COLORING AND DEGENERACY FOR DETERMINING VERY LARGE AND SPARSE DERIVATIVE MATRICES

ASHRAFUL HUQ SUNY
Bachelor of Science, Chittagong University of Engineering & Technology, 2013

A Thesis
Submitted to the School of Graduate Studies
of the University of Lethbridge
in Partial Fulfillment of the
Requirements for the Degree

MASTER OF SCIENCE

Department of Mathematics and Computer Science
University of Lethbridge
LETHBRIDGE, ALBERTA, CANADA

COLORING AND DEGENERACY FOR DETERMINING VERY LARGE AND SPARSE DERIVATIVE MATRICES

ASHRAFUL HUQ SUNY

Date of Defence: December 19, 2016

Dr. Shahadat Hossain Supervisor	Professor	Ph.D.
Dr. Daya Gaur Committee Member	Professor	Ph.D.
Dr. Robert Benkoczi Committee Member	Associate Professor	Ph.D.
Dr. Howard Cheng Chair, Thesis Examination Committee	Associate Professor	Ph.D.

Dedication

To

My Parents.

Abstract

Estimation of large sparse Jacobian matrix is a prerequisite for many scientific and engineering problems. It is known that determining the nonzero entries of a sparse matrix can be modeled as a graph coloring problem. To find out the optimal partitioning, we have proposed a new algorithm that combines existing exact and heuristic algorithms. We have introduced degeneracy and maximum k-core for sparse matrices to solve the problem in stages. Our combined approach produce better results in terms of partitioning than DSJM and for some test instances, we report optimal partitioning for the first time.

Acknowledgments

I would like to express my deep acknowledgment and profound sense of gratitude to my supervisor Dr. Shahadat Hossain, for his continuous guidance, support, cooperation, and persistent encouragement throughout the journey of my MSc program.

I wish to express my thanks to my supervisory committee members, Dr. Daya Gaur and Dr. Robert Benkoczi for their encouragement and suggestions.

I am grateful to my parents, friends and fellow graduate students for their inspiration and support.

Contents

List of Tables

List of Figures

List of Symbols

The next list describes several symbols that will be later used within the body of this thesis.

A, B, C, W	An uppercase letter is used to denote a matrix.
$A^\top$	Transpose of a matrix.
$i \perp j$	Column i and j are structurally orthogonal.
$i \not\perp j$	Column i and j are structurally dependent.
$a_{ij}, A(i,j)$	The (i,j) entry of A matrix.
$G(A)$	Column intersection graph of matrix A.
nnz	Number of nonzero elements in a matrix.
ρ_i	Number of nonzeroes in row i of a matrix.
ρ_{max}	Maximum number of nonzeroes in any row.
ρ_{min}	Minimum number of nonzeroes in any row.
$\overline{\rho}$	Arithmetic mean of number of nonzeroes in each row.

Chapter 1

Introduction

In many scientific and engineering problems, repeated evaluation of Jacobian matrices i.e. first order partial derivatives is a common subproblem. For large-scale problems, the Jacobian matrix is often sparse i.e. most of the matrix entries are zero. To obtain the Jacobian matrix, exploitation of prior known sparsity is often needed to avoid the computation involving known zero entries. Exploiting sparsity in computing the Jacobian matrix can be viewed as a partitioning problem [9]. With the known sparsity structure of the given sparse matrix $A \in \Re^{m \times n}$, we can partition the columns (rows) of A into p (q) groups such that each column (row) belongs to exactly one group and the columns (rows) in the same group are *structurally orthogonal* i.e. no two columns (rows) have nonzero entries in the same row (column) position. This type of partitioning is known as unidirectional partitioning. Then, the nonzeroes in each column group can be determined from finite difference approximation or forward mode of automatic differentiation and the nonzeroes in the row groups can be determined by the reverse mode of automatic differentiation [17]. Unidirectional partitioning may not be able to exploit the sparsity effectively if the matrix has both dense rows and dense columns. In this case bi-directional partitioning, i.e. row partitioning and column partitioning together is preferable [13]. In this thesis, we are considering unidirectional partitioning only. Both of the above partitioning problems can be formulated as graph coloring problems [7, 15, 22].

1.1 Motivation

The graph coloring problem deals with the assignment of minimum number of positive integers (colors) to the vertices of a graph such that colors on adjacent vertices are different. The graph coloring problem has applications in scheduling [25], timetabling [10], register allocation [5], train platforming [4], frequency assignment [11] and communication networks [34]. The graph coloring problem is known to be NP-hard [12] and has received much attention in the literature, not only for its real world applications but also for its computational difficulty. Exact algorithms [3, 30, 19] proposed for graph coloring are able to solve a problem when the problem instance is small. On the other hand, real-world applications usually deal with graphs of thousands and in recent years millions of vertices. Standard benchmark instances are solved using different heuristics [25, 8]. In this thesis, we are interested in the performance of available heuristics. Since we don't know the optimal coloring, it is really difficult to establish the effectiveness of the coloring algorithms. The following approaches can be used to find an optimal coloring:

1. Use a Set Covering (SC) formulation [30]. In the SC formulation, corresponding to each independent set, there is a decision variable. The objective is to minimize the number of independent sets that satisfies the coloring constraints by ensuring that each vertex belongs to at least one independent set. The decision variables are constrained to be $0-1$ variables. Using a column generation technique the method solves a relaxed problem for a subset of variables (columns). The "generation of new column" can be facilitated by solving a weighted independent set problem which is also an NP-hard problem. Gaur et al. [14] proposed a star bi-coloring column generation formulation for bi-directional determination of sparse Jacobian matrices.

2. Solve the problem in stages using a combined approach i.e. exact and heuristic algorithms together. The central idea in our combined approach is to identify a suitable submatrix of the given Jacobian sparsity pattern, find a structurally orthogonal partitioning of the columns of the submatrix and then extend the submatrix partitioning to

the given Jacobian sparsity pattern. All of these steps can also be described in terms of the recently proposed unified model called pattern graph [24]. In this thesis, we have used the combined approach.

1.2 Our Contribution

Specific contributions are given below.

1. We present a combined exact and heuristic algorithm for the determination of large sparse Jacobian matrices.

2. We have settled the coloring complexity of a number of benchmark problems from the literature. For those problems until now the optimal coloring was not verified.

3. Our combined approach is especially suitable for very large-scale problems. To the best of our knowledge, this is the first time that a combined approach has been proposed to very large and sparse Jacobian determination problem.

4. We give matrix interpretation of degeneracy and maximum k-core in relation to the structural orthogonal partitioning of columns.

1.3 Thesis organization

Including this chapter, there are four more chapters in this thesis organized as follows:

In Chapter 2, we introduce sparse Jacobian matrix followed by the description of Newton's method to solve systems of nonlinear equations. We then describe Finite Differencing (FD) and Automatic Differentiation (AD) along with forward mode and reverse mode. We also describe matrix partitioning techniques in this chapter. Finally, we provide basic graph concepts including graph coloring methods.

In Chapter 3, we present our combined approach for finding structurally orthogonal partitioning of columns. We describe the central ideas in our algorithm by using the duality

between matrices and graphs. Column intersection graph, smallest last ordering, degeneracy and maximum k-core are also covered in this chapter.

In Chapter 4, we describe our test data sets along with the data structures. Then we provide experimental results that demonstrate the efficacy of our algorithms.

Finally, we provide concluding remarks and directions for future research in this area in Chapter 5.

Chapter 2

Notations and Mathematical Preliminaries

In this chapter, we introduce basic notations, the problem of determination of sparse Jacobian matrices and review mathematical preliminaries necessary for this thesis. We also discuss finite differencing and algorithmic differentiation techniques in this chapter. Finally, we introduce basic graph concepts including vertex coloring.

2.1 Sparse Matrix

A matrix is called sparse if it is computationally advantageous to utilize the knowledge that many of its entries are zero. On the other hand, if most of the entries are nonzero then the matrix is called dense. In the Figure 2.1, the sparse matrix contains only 9 nonzero entries with 16 zero entries and the Figure 2.2 shows the sparsity structure of matrix *ibm32*.

$$\begin{bmatrix} 4 & 5 & 0 & 0 & 0 \\ 0 & 5 & 6 & 0 & 0 \\ 0 & 0 & 7 & 9 & 0 \\ 0 & 0 & 0 & 6 & 5 \\ 0 & 0 & 0 & 0 & 5 \end{bmatrix}$$

Figure 2.1: A sparse matrix

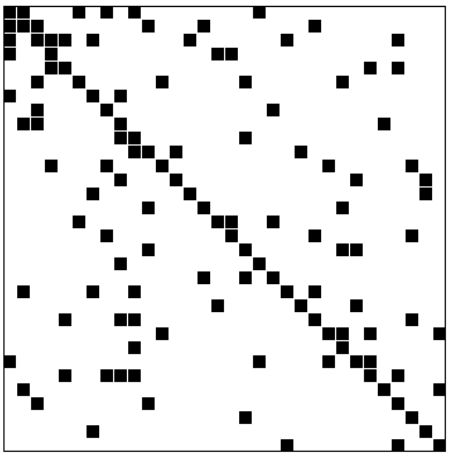

Figure 2.2: Sparsity structure. Matrix name: *ibm32*, Dimensions: 32×32, 126 nonzero elements are shown in black. Source: [1]

2.2 Jacobian Matrix

The matrix of all first-order partial derivatives of a vector-valued function is known as the Jacobian matrix. Let $F = (f_1, f_2, ..., f_m)^T$ be a mapping $F : \Re^n \to \Re^m$. If F is continuously differentiable then the Jacobian matrix J of F at a given vector x is given by

$$J(x) = F'(x) = \begin{pmatrix} \frac{\partial}{\partial x_1} f_1(x) & \cdots & \frac{\partial}{\partial x_n} f_1(x) \\ \vdots & \ddots & \vdots \\ \frac{\partial}{\partial x_1} f_m(x) & \cdots & \frac{\partial}{\partial x_n} f_m(x) \end{pmatrix} \tag{2.1}$$

Derivative information is often needed, for example in order to find the numerical solution of a system of nonlinear equations or to minimize a non-linear function of a large number of variables.

