Towards a Tighter Coupling of Bottom-Up and Top-Down Sparse Matrix Ordering Methods

Jürgen Schulze
University of Paderborn, Department of Computer Science
Fürstenallee 11, 33102 Paderborn, Germany

Abstract

Most state-of-the-art ordering schemes for sparse matrices are a hybrid of a bottom-up method such as minimum degree and a top down scheme such as George's nested dissection. In this paper we present an ordering algorithm that achieves a tighter coupling of bottom-up and top-down methods. In our methodology vertex separators are interpreted as the boundaries of the remaining elements in an unfinished bottom-up ordering. As a consequence, we are using bottom-up techniques such as quotient graphs and special node selection strategies for the construction of vertex separators. Once all separators have been found, we are using them as a skeleton for the computation of several bottom-up orderings. Experimental results show that the orderings obtained by our scheme are in general better than those obtained by other popular ordering codes.

1 Introduction

Cholesky's method [12] for solving a symmetric positive definite system $A \cdot x = b$, $A \in M(n, \mathbb{R})$, plays an important role in many scientific applications such as linear programming and structural engineering. The importance of this method is mainly due to its generality and robustness. If A is sparse, the amount of *fill* in the factor matrix L can be reduced significantly by reordering the columns and rows of A prior to factorization. Since the computation of a minimum fill ordering for general sparse matrices is an NP-complete problem [59], much effort has been devoted to the development of powerful heuristic algorithms. All heuristics are based on the observation that a symmetric $n \times n$ matrix A can be interpreted as the adjacency matrix of an undirected graph $G = (V, E)$, $E \subset V \times V$. In this graph theoretic context an ordering (or labeling) is a bijection $\pi : V \mapsto \{1, \ldots, n\}$.

As observed by Parter [49] and Rose [53], the column-wise factorization of A can be modeled by a sequence of *elimination graphs* G_k, $1 \leq k \leq n$. When column k is factorized, G_k is obtained from G_{k-1} ($G_0 = G$) by applying the following modifications to G_{k-1}: (1) Delete vertex v_k corresponding to column k in A, and (2) add edges so that the neighbors of v_k are pairwise connected (the neighbors of v_k form a *clique* in G_k). Each edge added in step (2) corresponds to a fill-entry in L.

One of the most popular ordering schemes is *minimum degree* [46, 58]. At stage k the basic minimum degree algorithm selects a vertex with minimum degree in G_{k-1}. This vertex is numbered next in the ordering and is eliminated from G_{k-1} to form the graph G_k. The whole selection/elimination process is then repeated for G_k. Another effective method for reducing fill is *nested dissection* [23, 24]. The method starts with computing a *vertex separator* S in G. All vertices in S are ordered after those in $G(V - S)$. The method is recursively applied to each component of $G(V - S)$ until a component

This work is supported by the German Federal Department of Science and Technology (PARALOR project).

consists of a single vertex or a clique. In contrast to minimum degree which uses local information of the elimination graphs to build π in a bottom-up manner, nested dissection uses global information of the original graph to build π in a top-down manner. It is well recognized that the quality of the orderings produced by bottom-up and top-down schemes is not uniformly good [9]. Therefore, most state-of-the-art ordering codes such as BEND [34], METIS [38], SCOTCH [50], SPOOLES [4], and WGPP [30] are using a hybrid of both schemes.

The main contribution of this paper is to present an ordering scheme that achieves a tighter coupling of bottom-up and top-down methods. In our methodology the vertex separators of an incomplete nested dissection ordering are interpreted as the boundaries of the remaining elements in an unfinished bottom-up ordering. Consequently, we apply bottom-up techniques such as quotient graphs and special node selection strategies to the construction of vertex separators. Once all separators have been found, we are using them as a skeleton for the computation of several bottom-up orderings.

The paper is organized as follows. In section 2 we introduce the concept of quotient graphs and review some of the most popular bottom-up ordering algorithms. Furthermore, we describe state-of-the-art algorithms for the construction of vertex separators and show how bottom-up and nested dissection are combined in existing hybrid schemes. In section 3 the fundamental concepts of our methodology are presented. Finally, section 4 provides us with computational results for some benchmark matrices.

2 Preliminaries

In this section we briefly review some of the fundamental concepts found in bottom-up and top-down sparse matrix ordering schemes. All schemes are described for the undirected graph $G = (V, E)$, $E \subset V \times V$, associated with the symmetric matrix A. We are using the following notations. Let v be a vertex of G. The set of vertices that are *adjacent* to v is denoted by $\mathrm{adj}_G(v)$. For a set $U \subset V$, the set of vertices adjacent to U is given by $\mathrm{adj}_G(U) = \left(\bigcup_{u \in U} \mathrm{adj}_G(u)\right) - U$. Note that we exclude all vertices in U itself. Therefore, the set $\mathrm{adj}_G(U)$ is also called the *boundary* of U. A graph G can be *weighted*, i. e. there may be weights attached to the vertices in V. We write $|v|$ to denote the weight of a vertex v. The weight of a subset U is then given by $|U| = \sum_{u \in U} |u|$. The *cardinality* of U is denoted by $\mathrm{card}(U)$. Some authors define $\mathrm{weight}(U)$ to denote the weight of a set U. However, in this work we consider unweighted graphs and weighted graphs interchangingly. To ease the presentation, $|U|$ is used to refer to the weight of U. Note that $|U|$ and $\mathrm{card}(U)$ are identical when the vertices have unit weight, i. e. when the graph is unweighted.

2.1 Quotient graphs

Quotient graphs play a crucial role in our methodology. Originally, these graphs have been used in bottom-up ordering algorithms such as minimum degree to represent the elimination graphs [25]. In the following, we are using Roman letters for graphs (i. e., G, V, E) and calligraphic letters for quotient graphs (i. e., $\mathcal{G}, \mathcal{X}, \mathcal{E}$).

Let us assume that $G_k = (V_k, E_k)$ has been obtained from G by eliminating the vertices $U_k \subset V$, i. e. $V = V_k \cup U_k$ and $V_k \cap U_k = \emptyset$. A set $D \subset U_k$ of eliminated vertices is called a *domain* in G (with respect to U_k), if the graph $G(D)$ is a connected subgraph of G such that $\mathrm{adj}_G(D) \subset V_k$, i. e. there are only uneliminated vertices on the boundary of D. The quotient graph $\mathcal{G}_k$ associated with G_k consists of nodes $\mathcal{X}_k = \mathcal{D}_k \cup \mathcal{V}_k$ with $\mathcal{D}_k = \{D;\ D \text{ is a domain with respect to } U_k\}$ and $\mathcal{V}_k = \{\{v\};\ v \in V_k\}$. In the context of quotient graphs a node $D \in \mathcal{D}_k$ is called *element* and a node $\{v\} \in \mathcal{V}_k$ *variable*.

The nodes $\mathcal{X}_k = \mathcal{D}_k \cup \mathcal{V}_k$ of $\mathcal{G}_k$ induce a partition of the vertices of G where $\mathcal{D}_k$ contains disjoint sets of eliminated vertices and $\mathcal{V}_k$ a variable for each uneliminated vertex. To ease the presentation we will not always distinguish between a variable $\{v\}$ and the uneliminated vertex v.

An edge of $\mathcal{G}_k$ is of two forms: A *domain-vertex* edge (D, v) where $v \in \mathrm{adj}_G(D)$, and a *vertex-vertex* edge (u, v) where $(u, v) \in E$ and there exists *no* domain $D \in \mathcal{D}_k$ such that $u \in \mathrm{adj}_G(D)$ and $v \in \mathrm{adj}_G(D)$. Note that there are no domain-domain edges in the quotient graph.

There is a close relationship between the elements of $\mathcal{G}_k$ and the cliques generated in G_k. The edge set E_k of the elimination graph G_k can be written as

$$E_k = (E \cap (V_k \times V_k)) \cup \bigcup_{D \in \mathcal{D}_k} (\mathrm{adj}_G(D) \times \mathrm{adj}_G(D)). \tag{2.1}$$

The first set contains all edges of G that are incident to uneliminated vertices. The second set contains the edges of all cliques generated during the elimination process. However, the clique edges are not explicitly stored in $\mathcal{G}_k$. Moreover, a clique is represented by an element $D \in \mathcal{D}_k$ and consists of all vertices/variables that are connected to D via a domain-vertex edge.

In terms of quotient graphs the elimination of a variable v is modeled as follows: (1) Remove v with all of its adjacent elements, (2) add a new element D_v and connect it to all variables that were adjacent to v or to an element, and (3) remove any vertex-vertex edge (u, w) where $u \in \mathrm{adj}_G(D_v)$ and $w \in \mathrm{adj}_G(D_v)$.

The elimination of v can be considered as merging v with all adjacent elements. The merging operation defines a *tree* on the vertices of G. Let D_u denote an element that is adjacent to v and that has been created by the elimination of u. Since D_u is absorbed by the new element D_v, we define v to be the parent of u. If one continues this way, the tree is built from the leaves up to the root. Note that any vertex v that is not adjacent to any element becomes a leaf of the tree. The tree is called *elimination tree* and plays an important role in the context of sparse direct solvers [20, 21, 45, 57].

In section 3 we consider quotient graphs that have a more general structure, i. e. there is no one-to-one relation between a variable of $\mathcal{G}$ and an uneliminated vertex of G. Each variable represents a subset V_i of uneliminated vertices such that two variables V_i, V_j are connected by a vertex-vertex edge, if $V_i \cap \mathrm{adj}_G(V_j) \neq \emptyset$. A variable V_i is connected to a domain D via a domain-vertex edge, if $V_i \cap \mathrm{adj}_G(D) \neq \emptyset$.

2.2 Bottom-up orderings

Bottom-up methods build the elimination tree from the leaves up to the root. In each iteration k a *greedy heuristic* is applied to G_{k-1} to select a vertex for elimination. This section briefly describes two of the most popular bottom-up algorithms, the minimum degree and the minimum deficiency ordering heuristics.

2.2.1 Minimum degree ordering

As mentioned above, at each iteration k the minimum degree (MD) algorithm eliminates a vertex v that minimizes $\deg_{G_{k-1}}(v) = |\mathrm{adj}_{G_{k-1}}(v)|$. The algorithm is a symmetric variant of the Markowitz scheme [46] and was first applied to sparse symmetric factorization by Tinney and Walker [58]. Over the years many enhancements have been proposed to the basic algorithm that have greatly improved its efficiency. In the following we briefly describe some of these enhancements. For a detailed survey the reader is referred to [27].

Perhaps one of the most important enhancements is the concept of *supernodes*. Two vertices u, v of an elimination graph G_k belong to the same supernode, if $\mathrm{adj}_{G_k}(u) \cup \{u\} = \mathrm{adj}_{G_k}(v) \cup \{v\}$. In this context the vertices u, v are called *indistinguishable*. Indistinguishable vertices possess two important properties: (1) they can be eliminated consecutively in a minimum degree ordering, and (2) they remain indistinguishable in all subsequent elimination graphs. As a consequence, all vertices that belong to a supernode $I \subset V_k$ can be replaced by a single logical node with weight $|I|$. This reduces the size of the elimination graphs and, therefore, the runtime of the minimum degree algorithm.

Closely related to the concept of supernodes is the notion of *external degrees* [42]. With our notations, the external degree of a vertex v in supernode I is $|\,\mathrm{adj}_{G_k}(I)|$. Instead of using true degrees, the vertex to be eliminated next is selected according to its external degree. The motivation is that the only edges added by the elimination of $v \in I$ are between vertices in $\mathrm{adj}_{G_k}(I)$. As a result, one obtains slightly better orderings [27].

A key feature of the minimum degree algorithm is that *one* vertex is eliminated in each step. Once the new elimination graph has been built, the degree of all vertices that were adjacent to the newly eliminated vertex have to be updated. The most time-consuming part of the minimum degree algorithm is this degree update step. In Liu's *multiple minimum degree* (MMD) algorithm [42] an independent set of minimum degree vertices is eliminated in each step. This multiple elimination technique reduces the amount of degree updates and leads to a significant acceleration of the minimum degree algorithm.

The efficiency of the minimum degree algorithm can be further improved when using *approximate degrees* [1, 29] rather than exact degrees. Once a vertex v has been eliminated from G_{k-1}, the new degree of a vertex $u \in \mathrm{adj}_{G_{k-1}}(v)$ is estimated by an upper bound. The upper bound is much less expensive to compute than the exact degree. The approximate minimum degree (AMD) algorithm proposed by Amestoy et al. [1] produces orderings that are of comparable quality to those obtained using exact degrees. Note that this is not always the case with the approximation proposed by [29].

2.2.2 Minimum deficiency ordering

A less popular bottom-up scheme is the *minimum deficiency* or *minimum local fill* (MF) heuristic [58]. Here, the exact amount of fill is used to select a vertex for elimination. Formally, the deficiency of a vertex v in G_k is defined as $\mathrm{def}_{G_k}(v) = |\{\{u, w\};\ u, w \in \mathrm{adj}_{G_k}(v), u \notin \mathrm{adj}_{G_k}(w)\}|$. The minimum deficiency algorithm has received much less attention because of its prohibitive runtime: The computation of $\mathrm{def}_{G_k}(v)$ is much more expensive than the computation of $\deg_{G_k}(v)$, and the elimination of v does not only affect the deficiency of all vertices in $\mathrm{adj}_{G_k}(v)$, but also the deficiency of all vertices that are adjacent to a vertex in $\mathrm{adj}_{G_k}(v)$. Furthermore, it was thought that minimum deficiency would only marginally improve the ordering quality compared to minimum degree [18]. However, recent studies have shown that much better orderings can be obtained from minimum deficiency [47, 48, 56].

In order to reduce the runtime of the minimum deficiency algorithm, Rothberg and Eisenstat [56] have developed several node selection strategies that rely on an approximate computation of the deficiencies. These selection strategies have proven to be very effective. Again, supernodes play a crucial role. Let v be the vertex whose elimination transforms G_{k-1} into G_k. Furthermore, let $u \in \mathrm{adj}_{G_{k-1}}(v)$ and let I_u denote the supernode with $u \in I_u$. Then, $\mathrm{def}_{G_k}(u)$ is bounded by $\frac{1}{2}d(d-1)$ where d denotes the external degree of u in G_k. Since all vertices in $\mathrm{adj}_{G_{k-1}}(v) - I_u$ are neighbors of u and connected to a clique in G_k, the upper bound can be tightened to $\mathrm{score}_{\mathrm{AMF}}(u) = \frac{1}{2}d(d-1) - \frac{1}{2}c(c-1)$ with $c = |\,\mathrm{adj}_{G_{k-1}}(v) - I_u|$. In Rothberg's and Eisenstat's *approximate minimum local fill* (AMF) algorithm, the vertex to be eliminated next is selected according to $\mathrm{score}_{\mathrm{AMF}}$. Further selection strategies that are based on function $\mathrm{score}_{\mathrm{AMF}}$ are proposed such as *approximate*

minimum mean local fill (AMMF) or *approximate minimum increase in neighbor degree* (AMIND). For more details consult [56].

2.3 Top-down orderings

The most popular top-down scheme is George's nested dissection (ND) algorithm [23, 24]. The basic idea of this approach is to find a subset of vertices S in G, whose removal partitions G in two subgraphs $G(B)$ and $G(W)$ with $V = S \cup B \cup W$ and $|B|, |W| \leq \alpha|V|$ for some $0 < \alpha < 1$. Such a partition of G is denoted by (S, B, W). The set S is called *vertex separator* of G. If we order the vertices in S after the *(black)* vertices in B and the *(white)* vertices in W, no fill-edge can occur between B and W. Typically, the columns corresponding to S constitute a full off-diagonal block in the Cholesky factor. Therefore, S is supposed to be small. Once S has been found, the algorithm is recursively applied to each connected component of $G(B)$ and $G(W)$ until a component consists of a single vertex or a clique. In this way the elimination tree is built from the root down to the leaves.

In contrast to the bottom-up methods introduced in section 2.2 the nested dissection algorithm is quite ill-specified [35]. For an effective implementation the following two important questions have to be answered: (1) How should the separators be determined, and (2) how important is a balanced partitioning. In the following we address question (1) and describe existing techniques for finding and improving vertex separators. Questions (1) and (2) are discussed again in sections 3 and 4.

2.3.1 Finding separators

Graph partitioning heuristics are usually divided into *construction* and *improvement heuristics*. A construction heuristic takes the graph as input and computes an initial separator from scratch. An improvement heuristic tries to minimize the size of a separator through a sequence of elementary steps. This section introduces two powerful construction heuristics, the *multilevel method* and the *domain decomposition approach*. The next section discusses improving a separator.

Multilevel method In recent years, multilevel algorithms have been applied successfully to the construction of *edge separators* [15, 33, 37]. Roughly speaking, a multilevel algorithm consists of three phases. In the first phase the original graph G is approximated by a sequence of smaller graphs that maintain the essential properties of G (coarsening phase). Then, an initial edge separator is constructed for the last graph in the sequence (partitioning phase). Finally, the edge separator is projected backwards to the next larger graph in the sequence until G is reached (uncoarsening phase). A local improvement heuristic such as Kernighan-Lin [39] or Fiduccia-Mattheyses [22] is used to refine the edge separator after each uncoarsening step. The software packages CHACO [32], METIS [38], and PARTY [52] provide a variety of methods for the coarsening, partitioning, and uncoarsening phase.

