preconditioning, we verify their practical relevance by considering several applications from science and engineering. In contrast to the test matrices in the previous chapters, the Jacobian matrices are provided as matrix-vector products and the right-hand sides are also available in these applications. In Sect. 5.1, the partial Jacobian computation is employed to determine sensitivities occurring in a distillation column in process engineering. This sparsity exploitation is already published in [50]. In Sect. 5.2, the full Jacobian computation for stencil-based computations and the partial Jacobian computation for preconditioning are assessed in carbon sequestration. In Sect. 5.3, the flow around an airfoil in aeronautical engineering is investigated. A preconditioned system of linear equations is solved by employing partial Jacobian computation to provide initially, potentially, and additionally required elements. In Sect. 5.4, we consider preconditioned systems while simulating blood flow in biomedical engineering.

5.1 Distillation in process engineering

A model of an industrial distillation column is considered by the institute Aachener Verfahrenstechnik–Process System Engineering at RWTH Aachen University. This model is formulated in CapeML [60], a domain-specific XML-based language used in process engineering. This CapeML model is transformed into another CapeML model using a prototype automatic differentiation tool called ADiCape [6, 49]. The new model is capable of evaluating derivatives of the original model. ADiCape is designed to be a part of the DyOS framework [11] for dynamic optimization in process engineering. This framework is developed by Aachener Verfahrenstechnik–Process System Engineering. Here, we are interested in computing a (proper) subset of the nonzero elements of a sparse Jacobian matrix. The partial Jacobian computation is employed to reduce the computational effort if only a subset is interesting.

Computational models in process engineering are often built up from a number of smaller components, the so-called sub-models, which are connected together to create a more complex structured model. The final compound model is often a subject for further investigations and optimization schemes. Only a subset of the whole system, e.g., one of its sub-models, needs to be investigated in more detail. The nonzero elements of Jacobian matrices illustrate the dependence and sensitivity of

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.



(a) (b)

Figure 5.1: (a) Scheme of an industrial distillation column. (b) Nonzero pattern of the Jacobian matrix where three different subsets of required elements are denoted by case 1 to case 3.

the system output variables. The sensitivity analysis can act as a first attempt to validate the implemented model. The behavior of a model might be not controllable if large sensitivities in the simulation's output are caused by small variations on model parameters. However, to preserve the actual behavior of the whole system, the component of interest may not be regarded separately from the rest of the model description. For such a task, it is sufficient to compute certain parts of the Jacobian matrix rather than all its nonzero elements.

The illustrating example is a computational model arising from an industrial process engineering problem. The model represents a hierarchically built distillation column for separating fluid mixtures of two pure liquids with different boiling points. The model of the distillation column is schematically depicted in Fig. 5.1(a). The distillation column consists of a number of column trays connected with a so-called swamp and a condenser. The mixture is heated so that one fluid boils and is transformed into vapor. The vapor is collected at the condenser and the liquid fluid remains in the swamp. Here, we are interested in investigating three different components separately: the connection component (case 1), the swamp (case 2), and the condenser (case 3). Conceptually, we ask for the influence of selected model parameters on the physical quantities representing the connection component, the swamp, or the condenser.

We are interested in selected nonzero elements of the Jacobian matrix associated with an instance of the distillation column given in Fig. 5.1(a). The sparsity pattern of this 206×237 Jacobian matrix A is shown in Fig. 5.1(b). If the sparsity is not exploited, $p_{\mathrm{d2}} = 237$ colors are needed to compute all nonzero elements of A. The bipartite graph corresponding to the Jacobian matrix was exploited in [51] by using the heuristic D2ColoringRestricted (Alg. 2.1) to compute all nonzero elements, i.e., $E_R = E$. This exploitation reduces the number of colors to $p_{\mathrm{d2}} = 14$. We are

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

now interested in the following three partial Jacobian computations where the required elements consist of all nonzero elements in specified blocks [50]. These blocks are emphasized in Fig. 5.1(b) using different shadings. The number of colors are as follows:

case	block	p_{d2}
1	(1:55,1:237)	2
2	(56:78,1:38)	11
3	(79:110,39:81)	10

Compared to computing all nonzero elements by D2ColoringRestricted with $p_{\mathrm{d2}} = 14$, the partial Jacobian computation reduces the number of colors p_{d2} further to $p_{\mathrm{d2}} = 2$ for case 1, $p_{\mathrm{d2}} = 11$ for case 2, and $p_{\mathrm{d2}} = 10$ for case 3.

5.2 Carbon sequestration in geosciene

The Institute for Applied Geophysics and Geothermal Energy (GGE) at RWTH Aachen University investigates the injection of CO_2 in the underground. The practical relevance arises from storing CO_2 in a reservoir. The injection is simulated by a two-phase flow model in porous media, e.g., sandstone. The two phases are a wetting and a non-wetting phase; in particular, water and gas. Before implementing a code in Fortran for high-performance computing, a prototype for three space dimensions created in MATLAB is investigated. However, the illustrating test instance is a two-dimensional simulation.

A system of coupled non-linear partial differential equations formulates the two-phase flow. These equations are semi-discretized in space using a stencil-based approach on an $M \times N$ grid. The boundary of the domain is specified by Dirichlet or Neumann boundary conditions. The time integration with the implicit Euler method leads to a system of non-linear algebraic equations

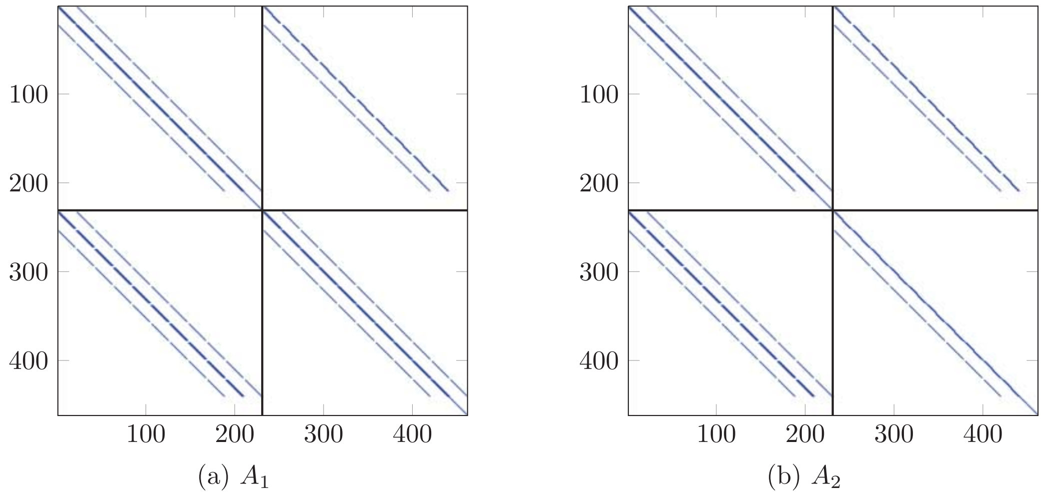
$$F(u) = 0 \ \text{ with } \ u = \begin{pmatrix} p_w \\ S_n \end{pmatrix} \in \mathbb{R}^{2MN} \ \text{ and } \ F = \begin{pmatrix} F_1 \\ F_2 \end{pmatrix} \in \mathbb{R}^{2MN},$$

where the variable $p_w \in \mathbb{R}^{MN}$ is the pressure for the wetting phase and $S_n \in \mathbb{R}^{MN}$ is the non-wetting saturation. In every time step, this system is solved by Newton's method. Therefore, in every Newton iteration a $2MN \times 2MN$ Jacobian matrix

$$A = \left(\begin{array}{c|c} \dfrac{\partial F_1}{\partial p_w} & \dfrac{\partial F_1}{\partial S_n} \\ \hline \dfrac{\partial F_2}{\partial p_w} & \dfrac{\partial F_2}{\partial S_n} \end{array} \right)$$

is determined with a transformed version of the original function F. For this transformation, the AD tool ADiMat [7] is applied.

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.



(a) A_1 (b) A_2

Figure 5.2: Sparsity pattern of Jacobian matrices A_1 and A_2 differently discretized by five-point stencil $\mathcal{N}_{5\mathrm{pt}}$ and two-point stencil $\mathcal{N}_{2\mathrm{pt}}$.