2.3 Newton's Method

Newton's methods are one of the classical methods to solve the systems of nonlinear equations. Newton's method iteratively finds the root of a real-valued function specified by $F(x) = 0$, where $F = (f_1, f_2, ..., f_m)^T$ is a mapping $F : \Re^n \to \Re^m$. The steps for solving a nonlinear system are as follows:

Algorithm: Newton's Method for systems of nonlinear equations

Input: Given an initial approximation $x \in \Re^n$

for $j \leftarrow 0$ *to convergence* **do**
- Evaluate $b = F(x)$;
- Determine $J = F'(x)$;
- Solve $Js = -b$ for s;
- Update $x \leftarrow x + s$;

The following example demonstrates the algorithm of Newton's method. Given a vector function

$$F(x) = \begin{bmatrix} x_1 + x_2 - 3 \\ x_1^2 + x_2^2 - 5 \end{bmatrix} \tag{2.2}$$

We consider solving $F(x) = 0$. The roots are $\begin{bmatrix} 1 & 2 \end{bmatrix}^T$ and $\begin{bmatrix} 2 & 1 \end{bmatrix}^T$ which can be verified directly. The Jacobian matrix is as follows

$$J(x) = \begin{bmatrix} 1 & 1 \\ 2x_1 & 2x_2 \end{bmatrix}$$

Let $x = \begin{bmatrix} 0 & 3 \end{bmatrix}^T$ be an initial approximation. For the first iteration of Newton's method we have,

$$F(y) \equiv b = \begin{bmatrix} 0 \\ 4 \end{bmatrix} \text{ and } J(x) \equiv J = \begin{bmatrix} 1 & 1 \\ 0 & 6 \end{bmatrix}$$

The value of s is then found by solving

$$Js = -b$$

$$\implies \begin{bmatrix} 1 & 1 \\ 0 & 6 \end{bmatrix} s = - \begin{bmatrix} 0 \\ 4 \end{bmatrix}$$

$$\implies s = \begin{bmatrix} 0.667 \\ -0.667 \end{bmatrix}$$

The updated value of x is $\begin{bmatrix} 0.667 & 2.333 \end{bmatrix}^T$. For the second iteration

$$b = \begin{bmatrix} 0 \\ .889 \end{bmatrix} \text{ and } J = \begin{bmatrix} 1 & 1 \\ 1.333 & 4.667 \end{bmatrix}$$

Hence

$$s = \begin{bmatrix} 0.267 \\ -0.267 \end{bmatrix}$$

Therefore new value of x will be $\begin{bmatrix} 0.934 & 2.066 \end{bmatrix}^T \cong \begin{bmatrix} 1 & 2 \end{bmatrix}^T$, which is closer to the original root of the function. From the algorithm, we can see that in each iteration we need to evaluate first order derivative matrix. So the computation of first order derivative i.e. the Jacobian matrix is an important step in finding the solutions of systems of nonlinear equations.

2.4 Finite Difference Approximation

The Jacobian matrix can be obtained by approximating it using a finite difference (FD) formula. Let A denote the Jacobian matrix $J(x)$ of a continuously differentiable mapping $F : \Re^n \to \Re^m$. An approximation to the j^{th} column of A, denoted by a_j, can be obtained from

$$a_j = \frac{\partial}{\partial x_i} F(x) \approx \frac{F(x+\varepsilon e_j) - F(x)}{\varepsilon}, 1 \leq j \leq n \tag{2.3}$$

where e_j is the j^{th} unit coordinate vector and ε is a positive increment. If $F(x)$ has already been evaluated, then we can estimate the partial derivatives in the j^{th} column of matrix A through the additional function evaluation $F(x+\varepsilon e_j)$. If the sparsity information is not known then we will need n extra function evaluations to determine A. Consider a vector function

$$F(x) = \begin{bmatrix} x_1 + x_2 - 5 \\ x_1^2 - x_2^2 - 5 \end{bmatrix} \tag{2.4}$$

The Jacobian matrix is

$$J(x) = F'(x) = \begin{bmatrix} 1 & 1 \\ 2x_1 & -2x_2 \end{bmatrix}$$

At some point $x = \begin{bmatrix} 1 & 2 \end{bmatrix}^T$, the value of $F(x)$ will be

$$F = \begin{bmatrix} -2 \\ -8 \end{bmatrix}$$

and the value of $J(x)$ will be

$$J = \begin{bmatrix} 1 & 1 \\ 2 & -4 \end{bmatrix}$$

The two unit coordinate vectors needed for this calculation are $e_1 = \begin{bmatrix} 1 & 0 \end{bmatrix}^T$ and $e_2 = \begin{bmatrix} 0 & 1 \end{bmatrix}^T$. Considering $\varepsilon = 0.1$, we get the first order derivative of $F(x)$ as $\begin{bmatrix} 1 & 1 \\ 2.1 & -4.1 \end{bmatrix}$.

The main advantage of using finite difference approximation is that it is easy to implement, but if ε is too large, then approximation may not be accurate due to truncation error. Also if ε is too small, then $F(x+\varepsilon e_j) - F(x)$ may cause loss of precision to round-off errors associated with finite precision calculations [31].

2.5 Automatic Differentiation

Automatic differentiation (AD), also called algorithmic differentiation is a chain rule based technique used for evaluating the derivatives of functions defined by computer programs [17]. Automatic differentiation uses exact formulas along with floating point values. Unlike finite difference approximation, the derivatives computed by using automatic differentiation are free from truncation errors. Let f be a function of the vector $y \in \Re^m$, which in turn a function of the vector $x \in \Re^n$. Using chain rule the derivative of f with respect to x

is then

$$\nabla_x f(y(x)) = \sum_{i=1}^{m} \frac{\partial f}{\partial y_i} \nabla y_i(x) \tag{2.5}$$

where ∇ indicates the gradient. Automatic differentiation is evaluated by performing a sequence of operations involving just one or two arguments at a time. Consider the following function of two variables

$$f(x_1, x_2) = x_1^2 \sin(x_2) + e^{x_1} \tag{2.6}$$

The steps involved in a computation of f is given as a sequence of arithmetic operations

$$x_3 = x_1 * x_1$$

$$x_4 = sin(x_2)$$

$$x_5 = v_3 * v_4$$

$$x_6 = e^{x_1}$$

$$x_7 = x_5 + x_6$$

Here, $(x_3, x_4, ..., x_6)$ are intermediate variables and (x_1, x_2) are independent variables. If we have the values of x_1 and x_2, then the result of computation is obtained in $f(x_1, x_2) = x_7$. It is possible to form different sequence of operations for the same function f. After forming a sequence, we can apply rules of differentiation to compute derivative of a function with respect to the independent variables x_1 and x_2. Automatic differentiation has two basic modes of operation known as *forward* mode and *reverse* mode.

2.5.1 Forward Mode

In the forward mode, also called forward accumulation, intermediate partial derivatives are obtained in the same order as the function values are determined. The computation of matrix-vector product Jv is done in the forward pass, where v is a n-vector and J is a Jacobian matrix. By initializing v to be unit coordinate vector e_i, where $i = 1, 2.., n$ all the

columns of J can be determined by n forward passes.

2.5.2 Reverse Mode

In the reverse mode, also called reverse accumulation, the intermediate partial derivatives are obtained in reverse order of function evaluation. The computation of Jw^T is done in the reverse pass, where w is a m-vector. By initializing w to be unit coordinate vector e_i, where $i = 1, 2.., m$ all the columns of J can be determined by m reverse passes.

Forward mode is more efficient than the reverse mode for function $f : \Re^n \rightarrow \Re^m$ with $m \gg n$ as only n passes are required, whereas reverse mode is more efficient than forward mode when $m \ll n$ as only m passes are necessary.

2.6 Matrix Partitioning

Let $A \in \Re^{m \times n}$ be a matrix whose sparsity pattern is given. The problem we address in this thesis is to determine sparse Jacobian matrices efficiently by exploiting known sparsity information. Matrix partitioning is to obtain vectors $s_1, s_2, ..., s_p$ so that the nonzero elements of the given matrix A are uniquely determined from the products $As_1, As_2, ..., As_p$ with p as small as possible. Curtis, Powell, and Reid [9] noted that sparsity of the Jacobian matrices can be exploited by partitioning the columns of the matrix into groups in such a way that columns in each group are structurally orthogonal. Then, the nonzero elements in each group can be determined through the forward mode of automatic differentiation or finite differencing. The concept of structurally orthogonal partitioning is illustrated using examples in the remainder of this chapter.

2.6.1 Unidirectional Partitioning

In unidirectional partitioning scheme, either the columns or the rows are partitioned into structurally orthogonal groups. Let us illustrates unidirectional partitioning with examples. Consider a sparse matrix

$$A = \begin{bmatrix} a_{11} & 0 & 0 & 0 \\ a_{21} & a_{22} & 0 & 0 \\ 0 & 0 & a_{33} & 0 \\ 0 & 0 & 0 & a_{44} \end{bmatrix}$$

and a seed matrix

$$S = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Each column of S corresponds to a group of structurally orthogonal columns. Then the product of sparse matrix and seed matrix will be

$$AS = \begin{bmatrix} a_{11} & 0 \\ a_{21} & a_{22} \\ a_{33} & 0 \\ 0 & a_{44} \end{bmatrix}$$

From the product AS, we can determine all the nonzero entries of A. Hence matrix A can be partitioned into two column groups. This is known as column partitioning.