For the computation of a nested dissection ordering we need vertex separators. Several nested dissection implementations [11, 15] first find an edge separator via a multilevel algorithm and then derive a vertex separator from the edge separator using a matching technique that is described in section 2.3.2. However, the size of an edge separator is only indirectly related to the size of a vertex separator. Therefore, the multilevel scheme has been extended to find vertex separators directly [31, 35]. In the extended scheme a variant of the Fiduccia-Mattheyses heuristic is applied to the refinement of vertex separators. This *vertex Fiduccia-Mattheyses* method is summarized in section 2.3.2.

Domain decomposition method In contrast to the multilevel method described above, Ashcraft and Liu [7] propose a two-level approach to construct a vertex separator. Analogous to the *domain*

decomposition methods for solving PDEs, the vertex set V of G is partitioned into $V = \widehat{V} \cup D_1 \cup \ldots \cup D_r$ with $\mathrm{adj}_G(D_i) \subset \widehat{V}$ for all $1 \leq i \leq r$. The set $\widehat{V}$ is called *multisector*. The removal of $\widehat{V}$ splits G into connected subgraphs $G(D_1), \ldots, G(D_r)$. Once a domain decomposition $(\widehat{V}, D_1, \ldots, D_r)$ has been found, a color from $\{\mathrm{BLACK}, \mathrm{WHITE}\}$ is assigned to each D_i. This induces a coloring of the vertices $v \in \widehat{V}$:

$$\mathrm{color}(v) = \begin{cases} \mathrm{BLACK}, \text{ if all } D \text{ with } v \in \mathrm{adj}_G(D) \text{ are colored BLACK} \\ \mathrm{WHITE}, \text{ if all } D \text{ with } v \in \mathrm{adj}_G(D) \text{ are colored WHITE} \\ \mathrm{GRAY}, \text{ otherwise.} \end{cases} \tag{2.2}$$

To ensure that the set $S = \{v \in \widehat{V};\ \mathrm{color}(v) = \mathrm{GRAY}\}$ constitutes a valid vertex separator of G for every coloring of $D_1, \ldots, D_r$, the vertices of the multisector $\widehat{V}$ are properly blocked to *segments*. This blocking technique is summarized in section 3.1. Its objective is to group into a segment adjacent variables that do not share at least one domain in their adjacency list. As a result, one obtains a partitioning $\widehat{V} = V_1 \cup \ldots \cup V_s$, $V_i \cap V_j = \emptyset$, of the multisector. Analogous to (2.2) a segment $V_i \subset \widehat{V}$ can be colored as follows:

$$\mathrm{color}(V_i) = \begin{cases} \mathrm{BLACK}, \text{ if all } D \text{ with } V_i \cap \mathrm{adj}_G(D) \neq \emptyset \text{ are colored BLACK} \\ \mathrm{WHITE}, \text{ if all } D \text{ with } V_i \cap \mathrm{adj}_G(D) \neq \emptyset \text{ are colored WHITE} \\ \mathrm{GRAY}, \text{ otherwise.} \end{cases} \tag{2.3}$$

Now, and because of the properties of the segments, the set $S = \{v \in \widehat{V};\ \exists\, V_i \subset \widehat{V} \text{ with } v \in V_i \text{ and } \mathrm{color}(V_i) = \mathrm{GRAY}\}$ constitutes a vertex separator of G for every coloring of $D_1, \ldots, D_r$. In section 3.1 the blocking of vertices to segments and the coloring of $D_1, \ldots, D_r$ is discussed again and illustrated on an example.

Ashcraft and Liu use a randomized greedy domain-growing algorithm to generate a domain decomposition. The sets $D_1, \ldots, D_r$ are colored using a Fiduccia-Mattheyses scheme that minimizes the size of the induced vertex separator S. Once S has been found, a sophisticated network-flow algorithm is used to refine S. For more information the reader is referred to [7, 8].

There is a close relationship between the sets $D_1, \ldots, D_r$ and the domains formed by a bottom-up ordering algorithm. Let $\widehat{U} = \bigcup_{i=1}^{r} D_i$. Since all graphs $G(D_i)$ are connected subgraphs of G with $\mathrm{adj}_G(D_i) \subset \widehat{V}$, each set D_i can be interpreted as a domain with respect to the eliminated vertices $\widehat{U}$. The multisector $\widehat{V}$ contains all uneliminated vertices of G, i. e. $V = \widehat{V} \cup \widehat{U}$. In section 3.1 we show that a domain decomposition $(\widehat{V}, D_1, \ldots, D_r)$ can be represented by a quotient graph. Even more, if $\widehat{V}$ is properly partitioned into segments, the quotient graph is bipartite, i. e. it contains domain-vertex edges only. This interesting connection encouraged us to use quotient graphs for the construction of vertex separators. The quotient graphs are obtained by node selection strategies similar to those found in bottom-up ordering algorithms.

2.3.2 Improving separators

To be more precise, we are interested in improving the partition (S, B, W) of a graph G rather than in improving the separator S. Usually, the quality of a partition is measured in terms of separator size and balance of the partition. To decide whether one partition (S, B, W) is better or worse than a second partition (S', B', W'), an evaluation function F is needed. This function weights the sometimes conflicting goals of minimizing separator size and maximizing partition balance. The choice of an appropriate evaluation function is not straightforward and must be considered in the context of the overall ordering process. Section 3.2 introduces the evaluation function that is used by our algorithm.

A partition (S, B, W) is usually improved by finding a new separator S' with the property that $|S'| \leq |S|$. We then have to examine the partition induced by S'. We hope that the improvement of S results in an improvement of the partition (S, B, W). There are two main improvement techniques that are in common use today. Firstly, *heuristic methods* that rely on finding a "nearby" separator S' by simple vertex moves and, secondly, *direct methods* that rely on finding a *vertex cover* in a bipartite subgraph of G to move a set of vertices simultaneously.

Heuristic methods The algorithms proposed by Kernighan-Lin [39] and Fiduccia-Mattheyses [22] are the most frequently used heuristics for refining edge separators of graphs. Both algorithms consist of two nested loops. In the inner loop, the Kernighan-Lin algorithm chooses pairs of vertices that belong to different components, swaps their positions logically and locks them. These logical swaps are repeated until all vertices are locked. The fundamental idea of the algorithm is to allow a deterioration of the cut size when swapping the vertices. The hope is that an intermediate increase of the cut size allows the discovery of better partitions in subsequent pair exchanges. Once all vertices are locked, the sequence of logical swaps is chosen that leads to the maximal decrease in cut size. The corresponding vertices are then physically swapped and the outer loop restarts the improvement process for the new partition.

To speed up the inner loop, Fiduccia and Mattheyses suggested to consider only simple vertex moves instead of pair exchanges. In this way, a single search for a better partition (i. e. one execution of the inner loop) can be performed in time bounded by the number of edges in the graph. In real implementations numerous extra "tricks" such as early termination, lazy evaluation of priorities, and randomization are used to improve both runtime as well as quality of the solutions [32].

The Fiduccia-Mattheyses algorithm can easily be generalized to vertex separators [5]. In this vertex Fiduccia-Mattheyses algorithm each move selects a vertex $v \in S$, transfers it to B (or W) and locks it. Once a vertex v has been transferred to B, all neighbors of v in W are pulled into S. The selection of v is based on a *gain-value* that measures the change in separator size. For a weighted graph the gain associated with moving v from S to B is defined as

$$\mathrm{gain}_{S \to B}(v) = |v| - \sum_{u \in \mathrm{adj}_G(v) \cap W} |u|.$$

The value $\mathrm{gain}_{S \to W}(v)$ is defined analogously. Since a vertex cannot move twice within the inner loop, it can be shown [35] that the total time required for computing and updating the gain-values is asymptotically bounded by the number of edges in the graph. If a *heap* data structure is used to sort the gain-values, the cost for one execution of the inner loop is given by $O(e \log n)$ where e denotes the number of edges and n the number of vertices in the graph. Note that we are considering weighted graphs. Therefore, we cannot use a linear time algorithm such as *bucket sort* as proposed by Fiduccia and Mattheyses.

Direct methods An alternate algorithm for refining a vertex separator S has been introduced by Liu [44]. Again, let (S, B, W) denote the partition of G and let B be the component with larger weight. If we define B_S to be the set of vertices in B that are adjacent to vertices in S, i. e. $B_S = \mathrm{adj}_G(S) \cap B$, then any vertex cover in the bipartite graph $H(S, B_S)$ constitutes a valid vertex separator of G. In order to obtain an improved separator, we must find a minimum cardinality or a minimum weight vertex cover, depending on whether G is weighted or not. In the unweighted case a *maximum matching* is determined in $H(S, B_S)$. The maximum matching is then used to construct a vertex cover so that every matching edge is incident to exactly one vertex of the cover. According to König [40] the

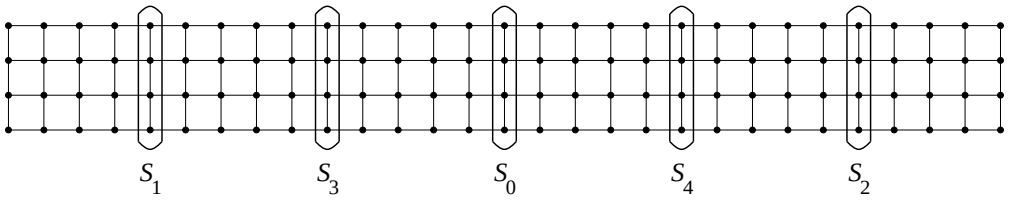

Fig. 2.1: Topmost separators of a rectangular grid. S_0 is the first separator, followed by the separators S_1, S_2, and S_3, S_4.

vertex cover must have minimum cardinality. If G is weighted, a network-flow technique is used to find a minimum weight vertex cover. The details are beyond the scope of this paper; more information can be found in [8]. Note that it is also possible to *pair* S with the smaller component W. However, in this case the refinement process can further increase the imbalance of the partition.

It turns out that the same vertex cover technique can be used to obtain a vertex separator from an edge separator. One simply computes a minimum vertex cover for the bipartite graph induced by the cut edges. Note that it also possible to use heuristics for finding an approximate minimal cardinality vertex cover [41].

2.4 Multisection orderings

The shortcomings of a bottom-up method such as minimum degree are largely due to the *local nature* of the algorithm. In [13] Berman and Schnitger describe a minimum degree elimination sequence for the $k \times k$ grid so that the number of factor entries and the number of factor operations is an order of magnitude higher than optimal. By construction, their scheme produces elements with a severe "fractal" boundary. Asymptotically optimal orderings for $k \times k$ grids are produced by George's nested dissection scheme [36]. If the separators are eliminated according to the given nested dissection order, elements with a "smooth" boundary are created.

The shortcomings of nested dissection are best illustrated on $h \times k$ grids with large aspect ratio (i. e. $h \ll k$). Here, minimum degree outperforms nested dissection. Since the nested dissection scheme still relies on the best separators of the grid (best in terms of separator size and partition balance), the bad performance cannot be attributed to the quality of the separators. Moreover, it is a question of how the separators are *numbered*. Figure 2.1 shows the topmost separators of a rectangular grid. If the separators are numbered according to the given nested dissection order, S_3 will be eliminated before S_1 and S_0. As a consequence one obtains an element whose boundary contains $2h$ vertices. However, if the separators are eliminated from "left to right" or "from both ends to the center" all elements will have a boundary of size h. Therefore, a *profile* or minimum degree ordering of the separator vertices will produce less fill than nested dissection.

Multisection schemes [9] provide a more effective way of numbering the vertices associated with a separator. Here, separators are only used to *split* the graph in multiple subgraph. As a result one obtains a domain decomposition $(\Phi, \Omega_1, \ldots, \Omega_q)$ of the original graph where Φ contains all vertices that belong to a separator. Note that we used Greek letters to distinguish the multisection domain decomposition $(\Phi, \Omega_1, \ldots, \Omega_q)$ from the separator domain decomposition $(\widehat{V}, D_1, \ldots, D_r)$ introduced in section 2.3.1. As described by Ashcraft and Liu [7], the latter domain decomposition can be used to construct the separators required by the former domain decomposition.

The fundamental property of a multisection ordering is that the vertices in the domains $\Omega_1, \ldots, \Omega_q$ are numbered before the vertices in the multisector Φ. Thus, the elimination sequence is *constrained*,

i. e. some vertices must be ordered before others. To order the vertices in the multisector we consider the elimination graph $G_\Phi = (\Phi, E_\Phi)$ with (cf. equation (2.1))

$$E_\Phi = (E \cap (\Phi \times \Phi)) \cup \bigcup_{i=1}^{q} (\mathrm{adj}_G(\Omega_i) \times \mathrm{adj}_G(\Omega_i)).$$

Note that G_Φ represents the structure of the *Schur complement matrix*. Different ordering methods can be used to number the vertices in the domains and the vertices in the Schur complement graph. Therefore, a multisection ordering is defined by three choices [9]:

1. How to determine the domain decomposition $(\Phi, \Omega_1, \ldots, \Omega_q)$?
2. What fill-reducing ordering to use to order the domains $\Omega_1, \ldots, \Omega_q$?
3. What fill-reducing ordering to use to order the multisector Φ?

Multisection orderings are denoted by $\mathrm{MS}(\mathrm{ord}_1, \mathrm{ord}_2)$ where ord_1 refers to the ordering method used to number the domains and ord_2 refers to the ordering method used to number the multisector. In the literature, there are a number of existing ordering schemes using the multisection approach. Two important examples are:

- Incomplete Nested Dissection [28] – $\mathrm{MS}(\mathrm{MMD}, \mathrm{ND})$ and $\mathrm{MS}(\mathrm{CMD}, \mathrm{ND})$

  In this scheme the multisector is constructed by the recursive bisection process of nested dissection. In contrast to the original nested dissection algorithm the recursion terminates after a few levels and the vertices in the remaining subgraphs/domains are numbered using either multiple minimum degree (MMD) [42] or *constrained minimum degree* (CMD) [43]. The vertices in the multisector are numbered according to the given nested dissection ordering.

- Local Nested Dissection [14] – $\mathrm{MS}(\mathrm{ND}, \mathrm{PROFILE})$

  The most successful ordering scheme for $h \times k$ grids with $h \ll k$ is local nested dissection. Here, the domain decomposition is constructed by a set of parallel and vertical dissectors that subdivide the rectangular grid in roughly square domains. Each square domain is then numbered by nested dissection. The vertices in the multisector are numbered by a profile ordering.

The $h \times k$ grid example gives rise to the assumption that the quality of an incomplete nested dissection ordering can be further improved when using minimum degree to order the separator vertices instead of following the given nested dissection ordering. This multisection ordering is referred to as $\mathrm{MS}(\mathrm{CMD}, \mathrm{MMD})$ and has been independently discovered by Ashcraft, Liu [6, 9], and Rothberg [54].

3 Our methodology

Two levels of hybridizing bottom-up and top-down methods can be found in the literature: incomplete nested dissection (i. e. $\mathrm{MS}(\mathrm{MMD}, \mathrm{ND})$ or $\mathrm{MS}(\mathrm{CMD}, \mathrm{ND})$) and minimum degree post-processing on an incomplete nested dissection (i. e. $\mathrm{MS}(\mathrm{CMD}, \mathrm{MMD})$). Our driving interest is to achieve an even tighter coupling of both methods. Roughly speaking, we want to explore how bottom-up methods can be used to improve the construction of vertex separators and how vertex separators can be used to improve the construction of bottom-up orderings.

To achieve the first goal we developed a new multisection scheme that is given in figure 3.1. Similar to the original scheme proposed by Ashcraft and Liu [9], the multisector Φ is constructed by a

```
MULTISECTION(ord_Φ, ord_1, ord_2)
01: Determine a domain decomposition (Φ, Ω_1, ..., Ω_q) of G by a recursive bisection
    process. Use node selection strategy ord_Φ to construct the vertex separators.
02: for each set Ω_i do
03:     Eliminate all vertices in Ω_i using node selection strategy ord_1.
04: Construct the elimination graph G_Φ.
05: Number the vertices in G_Φ using node selection strategy ord_2.
```

Fig. 3.1: Function MULTISECTION.

recursive bisection process (line 01). However, a new multilevel method is used to determine the vertex separators. In contrast to traditional methods that rely on a matching technique to coarsen a graph, the new method produces a sequence $\mathcal{G}_0, \mathcal{G}_1, \ldots, \mathcal{G}_t$ of quotient graphs where $\mathcal{G}_k$ is obtained from $\mathcal{G}_{k-1}$, $1 \leq k \leq t$, by the elimination of certain variables. The variables are chosen according to a node selection strategy ord_Φ that is similar to selection strategies found in bottom-up algorithms. Once a domain decomposition has been determined, the vertices in the domains $\Omega_1, \ldots, \Omega_q$ are eliminated according to selection strategy ord_1 (lines 02–03). Finally, the Schur complement graph G_Φ is constructed, and all vertices in G_Φ are eliminated according to selection strategy ord_2 (lines 04–05).

According to the classification scheme proposed by Ashcraft and Liu [9], our multisection ordering algorithm is defined by the three choices:

1. What node selection strategy to use to coarsen a quotient graph?
2. What node selection strategy to use to order the domains $\Omega_1, \ldots, \Omega_q$?
3. What node selection strategy to use to order the multisector Φ?

In our methodology multisection orderings are denoted by $\mathrm{MS}(\mathrm{ord}_\Phi, \mathrm{ord}_1, \mathrm{ord}_2)$. Again, ord_1 and ord_2 refer to the elimination methods applied to the vertices in the domains and the multisector. The additional parameter ord_Φ refers to the elimination method applied to the variables of a quotient graph. Thus, ord_Φ specifies the coarsening method in our multilevel algorithm.