This Jacobian matrix is divided into four quadrants: the derivative $\partial F_1/\partial p_w$ in the top left quadrant, $\partial F_1/\partial S_n$ in the top right, $\partial F_2/\partial p_w$ in the bottom left, and $\partial F_2/\partial S_n$ in the bottom right. The sparsity patterns of two illustrating examples are depicted in Fig. 5.2. These examples arise from two different discretizations resulting in 462×462 Jacobian matrices A_1 and A_2. In A_1, there are 3,236 nonzero elements whereas there are 2,666 nonzero elements in A_2. The four quadrants are indicated by crossed black lines. In these quadrants different stencils are used: the five-point stencil $\mathcal{N}_{5\mathrm{pt}}$ and stencils for an upwind scheme whose grid points are a subset of the grid points of the stencil $\mathcal{N}_{5\mathrm{pt}}$. The actually employed stencil is influenced by the input parameters. The Jacobian matrix A_1 in Fig. 5.2(a) is based on the five-point stencil $\mathcal{N}_{5\mathrm{pt}}$ in the north west, south east, and south west quadrant. In the north east, the two-point stencil $\mathcal{N}_{2\mathrm{pt}} = \{(m,n),(m,n-1)\}$ with the center (m,n) is used. In Fig. 5.2(b), the stencil $\mathcal{N}_{2\mathrm{pt}}$ is also employed in the south east quadrant of Jacobian matrix A_2 instead of stencil $\mathcal{N}_{5\mathrm{pt}}$. In particular, this leads to the following discretizations of the quadrants

$$A_1 = \left(\begin{array}{c|c} \mathcal{N}_{5\mathrm{pt}} & \mathcal{N}_{2\mathrm{pt}} \\ \hline \mathcal{N}_{5\mathrm{pt}} & \mathcal{N}_{5\mathrm{pt}} \end{array}\right) \quad \text{and} \quad A_2 = \left(\begin{array}{c|c} \mathcal{N}_{5\mathrm{pt}} & \mathcal{N}_{2\mathrm{pt}} \\ \hline \mathcal{N}_{5\mathrm{pt}} & \mathcal{N}_{2\mathrm{pt}} \end{array}\right).$$

This application is used to study both full Jacobian computation and partial Jacobian computation for preconditioning: First, we suppose that all nonzero elements of the Jacobian matrix can be stored. In this setting, we assess the number of colors required to compute all nonzero elements of the Jacobian matrices. Second, we suppose that we are not able to store all nonzero elements and evaluate the preconditioning techniques, which are introduced in the previous chapter, with different stencil combinations.

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

In the first part, the structure of Jacobian matrices is exploited for full Jacobian computation. Before using a coloring algorithm for stencil-based computation, we apply the coloring heuristic D2ColoringRestricted (Alg. 2.1) with $E_R = E$ to the bipartite graphs corresponding to the Jacobian matrices. This coloring heuristic determines a coloring with $p_{\mathrm{d2}} = 13$ colors for the Jacobian matrix A_1 in Fig. 5.2(a) and $p_{\mathrm{d2}} = 11$ colors for the Jacobian matrix A_2 in Fig. 5.2(b). For the stencil-based computation, rather than using the vanilla exploitation described in Sect. 3.1, we employ a hierarchical approach to reduce the number of colors. Therefore, the Jacobian matrices are sub-divided into a left part from column 1 to 231 and a right part from column 232 to 462. Moreover, it is enough for both halves to compute a coloring for the top left and bottom right quadrants, respectively. That is, using the coloring for the top left quadrant of the Jacobian matrix is sufficient to compress the left half of the matrix because the sparsity pattern in the top left and bottom left quadrants are identical. For the right half, coloring only the bottom right quadrant is sufficient since a coloring for a stencil is always suitable for a sub-stencil. The algorithms colorVSepAll and getTile are separately employed to the top left and bottom right quadrant. By accumulating the number of colors for the top left and bottom right quadrants, the minimal coloring needs only $p_{\mathrm{d2}} = 5 + 5 = 10$ colors to determine all nonzero elements of A_1 instead of $p_{\mathrm{d2}} = 13$. We consider the Jacobian matrix A_2 in Fig. 5.2(b). In the lower and upper quarter of the right half the two-point stencil is used. Rather than employing the five-point stencil $\mathcal{N}_{5\mathrm{pt}}$ to discretize the bottom right quadrant of the Jacobian matrix in Fig. 5.2(a), the simulation code employs the two-point stencil. If the neighborship relation is given as the two-point stencil $\mathcal{N}_{2\mathrm{pt}}$, a minimal coloring consists of $p_{\mathrm{d2}} = 2$ colors. Therefore, the number of colors can be reduced to $p_{\mathrm{d2}} = 5 + 2 = 7$. The differences between the minimal colorings and the colorings computed by a heuristic are 10 to 13 colors and 7 to 11 colors. Moreover, the same minimal coloring can be used to color a grid with increased grid size.

In the second part, the combination of preconditioning and partial Jacobian computation is evaluated. We suppose that not all nonzero elements of the Jacobian matrix can be stored. Hence, the access to the Jacobian matrix A is provided as a matrix-vector product $A \cdot v$. This Jacobian matrix is the coefficient matrix of the system of linear equations $A \cdot x = b$. The iterative solver Bi-CGSTAB is employed to solve this system. This solver stops if the relative residual norm is less than the threshold $\epsilon = 10^{-7}$ or if the number of matrix-vector products exceeds $\textsc{mvp} = 200$. To reduce the runtime, we determine a preconditioner using the incomplete LU factorization with level $\ell = 0$, ILU(0), i.e., no fill-in elements occur during the factorization. We consider the Jacobian matrix whose sparsity pattern is given in Fig. 5.2(a). The initially required elements R_{init} are specified in the top left and bottom right quadrants by the stencil combinations $\mathcal{N}_{5,1\mathrm{pt}}$ and $\mathcal{N}_{5,3\mathrm{pt}}$ for the full Jacobian matrix. Recall that the stencil combination $\mathcal{N}_{5,1\mathrm{pt}}$ specifies the nonzero elements of the main diagonal and the stencil combination $\mathcal{N}_{5,3\mathrm{pt}}$ the nonzero elements of the main diagonal and the diagonals above and below. We assume that these initially required elements can be stored.

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

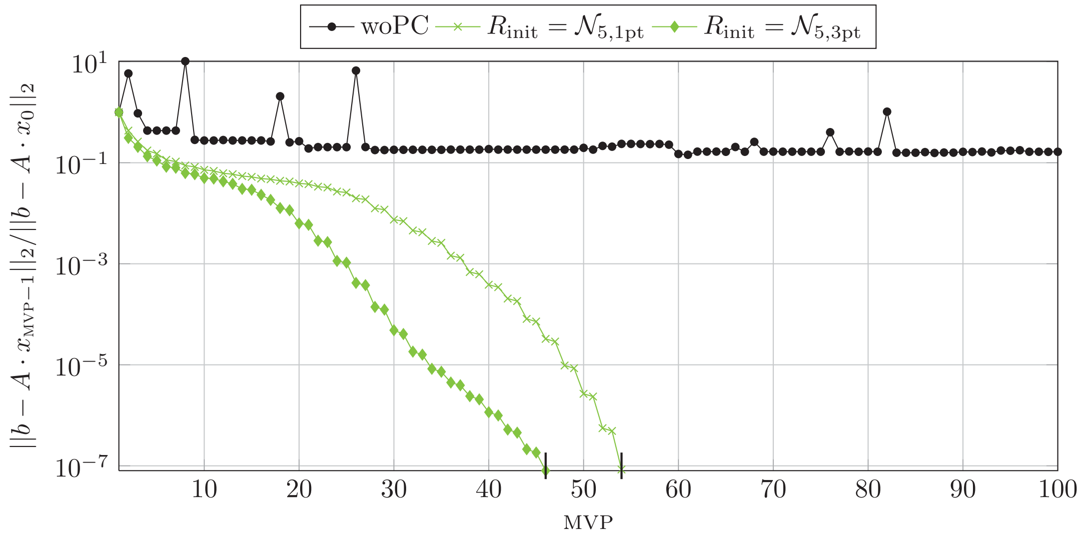


Figure 5.3: Convergence history of Bi-CGSTAB while solving a system of linear equations without preconditioning (woPC) and with the initially required elements R_{init} specified by stencil combinations $\mathcal{N}_{5,1\text{pt}}$ and $\mathcal{N}_{5,3\text{pt}}$.

The simulation of the CO_2 injection in the underground consists of several time steps. The convergence history for solving a system of linear equations arising in the first Newton iteration of the first time step is depicted in Fig. 5.3. The plot is cut after MVP = 100 matrix-vector products. The system is solved without preconditioning and using the nonzero elements specified by $\mathcal{N}_{5,1\text{pt}}$ and $\mathcal{N}_{5,3\text{pt}}$ for preconditioning. Solving the system of linear equations, the solution is divergent without using preconditioning. Using the stencil combination $\mathcal{N}_{5,3\text{pt}}$ leads to a better convergence rate than using $\mathcal{N}_{5,1\text{pt}}$. To obtain the same relative residual norm, MVP = 54 matrix-vector products are needed using $\mathcal{N}_{5,1\text{pt}}$ and MVP = 46 using $\mathcal{N}_{5,3\text{pt}}$. Recall from Sect. 3.1.2 that $p_{\text{d2}} = 2$ and $p_{\text{d2}} = 4$ colors are needed to determine R_{init} specified by $\mathcal{N}_{5,1\text{pt}}$ and $\mathcal{N}_{5,3\text{pt}}$, respectively. Comparing the overall number of matrix-vector products, MVP = 54 + 2 = 56 are needed for $\mathcal{N}_{5,1\text{pt}}$ and MVP = 46 + 4 = 50 for $\mathcal{N}_{5,3\text{pt}}$. Thus, there is a small reduction of matrix-vector products by using more nonzero elements for preconditioning.