In row partitioning, A sparse matrix is partitioned into row groups. Consider a sparse matrix

$$A = \begin{bmatrix} a_{11} & a_{12} & 0 & 0 \\ 0 & a_{22} & 0 & 0 \\ 0 & 0 & a_{33} & 0 \\ 0 & 0 & 0 & a_{44} \end{bmatrix}$$

and a seed matrix

$$W^T = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix}$$

The product AW from which the nonzeroes of A can be recovered is

$$AW = \begin{bmatrix} a_{11} & a_{21} \\ 0 & a_{22} \\ a_{33} & 0 \\ 0 & a_{44} \end{bmatrix}$$

2.6.2 Bidirectional Partitioning

If seed matrices $S \in \Re^{n \times p}$ and $W \in \Re^{m \times q}$ can be obtained such that all the nonzero elements of a given sparse matrix $A \in \Re^{m \times n}$ can be determined uniquely from the products $B = AS$ and $C^T = AW^T$, then the resulting partitioning is called as bidirectional partitioning. The following example will demonstrate bidirectional partitioning method. Let

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & 0 & 0 \\ a_{31} & 0 & a_{33} & 0 \\ a_{41} & 0 & 0 & a_{44} \end{bmatrix}, S = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \end{bmatrix} \text{ and } W^T = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix}$$

Now from the product AS we get

$$B = AS = \begin{bmatrix} a_{11} & a_{12} + a_{13} + a_{14} \\ a_{21} & a_{22} \\ a_{31} & a_{33} \\ a_{41} & a_{44} \end{bmatrix}$$

and from the product AW^T we get

$$C^T = AW^T = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \end{bmatrix}$$

Hence from matrices B and C^T, all the nonzero elements of sparse matrix A are recovered.

2.6.3 Direct Determination

In direct determination method, all the nonzero elements of a sparse matrix A can be read-off from the matrices $B = AS$ and $C^T = AW^T$ without any further arithmetic operation. Let us interpret the direct determination method with an example. Consider

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{21} & a_{22} & 0 & 0 & 0 \\ a_{31} & 0 & a_{33} & 0 & 0 \\ a_{41} & 0 & 0 & a_{44} & 0 \\ a_{51} & 0 & 0 & 0 & a_{55} \end{bmatrix}, S = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \end{bmatrix} \text{ and } W^T = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

We can obtain the matrix B and matrix C^T from the products AS and W^TA respectively.

$$B = AS = \begin{bmatrix} a_{11} & a_{12}+a_{13}+a_{14}+a_{15} \\ a_{21} & a_{22} \\ a_{31} & a_{33} \\ a_{41} & a_{44} \\ a_{51} & a_{55} \end{bmatrix} \text{ and } C^T = AW^T = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \end{bmatrix}$$

The nonzero elements of A can thus be read off from B and C^T without any further arithmetic operation.

2.6.4 Substitution Determination

In a substitution method, the nonzero elements of the matrix A are calculated by solving a triangular system of equations i.e. the ordering of the nonzero elements of A is such that every nonzero is determined using previously computed values. Let us demonstrate the substitution method with the help of an example from [21]. Consider

$$A = \begin{bmatrix} a_{11} & 0 & a_{13} \\ a_{21} & a_{22} & 0 \\ 0 & a_{32} & a_{33} \end{bmatrix} \text{ and } S = \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 0 & 1 \end{bmatrix}$$

Using the following reduced system the second row of A can be determined by solving for a_{21} and a_{22}

$$\begin{bmatrix} a_{21} & a_{22} & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} b_{21} & b_{22} \end{bmatrix}$$

Eliminating row 3 of S and transposing the system, we get

$$\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} a_{21} \\ a_{22} \end{bmatrix} = \begin{bmatrix} b_{21} \\ b_{22} \end{bmatrix}$$

which is an upper triangular system. Using the similar way, the nonzeroes of the other two rows of A can be obtained. For the above example, direct determination method will require three function evaluations, whereas substitution method will need two function evaluations.

2.7 Graph Concepts

A *graph* G is an ordered pair (V,E) where V is a finite and nonempty set called vertices or nodes and E is a set of unordered pairs of distinct vertices called edges. Two vertices u and v are *adjacent vertices* if and only if $\{u,v\} \in E$. The *degree* of a vertex u denoted by $deg(u)$, is the number of vertices adjacent to u. The smallest degree among the vertices of G is called the *minimum degree* of G, and is denoted by $\delta(G)$.

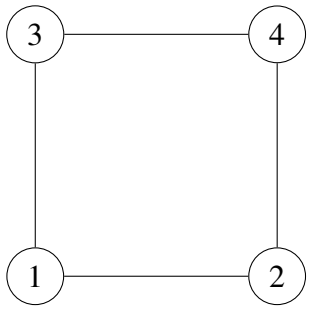

Figure 2.3: An undirected graph where the vertex set $\{1,2,3,4\}$ and edge set $\{\{1,2\},\{1,3\},\{2,4\},\{3,4\}\}$. Vertices are shown in circles and edges are represented as lines

2.8 Clique

A *complete graph* is a simple undirected graph in which every pair of distinct vertices is connected by a unique edge. The Figure (2.4) is an example of a complete graph. A *subgraph* $G' = (V', E')$ of a graph $G = (V, E)$ is a graph whose set of vertices and set of edges are all subsets of G i.e $V' \subseteq V$ and $E' \subseteq E$. A clique of graph G is a complete subgraph of G. Clique size is the number of vertices that form the clique. The clique with the maximum number of vertices is defined as maximum clique. The size of a maximum clique in G is known as the *clique number* of G, and denoted by $\omega(G)$. The Figure 2.4 shows a graph of clique of size four.

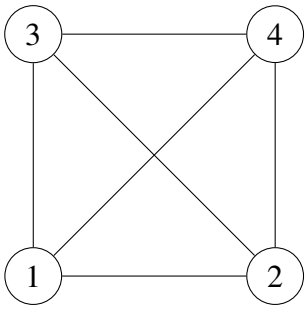

Figure 2.4: A clique of size 4

2.9 Graph Coloring

Graph coloring is a classical combinatorial optimization problem. Graph coloring refers to an assignment of a color to each vertex in such a way that no two adjacent vertices have the same color. A p-coloring of a graph $G = (V, E)$ is a function $\Phi : V \mapsto \{1, 2, .., p\}$ such that $\Phi(u) \neq \Phi(v)$, if $\{u, v\} \in E$. The *chromatic number* $\chi(G)$ is the smallest p for which G has a p-coloring. A coloring that uses $\chi(G)$ colors is known as optimal coloring. The Figure 2.4 illustrates p-coloring of a graph G using $p = 4$.

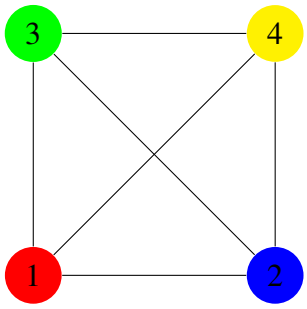

Figure 2.5: A 4-coloring graph

2.10 Graph Coloring Methods

The graph coloring problem is a well known NP-complete problem [12]. In this thesis, we have applied heuristic techniques to solve the partitioning problem because they are solvable in polynomial time and give good solutions but we want to know how good are the heuristics doing and this has motivated us to investigate exact coloring techniques. We will give a short description of both the techniques below.

2.10.1 Heuristic Methods

A heuristic is a technique to produce a solution of a problem in a reasonable time frame that is good enough for solving the problem at hand. A Heuristic does not guarantee that the solution is optimal. The heuristic method that we have used in this thesis is based on a greedy constructive algorithm [28]. The simplest greedy constructive algorithm is the greedy sequential algorithm (SEQ), where each vertex v_i $(i = 1,2,..,n)$ is assigned the lowest indexed color class which contains no vertices adjacent to v_i. Usually, the greedy algorithm is very fast, but the results produced by greedy can be very sensitive to some input parameter, like the input ordering of the vertices. It has been shown that pre-ordered sequence of vertices to SEQ algorithm can obtain better coloring [7].

2.10.2 Exact Methods

Exact methods are those methods that give an optimal solution for the given problem. Exact methods give upper and lower bounds of the problem and confirm that no better

solution could be found. Exact methods are “hard” and often not solvable in polynomial time. In graph coloring, exact coloring refers to coloring the graph such that the number of colors needed to color the graph is minimum. In our thesis, we have used Randall-Brown’s modified algorithm [3] as an exact method.

Chapter 3

Partitioning via Degeneracy

In this chapter, we describe our new algorithm that combines exact and heuristics approaches for finding structurally orthogonal partitioning of columns. First, we describe the central ideas in our algorithm informally in matrix notation. Following this, we introduce the notion of column intersection graph. Using the equivalence of structurally orthogonal column partitioning and vertex coloring of the column intersection graph [23], the new algorithm is introduced. The notion of "graph degeneracy" [26] is central to our algorithm. The Smallest Last Vertex Ordering described by Matula et al. [29] is used to compute the degeneracy.

The notion of "well-connectedness" in a graph or network is of fundamental interest in diverse areas, for example social, technological, biological. The pattern of nontrivial interactions expressed by the edges in a complex network is of practical interest when analyzing, for example, a large network of genomic quantities in a cell [18]. Structural measures such as cliques, degeneracy, ranking of nodes, etc have been developed and interpreted, apparently independently, in natural and social sciences to characterize the pattern of interactions. In the field of sociometry, the notion of "maximally cohesive subnetwork" which is precisely the notion of a clique is an important structural property of a network. A formal definition of a clique is given by Luce et al. [27]. Interestingly, the paper uses linear algebraic operations to compute shortest paths in a graph. Several recent works in complex networks use scale reduction methodology to maximum clique and coloring problem [32, 33]. In this thesis, we use degeneracy of a graph to handle very large instances of

partitioning problems.

3.1 Column Partitioning and Coloring

3.1.1 Column Partitioning

Given a sparse matrix $A \in \Re^{m \times n}$.