To achieve the second goal we generalized our multisection ordering scheme. In the new scheme the multisector Φ serves as a "skeleton" for the computation of several bottom-up orderings. The key idea is to eliminate some separators of Φ according to nested dissection and some according to minimum degree. Therefore, the new scheme is called *tristage multisection*. Tristage multisection schemes have originally been proposed by Ashcraft, Liu, and Eisenstat [10].

In section 3.1 we formally introduce our multilevel algorithm and present some node selection strategies to coarsen a quotient graph. Section 3.2 describes the local improvement heuristic that is used to refine a vertex separator after each uncoarsening step. We prove that the performance of our heuristic is asymptotically identical to the performance of the vertex Fiduccia-Mattheyses heuristic proposed by Ashcraft and Liu [5]. Finally, section 3.3 presents the tristage multisection scheme.

3.1 Finding a separator in the new multilevel scheme

The key idea of our multilevel scheme is to approximate the original graph G by a sequence of quotient graphs $\mathcal{G}_0, \mathcal{G}_1, \ldots, \mathcal{G}_t$. Each quotient graph $\mathcal{G}_k$, $0 \leq k \leq t$, has the fundamental property that any set of variables $\mathcal{S}_k$, whose removal separates $\mathcal{G}_k$ in two or more connected components, induces a vertex separator S in G. As a consequence, any improvement of $\mathcal{S}_k$ leads to an improvement of S.

```
SEPARATOR(ord_Φ)
01: Construct initial quotient graph G_0 from G.
02: k := 0;
03: while G_k not small enough do
04:     Construct coarser quotient graph G_{k+1} by eliminating some variables from G_k.
        The variables are chosen according to node selection strategy ord_Φ.
05:     k := k + 1;
06: end while
07: Determine a coloring for G_k that induces a small separator S_k.
08: while k > 0 do
09:     Extend the coloring of G_k to the nodes of G_{k-1}. This induces a separator S_{k-1} of G_{k-1}.
10:     Improve the coloring of G_{k-1} so that S_{k-1} is minimized.
11:     k := k - 1;
12: end while
13: Extend the coloring of G_0 to the vertices of G. This induces a separator S of G.
14: Improve S by applying a vertex cover technique.
```

Fig. 3.2: Function SEPARATOR.

This section is divided in two parts. The first part describes our multilevel scheme and the second part introduces some node selection strategies to coarsen the quotient graphs.

3.1.1 Description of the new multilevel scheme

Our multilevel scheme starts with the construction of an initial quotient graph $\mathcal{G}_0$ from G. Based on $\mathcal{G}_0$ a sequence of quotient graphs $\mathcal{G}_1, \ldots, \mathcal{G}_t$ is produced where $\mathcal{G}_k$ is obtained from $\mathcal{G}_{k-1}$, $1 \leq k \leq t$, by the elimination of certain variables. We want that each quotient graph $\mathcal{G}_k$ satisfies the following two conditions (this includes $\mathcal{G}_0$, i. e. $0 \leq k \leq t$):

(1) Let $\mathcal{V}_k = \{V_1, \ldots, V_s\}$ denote the set of variables in $\mathcal{G}_k$ and let $\mathcal{S}_k \subset \mathcal{V}_k$ be a separator for $\mathcal{G}_k$. If $\mathcal{S}_k$ consists of variables $V_{p_1}, \ldots, V_{p_t}$, then $S = V_{p_1} \cup \ldots \cup V_{p_t}$ constitutes a separator of G.

(2) $\mathcal{G}_k$ is bipartite, i. e. it only contains domain-vertex edges.

Condition (1) is crucial for the effectiveness of our multilevel scheme. It relates the minimization of the separator for the coarse graph $\mathcal{G}_k$ to the minimization of the separator for the original graph G. Note that in contrast to other multilevel methods, the separator for a coarse graph is truly a minimal vertex separator of the original graph. To find a separator $\mathcal{S}_k$ in the coarse graph $\mathcal{G}_k$ we are using the coloring technique proposed by Ashcraft and Liu [7]. Let us assume that a color from $\{\mathrm{BLACK}, \mathrm{WHITE}\}$ is assigned to each element $D \in \mathcal{D}_k$ of the quotient graph $\mathcal{G}_k$. Analogous to (2.3), this induces a coloring of the variables $\{V_1, \ldots, V_s\}$. As we will see soon, condition (2) guarantees that the gray colored variables constitute a separator of $\mathcal{G}_k$ for every coloring of the elements. Figure 3.2 summarizes the structure of our multilevel scheme. In the following we describe each step in more detail.

Construction of the initial quotient graph We start with computing a *maximal independent set* U of vertices in G. The vertices that belong to U are removed from G to obtain $r = \mathrm{card}(U)$ initial domains. All vertices that lie on the boundary of exactly one domain are merged with that domain. Let $D_1, \ldots, D_r$ denote the resulting domains and let $\widehat{V}$ denote the set of uneliminated vertices of G.

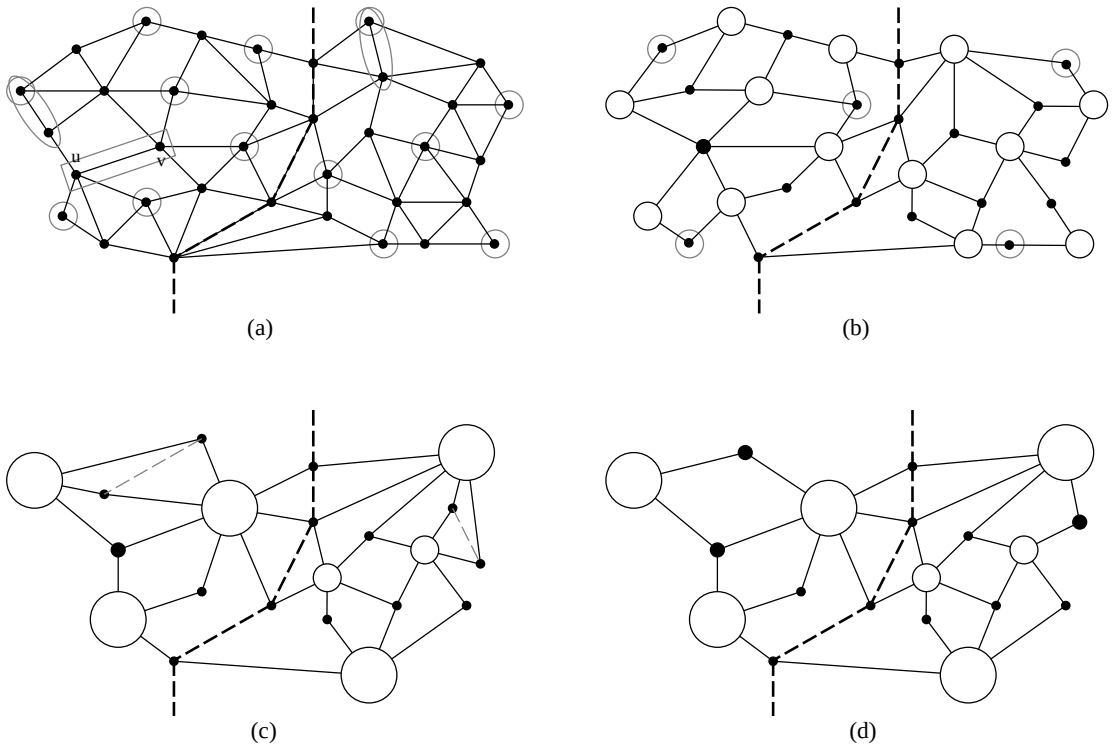

Fig. 3.3: The coarsening of a graph G. (a) shows the construction of the initial quotient graph $\mathcal{G}_0$. The vertices of G that belong to the independent set U are circled. These vertices represent the initial domains. Any vertex that lies on the boundary of only one domain is merged with that domain. This is indicated by an oval. All neighboring vertices that do not share a common adjacent domain are blocked to a segment. This is indicated by a rectangle. The initial quotient graph is shown in (b). Elements are denoted by white colored circles, variables by black colored circles. The larger the circle, the more vertices of G have been absorbed by the element/variable. (c) shows the quotient graph $\mathcal{G}_1$ obtained from $\mathcal{G}_0$ when eliminating all variables circled in (b). In (d) all indistinguishable variables of $\mathcal{G}_1$ (they are connected with a dotted line in (c)) have been replaced by a supervariable. The black dotted line represents a vertex separator for $\mathcal{G}_1$, $\mathcal{G}_0$, and G.

According to Ashcraft and Liu, the coloring rule (2.2) produces a valid vertex separator of G for every coloring of $D_1, \ldots, D_r$, if and only if

$$\forall\, u, v \in \widehat{V} : (u, v) \in E \Rightarrow \exists\, D \text{ with } u, v \in \mathrm{adj}_G(D). \tag{3.1}$$

Figure 3.3 (a) shows that the coloring technique fails, if (3.1) is not fulfilled. Let us assume that the three domains adjacent to u are colored black and the two domains adjacent to v are colored white. Due to (2.2) u is colored black and v is colored white. Since both vertices are connected by an edge in G, no separator is induced by this coloring. Such a situation cannot occur, if all vertices $u, v \in \widehat{V}$ with $(u, v) \in E$ share a common domain (for a formal proof the reader is referred to [7]).

To avoid this problem, the vertices in $\widehat{V}$ are blocked to segments $V_1, \ldots, V_s$ so that

$$\forall\, V_i, V_j \subset \widehat{V} : V_i \cap \mathrm{adj}_G(V_j) \neq \emptyset \Rightarrow \exists\, D \text{ with } V_i, V_j \cap \mathrm{adj}_G(D) \neq \emptyset. \tag{3.2}$$

The segments $V_1, \ldots, V_s$ are chosen to be the smallest segments that fulfill (3.2). In figure 3.3 (a) the vertices u, v are blocked to a segment. All other segments consist of a single vertex. Equa-

tion (3.2) guarantees that the coloring rule (2.3) produces a valid vertex separator for every coloring of $D_1, \ldots, D_r$.

We are now able to define the initial quotient graph $\mathcal{G}_0$. The elements are the domains $D_1, \ldots, D_r$ and the variables are the segments $V_1, \ldots, V_s$. The edges of $\mathcal{G}_0$ are as described in section 2.1. Obviously, $\mathcal{G}_0$ satisfies condition (1). Because of equation (3.2) all variables V_i, V_j with $V_i \cap \mathrm{adj}_G(V_j) \neq \emptyset$ share a common domain. As a consequence, there is no edge between two variables in $\mathcal{G}_0$. Thus, $\mathcal{G}_0$ also satisfies condition (2). Figures 3.3 (a) and (b) illustrate the construction of $\mathcal{G}_0$.

Construction of coarser quotient graphs Let $\mathcal{G}_k = (\mathcal{D}_k \cup \mathcal{V}_k, \mathcal{E}_k)$ denote a quotient graph that satisfies conditions (1) and (2). For the construction of $\mathcal{G}_{k+1}$ we first determine an independent set of variables $\mathcal{U} \subset \mathcal{V}_k$ in $\mathcal{G}_k$ using node selection strategy ord_Φ. In this context two variables $V_i, V_j \in \mathcal{V}_k$ are said to be independent, if there is no element $D \in \mathcal{D}_k$ such that $(D, V_i) \in \mathcal{E}_k$ and $(D, V_j) \in \mathcal{E}_k$. Once $\mathcal{U}$ has been found, a variable $V_i \in \mathcal{U}$ is merged with all adjacent elements to form a new element D_{V_i}. Each merging operation corresponds to an elimination step in a bottom-up algorithm. Note that there is some analogy with the multiple minimum degree algorithm proposed by Liu [42]. In each iteration of Liu's ordering algorithm an independent set of vertices with minimum degree is chosen for elimination. However, in our coarsening scheme the choice of variables is more relaxed and not limited to variables with minimum degree (cf. section 3.1.2).

Finally, all remaining variables that are adjacent to exactly one element are merged with that element. The resulting quotient graph is $\mathcal{G}_{k+1}$. It remains to show that $\mathcal{G}_{k+1}$ satisfies conditions (1) and (2). Obviously, $\mathcal{G}_{k+1}$ is bipartite. Because of $\mathcal{V}_{k+1} \subset \mathcal{V}_k$, every separator of $\mathcal{G}_{k+1}$ is a separator of $\mathcal{G}_k$, and since $\mathcal{G}_k$ satisfies condition (1), the condition is also satisfied by $\mathcal{G}_{k+1}$.

There is another analogy to minimum degree in our coarsening scheme: All variables of $\mathcal{G}_{k+1}$ that are adjacent to the same set of elements are replaced by a single *supervariable*. This replacement further reduces the number of nodes in $\mathcal{G}_{k+1}$ and leads to an acceleration of the coarsening process. The construction of new quotient graphs terminates, when $\mathcal{G}_{k+1}$ contains less than 200 elements. Figures 3.3 (b)–(d) illustrate the construction of $\mathcal{G}_{k+1}$.

Coloring of quotient graphs Again, let $\mathcal{G}_k = (\mathcal{D}_k \cup \mathcal{V}_k, \mathcal{E}_k)$ denote a quotient graph that satisfies conditions (1) and (2). To construct a separator $\mathcal{S}_k \subset \mathcal{V}_k$ of $\mathcal{G}_k$ we are using the coloring technique proposed by Ashcraft and Liu [7]. To be more precise, we are using the coloring rule

$$\mathrm{color}(V_i) = \begin{cases} \mathrm{BLACK}, \text{ if all } D \in \mathrm{adj}_{\mathcal{G}_k}(V_i) \text{ are colored BLACK} \\ \mathrm{WHITE}, \text{ if all } D \in \mathrm{adj}_{\mathcal{G}_k}(V_i) \text{ are colored WHITE} \\ \mathrm{GRAY}, \text{ otherwise.} \end{cases} \tag{3.3}$$

Note that $\mathcal{G}_k$ is bipartite, i. e. there is no edge between two variables. Therefore, no additional blocking of variables is necessary to ensure that the set $\mathcal{S}_k = \{V_i \in \mathcal{V}_k;\ \mathrm{color}(V_i) = \mathrm{GRAY}\}$ constitutes a valid separator of $\mathcal{G}_k$ for every coloring of the elements in $\mathcal{D}_k$.

To determine a coloring that minimizes the size of $\mathcal{S}_k$, we are using a Fiduccia-Mattheyses scheme that is introduced in section 3.2. The scheme requires an initial coloring of the nodes. In the case of $k = t$, i. e. $\mathcal{G}_k$ is the last quotient graph in the sequence, the initial coloring is defined as $\mathrm{color}(D) := \mathrm{BLACK}$ for all $D \in \mathcal{D}_k$ and $\mathrm{color}(V_i) := \mathrm{BLACK}$ for all $V_i \in \mathcal{V}_k$. (We also experimented with a random coloring of the elements, however, this did not lead to any improvements.) If $k < t$, the initial coloring is obtained by extending the coloring of $\mathcal{G}_{k+1}$ to the nodes of $\mathcal{G}_k$.

Refining the final separator The separator $\mathcal{S}_0 = \{V_{p_1}, \ldots, V_{p_t}\}$ of $\mathcal{G}_0$ induces the separator $S = V_{p_1} \cup \ldots \cup V_{p_t}$ of G. Thus, S is composed of boundary segments that belong to various domains. Often S can be improved by exchanging a boundary segment with the interior vertices of a domain. For this refinement process we apply the vertex cover technique described in section 2.3.2. Let us assume that the removal of S splits G in subgraphs $G(B)$ and $G(W)$ with $|B| \geq |W|$. We first pair S with B and compute a minimum weight vertex cover for the bipartite graph $H(S, B_S)$. If this cover does not improve S, we pair S with W. The whole process is repeated until none of the two minimum weight vertex covers improves the actual separator.

In multilevel schemes that rely on edge matchings the vertex cover technique can be applied after each uncoarsening step [35]. However, this is not possible in our coarsening scheme. Any bipartite subgraph $\mathcal{H}_k$ of $\mathcal{G}_k$ contains some elements on one side and some variables on the other side. Since we defined a separator $\mathcal{S}_k$ of $\mathcal{G}_k$ to contain only variables, a vertex cover in $\mathcal{H}_k$ does not induce a valid separator for $\mathcal{G}_k$. Furthermore, the weight of a variable is in general much smaller than the weight of an element. Thus, the original separator has already constituted a minimum weight vertex cover in $\mathcal{H}_k$ with high probability.

3.1.2 Node selection strategies to coarsen a quotient graph

In our coarsening scheme a quotient graph $\mathcal{G}_{k+1}$ is obtained from a quotient graph $\mathcal{G}_k$ by eliminating a set of independent variables $\mathcal{U} \subset \mathcal{V}_k$. This leads us to the following interesting questions: What node selection strategy should be used to find $\mathcal{U}$, and how does the strategy influence the construction of the separators? In this section we will address both questions.

Each separator $\mathcal{S}_k = \{V_{p_1}, \ldots, V_{p_t}\}$ of $\mathcal{G}_k$ induces a separator $S = V_{p_1} \cup \ldots \cup V_{p_t}$ of G. Thus, S is composed of variables that belong to the boundaries of certain elements. Since our primary goal is to find a small (i. e. light weighted) separator S, the elements of a quotient graph should be merged so that the newly formed elements have a small boundary. Additionally, it is important to avoid an unbalanced growing of the elements. Any separator that "touches" a large element will most likely contain a large boundary segment of that element. This may cripple our iterative improvement heuristic. To summarize, we must create quotient graphs that have a fair number of equally sized elements with small boundaries.