For general graphs, it was possible to obtain a lot of potentially and additionally required elements without increasing the number of colors. For our example, these elements do not occur in the top left and bottom right quadrant. That is, the minimal coloring consists of the smallest number of colors; thus, in this example, no extra nonzero element can be determined in these quadrants of the Jacobian matrix. Potentially and additionally required elements occur only in the top right and bottom left quadrant. These elements do not lead to a solution with less matrix-vector products. For this reason, we do not consider these elements further.

Finally, we compare how many Newton iterations NI and, especially, matrix-vector products MVP are needed to compute several time steps of the simulation. This comparison is carried out to evaluate the advantage of combining the preconditioning

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

	$t = 1$		$t = 2$		$t = 3$		$t = 4$		$t = 5$		$t = 1 \ldots 5$	
	NI	MVP	NI	MVP	NI	MVP	NI	MVP	NI	MVP	NI	MVP
woPC	12	1,712	12	1,968	14	2,192	15	2,231	13	2,029	66	10,132
$\mathcal{N}_{5,1\text{pt}}$	5	244	6	299	6	303	5	278	5	293	27	1,417
$\mathcal{N}_{5,3\text{pt}}$	5	197	6	253	5	208	5	220	5	239	26	1,117

Figure 5.4: Number of Newton iterations NI and accumulated number of matrix-vector products MVP in time step t by applying the iterative solver without preconditioning and using the nonzero elements specified by $\mathcal{N}_{5,1\text{pt}}$ and $\mathcal{N}_{5,3\text{pt}}$. Accumulated numbers for all five time steps are given in the last columns.

with the partial Jacobian computation. Newton's method in time step t stops if the relative residual $||u_{\text{NI}} - u_{\text{NI}-1}||_2$ is less than the threshold 10^{-6} or if the number of Newton iterations NI exceeds NI $= 20$. The settings for the iterative solver remain unmodified compared to solving systems of linear equations in the previous paragraphs. The number of Newton iterations and the accumulated number of matrix-vector products are compared using the iterative solver with and without preconditioning. Therefore, we consider the time steps $t = 1$ to $t = 5$ in Table 5.4. Solving the systems of linear equations without preconditioning, the number of matrix-vector products is significantly higher than using the described preconditioning methods. For time step $t = 1$, the number of matrix-vector products is reduced from 1,712 to 244 and 197, respectively. For time step $t = 2$ to $t = 5$, the relation is comparable. In the last two columns, the number of Newton iterations and matrix-vector products are accumulated for all five time steps. The reduction of matrix-vector products for time steps $t = 1 \ldots 5$ is from 10,132 to 1,417 when the required elements are specified by $\mathcal{N}_{5,1\text{pt}}$, i.e., a reduction to 14%. When the required elements are specified by $\mathcal{N}_{5,3\text{pt}}$, there is a further reduction to MVP $=$ 1,117 or 11%.

Using full Jacobian computation for the stencil-based computation, the number of colors is reduced. Carrying out preconditioning and partial Jacobian computation, the number of matrix-vector products is significantly decreased using specific stencil combinations.

5.3 Flow field around an airfoil in aeronautical engineering

Recall the motivating example in Sect. 4.3 from the field of aeronautical engineering. The flow field around an airfoil—for instance, the velocity and pressure distribution—is simulated by the flow solver Quadflow [9]. This solver is developed at the Department of Mechanics and the Institute of Geometry and Practical Mathematics at RWTH Aachen University within the collaborative research centre CRC 401 ("Flow Modulation and Fluid-Structure Interaction at Airplane Wings"). This simulation code solves Euler and Navier-Stokes equations for compressible fluid flows in two or

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

three dimensions. The flow field is computed in a steady flow simulation using an adaptive approach. Before solving non-linear systems by an approximate Newton's method, the outer part of the considered NACA0012 airfoil is discretized by an adaptive finite volume method. The coefficient matrices of the systems of linear equations are Jacobian matrices. More details about employing automatic differentiation to obtain Jacobian matrices in Quadflow are given in [5]. The size of the Jacobian matrices occurring in two-dimensional flow simulations around the NACA0012 airfoil are small enough, but shifting to a three-dimensional airfoil can exceed the available memory. Therefore, reducing the number of nonzero elements is mandatory.

The reduction in the number of colors using partial Jacobian computation for determining the main diagonal elements is evaluated in [42]. It was demonstrated that computing solely a subset of the nonzero elements reduces the number of colors in practice. The number of iterations for solving systems of linear equations was not studied in that work. We continue on these first results and study the approach for choosing the required elements as described in the previous chapter. In Quadflow, the Jacobian matrix is composed of local Jacobian matrices, each corresponding to a volume cell. In the following, the test instance is a 1600×1600 Jacobian matrix with 30,598 nonzero elements. A system of linear equations $A \cdot x = b$ is solved, where the coefficient matrix A is this Jacobian matrix. The iterative solver GMRES is used without restart to solve the system of linear equations, and it stops if the relative residual norm is less than the threshold $\epsilon = 10^{-6}$. To reduce the runtime, we determine a preconditioner using the incomplete LU factorization with level $\ell = 0$, ILU(0), i.e., no fill-in elements occur during the factorization.

We assess the combination of preconditioning and partial Jacobian computation. To solve the system of linear equations without preconditioning with a relative residual norm smaller than the threshold $\epsilon = 10^{-6}$, MVP $= 416$ matrix-vector products are needed. The relative residual norm is plotted versus the matrix-vector products in Fig. 5.5. The plot is cut after matrix-vector product MVP $= 100$. For preconditioning, we start with using the initially required elements $R_{\text{init}} = $ BLKDIAG$(A, 10)$, i.e., the initially required elements are the nonzero elements of the 10×10 blocks on the block diagonal. The coloring to determine R_{init} is computed by heuristic D2COLORINGRESTRICTED (Alg. 2.1) and consists of $p_{\text{d2}} = 28$ colors. Using R_{init} for preconditioning, the iterative solver computes a solution after MVP $= 76$ matrix-vector products. Due to the extension of the required elements with the potentially required elements R_{pot} determined by DETPOTREQELEMD2 (Alg. 4.2), the number of matrix-vector products is significantly reduced to MVP $= 31$. The number of nonzero elements is given at a glance:

$\lvert R_{\text{init}} \uplus F \rvert$	$\lvert R_{\text{init}} \uplus (F \cup R_{\text{pot}}) \rvert$	$\lvert R_{\text{init}} \uplus (F \cup R_{\text{add}\star}) \rvert$	$\lvert R_{\text{init}} \uplus (F \cup R_{\text{add}\circ}) \rvert$
10,991	22,726	12,338	17,006.

When the preconditioner is created from $R_{\text{init}} \uplus R_{\text{pot}}$ instead of solely from R_{init}, the number of nonzero elements is doubled from 10,991 to 22,726. If R_{pot} is substituted by R_{add}, the number of matrix-vector products is increased to MVP $= 66$ or

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

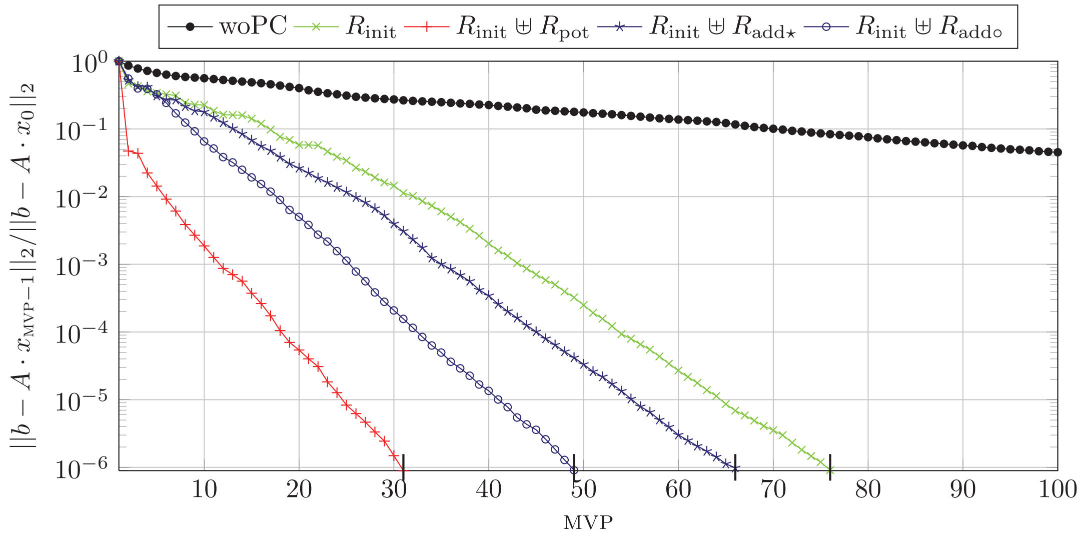


Figure 5.5: Relative residual norms for solving a system of linear equations plotted over matrix-vector products MVP. The system is solved without preconditioning (woPC) and using R_{init}, $R_{\text{init}} \uplus R_{\text{pot}}$, $R_{\text{init}} \uplus R_{\text{add}\star}$, and $R_{\text{init}} \uplus R_{\text{add}\circ}$ with $R_{\text{init}} = \text{BLKDIAG}(A, 10)$ for preconditioning.

