ON THE DETERMINATION OF SPARSE HESSIAN MATRICES USING MULTI-COLORING

NASRIN HAKIM MITHILA
Bachelor of Science, Military Institute of Science and Technology, 2012

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

ON THE DETERMINATION OF SPARSE HESSIAN MATRICES USING MULTI-COLORING

NASRIN HAKIM MITHILA

Date of Defence: December 20, 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 family.

Abstract

Efficient determination of large sparse Hessian matrices leads to solving many optimization problems. Exploiting sparsity and symmetry of the Hessian matrix can reduce the number of function evaluations required to determine the matrix. This sparse matrix determination problem can be posed as a graph coloring problem. Graph formulation of the problem using an appropriate model can lead to a better exposition of the matrix compression heuristics.

Acknowledgments

I would like to express my gratitude to everyone who have encouraged and guided me throughout this thesis work.

First and foremost, I would like to give my utmost thanks and appreciation to my supervisor, Dr. Shahadat Hossain, for his continuous guidance, helpful suggestions, and persistent encouragement throughout the journey of my MSc program.

I would also like to express sincere appreciation and obligation to my supervisory committee members, Dr. Daya Gaur and Dr. Robert Benkoczi for there encouragement and insightful advice.

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

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Many algorithms in optimization and partial differential equations require the repeated evaluation of gradient, Jacobian or Hessian matrices. These derivative matrices can be estimated through Automatic or Algorithmic Differentiation (AD) or Finite Differencing (FD) where the cost involved in computing is proportional to the number of columns in the matrix. Sparsity and symmetry in the derivative matrices can be exploited to solve problems economically: much faster and using far less memory than if all the entries of a matrix were stored and took part in explicit computations. It is very crucial for large-scale computational problems arising in structural design, circuit analysis, semiconductor modeling, and many other areas where solving problems is not computationally feasible without any sparse techniques. Taking the effective advantage of the structure of a sparse matrix requires a combination of numerical and combinatorial methods (graph algorithms and related data structures).

In a compression-reconstruction method, computing the sparse Jacobian or Hessian matrix, whose sparsity pattern is known a priori, involves partitioning the columns of the matrix into the fewest groups such that the nonzero unknowns in each group can be determined uniquely by a matrix-vector product of the form Ad, where A is the matrix to be determined and d is a direction determined by the columns in a group [13, 34, 24, 25].

For a twice continuously differentiable function $f : \mathbb{R}^n \mapsto \mathbb{R}$ it is usual for the second derivative matrix to have many zero elements in known positions with large n [33]. In this instance, efficient approach is to estimate second derivatives rather than evaluating the function with its derivatives in each iteration for unconstrained optimization. In this thesis we are concerned with the determination of symmetric Hessian matrices $\nabla f'(x)$ i.e., where ∇f is the gradient of f.

Sparse matrix partitioning problems are often modeled and effectively solved using specialized graph coloring problems and their algorithms. A natural representation of the structure of a Hessian matrix is to use its adjacency graph. There has been significant work on symmetry-exploiting determination methods where the vertices are partitioned into small number of groups or color classes such that the nonzero unknowns in each group can be determined directly or indirectly, via FD or AD [10, 15, 16, 29, 34]. Gebremedhin et al [15] present several heuristics for star coloring (direct determination) and acyclic coloring (determination via substitution) and analyze the performance of the implemented algorithms on an extensive test suite. Hossain and Steihaug [24] propose the pattern graph as a unifying framework for methods that exploit the sparsity by matrix compressions: row compression, column compression, or a combination of the two in sparse Jacobian matrix computation. From an algorithmic view point the structural correspondence between the matrix and its pattern graph leads to a better exposition of the compression heuristics and their efficient computer realization.