The creation of elements with small boundaries corresponds exactly to the objective of the minimum degree algorithm. Therefore, a suitable node selection strategy can be defined as follows: For each variable $V_i \in \mathcal{V}_k$ compute its degree

$$\deg(V_i) = \sum_{V_j \in \mathcal{M}_{V_i}} |V_j| \tag{3.4}$$

where $\mathcal{M}_{V_i}$ contains all variables that share a common domain with V_i, i. e. $V_j \in \mathcal{M}_{V_i} \Leftrightarrow \exists\ D \in \mathcal{D}_k$ with $V_i \in \mathrm{adj}_{\mathcal{G}_k}(D)$ and $\mathcal{V}_j \in \mathrm{adj}_{\mathcal{G}_k}(D)$. Then, sort the variables according to their degrees in ascending order and fill the independent set $\mathcal{U}$ starting with the first one in that order. This node selection strategy is called *minimum-degree-in-quotient-graph* (QMD).

Our second node selection strategy has been motivated by the heavy edge matching heuristic proposed by Karypis and Kumar [37]. Analogously to their heuristic an independent set of heavy weighted variables is eliminated in each coarsening step. However, this simple approach may result in a strong growing of only a few elements. Typically, a heavy weighted variable V_i represents a large boundary segment shared by two large elements/domains. If we remove V_i, the two elements will be merged with V_i to form an even larger element/domain. To avoid an unbalanced growing of elements

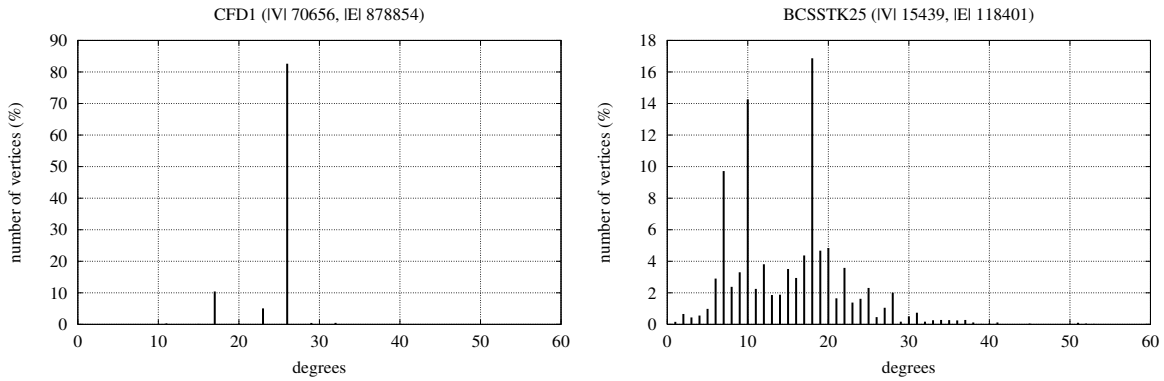

Fig. 3.4: Distribution of vertex degrees for CFD1 and BCSSTK25.

we relate the weight of V_i to the weight of the newly formed element. As a result one obtains a node selection strategy that is based on the score function

$$\mathrm{score}(V_i) = \frac{1}{|V_i|} \cdot \sum_{D \in \mathrm{adj}_{\mathcal{G}_k}(V_i)} |D|. \tag{3.5}$$

The score function is called *maximal-relative-decrease-of-variables-in-quotient-graph* (QMRDV). Note that the score function favors the construction of elements with small aspect ratio. This can be demonstrated as follows. Consider first a variable V_i that has four vertices and is adjacent to two domains, each a 4×4 grid. The score for V_i is $(16 + 16)/4 = 8$. If eliminated, it would result in an 4×9 grid. Next consider a variable V_j that has two vertices and is adjacent to two domains, each a 2×8 grid. The score for this variable is $(16 + 16)/2 = 16$. If eliminated, it would result in an 2×17 grid. In both cases the new grids are roughly of equal size. However, the elimination of V_i is preferred so that one obtains a new element with small aspect ratio.

We demonstrate the effectiveness of our selection strategies for two sample matrices. The first one (CFD1) has been extracted from a computational fluid dynamics application. The graph of CFD1 is very homogeneous. In figure 3.4 the scatter plot on the left shows that 83 % of the vertices have a degree of 26. The second matrix (BCSSTK25) belongs to the well-known Harwell-Boeing sparse matrix collection [19] and represents the finite element model of a tall skyscraper. In contrast to CFD1 the graph of BCSSTK25 is more heterogeneous. The scatter plot on the right of figure 3.4 shows that 14 % of the vertices have a degree of 10 and 17 % a degree of 18. For the remaining 69 % of the vertices the degree varies between one and 58.

Let us first discuss the question of how a node selection strategy influences the merging of elements during the coarsening process. In addition to QMD and QMRDV we are considering a random selection strategy QRAND. Figure 3.5 contains six scatter plots that show the size of all elements created during the coarsening process versus the size of their boundaries for each matrix and for each selection strategy. Since our goal is to cover the "geometry" of a graph by a fair number of equally sized elements that have small boundaries, the entries of the plots should by placed near to the origin of the coordinate system. A close look at the plots on the left of figure 3.5 demonstrates the superiority of QMD and QMRDV over QRAND. When using QRAND to coarsen the quotient graphs of CFD1, many elements of size > 1000 are created. Furthermore, many elements of size < 1000 have a larger boundary than the elements created by QMD or QMRDV. Note that neither QMD nor QMRDV create an element of size > 1000.

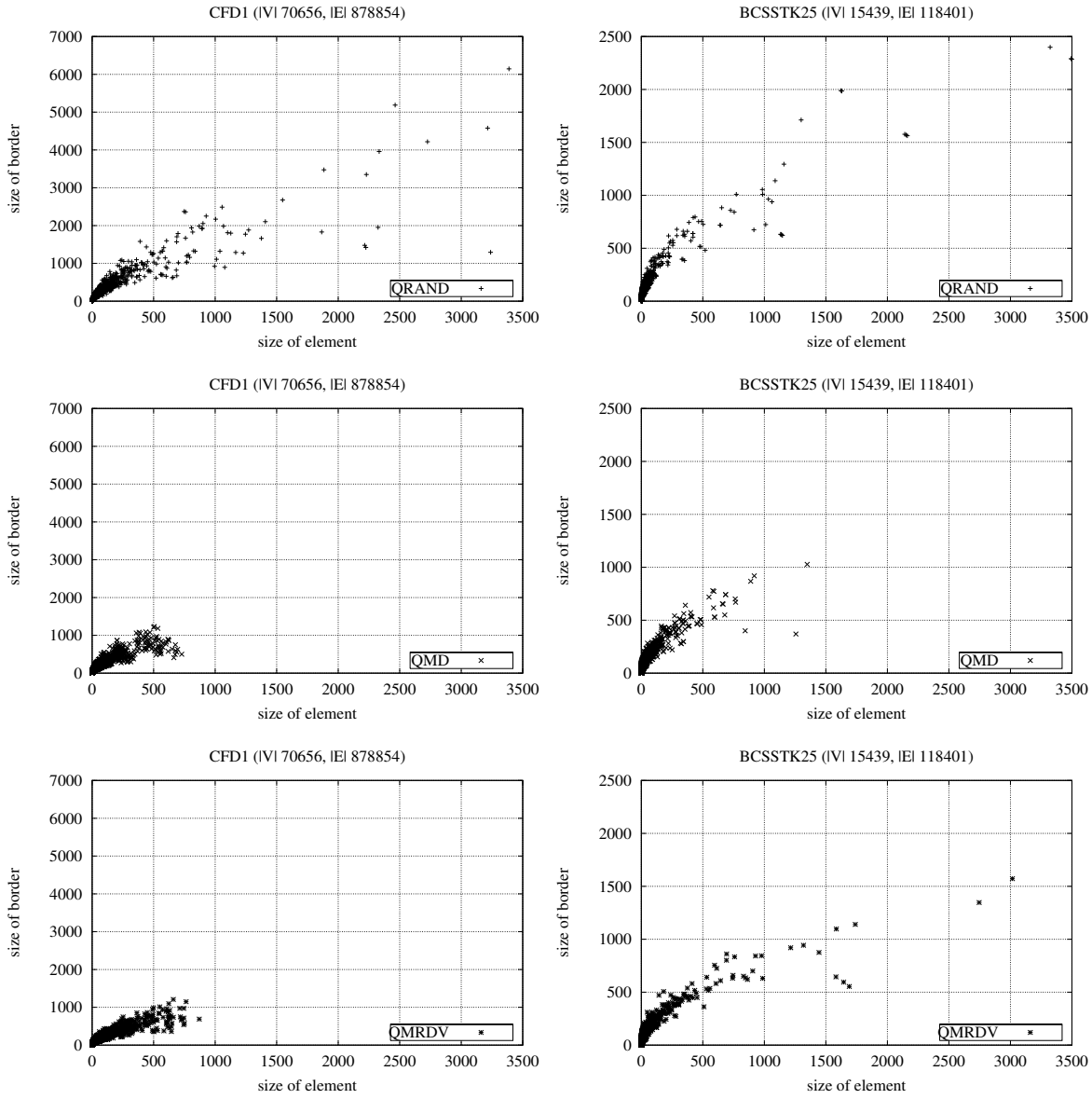

Fig. 3.5: Influence of the node selection strategies QRAND, QMD, and QMRDV on the merging of elements.

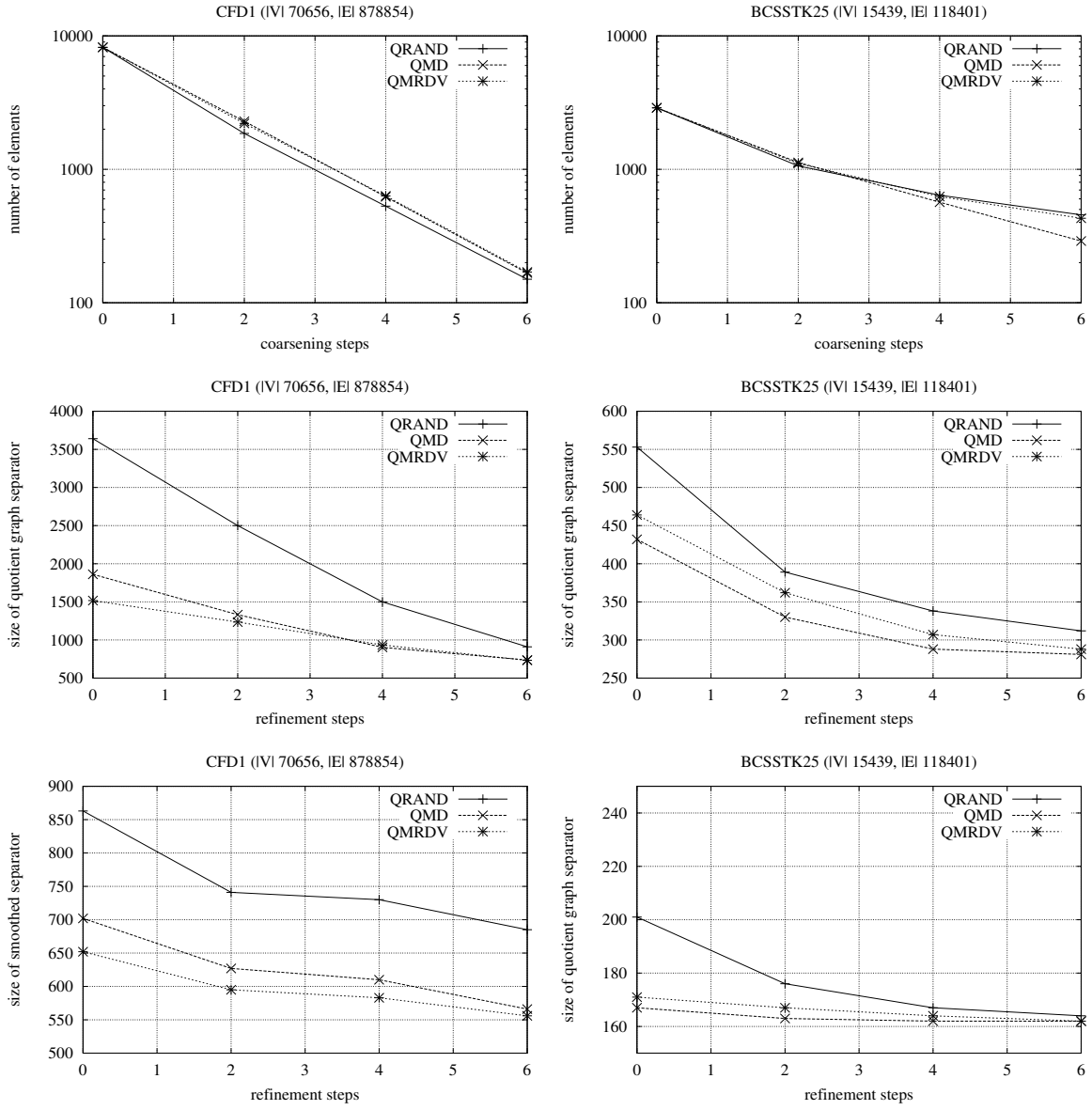

Fig. 3.6: Influence of the node selection strategies QRAND, QMD, and QMRDV on the construction of a vertex separator.

When considering more heterogeneous graphs such as BCSSTK25 the effectiveness of QMD and QMRDV seem to decrease at first glance. However, if one approximates the entries in each plot on the right of figure 3.5 by a line using quadratic regression, the gradient of this line will be much smaller in the plots for QMD and QMRDV. From this observation we can conclude that QMD and QMRDV create more elements with small boundaries. Note that BCSSTK25 is a tall skyscraper. Therefore, the geometry of BCSSTK25 favors the creation of elements with large aspect ratio. A further comparison of the plots on the right of figure 3.5 shows that the minimum degree coarsening strategy achieves a more balanced growing of the elements. In contrast to QRAND and QMRDV there are only two elements of size > 1000 produced by QMD. We observed this general tendency for several other heterogeneous graphs.

To summarize, figure 3.5 provides empirical evidence that QMD and QMRDV create more "well-shaped" elements as a random node selection strategy. When considering heterogeneous graphs, QMD outperforms QMRDV since it achieves a more balanced growing of the elements.

Let us now discuss the question of how the node selection strategy influences the construction of a vertex separator. The two plots on the top of figure 3.6 show the number of elements in the quotient graphs $\mathcal{G}_0, \ldots, \mathcal{G}_6$ according to the chosen node selection strategy. Note that we used a logarithmic scale for the y-axis. In the case of CFD1, the number of elements decreases almost linearly when progressing from one quotient graph to another. Note that most edge matching schemes have the same property. Interestingly, we cannot discover any differences between QRAND, QMD, and QMRDV from this plot, however, as illustrated in figure 3.5, the quality of the produced elements is much better when using QMD or QMRDV. The plot for BCSSTK25 demonstrates that the coarsening of a heterogeneous graph is in general much more difficult (the same holds for edge matching schemes). The problem is that heterogeneous graphs favor an unbalanced growing of the elements. If one represents the coarsening process as a tree, an unbalanced growing of the elements produces hairy subtrees, i. e. subtrees that contain long paths. As a consequence the number of elements is reduced only by a small fraction during each coarsening step. Since minimum degree is very effective in avoiding the creation of large elements (cf. figure 3.5), it is not surprising that from all three node selection strategies QMD reduces the number of elements most effectively. In fact, only for QMD the termination criterion of the coarsening process (less than 200 elements in a quotient graph) is met after six steps.

The two plots in the center of figure 3.6 illustrate that the disadvantages of a bad starting domain decomposition can be compensated (to a certain degree) by the multiple refinement process. For all three node selection strategies we computed the quotient graphs $\mathcal{G}_0, \ldots, \mathcal{G}_6$, determined an initial separator in $\mathcal{G}_6$ using our Fiduccia-Mattheyses heuristic (cf. section 3.2), and refined this separator, again using our Fiduccia-Mattheyses heuristic, until we reached $\mathcal{G}_{6-i}$, $i = 0, \ldots, 6$. Both plots report the sizes of the separators as a function of i. Note that all sizes represent the average over eleven runs and that the adjacency lists of CFD1 and BCSSTK25 have been randomly permuted prior to each run. In the case of QRAND, our improvement heuristic is unable to find a good separator for $\mathcal{G}_6$ (this holds for both CFD1 and BCSSTK25). We must attribute this to the badly shaped elements in $\mathcal{G}_6$. Although the gap between QRAND and QMD, QMRDV closes when progressing to finer quotient graphs, the separators for $\mathcal{G}_0$ are still slightly better in the case of QMD and QMRDV.