MVP = 49, depending on the employed algorithm for determining R_{add}. The algorithm DETADDREQELEM computes the additionally required elements denoted by the symbol $R_{\text{add}\star}$ and DETADDREQELEMCYCLE the additionally required elements $R_{\text{add}\circ}$. The number of nonzero elements is reduced to 12,338 or 17,006, respectively. In summary, the number of matrix-vector products is decreased by using the initially, potentially, and additionally required elements for preconditioning. Especially, the additionally required elements cause a good reduction of the matrix-vector products without increasing the number of nonzero elements too much.

5.4 Blood flow in biomedical engineering

The finite element flow solver XNS [3] is developed for large-scale simulations by the Chair for Computational Analysis of Technical Systems (CATS) at RWTH Aachen University. XNS uses a stabilized space-time Galerkin/least-squares discretization. Among other areas in science and engineering, XNS is able to model viscoelastic fluids such as blood. Under some assumptions the motion of blood can be described by the incompressible Navier-Stokes equations. The iterative solver GMRES is employed to solve the systems of linear equations.

In a previous project, we carried out sensitivity analyses for blood flow in artificial bypasses [52–54] by applying automatic differentiation to XNS. In particular, the AD tool Adifor2 [8] was employed. Here, the blood flow through the human aorta is simulated as non-steady state. Computing the full Jacobian matrix is based on previous work and was part of the interdisciplinary project "Towards a Computational Model

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

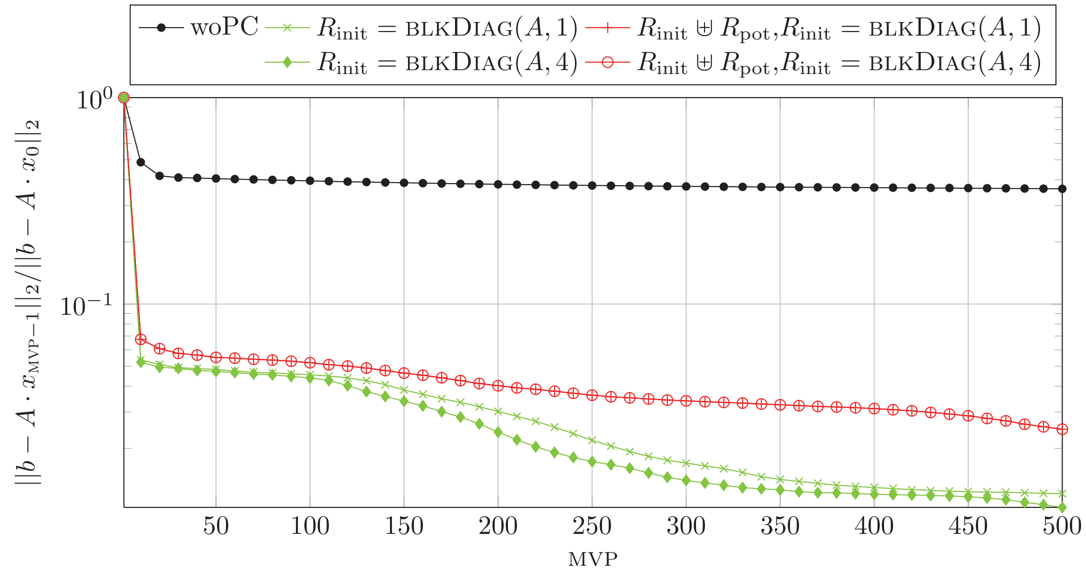


Figure 5.6: Relative residual norms for solving a system of linear equations plotted over matrix-vector products MVP. The system is solved without preconditioning (woPC) and using R_{init}, $R_{\text{init}} \uplus R_{\text{pot}}$ with $R_{\text{init}} = \text{BLKDIAG}(A, 1)$, and $R_{\text{init}} \uplus R_{\text{pot}}$ with $R_{\text{init}} = \text{BLKDIAG}(A, 4)$ for preconditioning.

of Blood Flow in the Left Human Heart, Aorta and Connecting Vessels" funded by JARA-HPC. Rather than performing a sensitivity analysis, we compute Jacobian matrices for Newton's method. We assume that there is not enough memory available to store all nonzero elements of the Jacobian matrix. First, while solving a system of linear equations, we assess the benefit of using initially and potentially required elements for preconditioning. Second, we consider the convergence rate while solving a non-linear system using Newton's method.

A system of linear equations $A \cdot x = b$ is solved where the coefficient matrix A is a 290,392 × 290,392 Jacobian matrix with 31,191,872 nonzero elements. The iterative solver GMRES is used without restart to solve this system. This solver stops if the relative residual norm is less than the threshold $\epsilon = 10^{-8}$ or if the number of matrix-vector products exceeds MVP = 500. To reduce the runtime, we determine a preconditioner using the incomplete LU factorization with level $\ell = 0$, ILU(0). The initially required elements are either the main diagonal elements, $R_{\text{init}} = \text{BLKDIAG}(A, 1)$, or the nonzero elements in the 4×4 blocks on the block diagonal, $R_{\text{init}} = \text{BLKDIAG}(A, 4)$. The convergence plot with the relative residual norms is given in Fig. 5.6. These norms are plotted over the matrix-vector products in steps of 10. If the system of linear equations is solved without preconditioning, the convergence rate is quite slow. A preconditioner determined by using the initially required elements $R_{\text{init}} = \text{BLKDIAG}(A, 1)$ improves the convergence rate of the solver. Although this rate of the iterative solver is quite slow compared to the other case studies, the difference of the relative residual norms is more than a factor of 10. Rather than $p_{\text{d2}} = 272$ colors to determine the full Jacobian matrix, $p_{\text{d2}} = 80$ colors are needed to compute R_{init}.

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

Next, the 4 × 4 blocks on the diagonal are considered as initially required elements, $R_{\text{init}} = \textsc{blkDiag}(A, 4)$. The number of colors does not increase due to the structure of the Jacobian matrix. This matrix consists of full 4 × 4 blocks. Computing the diagonal elements of such a block is as costly as computing all nonzero elements of this block. Unfortunately, the nonzero elements on the main diagonal of the Jacobian matrix seem to be very dominant so that there is only a small difference in the convergence rate starting with $\textsc{mvp} = 100$ between the initially required elements for block size $k = 1$ and $k = 4$. Due to the structure of the matrix, the potentially required elements R_{pot} for both sets of initially required elements, the main diagonal elements and the nonzero elements in the 4 × 4 blocks, are identical. That is, $R_{\text{init}} \uplus R_{\text{pot}}$, $R_{\text{init}} = \textsc{blkDiag}(A, 1)$, is identical to $R_{\text{init}} \uplus R_{\text{pot}}$, $R_{\text{init}} = \textsc{blkDiag}(A, 4)$. With the considered Jacobian matrix, the use of $R_{\text{init}} \uplus R_{\text{pot}}$ causes a slightly worse convergence rate than just using R_{init}. We do not consider additionally required elements, because it seems most unlikely that their use improves the convergence behavior compared to R_{pot}. The number of nonzero elements in R_{init} is $\textsc{nnz} = 290{,}392$ for block size $k = 1$ and $\textsc{nnz} = 1{,}161{,}568$ for $k = 4$. The number of nonzero elements in $R_{\text{init}} \uplus R_{\text{pot}}$ for block size $k = 1$ and $k = 4$ is $\textsc{nnz} = 7{,}358{,}912$. Recall that ILU(0) is employed; thus, no fill-in elements occur during the factorization.

After considering the convergence rate for solving the system of linear equations using preconditioning, we continue with the enclosing non-linear systems $F(u) = 0$. These non-linear systems are solved using Newton's method. The employed iterative solver GMRES for solving the systems of linear equations is used without preconditioning and with the already described preconditioners. We do not change the originally provided settings for the simulation; therefore, GMRES is used with restart 100. The iterative solver stops if the relative residual norm is less than the threshold $\epsilon = 10^{-8}$ or if the number of matrix-vector products exceeds $\textsc{mvp} = 500$. In every of the six considered time steps, the iterative solver always stops after reaching the maximum number of matrix-vector products. That is, the number of matrix-vector products is the same for solving each system of linear equations—with and without preconditioning. However, the convergence behavior changes depending on the preconditioning. The convergence behavior is given in Fig. 5.7. Using no preconditioner, Newton's method stagnates after $\textsc{ni} = 3$ Newton iterations in the first time step. In the following time steps, it stagnates after $\textsc{ni} = 4$. When $R_{\text{init}} = \textsc{blkDiag}(A, 1)$, $R_{\text{init}} = \textsc{blkDiag}(A, 4)$, or $R_{\text{init}} \uplus R_{\text{pot}}$, $R_{\text{init}} = \textsc{blkDiag}(A, 1)$ are used, the convergence rate is faster. Up to $\textsc{ni} = 6$ Newton iterations, there is no stagnation in any time step. The preconditioners lead to quite similar convergence rates. In time steps $t = 1$, $t = 3$, and $t = 4$, the preconditioner which is based on $R_{\text{init}} = \textsc{blkDiag}(A, 1)$ leads to the best convergence, and, in the other time steps, it is the preconditioner which is based on $R_{\text{init}} \uplus R_{\text{pot}}$.