Definition 3.1. Columns i and j of matrix A are called structurally orthogonal if there is no index l such that

$$a_{li} \neq 0 \quad and \quad a_{lj} \neq 0$$

We use the notation $i \perp j$ when column i is structurally orthogonal to column j. For example, consider the following sparse matrix.

$$\begin{bmatrix} x & 0 & x & 0 \\ 0 & x & 0 & x \\ x & 0 & x & 0 \\ 0 & x & 0 & x \end{bmatrix}$$

Here column 1 and column 2 are structurally orthogonal but column 1 and column 3 are not structurally orthogonal. A structurally orthogonal k-partitioning of columns of a sparse matrix A is the grouping of columns into k groups such that columns in the same group are mutually structurally orthogonal; k is the size of the partition. A structurally orthogonal k-partitioning where k is minimized is called a minimum or optimum structurally orthogonal partitioning.

Definition 3.2. Columns i and j are structurally dependent if they are not structurally orthogonal. In this case we use the notation $i \not\perp j$.

Definition 3.3. The degree of structural dependency of a column j denoted by $d(j)$, is the number of columns $i \neq j$ such that $j \not\perp i$.

Definition 3.4. A set of columns S is said to be mutually structurally dependent if for each pair columns $i \neq j$, where $i, j \in S$ and $i \not\perp j$.

Figure 3.1 displays a sparse matrix pattern. In Table 3.1, for each column, its degree of structural dependence and the columns that are structurally dependent on it are listed.

$$A = \begin{bmatrix} & x & x & x & & & & \\ & x & x & & & & & x \\ & & & x & & & & x \\ x & & & x & & & & \\ & & & & x & & & x \\ x & & & & & & x & \\ & & & & x & & x & \\ & & & & & x & x & \\ & & & & & x & x & \end{bmatrix}$$

Figure 3.1: A sparse matrix pattern

Table 3.1: Degree of structural dependency and structurally dependent columns

Column	*Degree of structural dependency*	*Structurally dependent columns*
1	2	{ 4, 8 }
2	3	{ 3, 4, 9 }
3	3	{ 2, 4, 9 }
4	3	{ 2, 3, 9 }
5	2	{ 7, 9 }
6	2	{ 7, 8 }
7	2	{ 5, 6 }
8	2	{ 1, 6 }
9	3	{ 2, 3, 4 }

A Jacobian matrix with sparsity pattern identical to matrix A can be determined with four extra function evaluations in FD or four forward passes in AD. To show that this is minimum possible, just observe that the columns $2, 3, 4$ and 9 are mutually structurally dependent. An optimal partitioning of the columns is:

$$\{1,2,5,6\},\{3\},\{4,7,8\},\{9\}$$

It has been shown that the unidirectional determination where the number of extra function evaluations or the number of AD forward passes are minimized is NP-hard [7, 23]. For matrices with many columns obtaining a minimum structurally orthogonal column partitioning is computationally impractical and therefore, heuristic algorithms are applied to solve the partitioning problem. If there is an effective procedure for finding the set of mutually structurally dependent columns of largest cardinality then these columns can be trivially partitioned and the partitioning can be extended to include the remaining columns while minimizing the size of the partition. Unfortunately, no such effective procedure exists for finding a maximum mutually structurally dependent set of columns [23].

Let ρ_i denote the number of nonzero entries in row i of the sparse matrix A. Then $\rho = maximum$ of ρ_i is a lower bound on the number of groups in a structurally orthogonal column partition. However, such a bound can be arbitrarily poor [23]. In our work, we consider a relaxation of mutual structural dependency of the columns. Specifically, we are interested in a sub-matrix A' of matrix A such that

- If S is a maximum mutually structurally dependent set of columns of A, then S is included in A'.
- A' is computationally easy to find.

If A' is small, then an algorithm for finding a minimum structurally orthogonal partitioning of columns can be specified by finding an optimum partitioning of the columns of A' and then extending the partition to matrix A. In Figure 3.1, the submatrix consisting of columns $2, 3, 4$ and 9 and rows $1, 2$ and 3 defines the maximum cardinality structurally dependent set of columns for matrix A. Now consider a column with a minimum degree in matrix A. Let us arbitrarily pick column 6. Removing column 6 reduces the degree of columns 7 and 8 by one. The degree of other columns is not affected. Again, we arbitrarily pick one of the minimum degree columns in the resulting matrix. Let the column be column number 7. In Table 3.2, the degree of the minimum degree column that has been removed from the matrix are listed.

Table 3.2: Steps of removing columns

Step	*Column removed*	*Minimum degree*
1	6	2
2	8	1
3	1	1
4	7	1
5	5	1
6	3	3
7	9	2
8	2	1
9	4	0

We have found the maximum of minimum degree 3 at step 6. This implies that in the submatrix of the remaining columns each column must have a degree at least 3. In the current example, the number of columns remaining is exactly equal to $1 + \textit{minimum degree} = 1 + 3 = 4$. In general, the number of columns remaining will be equal to or larger than $1 + \textit{minimum degree}$. In graph terminology, the quantity *minimum degree* where minimum degree is the maximum of minimum degree over all columns is termed "degeneracy number". In the next section, we use a graph formulation of the above procedure for finding a structurally orthogonal column partitioning. However, we emphasize that in a computer implementation a sparse matrix is preferable as many of the combinatorial tasks can be expressed as fundamental sparse matrix kernel operation such as sparse matrix-vector multiplication or sparse matrix-matrix multiplication [16, 23].

3.1.2 Coloring of Column Intersection Graph

Given a sparse matrix $A \in \Re^{m \times n}$, the column intersection graph of A is a graph $G(A) = (V, E)$ where for each unique column v_j, $j = 1, 2, .., n$ of matrix A there is a vertex $v_j \in V$ and $\{v_k, v_l\} \in E$ if and only if columns k and l share at least one nonzero in the same row. For example, consider the following sparse matrix.

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

The corresponding column intersection graph is shown in Figure 3.2.

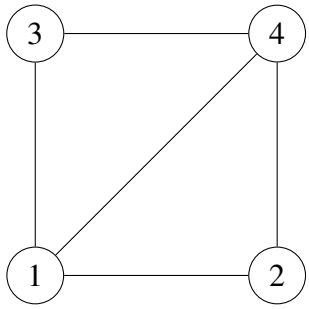

Figure 3.2: Column intersection graph

The mapping $\Phi : \{1,2,..,n\} \mapsto \{1,2,..,p\}$ is a p-coloring of the column intersection graph $G(A) = (V,E)$ if for each edge $\{v_i, v_j\} \in E$, $\Phi(v_i) \neq \Phi(v_j)$. Coleman and Moré [7] showed that structurally orthogonal partitioning of columns of matrix A is equivalent to the coloring of the graph $G(A)$. Hossain and Steihaug [23] proved that the general problem of direct determination of a sparse Jacobian matrix is equivalent to the coloring of element isolation graph associated with the matrix and that the column intersection graph coloring is a special case. We now introduce the notions of degeneracy and core as they apply to the column column intersection graph $G \equiv G(A) = (V,E)$. In Figure 3.3, we have showed the column intersection graph for the sparse matrix displayed in Figure 3.1.

In this thesis we have used *Smallest Last Ordering* (SLO) for calculating the degeneracy. Assume the vertices $V' = \{v_n, v_{n-1}, .., v_{i+1}\}$ have already been ordered. The i^{th} vertex in SLO is an unordered vertex u such that $deg(u)$ is minimum in $G[V \setminus V']$ where, $G[V \setminus V']$ is the graph obtained from G by removing the vertices of set V' from V. Let us demonstrate the SLO ordering for the column intersection graph shown in Figure 3.3. In the column intersection graph, we find minimum degree is 2 in vertices $1,5,6,7$ and 8. Let us arbitrarily

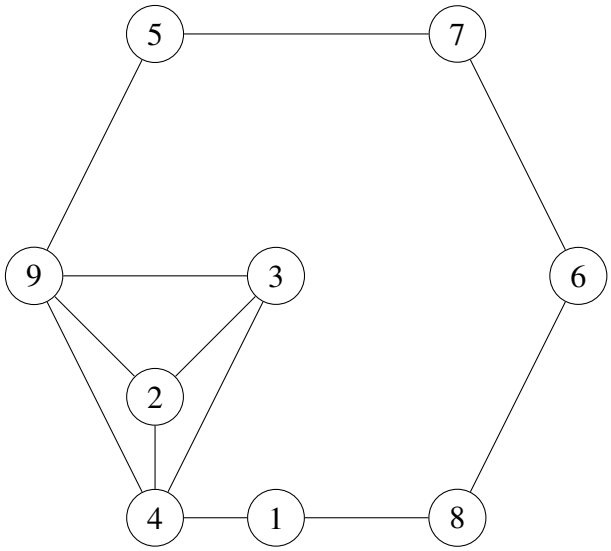

Figure 3.3: Column intersection graph of sparse matrix shown in Figure 3.1

select vertex 6 for SLO and remove vertex 6 from the column intersection graph. Now in the remaining column intersection graph, we find the minimum degree is 1 in vertices 7 and 8. Again, we arbitrarily pick vertex 8 for SLO and remove vertex 8 from the column intersection graph. By continuing those steps until all the vertices are ordered, we get the ordering of vertices $\{4,2,9,3,5,7,1,8,6\}$. Steps of smallest last ordering for the column intersection graph displayed in Figure 3.3 are shown in Table 3.3.