In our experiments we did not apply the vertex cover technique to the separators of the ending quotient graph $\mathcal{G}_{6-i}$. This leads us to the following interesting question: Is it really necessary to perform the multiple refinement steps before applying the vertex cover technique to a separator? Remember that in the two-level approach proposed by Ashcraft and Liu [7] the starting quotient graph $\mathcal{G}_6$ and the ending quotient graph $\mathcal{G}_{6-i}$ are identical (i. e. $i = 0$). The two plots at the bottom of figure 3.6 try to give an answer to this question. Here, we additionally applied the vertex cover technique to the separators of $\mathcal{G}_{6-i}$. Again, the sizes of the separators are reported as a function of i. The plot for CFD1

```
IMPROVECOLORING(G = (D ∪ V, E), color)
01: repeat
02:     color* := color;
03:     unmark all D ∈ D;
04:     while there are unmarked elements do
05:         select an unmarked element D;
06:         if (color(D) = BLACK) then
07:             color(D) := WHITE;
08:             for each variable V_i ∈ adj_G(D) do
09:                 UPDATE_{B→W}(V_i, D);
10:         else
11:             color(D) := BLACK;
12:             for each variable V_i ∈ adj_G(D) do
13:                 UPDATE_{W→B}(V_i, D);
14:         end else
15:         mark D;
16:         let (S*, B*, W*) denote the partition induced by color*;
17:         let (S, B, W) denote the partition induced by color;
18:         if F(S, B, W) < F(S*, B*, W*) then
19:             color* := color;
20:     end while
21:     color := color*;
22: until color has not been improved;
```

Fig. 3.7: Function IMPROVECOLORING.

illustrates that the multiple refinement process is indeed necessary to obtain a good vertex separator, especially if one starts with a bad domain decomposition as in the case of QRAND. This explains, why Ashcraft and Liu used a sophisticated network-flow algorithm to smooth a separator. In contrast to CFD1 the plot for BCSSTK25 seems to indicate that a good domain decomposition is sufficient to obtain small vertex separators with the vertex cover technique only. Note that in the case of QMD and QMRDV the refinement steps 4, 5, and 6 do not lead to any improvements. However, one should remember that BCSSTK25 is a tall skyscraper and, therefore, contains a wide spectrum of partitions with varying degrees of imbalance and consistently small separators. Therefore, we must attribute most of the effectiveness of the vertex cover technique to the special geometry of BCSSTK25.

3.2 Improving a separator in the new multilevel scheme

The performance of our multilevel quotient graph method crucially depends on the time required to refine a separator after each uncoarsening step (cf. figure 3.2, line 10). Similar to the two-level approach proposed by Ashcraft and Liu, a Fiduccia-Mattheyses scheme is used to optimize a coloring. In the following we present our iterative improvement heuristic and prove that the runtime of our heuristic is asymptotically identical to the runtime of the vertex Fiduccia-Mattheyses algorithm [5].

3.2.1 The generic improvement scheme

Figure 3.7 presents the structure of our improvement scheme. The algorithm consists of two nested loops. The inner loop computes a sequence of elements that change their colors. Before entering the inner loop, the actual coloring, i. e. the best coloring encountered so far, is saved in color*. Since the

algorithm allows a deterioration of the partition, the inner loop may never terminate. Therefore, each element is allowed to change its color only once. This constraint motivates the outer loop that starts the optimization process again until the actual coloring cannot be improved. In practice the outer loop is executed a small number of times, however, we cannot determine the exact number of iterations. Therefore, the runtime analysis presented in section 3.2.2 is focused on the inner while-loop.

In each iteration of the while-loop an unmarked element D is selected, whose color will be flipped (line 05). Note that each element is marked after it has changed its color (line 15). This guarantees that the color of an element is flipped only once. The element D is selected so that the weight of the induced separator S is decreased the most. In the case of a local minimum, D is chosen so that the deterioration of S is minimized. To be more precise, two heaps are used to store all unmarked black and all unmarked white elements. If none of the heaps is empty, a black element D_b and a white element D_w is chosen that decrease/increase the weight of S the most/least. According to the evaluation function F either D_b or D_w is assigned to D.

Once D has changed its color, functions $\textsc{Update}_{B\to W}$ and $\textsc{Udate}_{W\to B}$ apply the coloring rule (3.3) to each variable $V_i \in \mathrm{adj}_{\mathcal{G}}(D)$. If the new coloring induces a partition (S, B, W) that is better than the best partition (S^*, B^*, W^*) encountered so far, the coloring is saved in color^* (lines 16–19). For the evaluation of a partition we use the function

$$F(S,B,W) = |S| + \rho \cdot \max(0, \tau \cdot \max(|B|,|W|) - \min(|B|,|W|)) + \frac{\max(|B|,|W|) - \min(|B|,|W|)}{\max(|B|,|W|)}. \tag{3.6}$$

In F the weight of a separator is the dominating term. Only if the difference between the weight of B and the weight of W exceeds a certain threshold, the separator will be penalized. The tolerated imbalance is adjusted by $0 < \tau < 1$ and the penalty by $\rho > 0$. The third term is used as a tie-breaker for equally weighted separators, if the imbalance lies within the tolerated range.

An important implementation detail is the efficient calculation of the new partition weights when moving an element D from B to W or from W to B. The partition weights are required for the evaluation of F. Note that moving an element D from B to W or from W to B changes the actual coloring only locally: $\mathrm{color}(D)$ is flipped and some variables $V_i \in \mathrm{adj}_{\mathcal{G}}$ may change their colors. To calculate the weights of the new partition, we define $\Delta_S(D)$, $\Delta_B(D)$, and $\Delta_W(D)$ to be the changes in the respective weights, if D flips its color. Note that these quantities can be positive or negative. If all three values are known, the new partition can be evaluated using the values $|S| + \Delta_S(D)$, $|B| + \Delta_B(D)$, and $|W| + \Delta_W(D)$.

When changing the color of D, the Δ-values of any element D' that shares an adjacent variable V_i with D may have to be updated. This is checked with the help of functions $\textsc{Update}_{B\to W}$ and $\textsc{Update}_{W\to B}$. In both functions four cases are considered. The first two are related to the situation before, and the last two are related to the situation after the change of $\mathrm{color}(D)$. In the following, we only describe function $\textsc{Update}_{B\to W}$ (see figure 3.8). Function $\textsc{Update}_{W\to B}$ can be formulated in the same manner.

Let V_i denote the variable for which $\textsc{Update}_{B\to W}$ has been invoked. We assume that the color of D has not been changed yet (i. e. $\mathrm{color}(D) = \mathrm{BLACK}$). In lines 01–06 we consider the case where all elements in the neighborhood of V_i were colored black except for an element D'. If D' were also colored black, V_i would be moved from S to B. Therefore, $\Delta_S(D')$ contains the value $-|V_i|$ and $\Delta_B(D')$ the value $+|V_i|$. However, after the change of $\mathrm{color}(D)$, there are two white colored elements in the neighborhood of V_i. Therefore, $\Delta_S(D')$ and $\Delta_B(D')$ have to be corrected (lines 04–05). Now let us consider the case that no white colored element D' existed in the neighborhood of V_i

```
UPDATE_{B→W}(V_i, D)
/* Case 1: Before flipping D to WHITE there was only one other WHITE element.
     Search it and update its Δ_B and Δ_S values. */
if (there is only one D' ∈ adj_G(V_i), D' ≠ D, with color(D') = WHITE) then
     Δ_S(D') := Δ_S(D') + |V_i|;
     Δ_B(D') := Δ_B(D') − |V_i|;
end if
/* Case 2: Before flipping D to WHITE all elements were colored BLACK.
     Move V_i into the separator and update Δ_B and Δ_S of all BLACK elements. */
if (color(D') = BLACK for all D' ∈ adj_G(V_i), D' ≠ D) then
     color(V_i) := GRAY;
     for all D' ∈ adj_G(V_i), D' ≠ D do
          Δ_S(D') := Δ_S(D') − |V_i|;
          Δ_B(D') := Δ_B(D') + |V_i|;
     end for
end if
/* Case 3: After flipping D to WHITE there is only one remaining BLACK element.
     Search it and update its Δ_W and Δ_S values. */
if (there is only one D' ∈ adj_G(V_i), D' ≠ D, with color(D') = BLACK) then
     Δ_S(D') := Δ_S(D') − |V_i|;
     Δ_W(D') := Δ_W(D') + |V_i|;
end if
/* Case 4: After flipping D to WHITE all elements are colored WHITE.
     Remove V_i from the separator and update Δ_W and Δ_S of all WHITE elements. */
if (color(D') = WHITE for all D' ∈ adj_G(V_i), D' ≠ D) then
     color(V_i) := WHITE;
     for all D' ∈ adj_G(V_i), D' ≠ D do
          Δ_S(D') := Δ_S(D') + |V_i|;
          Δ_W(D') := Δ_W(D') − |V_i|;
     end for
end if
```

Fig. 3.8: Function $\text{UPDATE}_{B\to W}$.

(lines 07–15). Then, D is the first element in $\mathrm{adj}_{\mathcal{G}}(V_i)$ with $\mathrm{color}(D) = \mathrm{WHITE}$. As a consequence, V_i is moved from B to S. Once V_i has entered the separator, the cost of flipping the color of any other element to white is reduced by $|V_i|$ (lines 12–13).

We now assume that the color of D has been flipped to WHITE. In lines 16–21 we consider the case where all elements in the neighborhood of V_i are colored white except for an element D'. If D' should also be colored white, V_i will move from S to W. Therefore, the value $|V_i|$ is subtracted from $\Delta_S(D')$ and added to $\Delta_W(D')$ (lines 19–20). If there are no remaining black colored elements (lines 22-30), V_i is moved from S to W. However, V_i will move into S again, if any element in $\mathrm{adj}_{\mathcal{G}}(V_i)$ is colored black. Therefore, the cost of flipping the color of any other element to black is increased by $|V_i|$ (lines 27–28).

As mentioned above, function $\textsc{Update}_{W\to B}$ can be formulated in the same manner. We are now able to analyze the performance of our iterative improvement heuristic.

3.2.2 Timing analysis

The cost for one iteration of our improvement heuristic is given by the cost for one execution of the inner while-loop in function ImproveColoring (cf. figure 3.7). Critical are the selection of D (line 05) and the for-loops in lines 08–09 and 12–13. The execution of $\textsc{Update}_{B\to W}(V_i, D)$ or $\textsc{Update}_{W\to B}(V_i, D)$ can require $O(\deg_{\mathcal{G}}(V_i))$ time units, if one of the four if-clauses is true. In this case we speak of an *active* call to the function. Note that all four if-clauses can be evaluated in constant time. To do this, we maintain two counters $\#_B(V_i)$ and $\#_W(V_i)$ for each variable V_i that store the number of black and the number of white elements in $\mathrm{adj}_{\mathcal{G}}(V_i)$. Thus, only active calls to one of the two functions require $O(\deg_{\mathcal{G}}(V_i))$ time units. The following lemma shows that from all $\deg_{\mathcal{G}}(V_i)$ calls at most four can be active. Therefore, the total time required by the two for-loops is $O(e)$ where e denotes the number of edges in $\mathcal{G}$.

Lemma 1 *Let V_i denote a variable of the quotient graph $\mathcal{G}$. In function* ImproveColoring *there are at most four active calls to functions* $\textsc{Update}_{B\to W}(V_i, D)$ *and* $\textsc{Update}_{W\to B}(V_i, D)$.

Proof: The lemma is trivial for all variables V_i with $\deg_{\mathcal{G}}(V_i) \leq 4$. Therefore, let $d := \deg_{\mathcal{G}}(V_i)$ with $d > 4$. Furthermore, let a denote the initial number of white colored elements in $\mathrm{adj}_{\mathcal{G}}(V_i)$. Clearly, $0 \leq a \leq d$ and initially, $\#_W(V_i) = a$. Each call to $\textsc{Update}_{B\to W}(V_i, D)$ increases $\#_W(V_i)$ by one. The call is active, if $\#_W(V_i) \in \{0, 1\}$ when entering the function (cases 1 and 2 in figure 3.8) or if $\#_W(V_i) \in \{d-1, d\}$ when leaving the function (cases 3 and 4 in figure 3.8). Conversely, each call to $\textsc{Update}_{W\to B}(V_i, D)$ decreases $\#_W(V_i)$ by one. This call is active, if $\#_W(V_i) \in \{d-1, d\}$ when entering the function or if $\#_W(V_i) \in \{0, 1\}$ when leaving the function. In the following we name a call to $\textsc{Update}_{B\to W}(V_i, D)$ *upward move* and a call to $\textsc{Update}_{W\to B}(V_i, D)$ *downward move*. Since each element in $\mathrm{adj}_{\mathcal{G}}(V_i)$ changes its color exactly once, there are $d - a$ upward and a downward moves. According to a, five cases can be distinguished:

Case 1: $a = 0$

Then there are d upward moves from which four are active.

Case 2: $a = 1$

Then there are $d - 1$ upward moves. At most three of these moves can be active. Additionally, there is one downward move. This move can be active, too.

Case 3: $a = d$ (analogous to case 1)

Case 4: $a = d - 1$ (analogous to case 2)

Case 5: $2 \leq a \leq d-2$

In the following, we show that there are at most two active upward moves. We denote an upward move by a tuple $(i, i+1)$ where i denotes the number of white colored elements when entering function UPDATE$_{B \to W}$. We distinguish three cases:

Case 5.1: The first active upward move is $(0,1)$.
Then all a downward moves have been executed. From any remaining upward move only $(1,2)$ is active. Note that there cannot occur any active upward move $(d-2, d-1)$ or $(d-1, d)$, because at most $d-3$ upward moves are remaining after the move $(0,1)$.

Case 5.2: The first active upward move is $(1,2)$.
Then there is one remaining downward move and $\leq d-3$ remaining upward moves. Therefore, another upward move $(1,2)$ can occur or, alternatively, an upward move $(d-2, d-1)$. However, it is not possible to have a second $(1,2)$ move together with a $(d-2, d-1)$ move, since the number of available upward moves is bounded by $d-3$.

Case 5.3: The first active upward move is $(d-2, d-1)$.
Then there is at most one remaining upward move. This move can be active.

In the same manner it is shown that for $2 \leq a \leq d-2$ at most 2 active downward moves can occur. This proves the lemma.

■

Finally, we have to analyze the cost for the selection of element D (line 05). As described above, two heaps are maintained, one for all unmarked white and one for all unmarked black elements. Therefore, the selection of D requires $O(\log n)$ time units where n denotes the number of elements in $\mathcal{G}$. However, the heap management produces an additional overhead for each active call of UPDATE$_{B \to W}(V_i, D)$ and UPDATE$_{W \to B}(V_i, D)$. If $\Delta_S(D')$ changes for an element $D' \in \mathrm{adj}_{\mathcal{G}}(V_i)$, its heap position will have to be updated. Therefore, each active call can require $O(\deg_{\mathcal{G}}(V_i) \log n)$ additional time units. Since there are only four active calls for V_i, we obtain the following result:

Theorem 1 *Let $\mathcal{G} = (\mathcal{X}, \mathcal{E})$ be a quotient graph and* color *a coloring of the nodes of $\mathcal{G}$. Furthermore, let n denote the number of elements in $\mathcal{G}$ and e the number of edges in $\mathcal{G}$. Each execution of the inner loop in function* IMPROVECOLORING *requires $O(e \log n)$ time units.*

The theorem proves that the performance of our Fiduccia-Mattheyses scheme is asymptotically identical to the performance of the vertex Fiduccia-Mattheyses heuristic proposed by Ashcraft and Liu [5].

3.3 Tristage multisection

In this section we present a generalization of the multisection scheme of figure 3.1. The new scheme is called tristage multisection and has originally been proposed by Ashcraft, Liu, and Eisenstat [10]. The key idea of tristage multisection is to use the multisector Φ for the computation of a wide spectrum of bottom-up orderings.

3.3.1 Motivation and the basic tristage multisection scheme

It is well recognized that the elements created in the elimination process should have smooth boundaries, i. e. the value $|\mathrm{adj}_G(D)|/|D|$ should be small for all elements D. In a bottom-up algorithm such as minimum degree the node to be eliminated next is chosen according to a local greedy heuristic. As

demonstrated by Berman and Schnitger, this can give elements with a severe fractal boundary. The problem is that a bottom-up algorithm cannot forecast the consequences of a decision made at an early stage of the elimination process.

To overcome this blindness, vertex separators are used to split the graph in multiple subgraphs. The subgraphs can be interpreted as the remaining elements of an unfinished bottom-up ordering, and the vertex separators can be interpreted as the boundaries of these elements. Typically, the vertex separators are constructed by a recursive bisection process. The objective of each bisection step is to find a small vertex separator. Since our original goal is to find elements with small boundaries, we must hope that the recursive bisection process arranges the small separators in such a way that they form elements with small boundaries. In fact we discovered here another form of blindness that virtually any nested dissection algorithm has: it ignores the boundaries of the subgraphs that have to be split. Consequently, the boundaries of the subgraphs are ignored when eliminating the vertex separators according to the given nested dissection order (cf. figure 2.1). However, minimum degree naturally takes into account the boundaries of a graph.

There is one important example where the elimination sequence produced by nested dissection is asymptotically optimal: the $k \times k$ grid. Here, the automatic process of nested dissection splits a quadratic grid into four smaller quadratic grids. As a consequence, the given nested dissection order merges elements with small boundaries to new elements with small boundaries. This observation motivates the following strategy to improve the quality of a multisection ordering [3]: If the vertex separators are arranged so that the elimination sequence induced by nested dissection produces elements with large aspect ratio, then use minimum degree. However, if the elimination sequence produces roughly square domains, then stick to the given nested dissection order.

Theoretically, this strategy is supported by local nested dissection, the most successful ordering algorithm for $h \times k$ grids. Local nested dissection has the form $\mathrm{MS}(\mathrm{ND}, \mathrm{PROFILE})$, i. e. nested dissection is used to number the square domains, while the multisector vertices are numbered by a profile ordering. Replacing the profile ordering by a minimum degree ordering would have no significant influence on the ordering's fill.