In summary, solving the non-linear systems and the systems of linear equations while simulating the blood flow through the aorta, the convergence behavior is improved by using at least the main diagonal elements for preconditioning.

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

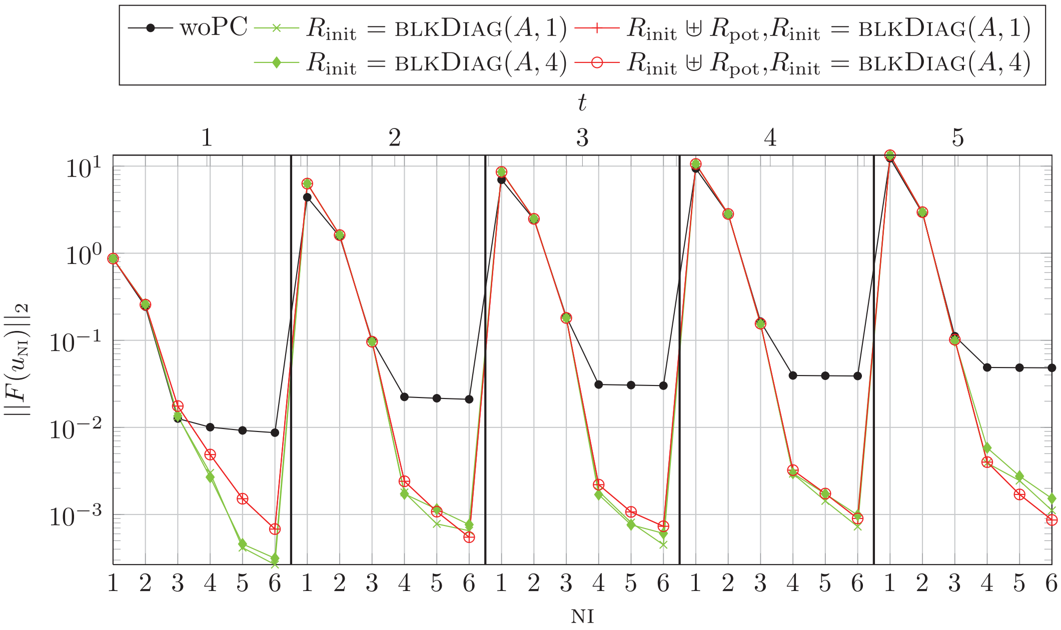


Figure 5.7: Absolute residual per iteration NI of Newton's method while solving nonlinear systems in time steps $t = 1, \dots, 5$. The systems of linear equations are solved without preconditioning (woPC) and using R_{init}, $R_{\text{init}} \uplus R_{\text{pot}}$ with $R_{\text{init}} = \textsc{blkDiag}(A,1)$, and $R_{\text{init}} \uplus R_{\text{pot}}$ with $R_{\text{init}} = \textsc{blkDiag}(A,4)$ for preconditioning.

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

6 Concluding summary

Memory consumption and computational effort are limiting factors in simulation and optimization. To address these challenges, in a first step, the computational effort to determine nonzero elements of a sparse Jacobian matrix is reduced using full and partial Jacobian computation. In a second step, for solving systems of linear equations, preconditioning is combined with the partial Jacobian computation to reduce the number of Jacobian matrix-vector products and the number of nonzero elements employed for preconditioning.

To reduce the computational effort for full and partial Jacobian computation, several new coloring algorithms are introduced in Chap. 3. For stencil-based computations, the sparsity of Jacobian matrices has been exploited to date in an unfavorable way by using either coloring algorithms for general graphs or explicit coloring formulae whose development is time-consuming. Instead, an exact sub-exponential divide-and-conquer coloring algorithm using separators is introduced. This new algorithm exploits the properties of the employed stencils and the regular grid. The runtime of this algorithm is much lower than using the exhaustive search, and the resulting colorings are again minimal. Compared to a coloring heuristic for general graphs, the runtime of this algorithm is often better and the number of colors is always minimal. To overcome the sub-exponential complexity of the divide-and-conquer coloring algorithm, a grid-size independent algorithm with linear complexity is presented. Although this approach is not proved to be applicable in general, it is successfully employed on all considered stencils. For general graphs in partial Jacobian computation, a two-sided coloring heuristic is introduced to combine rows and columns to linear combinations. It is demonstrated that this algorithm is comparable to state-of-the-art coloring heuristics for full Jacobian computation and its two-sided colorings for partial Jacobian computation are better than one-sided colorings. Using suitable vertex orderings for coloring heuristics is really important to decrease the number of colors. In this thesis, the influence of these vertex orderings is only sketched in brief. For specific matrix structures, the benefit of two-sided colorings compared to minimal one-sided colorings is assessed in terms of the number of colors. First, computing block diagonal elements of general Jacobian matrices is considered. Using the partial Jacobian computation for this structure, the number of colors may be reduced by using two-sided colorings. However, for the special case of the main diagonal, it is shown that two-sided colorings are not better than a minimal one-sided coloring. For stencil-based computations, a lower bound on the number of colors for two-sided colorings is introduced. There are stencils for which two-sided colorings cannot be better than minimal one-sided colorings. For other stencils, the existence of such a two-sided coloring seems unlikely.

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

In Chap. 4, the interplay between the partial Jacobian computation and preconditioning is evaluated. That is, opposed to currently employed methods for preconditioning, only a subset of the nonzero elements is used to determine a preconditioner. These elements are classified as initially, potentially, and additionally required elements. First, the initially required elements are selected, and a coloring is computed which is sufficient to determine these elements. Next, new algorithms are introduced to select further nonzero elements which are determinable with the same coloring. These elements are denoted as potentially required elements. The additionally required elements are a subset of the potentially required elements which do not cause any fill-in element while obtaining the preconditioner. Thus, these additionally required elements can be stored in memory. These elements are further restricted to obtain a beneficial block structure which is good for solving systems of linear equations in parallel. Overall, the additionally required elements do not cause any extra fill-in element, and the number of colors is not increased compared to the initially required elements. The number of colors to determine the nonzero elements and the number of matrix-vector products to solve a (preconditioned) system of linear equations are evaluated. In summary, the preconditioning is successfully combined with partial Jacobian computation to save computational effort and manage the limited memory.

The practical relevance of the previously described approaches is verified in Chap. 5 using application-oriented simulations. The partial Jacobian computation is useful to carry out a sensitivity analysis for a distillation column in process engineering. In carbon sequestration in applied geophysics, the new coloring algorithms for stencil-based computations and the combination of the partial Jacobian computation with preconditioning are successfully applied to decrease the computational effort. Fluid dynamic applications in aeronautical and biomedical engineering are used to show that the computational effort to solve systems of linear equations is decreased by using the initially, potentially and additionally required elements for preconditioning.

There are still some open issues which could be considered for further work. The universal applicability of the linear-time coloring algorithm for regular grids could be verified. The coloring algorithms for grids could be extended to cover non-regular grids and adaptivity. For the two-sided coloring algorithm for general Jacobian matrices, the influence of vertex orderings could be studied in more detail. The combination of preconditioning and partial Jacobian computation with respect to automatic differentiation provides a lot of opportunities for further investigations. Currently, a coloring for the initially required elements is determined, and then the additionally required elements are chosen. Instead, an approach for simultaneously combining the coloring and the selection employing the bipartite graph could be developed.

In summary, the sparsity of Jacobian matrices is successfully exploited using graph models and algorithms to decrease the computational effort. Furthermore, a standard preconditioning technique is adapted to deal with the available memory and overcome the excessive memory consumption. These approaches enable to simulate and optimize large-scale applications previously exceeding the resources.

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

Bibliography

[1] N. Alon, P. Seymour, and R. Thomas. A separator theorem for nonplanar graphs. *Journal of the American Mathematical Society*, 3(4):801–808, 1990.

[2] P. R. Amestoy, I. S. Duff, Y. Robert, F.-H. Rouet, and B. Uçar. On computing inverse entries of a sparse matrix in an out-of-core environment. Technical Report TR/PA/10/59, CERFACS, Toulouse, France, 2010.

[3] M. Behr and T. E. Tezduyar. Finite element solution strategies for large-scale flow simulations. *Computer Methods in Applied Mechanics and Engineering*, 112(1-4):3–24, 1994.

[4] M. Benzi. Preconditioning techniques for large linear systems: A survey. *Journal of Computational Physics*, 182(2):418–477, 2002.

[5] C. H. Bischof, H. M. Bücker, and A. Rasch. Enabling technologies for robust high-performance simulations in computational fluid dynamics. In W. Schröder, editor, *Summary of Flow Modulation and Fluid-Structure Interaction Findings*, volume 109 of *Notes on Numerical Fluid Mechanics and Multidisciplinary Design*, pages 153–180. Springer Berlin / Heidelberg, 2010.

[6] C. H. Bischof, H. M. Bücker, W. Marquardt, M. Petera, and J. Wyes. Transforming equation-based models in process engineering. In Bücker et al. [12], pages 189–198.

[7] C. H. Bischof, H. M. Bücker, and A. Vehreschild. A macro language for derivative definition in ADiMat. In Bücker et al. [12], pages 181–188.