Table 3.3: Steps of SLO

Step	*Column removed*	*Minimum degree*	*SLO ordering*
1	6	2	{6}
2	8	1	{8, 6}
3	1	1	{1, 8, 6}
4	7	1	{7, 1, 8, 6}
5	5	1	{5, 7, 1, 8, 6}
6	3	3	{3, 5, 7, 1, 8, 6}
7	9	2	{9, 3, 5, 7, 1, 8, 6}
8	2	1	{2, 9, 3, 5, 7, 1, 8, 6}
9	4	0	{4, 2, 9, 3, 5, 7, 1, 8, 6}

Let $G = (V,E)$ be a simple undirected graph, and let $V' \subseteq V$ be a subset of vertices of G. The subgraph induced by V' is denoted by $G[V']$ and let $\delta(G)$ denote the minimum degree

of G.

Definition 3.5. A subset $V' \subseteq V$ of vertices is said to be a k-core if $\delta(G[V']) \geq k$, and the degeneracy of G is the largest k for which G has a k-core.

In Table 3.3, we have found the maximum of minimum degree 3 at step 6 of SLO ordering and the number of remaining columns at step 6 is 4. Hence the degeneracy of the graph shown in Figure 3.3 is $3(=k)$. The subgraph induced by vertices $2,3,4$ and 9 is the k-core where $k(=3)$ is maximized is shown in Figure 3.4.

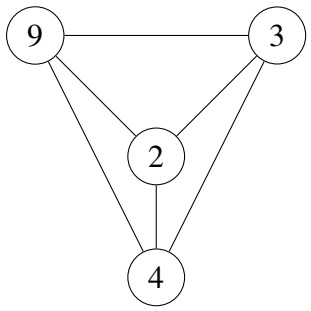

Figure 3.4: Induced subgraph of Column intersection graph shown in Figure 3.3

We emphasize that in a SLO ordered vertices it is possible that there are indices $l' < l$ such that

$$v_1, v_2, ..., v_{l'}, ..., v_l, ..., v_n$$

$$k = \delta(G[\{v_1, v_2, ..., v_{l'}\}]) = \delta(G[\{v_1, v_2, ..., v_{l'}, ..., v_l\}])$$

and that k is maximized. In this case the k-core is defined to be the subgraph $G[\{v_1, v_2, ..., v_{l'}, ..., v_l\}]$ where index l is maximized. In can be shown that the SLO ordering yields the k-core for graph $G(A)$ [29, 24]. We make the following observations on degeneracy and the associated subgraph in relation to the derivation of our algorithm for column partitioning of large sparse matrix A. Most of these observations follows from the definition of degeneracy, maximum k-core and smallest last ordering. Details can be found in [24].

1. If $C \subseteq V$, for $G(A) = (V,E)$ induces a maximum clique in $G(A)$, then C induces a maximum clique in the maximum k-core of $G(A)$. As a consequence, a lower bound

on the number of colors in an optimal coloring of $G(A)$ can be obtained from a lower bound for the maximum k-core.

2. A coloring for $G(A)$ can be obtained by coloring the maximum k-core first and then extending the coloring to the uncolored vertices. Since A is large and sparse, it is computationally infeasible to use an exact method to optimally color $G(A)$. On the other hand, if the maximum k-core is sufficiently small it may be possible to optimally color it and then extending this coloring to the vertices that are outside of the maximum k-core. Consequently, if the optimal coloring of the maximum k-core uses χ_k colors and if this coloring can be extended to all other vertices of $G(A)$ without increasing the number of colors then we have an optimal coloring of graph $G(A)$ [24].

3.2 Combined Coloring

3.2.1 Degeneracy and Core

Let $G(A)$ be the column intersection graph for a sparse matrix A and H be the induced subgraph such that $H \subseteq G(A)$. The following algorithm shows the major computational steps of finding k-core and degeneracy. The algorithm we have used here is based on smallest last ordering algorithm.

We have implemented the the algorithm using the compressed data structure used in DSJM [20]. The SLO ordering in our implementation uses bucket heap data structure of DSJM. Details of bucket heap data structure can be found in the user manual of DSJM. The overall computational complexity of this algorithm is $O(\sum_{i=1}^{n} \rho_i^2)$ [20, 24].

3.2.2 Greedy Coloring

Let $G(A)$ be the column intersection graph for a sparse matrix A. After the vertices have been ordered using *Compute_Degeneracy* algorithm, the sequential algorithm will access the vertices in the given order and will assign the smallest available color to the vertices. A greedy partitioning heuristic to find a structurally orthogonal mapping was proposed in [9].

Algorithm: Compute_Degeneracy

Input : $G(A) = (V, E)$ and
degree array such that $degree[j]$ is the degree of vertex corresponding to column j of matrix A

Output: *order* array such that j^{th} column in SLO order is $order[j]$;
k, the degeneracy of $G(A)$ i.e. $k = max\{\delta(H)\}$, $H \subseteq G(A)$ and
c_index, the maximum k-core is the subgraph induced columns $order[1]$ ... $order[c\_index]$

1 Set V' to $\emptyset$;
2 $slo\_order \leftarrow |V|$;
3 $k\_deg \leftarrow 0$;
4 Construct a *min_heap* Q of degree information from *degree* array ;
5 $H \leftarrow G$;
6 **while** Q *is not empty* **do**
7 Let j be the column such that its degree k is minimum in the induced graph H ;
8 $order[slo\_order] \leftarrow j$;
9 $k\_deg[slo\_order] \leftarrow k$;
10 $slo\_order \leftarrow slo\_order - 1$;
11 Remove column j from Q;
12 $H \leftarrow H \setminus \{j\}$;
13 Update the degree of the neighbors of column j in H;
14 Let l be the largest index such that $k\_deg[l]$ is maximum;
15 $k \leftarrow k\_deg[l]$;
16 $c\_index \leftarrow l$;

Our algorithm for finding structurally orthogonal partitioning is an adaption of sequential greedy color implementation of DSJM. Major computational steps of finding structurally orthogonal partitioning are given below:

Algorithm: Sequential_Greedy_Coloring

Input : *order* array containing a permutation of columns 1... n
Output: *color* array defining an assignment of colors to columns such that $color[j] \neq color[l]$ if columns j and l are structurally dependent

```
1 for i ← 1 to n do
2     color[i] ← n
3 for j ← 1 to n do
4     colm ← order[j] ;
5     Let neighbors be the set of columns such that l ∈ neighbors implies l ⊁ colm and
        let c be the smallest color index that has not been assigned to any column in
        neighbors;
6     Assign color index c to column colm;
```

We have used the package DSJM for implementing *Sequential_Greedy_Coloring*. The major computational cost of the *Sequential_Greedy_Coloring* algorithm is $O(\sum_{\{i|a_{ij}\neq 0\}} \rho_i)$ to find the neighbors of *colm*. For all columns in *order* array the cost is proportional to $O(\sum_{i=1}^{n} \sum_{\{i|a_{ij}\neq 0\}} \rho_i) = O(\sum_{i=1}^{n} \rho_i^2)$. So, the total computational cost of *Sequential_Greedy_Coloring* is $O(\sum_{i=1}^{m} \rho_i^2)$ [20]. The following Figure 3.5 shows the column intersection graph $G(B)$ of matrix B where columns $(1,2,3,4,5)$ are grouped into two groups {{1,4,5}, {2, 3}} by using the *Sequential_Greedy_Coloring* algorithm.

$$B = \begin{bmatrix} a_{11} & a_{12} & 0 & 0 & 0 \\ a_{21} & 0 & a_{23} & 0 & 0 \\ 0 & 0 & a_{33} & a_{34} & 0 \\ 0 & a_{42} & 0 & a_{44} & 0 \\ 0 & a_{52} & 0 & 0 & a_{55} \end{bmatrix}$$

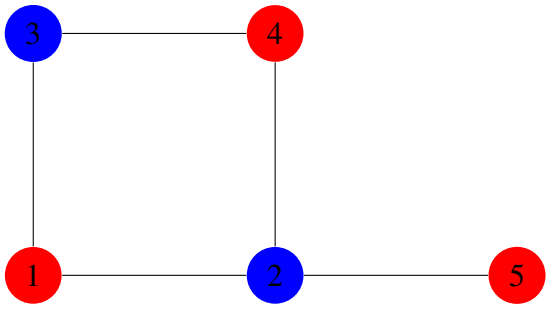

Figure 3.5: Sequential coloring example

3.2.3 Exact Coloring

For exact coloring, we have used the Randall-Brown's modified algorithm (RB) by Brélaz [3] which used DSATUR heuristics. The algorithm divides the graph coloring instance into a series of subproblems. Each subproblem corresponds to a partial coloring of the graph. At each step, there is an upper bound (UB) on the number of colors needed to color the graph. If the subproblem uses p colors such that $p < UB$, then a better coloring is found and UB is set to p. If the graph is not completely colored and the number of colors used is less than UB, then new subproblems are created. An uncolored vertex v_i is selected for branching and for each feasible color out of p colors a subproblem is created to assign that color to v_i. Then another subproblem is created to assign color $p+1$ to v_i. The choice of branch node i is critical, Brélaz suggested to choose the vertex adjacent to the largest number of differently colored nodes.

In this thesis, we have used the Randall-Brown's modified algorithm implemented by Mehrotra and Trick [30]. They implemented the algorithm by finding the maximum clique in the graph using the unweighted clique finding algorithm. They generated $10,000$ subproblems and the remaining vertices were dynamically ordered in terms of the number of adjacent colors. The subproblems were created as in the basic DASTUR algorithm and solved by using depth-first search.

3.2.4 Combined Coloring

Heuristic algorithms are frequently used for graph coloring problem but they do not provide any guarantee of the quality of the optimal solution. Again exact algorithms can not be applicable when the graph is large due to their higher computation cost. In this thesis, we have proposed a new approach which combines the sequential greedy coloring and RB. The algorithm of our combined approach is given below:

Algorithm: Combined_Coloring

Input : *order* array such that j^{th} column in SLO order is $order[j]$ and *c_index*, the maximum *k*-core is the subgraph induced columns $order[1]$ ... $order[c\_index]$

Output: *color* array containing the color of columns 1...*n*

1 Apply RB to color the *k*-core of graph $G(A)$;
2 Update the *color* array with color assignment of columns in the *k*-core;
3 Apply *Sequential_Greedy_Coloring* to color the remaining vertices;
4 Return the *color* array;

In our combined approach, we solve the problem in stages. First, we apply RB algorithm to color the maximum *k*-core graph. Maximum *k*-core graph is obtained by applying the *Compute_Degeneracy* algorithm. Figure 3.6 shows the coloring of maximum *k*-core graph using RB algorithm on it. Maximum *k*-core graph requires at least four colors according to RB algorithm.