To implement this strategy we must know the aspect ratio of the elements produced by the recursive bisection process. However, aspect ratio is not well defined for general graphs. A simple solution to this problem is as follows [3]: Due to the recursive bisection process, the separators of Φ can be represented as a binary tree T. The nodes in level j of the tree represent the separators constructed in recursion level j. The root of the tree is located in level 0 and represents the topmost separator. For each separator S let T_S denote the binary subtree rooted at S. For each such subtree compute a minimum degree ordering and compare it to the given nested dissection order. Then, choose the maximal subtrees for which nested dissection is better than minimum degree. Let the remaining separators form the multisector $\Phi' \subset \Phi$. Now, eliminate all separators in $\Phi - \Phi'$ according to nested dissection and all separators in Φ' according to minimum degree.

If the separator tree has k levels, the time required to compute the minimum degree orderings for all subtrees is roughly k times the cost of a minimum degree ordering for T. This can be seen as follows: When going down one level in the tree, the number of subtrees for which a minimum degree ordering has to be computed doubles. However, at the same time the number of nodes in the subtrees halves. Therefore, the time required to compute the minimum degree orderings for the subtrees of level j is roughly the same for all $j = 0, \ldots, k$. Since the time required to compute the minimum degree orderings in level j is bounded above by the time required to compute a minimum degree ordering for T, the statement follows immediately. Together with the minimum degree orderings on the domains and the minimum degree ordering on the multisector Φ', the overall runtime of this simple tristage multisection algorithm is roughly $k + 1$ times the cost of a single minimum degree ordering.

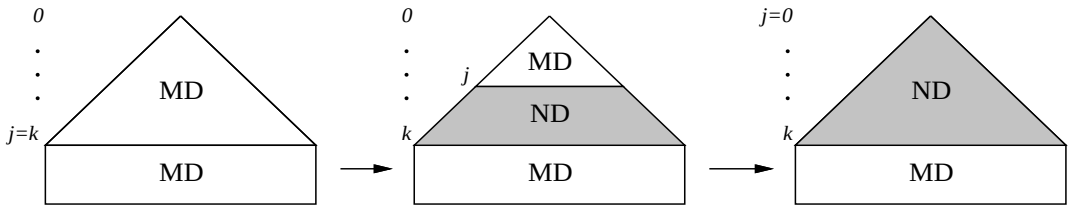

Fig. 3.9: Spectrum of orderings for $j = k$ (left) to $j = 0$ (right).

3.3.2 The tristage multisection scheme of Ashcraft, Liu, and Eisenstat

A much more efficient scheme has been proposed by Ashcraft, Liu, and Eisenstat [3, 10]. Let k denote the maximal level of the separator tree, and let $j \in \{0, \ldots, k\}$. In their tristage multisection scheme all orderings are evaluated that can be created as follows:

(1) Eliminate all vertices in the domains using minimum degree.

(2) Eliminate all vertex separators in the lower levels $k, \ldots, j-1$ of Φ according to the given nested dissection order.

(3) Eliminate all vertex separators in the upper levels $j, \ldots, 0$ of Φ using minimum degree.

From all $k+1$ orderings obtained this way, the best one is chosen. Figure 3.9 shows that there is a wide spectrum of orderings of this form. Each ordering is symbolized by a triangle and a rectangle below the triangle. The triangle represents the separator tree and contains all vertices of the multisector. The rectangle contains the vertices of the domains. On the left side of the spectrum, i. e. $j = k$, minimum degree is used to order both the domains as well as the separators in Φ. As a result one obtains a multisection ordering of the form $\mathrm{MS}(\mathrm{MD}, \mathrm{MD})$. At the opposite side of the spectrum, i. e. $j = 0$, the separators are eliminated according to the given nested dissection order and one obtains an incomplete nested dissection ordering, i. e. a multisection ordering of the form $\mathrm{MS}(\mathrm{MD}, \mathrm{ND})$.

The bulk of the computation time is spent during step (1) where the vertices in the domains are numbered using minimum degree. Since each ordering uses the minimum degree ordering on the domains, it can be computed once and "spliced" into the other orderings. Note that there is also a lot of overlap between the orderings on Φ. For $j \in \{0, \ldots, k\}$ let $\Phi_j \subset \Phi$ denote the multisector that contains all separators in levels $0, \ldots, j$, i. e. $\Phi_k = \Phi$. If the sequence of orderings is evaluated from $j = k$ down to $j = 0$, the nested dissection ordering on the separators $\Phi - \Phi_j$ (cf. step (2)) can be built using the nested dissection ordering on $\Phi - \Phi_{j+1}$. Therefore, only the minimum degree ordering on the separators Φ_j (cf. step (3)) need to be computed for every ordering.

Figure 3.10 presents an efficient way to evaluate all $k + 1$ orderings of the spectrum. Similar to the multisection scheme in figure 3.1 parameters ord_Φ, ord_1, and ord_2 are used to specify the node selection strategies. Again, the multisection domain decomposition is constructed by a recursive bisection process that uses the multilevel scheme introduced in section 3.1. Once a multisector Φ_k has been found, the vertices in the domains $\Omega_1, \ldots, \Omega_q$ are eliminated according to ord_1. The number of floating point operations required to factor these vertices is stored in ops_0. As a by-product one obtains the Schur complement graph G_{Φ_k}. The various orderings for the separator vertices in G_{Φ_k} are then evaluated in the for-loop of lines 04–13.

Each iteration j, $j < k$, of the for-loop starts with the construction of the actual elimination graph G_{Φ_j}. This graph is obtained from $G_{\Phi_{j+1}}$, the actual elimination graph of iteration $j+1$, by eliminating all separators in level $j + 1$. Variable ops_1 is used to sum up the factor operations required by all these nested dissection steps. In the first iteration of the for-loop, i. e. $j = k$, we have $G_{\Phi_j} = G_{\Phi_k}$. Once the

TRISTAGEMULTISECTION($\mathrm{ord}_\Phi, \mathrm{ord}_1, \mathrm{ord}_2$)
01: Determine a domain decomposition $(\Phi_k, \Omega_1, \ldots, \Omega_q)$ of G by a recursive bisection process. Use node selection strategy ord_Φ to construct the vertex separators.
02: **for** each set Ω_i **do**
03: Eliminate all vertices in Ω_i using node selection strategy ord_1.
04: Store operation count in ops_0 and construct elimination graph G_{Φ_k}.
03: $\mathrm{ops}_1 := 0$; $\mathrm{ops}^* := \infty$;
04: **for** $(j := k)$ **downto** 0 **do**
05: **if** $(j < k)$ **then**
06: Eliminate from $G_{\Phi_{j+1}}$ all separators in level $j+1$ to obtain the actual elimination graph G_{Φ_j}. Add operation count to ops_1.
07: **end if**
08: Order the vertices in G_{Φ_j} using node selection strategy ord_2.
Store operation count in ops_2.
09: **if** $(\mathrm{ops}_0 + \mathrm{ops}_1 + \mathrm{ops}_2 < \mathrm{ops}^*)$ **then**
10: $j^* := j$;
11: $\mathrm{ops}^* := \mathrm{ops}_0 + \mathrm{ops}_1 + \mathrm{ops}_2$;
12: **end if**
13: **end for**
14: Splice together the bottom-up orderings on $\Omega_1, \ldots, \Omega_q$, the nested dissection ordering on $\Phi_k - \Phi_{j^*}$, and the bottom-up ordering on Φ_{j^*}.

Fig. 3.10: Function TRISTAGEMULTISECTION.

actual elimination graph G_{Φ_j} has been constructed, all separators in G_{Φ_j} are eliminated using node selection strategy ord_2. The number of floating point operations required by this elimination step is stored in ops_2. Note that we must use a copy of G_{Φ_j}, since the graph is still needed in iteration $j-1$. Once all vertices in G_{Φ_j} have been ordered, a new three-level multisection ordering is completed. If it is the best ordering encountered so far, the level number j is stored in j^*.

After the termination of the for-loop, j^* defines the level at which the ordering of vertex separators should switch from nested dissection to minimum degree. Therefore, the best ordering of the spectrum can be obtained as described in line 14.

The overall cost for evaluating all $k+1$ orderings is bounded above by roughly the cost of three bottom-up orderings. This can be seen as follows: When constructing the actual elimination graph G_{Φ_j} from $G_{\Phi_{j+1}}$ the number of nodes halves. Therefore, the cost of computing a bottom-up ordering for G_{Φ_j} is roughly half the cost of computing a bottom-up ordering for $G_{\Phi_{j+1}}$. As a consequence, the cost of line 08 is bounded above by the cost of two bottom-up orderings. Together with the cost for computing the bottom-up orderings on the domains (lines 02–03) and the cost for constructing the actual elimination graph G_{Φ_j} (line 06) we obtain the desired result.

4 Computational results

In this section we empirically evaluate the multisection and tristage multisection algorithms of our methodology. Our primary metric for measuring the quality of an ordering is the number of floating point operations required to factor matrix A. This number is closely related to the time required for the overall factorization process. All results have been generated on a SUN Ultra with 200MHz UltraSPARC processor.

4.1 The programs pord and multipord

For our computational experiments we have developed two programs, called **pord** and **multipord** (Paderborn ORDering tools). The programs implement the functions MULTISECTION and TRISTAGE-MULTISECTION, respectively. Additionally, both programs perform a preprocessing and a postprocessing step. During the preprocessing step the programs attempt to compress the graph G of matrix A by identifying indistinguishable vertices. Especially matrices arising from finite element and finite difference discretizations can contain multiple columns with identical adjacency structure. The compressed graph G_c is formed by merging indistinguishable vertices of G to a single weighted node. The weight of this node is equal to the number of merged vertices. Functions MULTISECTION and TRISTAGEMULTISECTION are then invoked for the compressed graph. As a result one obtains an ordering π_c for G_c that will be expanded to an ordering π for G during the postprocessing step. More information concerning compressed graphs can be found in [2, 16, 35].

Although the main algorithmic components of our approach have been introduced in section 3, a number of details have been left out. The following list describes some important parameters of **pord** and **multipord** that can greatly influence both runtime and ordering quality. For most parameters default values are provided that are used throughout our computational experiments.

- In functions MULTISECTION and TRISTAGEMULTISECTION the selection startegies ord_1 and ord_2 can take the values AMD, AMF, AMMF, AMIND, or MF. Note that we are using constrained versions of the selection strategies in the case of ord_1, i. e. the multisector vertices are included when computing the score of a domain vertex. In function MULTISECTION the selection strategy ord_2 can also be set to ND. The separators are then eliminated according to the given nested dissection order. In both functions ord_Φ can take a value from $\{\mathrm{QMD}, \mathrm{QMRDV}, \mathrm{QRAND}\}$. Although QMD produces slightly better domain decompositions for heterogeneous graphs than QMRDV (cf. figure 3.5), we set ord_Φ to QMRDV in all our experiments, since the computation of $\mathrm{score}_{\mathrm{QMRDV}}(V_i)$ is much cheaper than the computation of $\deg(V_i)$.
- In functions MULTISECTION and TRISTAGEMULTISECTION the recursive bisection process continues until a subgraph has 100 or fewer vertices. However, at most 255 separators are constructed. We used this threshold to improve the efficiency of our programs. We observed no further improvements when more separators were constructed.
- In function SEPARATOR (cf. figure 3.2) the coarsening process terminates when a quotient graph has 200 or fewer elements. This provides some degrees of freedom to our coloring heuristic. In our implementation of IMPROVECOLORING (cf. figure 3.7) there is an early termination of the inner while-loop, if no better partition is found in 100 subsequent iterations.
- In our evaluation function (3.6) parameters τ and ρ are set to 0.5 and 100, respectively. The value for τ allows an imbalance of 50 % without penalizing the separator. This high imbalance is motivated by our interpretation of vertex separators. Remember that we interpret vertex separators as the boundaries of the remaining elements in an unfinished bottom-up ordering. In this context it is important to have elements with small boundaries and not to have equally sized elements. Therefore, the minimization of partition imbalance is only a secondary objective in our nested dissection process. Empirically, our choice of τ is also supported by the experiments reported in [55].

Numerous experiments have shown that the default parameters described above are very effective on a wide range of matrix types. Note that we did not provide default parameters for ord_1 and ord_2. In section 4.3 we explore the influence of these parameters on the quality of the produced orderings.

4.2 The benchmark matrices

For our computational experiments we have tried to select a realistic set of benchmark matrices. The first two matrices are Laplace matrices that correspond to a 127×127 grid problem with five point difference operator (GRID) and nine point difference operator (MESH). The 15 BCSSTK matrices have been selected from the Harwell-Boeing sparse matrix collection. A detailed description of the matrices can be found in [19]. MAT02HBF and MAT03HBF have been supplied by a consulting agency of the German car industry. The matrices have been extracted from a crash simulation tool. The matrices BRACK2, CRACK, WAVE, HERMES, CYL3, and DIME20 are FEM meshes. All meshes come from a 3-D problem domain, except for CRACK, which is a 2-D FEM mesh. The first three matrices (BRACK2, CRACK, and WAVE) come from Carnegie-Mellon-University. HERMES comes from University of Michigan and is a model of the European space shuttle. The last two matrices (CYL3 and DIME20) have been given to us by C. Walshaw at University of Southampton. All other matrices can be found in Tim Davis' sparse matrix collection [17]. Most of these matrices have been extracted from commercial structural analysis and computational fluid dynamics applications.

Table 4.1 reports relevant statistics about our test matrices. The first four columns give the number of vertices and edges in G and G_c. The last two columns show the number of factor entries (scaled by 10^3) and the number of floating point operations (scaled by 10^6) when ordering the matrices using our implementation of the approximate minimum degree algorithm (AMD) proposed by Amestoy et al. [1]. AMD will serve as a point of reference during our computational experiments.

All results reported in this paper come from an ordering of the unpermuted input matrix A. It is well known that ordering methods are very sensitive to the initial ordering of A. However, in most practical situations the initial ordering is far away from random and especially bottom-up methods can benefit from it.

4.3 Results for pord and multipord

In this section we present experimental results for the programs **pord** and **multipord**. Table 4.2 shows the number of floating point operations (scaled by 10^6) when ordering the benchmark matrices using **pord**. The numbers in brackets give the operation counts relative to AMD. The function MULTISECTION has been executed with parameters $\mathrm{ord}_1 = \mathrm{ord}_2 = \mathrm{AMD}$ (column 1), $\mathrm{ord}_1 = \mathrm{ord}_2 = \mathrm{AMMF}$ (column 2), $\mathrm{ord}_1 = \mathrm{AMMF}$, $\mathrm{ord}_2 = \mathrm{MF}$ (column 3), and $\mathrm{ord}_1 = \mathrm{AMMF}$, $\mathrm{ord}_2 = \mathrm{ND}$ (column 4). Although more parameter combinations are possible, we limit or study to these four. In all experiments, ord_Φ has been set to QMRDV.

Rothberg and Eisenstat [56] have shown that approximate minimum mean local fill (AMMF) and, especially, exact minimum local fill (MF) produce significantly better orderings than minimum degree. This is the motivation for using the two selection strategies in our multisection scheme. Indeed, a comparison of columns 1 and 2 of table 4.2 demonstrates that the operation counts are reduced when switching from $\mathrm{ord}_1 = \mathrm{ord}_2 = \mathrm{AMD}$ to $\mathrm{ord}_1 = \mathrm{ord}_2 = \mathrm{AMMF}$. A further reduction is achieved, when the vertex separators in the Schur complement graph are numbered using selection strategy MF. Typically, the separator vertices are merged to a few indistinguishable nodes so that the Schur complement graph is small. Therefore, we can afford using MF on the multisector vertices. Note that this combination can be very effective because most of the factorization work is done for the columns of the multisector. With this parameter setting the number of floating point operations is reduced by 41 % compared to AMD.

A comparison of columns 2 and 4 demonstrates that the quality of an ordering can deteriorate when eliminating the separator vertices according to the given nested dissection order. However,

	G		G_c		AMD	
Matrix	$\|V\|$	$\|E\|$	$\|V_c\|$	$\|E_c\|$	NZL/10^3	OPS/10^6
GRID127x127	16129	32004	16129	32004	346	27
MESH127x127	16129	63756	16129	63756	527	43
BCSSTK15	3948	56934	3948	56934	624	155
BCSSTK16	4884	142747	1778	18251	763	162
BCSSTK17	10974	208838	5219	40531	985	138
BCSSTK18	11948	68571	10926	61086	625	127
BCSSTK23	3134	42044	2930	17628	428	125
BCSSTK24	3562	78174	892	6378	270	31
BCSSTK25	15439	118401	13183	80982	1464	316
BCSSTK29	13992	302748	10202	156923	1760	467
BCSSTK30	28924	1007284	9289	111442	3786	947
BCSSTK31	35588	572914	17403	144403	5281	2593
BCSSTK32	44609	985046	14821	113487	5002	989
BCSSTK33	8738	291583	4344	82142	2480	1140
BCSSTK35	30237	709963	6611	32967	2725	399
BCSSTK36	23052	560044	4351	18583	2719	616
BCSSTK37	25503	557737	7093	44462	2755	535
BCSSTK38	8032	173714	3456	40656	718	115
MAT02HBF	46949	1117809	6707	19938	5057	1344
MAT03HBF	73752	1761718	10536	31438	10061	4184
STRUCT3	53570	560062	41644	340543	5040	1096
STRUCT4	4350	116724	4350	116724	2357	2004
PWT	36519	144794	36515	144774	1556	173
BRACK2	62631	366559	62631	366559	7275	3085
CRACK	10240	30380	10240	30380	163	8
3DTUBE	45330	1584144	15909	181865	26310	30053
CFD1	70656	878854	70656	878854	37663	44556
CFD2	123440	1482229	123440	1482229	74884	136477
CYL3	232362	457853	232362	457853	77440	208480
DIME20	224843	336024	224843	336024	3430	330
GEARBOX	153746	4463329	56175	693142	46325	41121
NASASRB	54870	1311227	24954	275813	11624	4538
WAVE	156317	1059331	156316	1059325	114930	372458
PWTK	217918	5708253	41531	221130	60305	49086
HERMES	320194	3722641	320194	3722641	323055	1434744

Tab. 4.1: Statistics for benchmark matrices.

there are only three matrices (BCSSTK17, BCSSTK25, and BCSSTK30) where the nested dissection variant of **pord** does not produce as good an ordering as AMD. The average number of floating point operations is still reduced by 33 % compared to AMD. Note that the graphs of BCSSTK17 (model of a pressure vessel), BCSSTK25 (model of a tall skyscraper), and BCSSTK30 (model of an off-shore generator platform) have large aspect ratio so that the topmost separators of the multisector are placed next to each other similar to the separators in the $h \times k$ grid of figure 2.1. This explains the bad performance of nested dissection for these graphs. However, there are also matrices for which the number of floating point operations is reduced when following the given nested dissection order (CYL3 and WAVE).