[8] C. H. Bischof, A. Carle, P. Khademi, and A. Mauer. ADIFOR 2.0: Automatic differentiation of Fortran 77 programs. *IEEE Computational Science & Engineering*, 3(3):18–32, 1996.

[9] F. Bramkamp, P. Lamby, and S. Müller. An adaptive multiscale finite volume solver for unsteady and steady state flow computations. *Journal of Computational Physics*, 197:460–490, 2004.

[10] D. Brélaz. New methods to color the vertices of a graph. *Communications of the ACM*, 22:251–256, 1979.

[11] M. Brendel, J. Oldenburg, M. Schlegel, and K. Stockmann. *DyOS 2.1 User's Guide*. Process Systems Engineering, RWTH Aachen University, Aachen, 2002.

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

[12] H. M. Bücker, G. F. Corliss, P. D. Hovland, U. Naumann, and B. Norris, editors. *Automatic Differentiation: Applications, Theory, and Implementations*, volume 50 of *Lecture Notes in Computational Science and Engineering*. Springer, New York, NY, 2005.

[13] H. M. Bücker and M. Lülfesmann. Combinatorial optimization of stencil-based Jacobian computations. In J. V. Aguiar, editor, *Proceedings of the 2010 International Conference on Computational and Mathematical Methods in Science and Engineering, Almería, Andalucía, Spain, June 26–30, 2010*, volume 1, pages 284–295, 2010.

[14] H. M. Bücker and M. Lülfesmann. Combinatorial optimization of stencil-based Jacobian computations revisited. Preprint of the Institute for Scientific Computing RWTH–CS–SC–10–09, RWTH Aachen University, Aachen, 2010.

[15] A. Calotoiu. Bipartite graph coloring for compressed sparse Jacobian computation. Master's thesis, Computer Science and Engineering Department, University Politehnica of Bucharest, 2009.

[16] T. F. Coleman, B. S. Garbow, and J. J. Moré. Software for estimating sparse Jacobian matrices. *ACM Transactions on Mathematical Software*, 10(3):329–345, 1984.

[17] T. F. Coleman, B. S. Garbow, and J. J. Moré. Software for estimating sparse Hessian matrices. *ACM Transactions on Mathematical Software*, 11(4):363–377, 1985.

[18] T. F. Coleman and J. J. Moré. Estimation of sparse Jacobian matrices and graph coloring problems. *SIAM Journal on Numerical Analysis*, 20(1):187–209, 1983.

[19] T. F. Coleman and A. Verma. The efficient computation of sparse Jacobian matrices using automatic differentiation. *SIAM Journal on Scientific Computing*, 19(4):1210–1233, 1998.

[20] T. F. Coleman and A. Verma. ADMIT-1: automatic differentiation and MATLAB interface toolbox. *Transactions on Mathematical Software*, 26(1):150–175, 2000.

[21] J. K. Cullum and M. Tuma. Matrix-free preconditioning using partial matrix estimation. *BIT Numerical Mathematics*, 46:711–729, 2006.

[22] A. R. Curtis, M. J. D. Powell, and J. K. Reid. On the estimation of sparse Jacobian matrices. *Journal of the Institute of Mathematics and Applications*, 13:117–119, 1974.

[23] T. A. Davis and Y. Hu. The University of Florida Sparse Matrix Collection. *ACM Transactions on Mathematical Software*, 38:1–25, 2011.

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

[24] R. Diestel. *Graph Theory*, volume 173 of *Graduate Texts in Mathematics*. Springer-Verlag, Heidelberg, 2010.

[25] G. Fertin, E. Godard, and A. Raspaud. Acyclic and k-distance coloring of the grid. *Information Processing Letters*, 87(1):51–58, 2003.

[26] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP–Completeness*. Freeman, San Francisco, 1979.

[27] A. H. Gebremedhin, F. Manne, and A. Pothen. What color is your Jacobian? Graph coloring for computing derivatives. *SIAM Review*, 47(4):629–705, 2005.

[28] A. H. Gebremedhin, D. Nguyen, M. M. A. Patwary, and A. Pothen. Colpack: Graph coloring software for sparse derivative matrix computation and beyond. Technical report. Submitted for publication.

[29] D. Goldfarb and P. L. Toint. Optimal estimation of Jacobian and Hessian matrices that arise in finite difference calculations. *Mathematics of Computation*, 43(167):69–88, 1984.

[30] M. Goyal and S. Hossain. Bi-directional determination of sparse Jacobian matrices: Approaches and algorithms. *Electronic Notes in Discrete Mathematics*, 25:73–80, 2006. CTW2006 - Cologne-Twente Workshop on Graphs and Combinatorial Optimization.

[31] A. Griewank and A. Walther. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. Number 105 in Other Titles in Applied Mathematics. SIAM, Philadelphia, PA, 2nd edition, 2008.

[32] B. Grünbaum and G. Shephard. *Tilings and patterns: An introduction*. A Series of books in the mathematical sciences. W.H. Freeman, 1989.

[33] S. Hossain and T. Steihaug. Computing a sparse Jacobian matrix by rows and columns. *Optimization Methods and Software*, 10:33–48, 1998.

[34] S. Hossain and T. Steihaug. Sparsity issues in the computation of Jacobian matrices. In *ISSAC '02: Proceedings of the 2002 International Symposium on Symbolic and Algebraic Computation*, pages 123–130, New York, NY, USA, 2002. ACM Press.

[35] S. Hossain and T. Steihaug. Graph coloring in the estimation of sparse derivative matrices: Instances and applications. *Discrete Applied Mathematics*, 156(2):280–288, 2008.

[36] D. Hysom and A. Pothen. Level-based incomplete LU factorization: Graph model and algorithms. Technical Report UCRL-JC-150789, Lawrence Livermore National Labs, 2002.

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

[37] K. Kawarabayashi and B. Reed. A separator theorem in minor-closed classes. In *Proceedings of the 2010 IEEE 51st Annual Symposium on Foundations of Computer Science*, FOCS '10, pages 153–162, Washington, DC, USA, 2010. IEEE Computer Society.

[38] R. J. Lipton and R. E. Tarjan. A separator theorem for planar graphs. *SIAM Journal on Applied Mathematics*, 36(2):177–189, 1979.

[39] R. J. Lipton and R. E. Tarjan. Applications of a planar separator theorem. *SIAM Journal on Computing*, 9(3):615–627, 1980.

[40] M. Lülfesmann. Graphfärbung zur partiellen Berechnung von Jacobi-Matrizen. Diploma thesis, Department of Computer Science, RWTH Aachen University, 2006. In German.

[41] M. Lülfesmann. Partielle Berechnung von Jacobi-Matrizen mittels Graphfärbung. In *Informatiktage 2007, Fachwissenschaftlicher Informatik-Kongress, 30. und 31. März 2007, B-IT Bonn-Aachen International Center for Information Technology, Bonn*, volume S-5 of *Lecture Notes in Informatics - Seminars*, pages 21–24. Gesellschaft für Informatik e.V., 2007. In German.

[42] M. Lülfesmann. Graphfärbung zur Berechnung benötigter Matrixelemente. *Informatik-Spektrum*, 31(1):50–54, 2008. In German.

[43] M. Lülfesmann and K. Kawarabayashi. Sub-exponential graph coloring algorithm for stencil-based Jacobian computations. Preprint of the Institute for Scientific Computing RWTH–CS–SC–11–01, RWTH Aachen University, Aachen, 2011.

[44] D. Matula, G. Marble, and J. Isaacson. Graph coloring algorithms. In R. Read, editor, *Graph Theory and Computing*, pages 109–122. Academic Press, New York, 1972.

[45] J. A. Meijerink and H. A. van der Vorst. An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix. *Mathematics of Computation*, 31:148–162, 1977.

[46] D. K. Melgaard and R. F. Sincovec. General software for two-dimensional nonlinear partial differential equations. *ACM Transactions on Mathematical Software*, 7(1):106–125, 1981.

[47] G. L. Miller, S.-H. Teng, W. Thurston, and S. A. Vavasis. Geometric separators for finite-element meshes. *SIAM Journal on Scientific Computing*, 19(2):364–386, 1998.

[48] G. N. Newsam and J. D. Ramsdell. Estimation of sparse Jacobian matrices. *SIAM Journal on Algebraic and Discrete Methods*, 4(3):404–417, 1983.

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

[49] M. Petera. *Automatic Differentiation of the CapeML High-Level Language for Process Engineering.* Dissertation, Department of Computer Science, RWTH Aachen University, 2011.

[50] M. Petera, M. Lülfesmann, and H. M. Bücker. Partial Jacobian computation in the domain-specific program transformation system ADiCape. In M. Ganzha and M. Paprzycki, editors, *Proceedings of the International Multiconference on Computer Science and Information Technology, Mrągowo, Poland, October 12–14, 2009*, volume 4, pages 595–599, Los Alamitos, CA, USA, 2009. IEEE Computer Society.