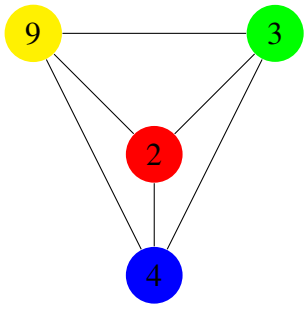

Figure 3.6: Coloring of maximum *k*-core graph using RB. Maximum *k*-core graph is obtained by using the *Compute_Degeneracy* on Figure 3.3

Then we extend the coloring information of maximum *k*-core to the uncolored vertices by using *Sequential_Greedy_Coloring* algorithm. Figure 3.7 shows the coloring of using

Sequential_Greedy_Coloring algorithm. In this case, we get four colors. So in this example, the number of colors is equal to be the number of colors needed in RB. Hence we get an optimal partitioning of columns

$$\{1,2,5,6\},\{3\},\{4,7,8\},\{9\}$$

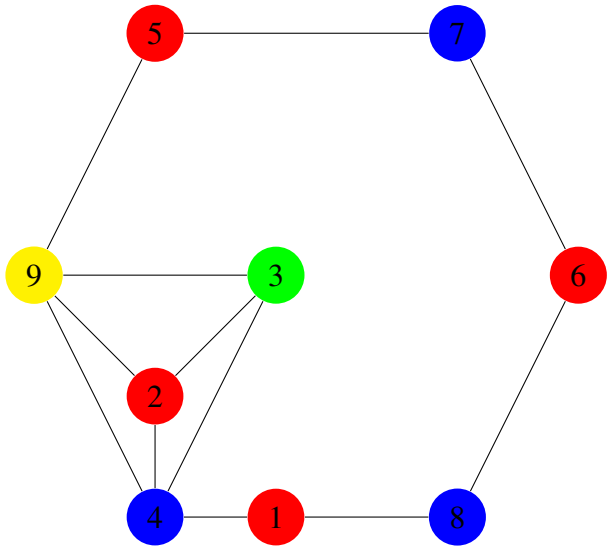

Figure 3.7: Combined coloring result of sparse matrix shown in Figure 3.1

Chapter 4

Numerical Experiments

In this chapter, we present numerical results of applying algorithms proposed in this thesis on practical instances. In Section 4.1, we give details of test data sets that we have used in our experiments. The data structures used in this thesis are briefly covered in Section 4.2. Finally, we describe numerical experiments and summary of experimental results in Section 4.4.

4.1 Test Data Sets

In this section, we discuss two different sets of sparse matrices for our experimental results. Table 4.1 and Table 4.2 lists matrices along with their structural properties. The data set in Table 4.1 is obtained from University of Florida Sparse Matrix Collection [2] and the data set in Table 4.2 is collected from Matrix Market Collection [1]. In tables, Table 4.1 and Table 4.2, columns labelled *m*, *n*, *nnz* are used to represent the number of rows, columns and total number of nonzeroes in the matrix respectively. Columns ρ_{max} and ρ_{min} represent maximum and minimum number of nonzeroes in any row of the matrix and $\overline{\rho}$ denotes the arithmetic mean of nonzeroes in each row of the matrix.

Table 4.1: Matrix Data Set - 1

Matrix Name	*m*	*n*	*nnz*	ρ_{max}	$\overline{\rho}$	ρ_{min}
cage11	39082	39082	559722	31	14	3
Continued on next page						

Table 4.1 – continued from previous page

Matrix Name	*m*	*n*	*nnz*	ρ_{max}	$\overline{\rho}$	ρ_{min}
cage12	130228	130228	2032536	33	15	5
fidap002	441	441	26831	125	60	28
fidap003	1821	1821	52659	62	28	8
fidap005	27	27	279	15	10	6
fidap015	6867	6867	96421	18	14	4
fidap018	5773	5773	69335	18	12	1
fidap022	839	839	22613	62	26	8
fidap029	2870	2870	23754	9	8	1
fidap031	3909	3909	115299	75	29	9
fidap033	1733	1733	20315	18	11	4
fidap035	19716	19716	218308	18	11	1
fidapm05	42	42	520	21	12	6

Table 4.2: Matrix Data Set - 2

Matrix Name	*m*	*n*	*nnz*	ρ_{max}	$\overline{\rho}$	ρ_{min}
arc130	130	130	1282	124	9	1
can1054	1054	1054	12196	35	11	6
can1072	1072	1072	12444	35	11	6
can634	634	634	7228	28	11	2
dwt1007	1007	1007	8575	10	8	3
dwt1242	1242	1242	10426	12	8	2
Continued on next page						

Table 4.2 – continued from previous page

Matrix Name	*m*	*n*	*nnz*	ρ_{max}	$\bar{\rho}$	ρ_{min}
dwt2680	2680	2680	25026	19	9	4
dwt162	162	162	1182	9	7	2
dwt193	193	193	3493	30	18	8
dwt221	221	221	1629	12	7	4
dwt307	307	307	2523	9	8	6
dwt361	361	361	2953	9	8	4
dwt59	59	59	267	6	4	2
e20r0000	4241	4241	131556	62	31	8
e20r0100	4241	4241	131556	62	31	8
e30r0000	9661	9661	306356	62	31	8
e30r0100	9661	9661	306356	62	31	8
fs5411	541	541	4285	11	7	1
fs5412	541	541	4285	11	7	1
ibm32	32	32	126	3	2	8
impcolb	59	59	312	5	2	7
impcolc	137	137	411	3	1	8
impcold	425	425	1339	10	3	1
lunda	147	147	2449	21	16	5
lundb	147	147	2441	21	16	5
west0067	67	67	294	4	1	6
west0381	381	381	2157	5	1	25

4.2 Data Structure

Generally, a matrix is stored using a two-dimensional array but it is slow and inefficient for large sparse matrices as the processing and memory are wasted on the zero entries. So it is beneficial to use specialized algorithms and data structures to take advantage of sparse structure for storing and manipulating sparse matrices on a computer. In this thesis, we have used a data structure proposed in [20]. This data structure takes only $3 \times nnz + m + n + 2$ memory locations where nnz is the total number of nonzeroes, m is the number of rows and n is the number of columns of a sparse matrix. The total data structure is divided into two parts: *Compressed Column Storage* and *Compressed Row Storage*.

4.2.1 Compressed Column Storage

The compressed Column Storage (CCS) format represents a matrix by three one dimensional arrays, named as row_ind, col_ptr and $value$. Row indices of nonzero elements of each column are stored into row_ind array. The starting index of each column is stored in col_ptr array and nonzero elements are stored into $value$ array. Row indices of nonzero elements in column j can be found in between $row\_ind[col\_ptr[j]]$ and $row\_ind[col\_ptr[j+1]-1]$. Therefore for a sparse matrix $A \in \Re^{m \times n}$, the total amount of memory required to store compressed column storage is $2nnz + n + 1$.

4.2.2 Compressed Row Storage

In Compressed Row Storage (CRS), column indices of nonzero elements of each row are stored in col_ind and the starting index of each row is stored in row_ptr. Like CCS, CRS also uses $value$ array to store nonzero elements. Column indices of each row i can be found in between $col\_ind[row\_ptr[i]]$ and $col\_ind[row\_ptr[i+1]-1]$. Hence the CRS uses $2nnz + m + 1$ memory locations to store a sparse matrix. Following example demonstrate the CCS and CRS data structures. Let

$$A = \begin{bmatrix} a_{11} & 0 & a_{13} & a_{14} & 0 & 0 \\ 0 & a_{22} & 0 & 0 & 0 & a_{26} \\ a_{31} & 0 & a_{33} & 0 & a_{35} & 0 \\ 0 & a_{42} & 0 & a_{44} & 0 & 0 \\ a_{51} & 0 & a_{53} & 0 & a_{55} & 0 \\ 0 & a_{62} & 0 & a_{64} & 0 & a_{66} \end{bmatrix}$$

The corresponding data structure of the matrix A is as follows:

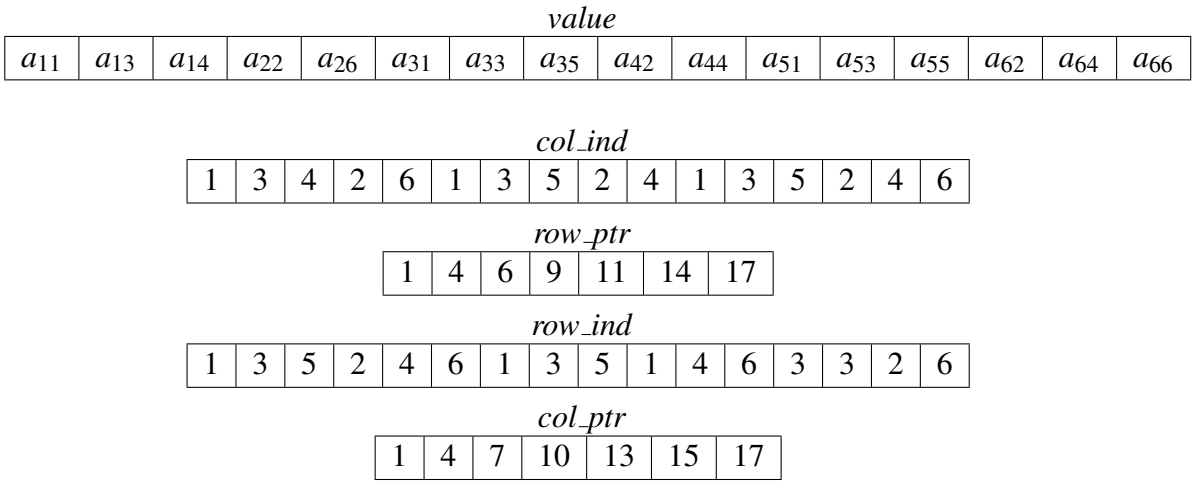

Figure 4.1: CRS and CCS data structures

4.3 Test Environment

All the experiments were done on a 64 bit Ubuntu 16.04 with 2.6 GHz Intel Core i5-3230M, 4 GB RAM, and 3072K L3 cache.