A comparison of the last three columns of table 4.2 shows that the quality of the orderings produced by **pord** can vary significantly even when using the same multisector and the same selection strategy ord_1. Since most of the factorization work is done on the columns of the multisector, much effort should be devoted to the elimination of the multisector vertices. As described in section 3.3 the optimal elimination sequence depends on the arrangement of the vertex separators. The program **multipord** allows us to evaluate a wide spectrum of elimination sequences.

Table 4.3 reports the operation counts of all orderings produced in the for-loop of function TRISTAGEMULTISECTION. The function has been executed with parameters $\mathrm{ord}_1 = \mathrm{ord}_2 = \mathrm{AMMF}$ and $\mathrm{ord}_\Phi = \mathrm{QMRDV}$. The numbers in the last column ($j = 0$) correspond to the numbers in column $(\mathrm{AMMF}, \mathrm{ND})$ of table 4.2. The leftmost numbers can be found in the corresponding rows of column $(\mathrm{AMMF}, \mathrm{AMMF})$. Note that at most 255 vertex separators are constructed in our bisection process so that $k \leq 7$. Furthermore, the construction of vertex separators stops, if a subgraph has 100 or fewer vertices. The operation counts of the best orderings are printed in boldface. Exactly these orderings are returned by function TRISTAGEMULTISECTION. As will be shown in table 4.5, the runtime of **multipord** does only marginally increase compared to **pord**.

4.4 Comparison with other ordering codes

Table 4.4 shows the number of floating point operations for the programs METIS [38] (version 4.0), SCOTCH [50] (version 3.3), SPOOLES [4] (version 2.2), **pord**, and **multipord**. Again, the numbers in brackets give the operation counts relative to AMD. METIS and SCOTCH represent state-of-the-art implementations of incomplete nested dissection. For the construction of vertex separators both programs are using a multilevel approach that is based on a matching technique. The separators are numbered according to the given nested dissection order. The two programs differ in how the domains are numbered. METIS uses multiple minimum degree, while SCOTCH relies on a constrained version of the approximate minimum degree algorithm proposed by Amestoy et al. [51].

SPOOLES implements the two-level approach described in section 2.3. The vertices in the domains as well as the vertices in the multisector are numbered using multiple minimum degree. However, a constrained version of MMD is applied to the vertices in the domains. As mentioned above, **pord** and **multipord** implement the functions MULTISECTION and TRISTAGEMULTISECTION, respectively. The results reported for **pord** correspond to the numbers in column $(\mathrm{AMMF}, \mathrm{AMMF})$ of table 4.2. The results reported for **multipord** correspond to the operation counts of the best orderings obtained by TRISTAGEMULTISECTION (cf. table 4.3).

METIS, SCOTCH, and SPOOLES recursively split a subgraph until it has 200 or fewer vertices. In **pord** and **multipord** the recursion continues until a subgraph has 100 or fewer vertices. However, at most 255 separators are constructed. As mentioned in section 2.3 SPOOLES uses a randomized greedy domain-growing algorithm to construct a domain decomposition. To reduce the variations due to different random numbers we have run it several times for each matrix. The results reported

Matrix	(AMD, AMD)	(AMMF, AMMF)	(AMMF, MF)	(AMMF, ND)
GRID127x127	16 (0.59)	16 (0.59)	16 (0.59)	17 (0.63)
MESH127x127	36 (0.84)	38 (0.88)	35 (0.81)	39 (0.91)
BCSSTK15	93 (0.60)	79 (0.51)	86 (0.55)	82 (0.53)
BCSSTK16	112 (0.69)	117 (0.72)	115 (0.71)	135 (0.83)
BCSSTK17	124 (0.90)	123 (0.89)	120 (0.87)	172 (1.25)
BCSSTK18	87 (0.68)	85 (0.67)	82 (0.65)	96 (0.76)
BCSSTK23	98 (0.78)	85 (0.68)	98 (0.78)	90 (0.72)
BCSSTK24	31 (1.00)	30 (0.97)	30 (0.97)	31 (1.00)
BCSSTK25	256 (0.81)	207 (0.65)	230 (0.73)	355 (1.12)
BCSSTK29	360 (0.77)	326 (0.70)	277 (0.59)	336 (0.72)
BCSSTK30	722 (0.76)	702 (0.74)	707 (0.74)	982 (1.04)
BCSSTK31	1226 (0.47)	1215 (0.47)	1184 (0.46)	1291 (0.50)
BCSSTK32	827 (0.84)	774 (0.78)	767 (0.77)	969 (0.98)
BCSSTK33	740 (0.65)	626 (0.55)	619 (0.54)	644 (0.56)
BCSSTK35	374 (0.94)	367 (0.92)	369 (0.92)	384 (0.96)
BCSSTK36	461 (0.75)	458 (0.74)	460 (0.75)	500 (0.81)
BCSSTK37	404 (0.75)	403 (0.75)	388 (0.72)	445 (0.83)
BCSSTK38	91 (0.79)	91 (0.79)	89 (0.77)	106 (0.92)
MAT02HBF	1091 (0.81)	1088 (0.80)	1093 (0.81)	1222 (0.91)
MAT03HBF	2654 (0.63)	2723 (0.65)	2473 (0.59)	2666 (0.64)
STRUCT3	717 (0.65)	730 (0.67)	664 (0.61)	731 (0.67)
STRUCT4	574 (0.29)	541 (0.27)	617 (0.31)	504 (0.25)
PWT	108 (0.62)	109 (0.63)	107 (0.62)	110 (0.64)
BRACK2	1923 (0.62)	1610 (0.52)	1661 (0.53)	1982 (0.64)
CRACK	7 (0.87)	7 (0.87)	6 (0.75)	7 (0.87)
3DTUBE	13235 (0.44)	14839 (0.49)	11437 (0.38)	12303 (0.41)
CFD1	9885 (0.22)	8814 (0.20)	8799 (0.20)	11302 (0.25)
CFD2	34421 (0.25)	27978 (0.20)	27902 (0.20)	28636 (0.21)
CYL3	57971 (0.29)	45386 (0.22)	41372 (0.20)	39791 (0.19)
DIME20	175 (0.53)	175 (0.53)	167 (0.51)	191 (0.58)
GEARBOX	18034 (0.44)	17404 (0.42)	17505 (0.43)	17987 (0.44)
NASASRB	2839 (0.63)	2613 (0.58)	2582 (0.56)	3437 (0.76)
WAVE	160843 (0.43)	119694 (0.32)	108804 (0.29)	97480 (0.26)
PWTK	23019 (0.47)	22658 (0.46)	22323 (0.45)	23119 (0.47)
HERMES	326902 (0.23)	266255 (0.19)	268303 (0.19)	265133 (0.18)
MEAN	(0.63)	(0.60)	(0.59)	(0.67)

Tab. 4.2: Number of floating point operations (scaled by 10^6) for **pord**. Function MULTISECTION has been called with parameters $\mathrm{ord}_1 = \mathrm{ord}_2 = \mathrm{AMD}$ (column 1), $\mathrm{ord}_1 = \mathrm{ord}_2 = \mathrm{AMMF}$ (column 2), $\mathrm{ord}_1 = \mathrm{AMMF}$, $\mathrm{ord}_2 = \mathrm{MF}$ (column 3), and $\mathrm{ord}_1 = \mathrm{AMMF}$, $\mathrm{ord}_2 = \mathrm{ND}$ (column 4).

	Iteration j							
Matrix	7	6	5	4	3	2	1	0
GRID127x127	–	**16**	**16**	**16**	**16**	17	17	17
MESH127x127	–	38	45	42	**35**	36	39	39
BCSSTK15	–	–	–	79	**78**	**78**	82	82
BCSSTK16	–	–	–	**117**	120	130	135	135
BCSSTK17	–	–	**123**	137	156	171	171	172
BCSSTK18	–	85	87	**81**	86	91	96	96
BCSSTK23	–	–	–	–	85	**84**	91	90
BCSSTK24	–	–	–	–	**30**	31	31	31
BCSSTK25	–	**207**	216	244	309	326	355	355
BCSSTK29	–	326	**309**	321	330	316	336	336
BCSSTK30	–	**702**	725	809	913	973	982	986
BCSSTK31	**1215**	1266	1317	1261	1289	1285	1291	1291
BCSSTK32	**774**	**774**	794	833	929	968	969	969
BCSSTK33	–	–	626	626	**615**	644	644	644
BCSSTK35	–	**367**	376	378	383	385	384	384
BCSSTK36	–	–	**458**	476	475	497	500	500
BCSSTK37	–	403	**393**	403	413	433	445	445
BCSSTK38	–	–	**91**	93	97	103	106	106
MAT02HBF	–	**1088**	1107	1128	1140	1192	1222	1222
MAT03HBF	2723	2726	**2542**	2790	2619	2659	2663	2666
STRUCT3	730	**691**	712	707	725	731	731	731
STRUCT4	–	–	–	541	542	523	**503**	504
PWT	109	111	**108**	**108**	109	110	110	110
BRACK2	**1610**	1646	1633	1655	1817	1981	1982	1982
CRACK	–	7	7	**6**	7	7	7	7
3DTUBE	14839	14953	12907	12366	13392	**12303**	**12303**	**12303**
CFD1	**8814**	8961	10011	10533	10798	10947	11302	11302
CFD2	27978	26379	27616	**26141**	26554	28371	28636	28636
CYL3	45386	40806	41324	41525	40623	40190	**39791**	**39791**
DIME20	175	**174**	182	183	187	190	191	191
GEARBOX	**17404**	17643	17658	17748	17791	17987	17987	17987
NASASRB	**2613**	2706	2767	2955	3201	3280	3437	3437
WAVE	119694	98274	100509	101306	99559	97463	**97472**	97480
PWTK	**22658**	22680	22868	23068	23088	23042	23119	23119
HERMES	266255	260469	260060	**250740**	252313	255478	265133	265133

Tab. 4.3: Number of floating point operations (scaled by 10^6) for **multipord**. Function TRISTAGEMULTISECTION has been called with parameters $\mathrm{ord}_1 = \mathrm{ord}_2 = \mathrm{AMMF}$. The operation counts of the best orderings are printed in boldface. Exactly these orderings are returned by TRISTAGEMULTISECTION.

Name	METIS-4.0	SCOTCH-3.3	SPOOLES-2.2	**pord**	**multipord**
GRID127x127	22 (0.81)	25 (0.93)	20 (0.74)	16 (0.59)	16 (0.59)
MESH127x127	37 (0.86)	40 (0.93)	43 (1.00)	38 (0.88)	35 (0.81)
BCSSTK15	87 (0.56)	93 (0.60)	95 (0.61)	79 (0.51)	78 (0.50)
BCSSTK16	140 (0.86)	140 (0.86)	129 (0.79)	117 (0.72)	117 (0.72)
BCSSTK17	184 (1.33)	161 (1.16)	135 (0.98)	123 (0.89)	123 (0.89)
BCSSTK18	101 (0.80)	77 (0.60)	84 (0.66)	85 (0.67)	81 (0.64)
BCSSTK23	98 (0.78)	94 (0.75)	91 (0.72)	85 (0.68)	84 (0.67)
BCSSTK24	34 (1.09)	35 (1.13)	38 (1.22)	31 (0.97)	30 (0.97)
BCSSTK25	380 (1.20)	348 (1.10)	235 (0.74)	207 (0.65)	207 (0.65)
BCSSTK29	345 (0.74)	327 (0.70)	341 (0.73)	326 (0.70)	309 (0.66)
BCSSTK30	1203 (1.27)	1114 (1.18)	833 (0.88)	702 (0.74)	702 (0.74)
BCSSTK31	1165 (0.45)	1219 (0.47)	1530 (0.59)	1215 (0.47)	1215 (0.47)
BCSSTK32	1213 (1.23)	1175 (1.19)	866 (0.88)	774 (0.78)	774 (0.78)
BCSSTK33	909 (0.78)	674 (0.59)	739 (0.65)	626 (0.55)	615 (0.54)
BCSSTK35	523 (1.31)	422 (1.06)	393 (0.98)	367 (0.92)	367 (0.92)
BCSSTK36	615 (1.00)	583 (0.95)	496 (0.81)	458 (0.74)	458 (0.74)
BCSSTK37	694 (1.30)	653 (1.22)	433 (0.81)	403 (0.75)	397 (0.74)
BCSSTK38	135 (1.17)	108 (0.94)	103 (0.90)	91 (0.79)	91 (0.79)
MAT02HBF	1192 (0.87)	1099 (0.82)	1156 (0.86)	1088 (0.81)	1088 (0.81)
MAT03HBF	2724 (0.65)	2924 (0.70)	3607 (0.86)	2723 (0.65)	2542 (0.61)
STRUCT3	826 (0.75)	857 (0.78)	773 (0.70)	730 (0.67)	691 (0.63)
STRUCT4	535 (0.27)	541 (0.27)	691 (0.34)	541 (0.27)	503 (0.25)
PWT	110 (0.64)	101 (0.58)	108 (0.62)	109 (0.63)	108 (0.62)
BRACK2	1908 (0.62)	1821 (0.59)	1900 (0.62)	1610 (0.52)	1610 (0.52)
CRACK	7 (0.87)	7 (0.87)	7 (0.87)	7 (0.87)	6 (0.75)
3DTUBE	12071 (0.40)	15834 (0.53)	15523 (0.52)	14839 (0.49)	12303 (0.41)
CFD1	16345 (0.37)	15027 (0.34)	10509 (0.24)	8814 (0.20)	8814 (0.20)
CFD2	31024 (0.23)	35659 (0.26)	35798 (0.26)	27978 (0.20)	26141 (0.19)
CYL3	32164 (0.15)	31670 (0.15)	80818 (0.39)	45386 (0.22)	39791 (0.19)
DIME20	196 (0.59)	181 (0.55)	235 (0.71)	175 (0.53)	175 (0.53)
GEARBOX	20390 (0.50)	24755 (0.60)	21516 (0.52)	17404 (0.42)	17404 (0.42)
NASASRB	3494 (0.77)	3748 (0.83)	2801 (0.62)	2613 (0.58)	2613 (0.58)
WAVE	120180 (0.32)	98547 (0.26)	188316 (0.50)	119694 (0.32)	97463 (0.26)
PWTK	22039 (0.45)	23275 (0.47)	28313 (0.58)	22658 (0.46)	22658 (0.46)
HERMES	258970 (0.18)	368863 (0.26)	518549 (0.36)	266255 (0.19)	250740 (0.17)
MEAN	(0.75)	(0.72)	(0.69)	(0.60)	(0.58)

Tab. 4.4: Comparison of operation counts.