[51] M. Petera, A. Rasch, and H. M. Bücker. Exploiting Jacobian sparsity in a large-scale distillation column. In D. H. van Campen, M. D. Lazurko, and W. P. J. M. van den Oever, editors, *Proceedings of the Fifth EUROMECH Nonlinear Dynamics Conference, ENOC 2005, Eindhoven, NL, August 7–12, 2005*, pages 825–827, Eindhoven, NL, 2005. Eindhoven University of Technology.

[52] M. Probst, M. Lülfesmann, H. M. Bücker, M. Behr, and C. H. Bischof. Sensitivity of shear rate in artificial grafts using automatic differentiation. *International Journal for Numerical Methods in Fluids*, 62(9):1047–1062, 2010.

[53] M. Probst, M. Lülfesmann, M. Nicolai, H. M. Bücker, M. Behr, and C. H. Bischof. Sensitivity of optimal shapes of artificial grafts with respect to flow parameters. *Computer Methods in Applied Mechanics and Engineering*, 199(17–20):997–1005, 2010.

[54] M. Probst, M. Lülfesmann, M. Nicolai, H. M. Bücker, M. Behr, and C. H. Bischof. On the influence of constitutive models on shape optimization for artificial blood pumps. In G. Leugering, S. Engell, A. Griewank, M. Hinze, R. Rannacher, V. Schulz, M. Ulbrich, and S. Ulbrich, editors, *Constrained Optimization and Optimal Control for Partial Differential Equations*, number 160 in International Series of Numerical Mathematics, pages 611–622. Birkhäuser, Basel, 2012.

[55] L. B. Rall. *Automatic Differentiation: Techniques and Applications*, volume 120 of *Lecture Notes in Computer Science*. Springer Verlag, Berlin, 1981.

[56] D. J. Rose and R. E. Tarjan. Algorithmic aspects of vertex elimination on directed graphs. *SIAM Journal on Applied Mathematics*, 34(1):176–197, 1978.

[57] Y. Saad. *Iterative Methods for Sparse Linear Systems.* Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2nd edition, 2003.

[58] Y. Saad and M. H. Schultz. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal on Scientific and Statistical Computing*, 7(3):856–869, 1986.

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

[59] J. M. Tang and Y. Saad. A probing method for computing the diagonal of a matrix inverse. *Numerical Linear Algebra with Applications*, 19(3):485–501, 2012.

[60] L. v. Wedel. CapeML – A Model Exchange Language for Chemical Process Modeling. Technical report, Process Systems Engineering, RWTH Aachen University, 2002.

[61] H. A. van der Vorst. Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems. *SIAM Journal on Scientific and Statistical Computing*, 13:631–644, 1992.

[62] D. J. A. Welsh and M. J. D. Powell. An upper bound for the chromatic number of a graph and its applications to timetabling problems. *The Computer Journal*, 10:85–87, 1967.

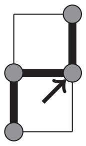
Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

Appendix

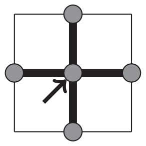
A.1 Overview of stencils

The stencils and stencil combinations which are employed in this thesis are given at a glance. Each center is highlighted by an arrow.

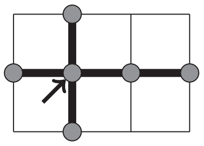
A.1.1 Full Jacobian computation



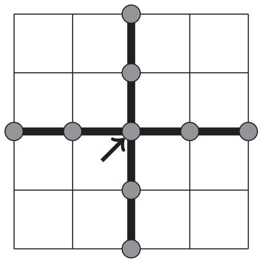
$\mathcal{N}_{4\text{pt}}$



$\mathcal{N}_{5\text{pt}}$

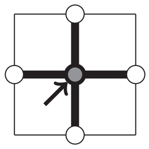


$\mathcal{N}_{6\text{pt}}$

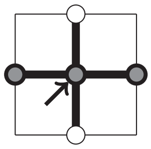


$\mathcal{N}_{9\text{pt}}$

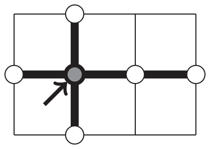
A.1.2 Partial Jacobian computation



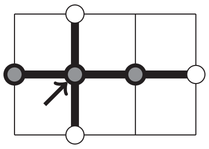
$\mathcal{N}_{5,1\text{pt}}$



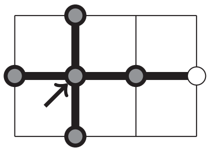
$\mathcal{N}_{5,3\text{pt}}$



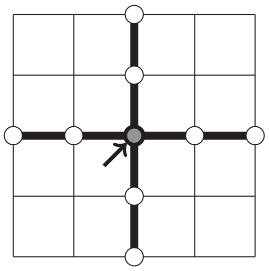
$\mathcal{N}_{6,1\text{pt}}$



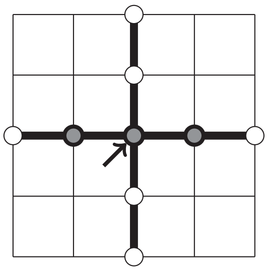
$\mathcal{N}_{6,3\text{pt}}$



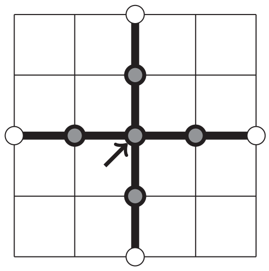
$\mathcal{N}_{6,5\text{pt}}$



$\mathcal{N}_{9,1\text{pt}}$



$\mathcal{N}_{9,3\text{pt}}$



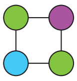
$\mathcal{N}_{9,5\text{pt}}$

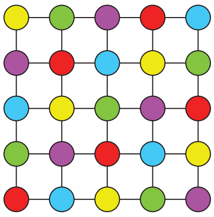
Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

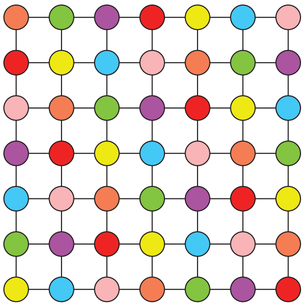
A.2 Colored tiles

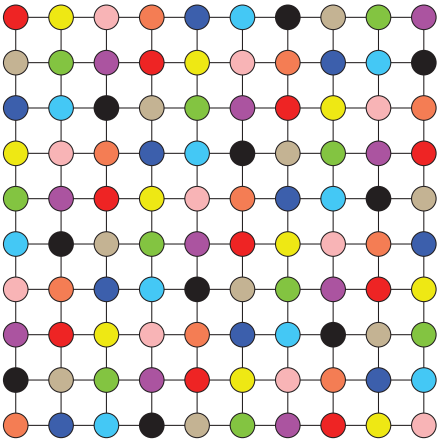
The given colored tiles are computed by the algorithm GETTILE (Alg. 3.4). These colored tiles correspond to the entries in the Tables 3.7(a) and 3.7(b).

A.2.1 Full Jacobian computation

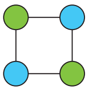

$\mathcal{N}_{4\text{pt}}$

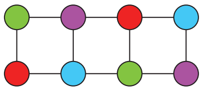

$\mathcal{N}_{5\text{pt}}$

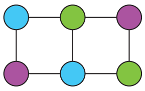

$\mathcal{N}_{6\text{pt}}$

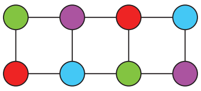

$\mathcal{N}_{9\text{pt}}$

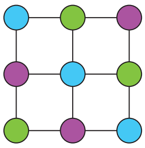
A.2.2 Partial Jacobian computation

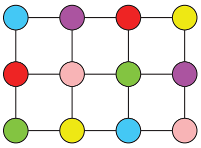

$\mathcal{N}_{5,1\text{pt}}$


$\mathcal{N}_{5,3\text{pt}}$


$\mathcal{N}_{6,1\text{pt}}$


$\mathcal{N}_{6,3\text{pt}}$


$\mathcal{N}_{9,1\text{pt}}$


$\mathcal{N}_{9,3\text{pt}}$

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

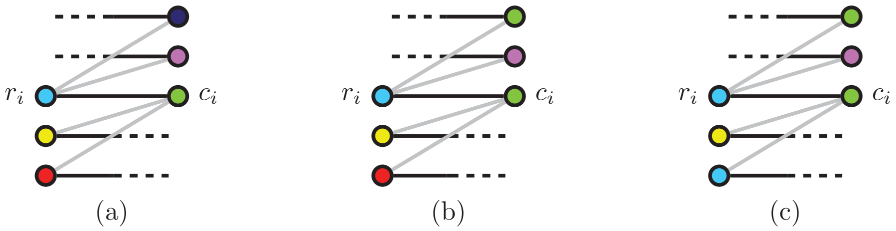
A.3 Proof of Lemma 3.8

Proof. Given a minimal restricted star bicoloring Φ_{sb}, this mapping Φ_{sb} is transformed into a restricted distance-2 coloring Φ_c where the number of colors does not increase. Therefore, two steps are carried out: First, the mapping Φ_{sb} is transformed into a restricted star bicoloring $\Phi^{\star}_{\mathrm{sb}}$ where one of both incident vertices of each edge in E_D is colored with a nonzero color and the other vertex with color zero. Second, the coloring $\Phi^{\star}_{\mathrm{sb}}$ is transformed into a restricted star bicoloring Φ'_{sb} where all vertices $r_i \in V_r$ are colored with color zero, $\Phi'_{\mathrm{sb}}(r_i) = 0, 1 \leq i \leq |V_r|$. We show that the mapping Φ'_{sb} is also a restricted distance-2 coloring of the column vertices.