4.4 Test Results

Our implementation depends heavily on the use of efficient sparse data structures and operations on sparse matrices. We have used C++ for implementing our algorithms and we compare our results with DSJM [20]. Our test results are shown in Table 4.3 and 4.4.

In both tables, matrices are listed under *Matrix Name*, $\rho_{\max}$ column represents maximum number of nonzeroes in any row of the matrix, degeneracy number is shown in *Degeneracy* column and *Core Size* denotes the number of columns in the maximum k-core. Entries in column labelled *DSJM* denotes the number of colors required using SLO in DSJM. The number of colors required to color the maximum k-core by using Randall-Brown's modified algorithm is presented in the *RB* column. Column *CM* is used to represent the total number of colors required using our combined approach. The number in boldface represents the optimal coloring (partitioning) for the respective problem instance.

4.4.1 Numerical Experiments

Table 4.3: Coloring results for data set - 1

Matrix Name	ρ_{max}	*Degeneracy*	*Core Size*	*DSJM*	*RB*	***CM***
cage11	31	141	832	62	56*	**60**
cage12	33	162	1710	68	61*	**65**
fidap002	125	129	275	125	125	**125**
fidap003	62	65	482	62	54	**62**
fidap005	15	14	21	15	15	**15**
fidap015	18	25	4482	22	18	**18**
fidap018	18	25	2710	22	18	**18**
fidap022	62	70	155	64	62	64
fidap029	9	12	2643	12	9	**9**
fidap031	75	107	2322	93	90*	**90**
fidap033	18	23	1054	21	18	**18**
fidap035	18	25	7252	22	18	**18**

Continued on next page

Table 4.3 – continued from previous page

Matrix Name	ρ_{max}	*Degeneracy*	*Core Size*	*DSJM*	*RB*	***CM***
fidapm05	21	20	30	21	21	**21**

* - Terminated after 1 hour

Table 4.4: Coloring results for data set - 2

Matrix Name	ρ_{max}	*Degeneracy*	*Core Size*	*DSJM*	*RB*	***CM***
arc130	124	123	124	124	124	**124**
can1054	35	36	223	35	35	**35**
can1072	35	36	220	35	35	**35**
can634	28	28	92	28	27	**28**
dwt1007	10	12	893	11	10	11
dwt1242	12	15	322	14	12	**13**
dwt2680	19	21	89	19	18	**19**
dwt162	9	10	59	10	9	**9**
dwt193	30	34	81	31	30	**30**
dwt221	12	12	173	13	12	**12**
dwt307	9	14	144	12	9	**9**
dwt361	9	12	277	10	9	**9**
dwt59	6	6	54	7	6	**6**
e20r0000	62	80	3302	70	67*	**67**
e20r0100	62	80	3302	71	67*	**67**
e30r0000	62	80	8442	70	68*	**68**
e30r0100	62	80	8442	71	68*	**68**

Continued on next page

Table 4.4 – continued from previous page

Matrix Name	ρ_{max}	*Degeneracy*	*Core Size*	*DSJM*	*RB*	***CM***
fs5411	11	14	541	13	12*	**12**
fs5412	11	14	541	13	12*	**12**
ibm32	8	8	22	8	7	**8**
impcolb	7	10	15	11	10	**10**
impcolc	8	8	63	8	7	**8**
impcold	10	10	193	11	10	**10**
lunda	21	26	90	24	21	**21**
lundb	21	26	90	24	21	**21**
west0067	6	9	54	9	8	**8**
west0381	25	31	105	30	28	**29**

* - Terminated after 1 hour

4.4.2 Summary of Experimental Results

ρ_{max} is shown to be a lower bound on the number of groups in a structurally orthogonal partition of the columns [6]. We found ρ_{max} to be equal to the chromatic number for 9 matrices out of 13 matrices in data set 1 and in data set 2, it is true for 16 matrices out of 27 matrices.

We observe that our combined approach is more efficient than DSJM in terms of numbers of partitions (colors). In data set 1, we found fewer number of colors than DSJM in 8 matrices and for 5 matrices, we found the equal number of colors as in DSJM. In data set 2, our combined approach obtained fewer number of colors in 19 matrices while 9 matrices received the same number of colors as in DSJM.

We note that it may be possible to obtain multiple k-core of a column intersection

graph. For example, consider the matrix *ibm*32. Its degeneracy number is 8. In Figure 4.2, we see that maximum of minimum degree (degeneracy) is found at positions where the total number of columns is 11, 21 and 22. So we found three different sizes of k-core but we select the core size to be 22 since it has maximum number of columns. Therefore, we can say that maximum k-core is unique.

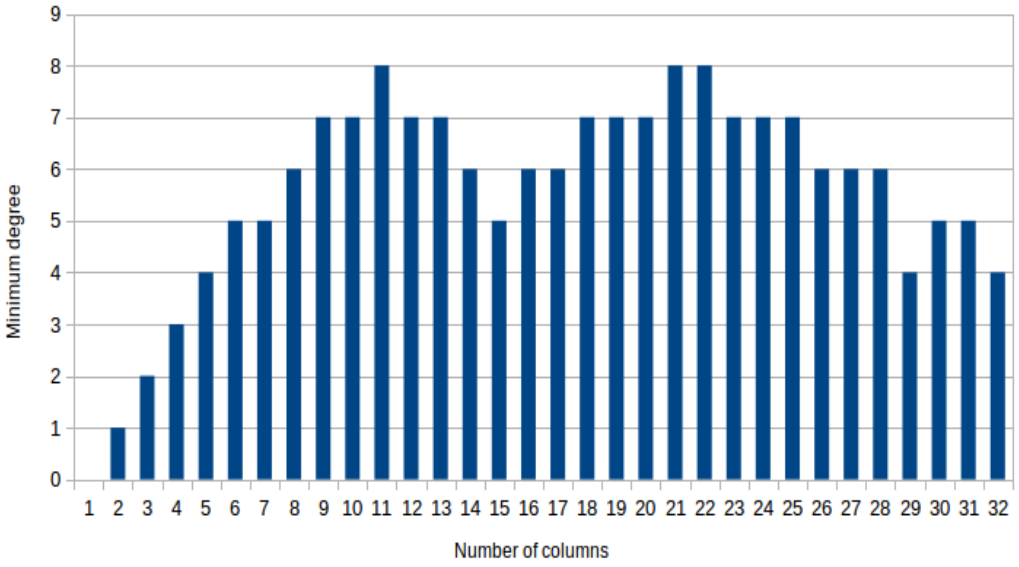

Figure 4.2: Finding degeneracy and core size of *ibm32*

From Tables 4.3 and 4.4, we found optimal coloring equal to be ρ_{max} for 25 matrices by applying the RB algorithm on the maximum k-core and out of these 24 matrices, 20 matrices receive the equal number of colors as in RB when extended to the whole matrix (CM). Consider a matrix $fidap035$. The Figure 4.3 shows the maximum core region in red color for the $fidap035$ matrix. Core region contains only 7252 columns whereas the total number of columns in $fidap035$ is 19716. In this case, using maximum k core half of the problem size is reduced. Hence by using the maximum core graph we can significantly reduce the size of the problem.

One of the central contributions of our work is that we have solved a number of previously unsolved instances by showing optimal partitioning. For instances *fidap015, fidap018, fidap029, fidap033, fidap035* of Table 4.3 and instances *dwt162, dwt193, dwt221,*

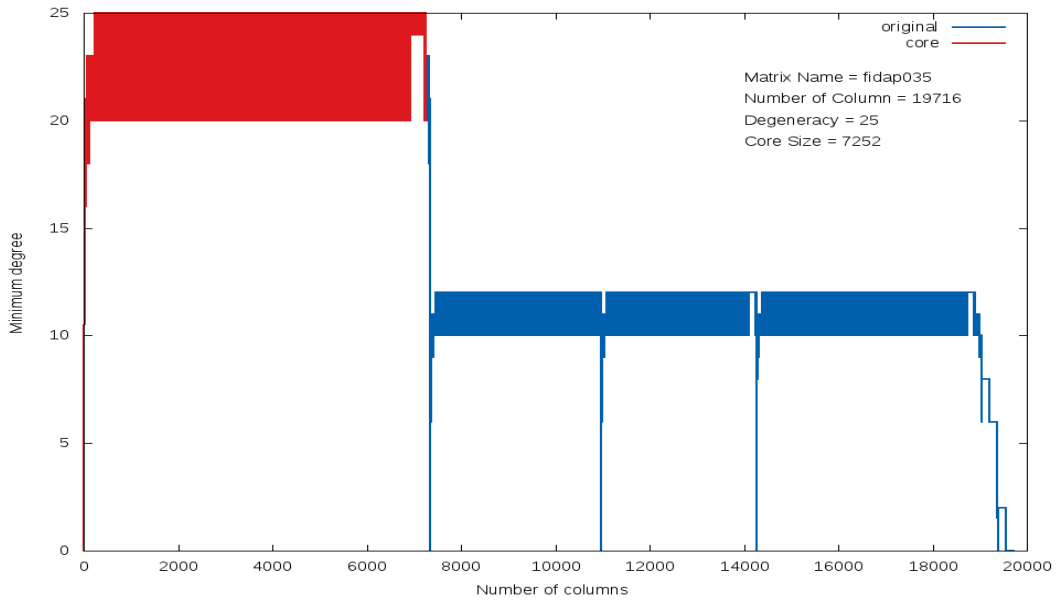

Figure 4.3: Finding degeneracy and core size of *fidap035*

dwt307, dwt361, dwt59, impcolb, impcold, lunda, lundb, west0067 of Table 4.4 optimal partitioning were not known. We report optimal partitioning of those instances for the first time. Specifically, on problem *west*0067 of Table 5.4, $\rho_{max} = 6$ while the optimal partitioning is confirmed by having the maximum k-core partitioning (RB) extended to the whole matrix without the addition of groups.