Matrix	METIS-4.0		SCOTCH-3.3		SPOOLES-2.2		**pord**		**multipord**	
GRID127x127	0.81	(4.1)	2.22	(11.1)	1.58	(7.9)	1.27	(6.4)	1.35	(6.7)
MESH127x127	1.04	(4.7)	2.90	(13.2)	2.36	(10.1)	1.45	(6.6)	1.58	(7.2)
BCSSTK15	0.52	(4.3)	1.96	(16.3)	1.13	(9.4)	0.52	(4.3)	0.60	(5.0)
BCSSTK16	0.21	(2.6)	0.52	(6.5)	0.48	(6.0)	0.19	(2.4)	0.25	(3.1)
BCSSTK17	0.76	(5.1)	1.50	(10.0)	1.13	(7.5)	0.61	(4.1)	0.71	(4.7)
BCSSTK18	1.02	(4.1)	3.70	(14.8)	2.84	(11.4)	1.25	(5.0)	1.57	(6.3)
BCSSTK23	0.25	(2.5)	1.23	(12.3)	0.58	(5.8)	0.29	(2.9)	0.37	(3.7)
BCSSTK24	0.09	(3.0)	0.20	(6.7)	0.16	(5.3)	0.07	(2.3)	0.11	(3.7)
BCSSTK25	1.63	(4.9)	5.81	(17.6)	3.71	(11.2)	1.69	(5.1)	1.90	(5.8)
BCSSTK29	1.65	(5.5)	9.09	(30.3)	3.29	(11.0)	1.35	(4.5)	1.66	(5.5)
BCSSTK30	2.26	(4.2)	4.73	(8.9)	4.30	(8.1)	1.69	(3.2)	1.85	(3.5)
BCSSTK31	3.35	(5.2)	6.61	(10.2)	5.60	(8.6)	2.96	(4.6)	3.44	(5.3)
BCSSTK32	2.86	(4.8)	4.95	(8.2)	4.30	(7.2)	2.28	(3.8)	2.90	(4.8)
BCSSTK33	0.77	(3.7)	2.56	(12.2)	2.61	(12.4)	0.78	(3.7)	0.91	(4.3)
BCSSTK35	0.91	(2.9)	1.57	(5.1)	1.70	(5.5)	0.81	(2.6)	0.95	(3.1)
BCSSTK36	0.56	(2.4)	0.95	(4.1)	1.17	(5.1)	0.48	(2.1)	0.61	(2.7)
BCSSTK37	1.03	(3.7)	1.90	(6.8)	1.71	(6.1)	0.87	(3.1)	0.97	(3.5)
BCSSTK38	0.85	(5.7)	1.63	(10.8)	1.00	(6.7)	0.48	(3.2)	0.61	(4.1)
MAT02HBF	0.90	(2.2)	1.37	(3.3)	1.73	(4.2)	0.87	(2.2)	0.97	(2.4)
MAT03HBF	1.15	(1.9)	2.35	(3.9)	2.70	(4.4)	1.46	(2.4)	1.66	(2.7)
STRUCT3	5.18	(4.9)	18.40	(17.5)	14.00	(13.3)	7.52	(7.2)	8.81	(8.4)
STRUCT4	1.37	(5.1)	3.58	(13.3)	5.43	(20.1)	1.06	(3.9)	1.33	(4.9)
PWT	0.96	(1.7)	6.30	(11.1)	6.30	(11.1)	3.87	(6.8)	4.03	(7.1)
BRACK2	7.14	(3.5)	23.95	(11.7)	17.32	(8.4)	12.48	(6.1)	14.11	(6.9)
CRACK	0.59	(3.3)	1.69	(9.4)	1.06	(5.9)	0.87	(4.8)	0.98	(5.4)
3DTUBE	3.73	(3.9)	7.57	(8.0)	6.70	(7.1)	2.81	(3.0)	3.40	(3.6)
CFD1	12.78	(4.3)	37.57	(12.5)	36.73	(12.2)	15.71	(5.2)	17.43	(5.8)
CFD2	22.34	(5.0)	65.29	(14.5)	64.11	(14.2)	27.61	(6.1)	31.64	(7.0)
CYL3	21.99	(1.3)	77.81	(4.6)	71.53	(4.2)	68.59	(4.0)	71.99	(4.2)
DIME20	13.70	(3.4)	38.00	(9.5)	37.50	(9.4)	32.78	(8.2)	33.07	(8.3)
GEARBOX	18.61	(6.5)	27.83	(9.6)	21.36	(7.4)	13.85	(4.8)	14.86	(5.2)
NASASRB	6.30	(6.3)	9.54	(9.5)	7.95	(7.9)	4.68	(4.7)	5.77	(5.8)
WAVE	21.32	(3.6)	72.51	(12.2)	67.81	(11.4)	35.79	(6.0)	38.59	(6.5)
PWTK	16.74	(7.4)	11.53	(5.1)	10.50	(4.7)	6.80	(3.0)	8.01	(3.6)
HERMES	61.78	(3.9)	181.79	(11.4)	220.68	(13.9)	104.81	(6.6)	110.91	(7.0)
MEAN		(4.0)		(10.6)		(8.7)		(4.4)		(5.1)

Tab. 4.5: Comparison of runtimes.

in column 3 present the medians over eleven runs. All other results have been obtained by a single execution of the ordering code.

The last column of table 4.4 shows that **multipord** achieves the highest reduction of operation count. While the other popular ordering codes reduce the average number of floating point operations by 25 % (METIS), 28 % (SCOTCH), and 31 % (SPOOLES) compared to MMD, **multipord** achieves an improvement of 42 %. Note that all orderings produced by **multipord** are better than the corresponding AMD orderings. Thus, in contrast to the other three ordering codes, **multipord** consistently outperforms AMD. However, it is also interesting to compare the results obtained by the nested dissection variant of **pord** (column 4 of table 4.2) with the results obtained by METIS and SCOTCH. Of the three state-of-the-art nested dissection codes, **pord** achieves the highest average improvement. Compared to AMD it reduces the operation count by 33 %. While there are only three matrices where **pord** does not produce as good an ordering as AMD, there are eight such cases for METIS and seven such cases for SCOTCH.

Table 4.5 provides us with the runtimes required by the five methods. The numbers in brackets give the runtimes relative to AMD. As can be seen, METIS takes modest amount of CPU times. Due to the more complicated coarsening and refinement process, the ordering times for **pord** can increase by an factor of two (i. e. BRACK2, HERMES) or more (i. e. CYL3) compared to METIS. Note that the ordering times for SCOTCH and SPOOLES differ only by a small fraction. In general, the ordering times for **pord** lie between the ordering times for METIS and SCOTCH/SPOOLES. We have observed this general tendency across the entire set of benchmark matrices. Columns 3 and 4 demonstrate that **multipord** requires only a marginally higher amount of CPU times than **pord**.

5 Conclusion

We have described two modifications to the basic multisection ordering heuristic that achieve a tighter coupling of bottom-up and top-down methods. In one of these modifications we are using bottom-up techniques such as quotient graphs and special node selection strategies for the construction of vertex separators. The idea is that vertex separators can be interpreted as the boundaries of the remaining elements in an unfinished bottom-up ordering. In the second modification we are using the vertex separators as a skeleton for the computation of a wide spectrum of bottom-up orderings. Here, the motivation is that nested dissection ignores the boundaries of elements which minimum degree naturally takes into account.

Both methods, bottom-up as well as top-down, have their own odds and ends. In multisection ordering schemes the advantages of one method are used to reduce the disadvantages of the other. Our intention was to push this development one step further.

Acknowledgement

We would like to thank Patrick Amestoy and Cleve Ashcraft for many helpful comments on an earlier draft of this paper. Cleve also provided many suggestions on the algorithmic components of our ordering heuristic.

References

[1] P.R. Amestoy, T.A. Davis, I.S. Duff, *An approximate minimum degree ordering algorithm*, SIAM J. Matrix Anal. Appl., Vol. 17, 886–905, 1996.

[2] C. Ashcraft, *Compressed graphs and the minimum degree algorithm*, SIAM J. Sci. Comput., Vol. 16, No. 6, 1404–1411, 1995.

[3] C. Ashcraft, *Sparse Direct Methods, Volume 1: Orderings for Matrices with Symmetric Structure*, Preprint, February, 2000.

[4] C. Ashcraft, R. Grimes, *SPOOLES: an object-oriented sparse matrix library*, 9th SIAM Conference on Parallel Processing for Scientific Computing, March 1999, San Antonio, Texas.

[5] C. Ashcraft, J.W.H. Liu, *A partition improvement algorithm for generalized nested dissection*, Techn. Rep. BCSTECH-94-020, Boeing Computer Services, Seattle, 1994.

[6] C. Ashcraft, J.W.H. Liu, *Generalized nested dissection: Some recent progress*, Mini Symposium 5th SIAM Conference on Applied Linear Algebra, Snowbird, Utah, 1994.

[7] C. Ashcraft, J.W.H. Liu, *Using domain decomposition to find graph bisectors*, BIT 37, 506–534, 1997.

[8] C. Ashcraft, J.W.H. Liu, *Applications of the Dulmage-Mendelsohn decomposition and network flow to graph bisection improvement*, SIAM J. Matrix Anal. Appl., Vol. 19, 325–354, 1998.

[9] C. Ashcraft, J.W.H. Liu, *Robust ordering of sparse matrices using multisection*, SIAM. J. Matrix Anal. Appl., Vol. 19, No. 3, 816–832, 1998.

[10] C. Ashcraft, J.W.H. Liu, S.C. Eisenstat, *Practical extensions of the multisection ordering for sparse matrices*, 6th SIAM Conference on Applied Linear Algebra, Snowbird, Utah, October 29, 1997.

[11] S.T. Barnard, H.D. Simon, *A fast multilevel implementation of recursive spectral bisection*, Proc. of 6th SIAM Conf. Parallel Processing for Scientific Computing, 711–718, 1993.

[12] Benoit, *Note sur une méthode de résolution des équations normales etc. (Procédé du commandant Cholesky)*, Bull. géodésique 3, 67–77, 1924.

[13] P. Berman, G. Schnitger, *On the performance of the minimum degree ordering for Gaussian elimination*, SIAM J. Matrix Anal. Appl., Vol. 11, No. 1, 83–88, 1990.

[14] M.V. Bhat, W.G. Habashi, J.W.H. Liu, V.N. Nguyen, M.F. Peeters, *A note on nested dissection for rectangular grids*, SIAM J. Matrix Anal. Appl., Vol. 14, No. 1, 253–258, 1993.

[15] T. Bui, C. Jones, *A heuristic for reducing fill-in in sparse matrix factorization*, Proc. 6th SIAM Conference on Parallel Processing for Scientific Computing, 445–452, 1993.

[16] A.C. Damhaug, *Sparse solution of finite element equations*, PhD Thesis, Department of Structual Engeneering, The Norwegian Institute of Technology, Trondheim, Norway, 1992.

[17] T. Davis, *University of Florida Sparse Matrix Collection*, http://www.cise.ufl.edu/~davis/sparse/, ftp://ftp.cise.ufl.edu/pub/faculty/davis/matrices, NA Digest, Vol. 92, No. 42, October 16, 1994, NA Digest, Vol. 96, No. 28, July 23, 1996, and NA Digest, Vol. 97, No. 23, June 7, 1997.

[18] I.S. Duff, A.M. Erisman, J.K. Reid, *Direct Methods for Sparse Matrices*, Oxford University Press, Oxford, 1987.

[19] I.S. Duff, R.G. Grimes, J.G. Lewis, *Users' guide for the Harwell-Boeing sparse matrix collection*, Technical Report TR/PA/92/86, Res. and Techn. Division, Boeing Computer Services, Seattle, 1992.

[20] I.S. Duff, J.K. Reid, *The multifrontal solution of indefinite sparse symmetric linear equations*, ACM Trans. Math. Software, 3, 302–325, 1983.

[21] S.C. Eisenstat, M.H. Schultz, A.H. Sherman, *Applications of an element model for Gaussian elimination*, in *Sparse Matrix Computations*, J. Bunch, D. Rose (Eds), Academic Press, New York, 85–96, 1976.

[22] C.M. Fiduccia, R.M. Mattheyses, *A linear-time heuristic for improving network partitions*, 19th IEEE Design Automation Conference, 175–181, 1982.

[23] A. George, *Nested dissection of a regular finite element mesh*, SIAM J. Numer. Anal., Vol. 10, No. 2, 345–363, 1973.

[24] A. George, J.W.H. Liu, *An automatic nested dissection algorithm for irregular finite element problems*, SIAM J. Numer. Anal., Vol. 15, No. 5, 1053–1069, 1978.

[25] A. George, J.W.H. Liu, *A fast implementation of the minimum degree algorithm using quotient graphs*, ACM Trans. Math. Software, Vol. 6, 337–358, 1980.

[26] A. George, J.W.H. Liu, *Computer Solution of Large Sparse Positive Definite Systems*, Prentice-Hall, Englewood Cliffs, NJ, 1981.

[27] A. George, J.W.H. Liu, *The evolution of the minimum degree ordering algorithm*, SIAM Review, Vol. 31, No. 1, 1–19, 1989.

[28] A. George, J.W. Poole, R. Voigt, *Incomplete nested dissection for solving n by n grid problems*, SIAM J. Numer. Anal., Vol. 15, 663–673, 1978.

[29] J.R. Gilbert, C. Moler, R. Schreiber, *Sparse matrices in MATLAB: design and implementation*, SIAM J. Matrix Anal. Appl., Vol. 13, 333–356, 1992.

[30] A. Gupta, *WGPP: Watson graph partitioning (and sparse matrix ordering) package, users manual*, IBM T.J. Watson Research Center, Research Report RC 20453, New York, 1996.

[31] A. Gupta, *Fast and effective algorithms for graph partitioning and sparse matrix ordering*, IBM T.J. Watson Research Center, Research Report RC 20496, New York, 1996.

[32] B. Hendrickson, R. Leland, *The CHACO user's guide*, Techn. Rep. SAND94-2692, Sandia Nat. Lab., 1994.

[33] B. Hendrickson, R. Leland, *A multilevel algorithm for partitioning graphs*, Proc. of Supercomputing'95, 1995.

[34] B. Hendrickson, E. Rothberg, *Effective sparse matrix ordering: just around the BEND*, Proc. of 8th SIAM Conf. Parallel Processing for Scientific Computing, 1997.

[35] B. Hendrickson, E. Rothberg, *Improving the runtime and quality of nested dissection ordering*, SIAM J. Sci. Comput., Vol. 20, No. 2, 468–489, 1998.

[36] A.J. Hoffman, M.S. Martin, D.J. Rose, *Complexity bounds for regular finite difference and finite element grids*, SIAM J. Numer. Anal., Vol. 10, No. 2, 364–369, 1973.

[37] G. Karypis, V. Kumar, *A fast and high quality multilevel scheme for partitioning irregular graphs*, SIAM J. Sci. Comput., Vol. 20, No. 1, 1999.

[38] G. Karypis, V. Kumar, *METIS: a software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices (Version 4.0)*, Techn. Rep., Dept. of Computer Science, Univ. of Minnesota, 1998.

[39] B.W. Kernighan, S. Lin, *An effective heuristic procedure for partitioning graphs*, The Bell Systems Technical Journal, 291–308, 1970.

[40] D. König, *Über Graphen und ihre Anwendung auf Determinantentheorie und Mengenlehre*, Math. Ann., 77, 453–465, 1916.

[41] C.E. Leiserson, J.G. Lewis, *Ordering for parallel sparse symmetric factorization*, in *Parallel Processing for Scientific Computing*, SIAM, Philadelphia, 27–31, 1989.

[42] J.W.H. Liu, *Modification of the minimum-degree algorithm by multiple elimination*, ACM Trans. Math. Software, Vol. 11, No. 2, 141–153, 1985.

[43] J.W.H. Liu, *The minimum degree ordering with constraints*, SIAM J. Sci. Stat. Comput., Vol. 10, No. 6, 1136–1145, 1989.

[44] J.W.H. Liu, *A graph partitioning algorithm by node separators*, ACM Trans. Math. Software, Vol. 15, No. 3, 198–219, 1989.

[45] J.W.H. Liu, *The role of elimination trees in sparse factorization*, SIAM J. Matrix Anal. Appl., Vol. 11, No. 1, 134–172, 1990.

[46] H.M. Markowitz, *The elimination form of the inverse and its application to linear programming*, Management Science, Vol. 3, 255–269, 1957.

[47] C. Meszaros, *The inexact minimum local fill-in ordering algorithm*, Techn. Report WP 95 7, Computer and Automation Research Institute, Hungarian Academy of Sciences, Budapest, 1995.

[48] E. Ng, P. Raghavan, *Performance of greedy heuristics for sparse Cholesky factorization*, SIAM J. Matrix Anal. Appl., Vol. 20, 902–914, 1998.

[49] S.V. Parter, *The use of linear graphs in Gauss elimination*, SIAM Review, Vol. 3, 119–130, 1961.

[50] F. Pellegrini, J. Roman, *Sparse matrix ordering with SCOTCH*, Proc. HPCN'97, LNCS 1225, 370–378, 1997.

[51] F. Pellegrini, J. Roman, P. Amestoy, *Hybridizing nested dissection and halo approximate minimum degree for efficient sparse matrix ordering*, Proc. Irregular'99, LNCS 1586, 986–995, 1999.

[52] R. Preis, R. Diekmann, *The PARTY partitioning library user guide – version 1.1*, Techn. Rep., Dept. of Computer Science, Univ. of Paderborn, 1996.

[53] D.J. Rose, *A graph-theoretic study of the numerical solution of sparse positive definite systems of linear equations*, in *Graph-Teory and Computing*, R. Read (Ed.), Academic Press, New York, 183–217, 1972.

[54] E. Rothberg, *Robust ordering of sparse matrices: a minimum degree, nested dissection hybrid*, Silicon Graphics manuscript, 1995.

[55] E. Rothberg, *Exploring the tradeoff between imbalance and separator size in nested dissection ordering*, Silicon Graphics manuscript, 1996.

[56] E. Rothberg, S.C. Eisenstat, *Node selection strategies for bottom-up sparse matrix ordering*, SIAM J. Matrix Anal. Appl., Vol. 19, No. 3, 682–695, 1998.

[57] B. Speelpenning, *The generalized element model*, Techn. Rep. UIUCDCS-R-78-946, Dept. of Computer Science, Univ. of Illinois, 1978.

[58] W.F. Tinney, J.W. Walker, *Direct solutions of sparse network equations by optimally ordered triangular factorization*, Proc. of the IEEE, Vol. 55, 1801–1809, 1967.

[59] M. Yannakakis, *Computing the minimum fill-in is NP-complete*, SIAM J. Alg. Disc. Meth., Vol. 2, No. 1, 77–79, 1981.