1. The coloring Φ_{sb} is transformed into another star bicoloring $\Phi^{\star}_{\mathrm{sb}}$ where for each edge $(r_i, c_i) \in E_D, 1 \leq i \leq |V_r|$, holds either $\Phi^{\star}_{\mathrm{sb}}(r_i) \neq 0$ and $\Phi^{\star}_{\mathrm{sb}}(c_i) = 0$ or $\Phi^{\star}_{\mathrm{sb}}(r_i) = 0$ and $\Phi^{\star}_{\mathrm{sb}}(c_i) \neq 0$. Therefore, we iterate over all edges $(r_i, c_i) \in E_D$ with $\Phi_{\mathrm{sb}}(r_i) \neq 0$ and $\Phi_{\mathrm{sb}}(c_i) \neq 0$ and recolor one of both vertices with color zero. To show the correctness of the new star bicoloring $\Phi^{\star}_{\mathrm{sb}}$, we have to consider four cases based on the paths $(c_i, r_i, c_j), \forall c_j \in N_2(c_i, G)$, and $(r_i, c_i, r_k), \forall r_k \in N_2(r_i, G)$:
   - The vertex r_i is differently colored from its distance-2 neighbors r_k; the vertex c_i is differently colored from its distance-2 neighbors c_j. In this scenario, the condition 3a and the condition 3b from Def. 2.12 hold regardless if either vertex r_i or vertex c_i are colored with color zero, i.e, either $\Phi^{\star}_{\mathrm{sb}}(r_i) = 0$ or $\Phi^{\star}_{\mathrm{sb}}(c_i) = 0$. An example for this setting is given in Fig. A.1(a).
   - The vertex r_i is differently colored from its distance-2 neighbors r_k; there is a distance-2 neighbor c_j of the vertex c_i with $\Phi_{\mathrm{sb}}(c_i) = \Phi_{\mathrm{sb}}(c_j)$. This case is illustrated in Fig. A.1(b). The vertex r_i must keep the nonzero color, otherwise condition 3a would be broken, because a path (c_i, r_i, c_j) with $\Phi^{\star}_{\mathrm{sb}}(r_i) = 0$ and $\Phi^{\star}_{\mathrm{sb}}(c_i) = \Phi^{\star}_{\mathrm{sb}}(c_j)$ would occur. However, the vertex c_i can be assigned the color zero without generating an invalid coloring.
   - There is a distance-2 neighbor r_k of the vertex r_i with $\Phi\mathrm{sb}(r_i) = \Phi_{\mathrm{sb}}(r_k)$; the vertex c_i is differently colored from its distance-2 neighbors c_j. That is analogous to the previous case by swapping the row and column vertices.
   - The vertex r_i is identically colored to a distance-2 neighbor r_k; the vertex c_i is identically colored to a distance-2 neighbor c_j. This case is depicted in Fig. A.1(c). The vertices r_i or c_i cannot be colored with the color zero without breaking condition 3a or condition 3b of Def. 2.12. This does not matter, because this case is not a valid restricted star bicoloring due to condition 3c and cannot exist.

   During this transformation step, the number of colors is not increased, because the transformation does not assign any nonzero color.
2. This valid restricted star bicoloring $\Phi^{\star}_{\mathrm{sb}}$ with either $\Phi^{\star}_{\mathrm{sb}}(r_i) \neq 0$ and $\Phi^{\star}_{\mathrm{sb}}(c_i) = 0$ or $\Phi^{\star}_{\mathrm{sb}}(c_i) \neq 0$ and $\Phi^{\star}_{\mathrm{sb}}(r_i) = 0$ is the starting point for the next transformation step. This coloring is transformed into another restricted star bicoloring Φ'_{sb}

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.



(a) (b) (c)

Figure A.1: Star bicolorings Φ_{sb} for vertices $r_i \in V_r$ and $c_i \in V_c$ and its distance-2 neighbors.

with $\Phi'_{\text{sb}}(r_i) = 0$, $1 \leq i \leq |V_r|$. The transformation works as follows: We iterate over the edges $(r_i, c_i) \in E_D$ with $\Phi^\star_{\text{sb}}(r_i) \neq 0$ and $\Phi^\star_{\text{sb}}(c_i) = 0$ and swap $\Phi^\star_{\text{sb}}(c_i)$ and $\Phi^\star_{\text{sb}}(r_i)$. Hence, we obtain the restricted star bicoloring Φ'_{sb}. The condition 1 of Def. 2.12 holds due to the different colors used for coloring the vertices in V_r and V_c. The condition 2 is also not violated, because every edge $(r_i, c_i) \in E_D$ has an incident vertex c_i with $\Phi'_{\text{sb}}(c_i) \neq 0$. The condition 3b is not active anymore, because every vertex in V_c is colored with a nonzero color. The condition 3c is as well not active anymore, because every vertex in V_r is colored with color zero. Finally, the path (c_i, r_i, c_j) is considered to show that the condition 3a is not violated. This path is extended to (c_i, r_i, c_j, r_j), because the color swapped from vertex r_j to vertex c_j is relevant to show that this condition is valid. We consider the edge (r_i, c_i) before swapping the colors. A coloring with $\Phi^\star_{\text{sb}}(r_i) = 0$ and $\Phi^\star_{\text{sb}}(c_i) \neq 0$ cannot violate condition 3a, because the colors are not swapped. For a coloring with $\Phi^\star_{\text{sb}}(r_i) \neq 0$ and $\Phi^\star_{\text{sb}}(c_i) = 0$, there are two cases how the vertices r_j and c_j can be colored.

- $\Phi^\star_{\text{sb}}(r_j) = 0$ and $\Phi^\star_{\text{sb}}(c_j) \neq 0$: The nonzero colors for rows and columns are different. Thus, the vertices r_i and c_j are colored with different nonzero colors. After swapping the colors, the vertices are colored as follows: $\Phi'_{\text{sb}}(c_i) \neq \Phi'_{\text{sb}}(c_j)$ with $\Phi'_{\text{sb}}(c_i) \neq 0$ and $\Phi'_{\text{sb}}(c_j) \neq 0$. Hence, condition 3a is not violated.
- $\Phi^\star_{\text{sb}}(r_j) \neq 0$ and $\Phi^\star_{\text{sb}}(c_j) = 0$: In this case, we swap not only $\Phi^\star_{\text{sb}}(c_i)$ and $\Phi^\star_{\text{sb}}(r_i)$ but also $\Phi^\star_{\text{sb}}(c_j)$ and $\Phi^\star_{\text{sb}}(r_j)$. Before swapping the colors, the vertices r_i and r_j are colored with different nonzero colors due to $\Phi^\star_{\text{sb}}(c_j) = 0$ and condition 3b. After swapping the colors, the vertices c_i and c_j are colored with different nonzero colors. The condition 3a is not violated.

Hence, by swapping the nonzero colors from the row vertices to the column vertices, the condition 3a cannot be violated. Furthermore, the number of colors does not change.

Lastly, we consider a special case: The vertices r_i and c_i are isolated. That is, the vertices do not have any distance-2 neighbor. The condition 3 does not affect in this situation. There are two possible colorings: either $\Phi^\star_{\text{sb}}(r_i) = 0$ and $\Phi^\star_{\text{sb}}(c_i) \neq 0$ or $\Phi^\star_{\text{sb}}(r_i) \neq 0$ and $\Phi^\star_{\text{sb}}(c_i) = 0$. In the first case, this is already a valid coloring for

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

Φ'_{sb}. In the second one, the colors of the vertices r_i and c_i must be swapped to get a correct coloring Φ'_{sb}. In both cases the condition 1 and condition 2 hold.

After coloring all vertices in V_r with the color zero and all vertices in V_c with nonzero colors, we show that the restricted star bicoloring Φ'_{sb} is also a restricted distance-2 coloring. Therefore, we compare condition 1 of Def. 2.10 with condition 2 of Def. 2.12 and condition 2 with condition 3a. We can directly observe that the coloring Φ'_{sb} does not violate the conditions of the restricted distance-2 coloring. Hence, the property $\chi_{\mathrm{sb}} \geq p_{\mathrm{d2}}$ holds. □

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

Lebenslauf

Persönliche Daten:

Name:	Michael Lülfesmann
Geburtsdatum:	30. Mai 1981
Geburtsort:	Bielefeld
Staatsangehörigkeit:	deutsch

Qualifikationen:

2000	Abitur am Gymnasium Melle
2001 – 2006	Studium der Informatik an der RWTH Aachen University mit Abschluss Diplom-Informatiker
2006	Beginn der Promotion am Lehrstuhl für Informatik 12 (Hochleistungsrechnen), RWTH Aachen University

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.

Dieses Werk ist copyrightgeschützt und darf in keiner Form vervielfältigt werden noch an Dritte weitergegeben werden.
Es gilt nur für den persönlichen Gebrauch.