Table 4.5: Timing results

Matrix Name	*RD Time*	***CM Time***
cage11	-	0.324
cage12	-	1.203
fidap002	0.05	0.026
fidap003	0.983	0.116
fidap005	0.00001	0.00001
fidap015	4.966	2.812

Continued on next page

Table 4.5 – continued from previous page

Matrix Name	*RD Time*	***CM Time***
fidap018	4.03	1.584
fidap022	123.6	0.01
fidap029	1.55	1.11
fidap031	-	1.3
fidap033	0.917	0.44
fidap035	-	4.5
fidapm05	0.00001	0.00001
arc130	0.00001	0.00001
can1054	0.34	0.003
can1072	0.34	0.002
can634	0.00001	0.00001
dwt1007	0.216	.002
dwt1242	15.788	.003
dwt2680	0.00001	0.00001
dwt162	0.00001	0.00001
dwt193	0.00001	0.00001
dwt221	0.00001	0.00001
dwt307	0.00001	0.00001
dwt361	0.166	0.001
dwt59	0.00001	0.00001
e20r0000	-	1.99

Continued on next page

Table 4.5 – continued from previous page

Matrix Name	*RD Time*	***CM Time***
e20r0100	-	2.1
e30r0000	-	5.3
e30r0100	-	5.5
fs5411	-	0.1
fs5412	-	0.1
ibm32	0.00001	0.00001
impcolb	0.00001	0.00001
impcolc	0.00001	0.00001
impcold	0.00001	0.00001
lunda	0.00001	0.00001
lundb	0.00001	0.00001
west0067	0.00001	0.00001
west0381	0.00001	0.00001

- Represents that no result was found in 1 hours

Table 4.5 lists a comparison for running time for RD and proposed combined method. The running time for both the algorithms is given in seconds. Reported time discards the running time for I/O operations (e.g reading the matrix description from file). Table 4.5 clearly shows that our combined method is efficient in terms of running time, as it is several order of magnitude of faster than RD. We also observe that Randall-Brown's algorithm can not solve all test matrices. For matrices cage11, cage12, fidap031, fidap035, e20r0000, e20r0100, e30r0000, e30r0100, fs5411 and fs5412, RD can not able to solve them. Those problems are solved in a reasonable amount of time by our proposed combined method.

Chapter 5

Conclusion and Future works

5.1 Conclusion

Finding optimal coloring on large instances is hard to approximate. This thesis has been an effort to provide optimal coloring for large sparse instances. Using the concept of degeneracy and maximum k-core, we propose a new algorithm by combing existing exact and heuristic algorithms. Our exact algorithm is based on Randall-Brown's modified algorithm and greedy coloring is used as a heuristic algorithm. In this thesis,

- We have presented a combined algorithm to find optimal coloring on large instances.
- We have shown optimal coloring for some test problems that were not known.
- We have shown better coloring for some test problems than the existing known coloring results.

5.2 Future works

Work presented in this thesis deserve further study and extension. There are many opportunities for extending the scope of this thesis. A natural extension involves a generalization of maximum k-core and degeneracy to Hessian matrix determination and row-column compression [13] for Jacobian's. Extension to distance coloring of a general graph is interesting in its own right.

Bibliography

[1] The Matrix Market Collection. http://math.nist.gov/MatrixMarket/matrices.html. Accessed: 2016-10-28.

[2] The University of Florida Sparse Matrix Collection. http://www.cise.u.edu/research/sparse/matrices/. Accessed: 2016-10-28.

[3] Daniel Brélaz. New methods to color the vertices of a graph. *Communications of the ACM*, 22(4):251–256, 1979.

[4] Alberto Caprara, Leo Kroon, Michele Monaci, Marc Peeters, and Paolo Toth. Passenger railway optimization. *Handbooks in operations research and management science*, 14:129–187, 2007.

[5] Fred C Chow and John L Hennessy. The priority-based coloring approach to register allocation. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 12(4):501–536, 1990.

[6] Thomas F Coleman, Burton S Garbow, and Jorge J Moré. Software for estimating sparse Jacobian matrices. *ACM Transactions on Mathematical Software (TOMS)*, 10(3):329–345, 1984.

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

[8] Joseph C Culberson and Feng Luo. Exploring the k-colorable landscape with iterated greedy. *Cliques, coloring, and satisfiability: second DIMACS implementation challenge*, 26:245–284, 1996.

[9] AR Curtis, Michael JD Powell, and John K Reid. On the estimation of sparse Jacobian matrices. *J. Inst. Math. Appl*, 13(1):117–120, 1974.

[10] Dominique de Werra. An introduction to timetabling. *European journal of operational research*, 19(2):151–162, 1985.

[11] Andreas Gamst. Some lower bounds for a class of frequency assignment problems. *IEEE transactions on vehicular technology*, 35(1):8–14, 1986.

[12] Michael R Gary and David S Johnson. Computers and intractability: A guide to the theory of np-completeness, 1979.

[13] Daya R Gaur, Shahadat Hossain, and Anik Saha. Determining sparse Jacobian matrices using two-sided compression: An algorithm and lower bounds. In *Mathematical and Computational Approaches in Advancing Modern Science and Engineering*, pages 425–434. Springer, 2016.

[14] Daya R. Gaur, Shahadat Hossain, and Rishi R. Singh. Star Bi-Coloring of Bipartite Graphs using Column Generation. In *CTW 2015: 13th Cologne Twente Workshop on Graphs and Combinatorial Optimization, Istanbul, Turkey, May 26-28, 2015*, pages 250–253, 2016.

[15] Assefaw Hadish Gebremedhin, Fredrik Manne, and Alex Pothen. What color is your Jacobian? graph coloring for computing derivatives. *SIAM review*, 47(4):629–705, 2005.

[16] John R Gilbert, Viral B Shah, and Steve Reinhardt. A unified framework for numerical and combinatorial computing. *Computing in Science & Engineering*, 10(2):20–25, 2008.

[17] Andreas Griewank and Andrea Walther. *Evaluating derivatives: principles and techniques of algorithmic differentiation*. SIAM, 2008.

[18] Peter Grindrod and Milla Kibble. Review of uses of network and graph theory concepts within proteomics. *Expert review of proteomics*, 1(2):229–238, 2004.

[19] Pierre Hansen, Martine Labbé, and David Schindl. Set covering and packing formulations of graph coloring: algorithms and first polyhedral results. *Discrete Optimization*, 6(2):135–147, 2009.

[20] Mahmudul Hasan, Shahadat Hossain, Ahamad Imtiaz Khan, Nasrin Hakim Mithila, and Ashraful Huq Suny. Dsjm: A software toolkit for direct determination of sparse Jacobian matrices. In *Mathematical Software–ICMS 2016: 5th International Conference, Berlin, Germany, July 11-14, 2016, Proceedings*, volume 9725, page 275. Springer, 2016.

[21] Shahadat Hossain and Trond Steihaug. Reducing the number of AD passes for computing a sparse Jacobian matrix. In *Automatic differentiation of algorithms*, pages 263–270. Springer, 2002.

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

[23] Shahadat Hossain and Trond Steihaug. Optimal direct determination of sparse Jacobian matrices. *Optimization Methods and Software*, 28(6):1218–1232, 2013.

[24] Shahadat Hossain and Ashraful Huq Suny. Column Partitioning, Degeneracy and Coloring in the Determination of Large-scale Derivative Matrices. Article in preparation, 2016.

[25] Frank Thomson Leighton. A graph coloring algorithm for large scheduling problems. *Journal of research of the national bureau of standards*, 84(6):489–506, 1979.

[26] Don R Lick and Arthur T White. k-Degenerate graphs. *Canadian J. of Mathematics*, 22:1082–1096, 1970.

[27] R Duncan Luce and Albert D Perry. A method of matrix analysis of group structure. *Psychometrika*, 14(2):95–116, 1949.

[28] Enrico Malaguti and Paolo Toth. A survey on vertex coloring problems. *International Transactions in Operational Research*, 17(1):1–34, 2010.

[29] David W Matula and Leland L Beck. Smallest-last ordering and clustering and graph coloring algorithms. *Journal of the ACM (JACM)*, 30(3):417–427, 1983.

[30] Anuj Mehrotra and Michael A Trick. A column generation approach for graph coloring. *informs Journal on Computing*, 8(4):344–354, 1996.

[31] Jorge Nocedal and Stephen Wright. *Numerical optimization*. Springer Science & Business Media, 2006.

[32] Ryan A Rossi and Nesreen K Ahmed. Coloring large complex networks. *Social Network Analysis and Mining*, 4(1):1–37, 2014.

[33] Anurag Verma, Austin Buchanan, and Sergiy Butenko. Solving the maximum clique and vertex coloring problems on very large sparse networks. *INFORMS Journal on Computing*, 27(1):164–177, 2015.

[34] T-K Woo, Stanley YW Su, and Richard Newman-Wolfe. Resource allocation in a dynamically partitionable bus network using a graph coloring algorithm. *IEEE Transactions on Communications*, 39(12):1794–1801, 1991.