1.1 Our Contribution

In this thesis we assume that the sparsity pattern of the Hessian matrix is known and the problem lies in partitioning the columns of the matrix into groups to determine each nonzero entries of the matrix. We use direct method as the evaluation scheme which corresponds to solving a diagonal system of equations. We employ pattern graph model to formulate Hessian matrix determination problem. The underlying mapping is defined on the nonzero unknowns of the Hessian matrix such that sparsity can be exploited at element level. Further, the mapping induces "multi-coloring" i.e., nonzero unknowns can assume more than one color or belong to more than one group as long as each non-zero can be determined directly. We use data structures where the sparse matrix is represented using Compressed Row Storage (CRS) and Compressed Column Storage (CCS) storage scheme to implement the direct determination algorithms. Results from experiments on various test matrices demonstrate the effectiveness of our method. Specific contributions are given below:

1. We extend the definition of pattern graph in [24] to incorporate symmetry.
2. We show that the symmetric direct cover is equivalent to a multi-coloring problem on the pattern graph.
3. We propose an algorithm to identify the reduced seed matrix corresponding to a direct cover for Hessian matrix H. Each nonzero element is determined directly from this cover.
4. We present results of numerical testing using a suite of large-scale practical problem instance to validate our approach.

Parts of the work contained in this thesis have been published in [21].

1.2 Thesis organization

Including this chapter, there are five more chapters in this thesis organized as follows.

In Chapter 2, we describe Newton's method to solve unconstrained minimization problem. We review some basic concepts used in this thesis and a finite difference approximation scheme that is suitable for exploiting known sparsity is introduced. For completeness we also describe Automatic Differentiation (AD), which is also applicable to our method to avoid truncation error. Finally we describe the variants of matrix partitioning problem.

In Chapter 3, we provide some basic graph theory definitions and notations. This is followed by a brief description of contemporary graph formulations for large scale sparse matrix determination problem and a detailed comparison among them.

In Chapter 4, we describe the main result of our work for determining symmetric direct cover and the proposed algorithm to determine the Hessian matrix.

In Chapter 5, we provide experimental results that demonstrate the efficacy of our algorithms.

Finally in Chapter 6, we provide concluding remarks and directions for future research in this area.

Chapter 2

Preliminaries

In this chapter, we give the motivation to determine Hessian matrices, discuss about various sparse matrix determination problems and review mathematical preliminaries necessary for this thesis.

2.1 Newton's Method for Unconstrained Minimization

Derivative information is needed in the solution of systems of nonlinear equations and in the unconstrained minimization problems. We consider the unconstrained minimization problem

$$\min_{x \in \mathbb{R}^n} f(x)$$

At point $x = \bar{x}$, $f(x)$ can be approximated by:

$$q(x) = f(\bar{x}) + \nabla f(\bar{x})^T (x - \bar{x}) + \frac{1}{2}(x - \bar{x})^T H(\bar{x})(x - \bar{x}) \tag{2.1}$$

which is the quadratic Taylor expansion of $f(x)$ where $\nabla f(x)$ is the first derivative (the gradient vector) of $f(x)$ and $H(x) = \nabla^2 f(x)$ is the second derivative (the Hessian matrix) of $f(x)$. The quadratic function $q(x)$ is minimized by solving $\nabla q(x) = 0$. We have to solve

$$\nabla f(\bar{x}) + H(\bar{x})(x - \bar{x}) = 0 \tag{2.2}$$

since the gradient of $q(x)$ is:

$$\nabla q(x) = \nabla f(\bar{x}) + H(\bar{x})(x - \bar{x}) \tag{2.3}$$

Equation (2.2) yields,

$$x - \bar{x} = -H(\bar{x})^{-1} \nabla f(\bar{x}) \tag{2.4}$$

Equation (2.4) is commonly called the Newton step. This leads to the following algorithm for solving the above minimization problem.

Algorithm: Newton's Method

```
  Input: Given an initial approximation $x_0$
1 for $k \leftarrow 0$ to max_iter do
2     Solve $s_k = -H(x_k)^{-1} \nabla f(x_k)$
3     if $s_k = 0$ then
4         break;
5     Set $x_{k+1} \leftarrow x_k + s_k$
```

One of the main difficulties in using Newton's method on practical problems is the need for derivative information at each iteration. Fortunately in many large-scale problems, the Hessian matrix is sparse or structured. By exploiting this sparsity and symmetry, we can efficiently determine the Hessian matrix and thus significantly reduce the overall computational cost of the solution process.

2.2 Computing Partial Derivatives

In this section, we introduce some basic terminologies related to this thesis and discuss different methods for obtaining derivative information. In section 2.1 we have observed that the gradient and Hessian are useful quantities in modeling and solving nonlinear problems. In practical problems the analytical derivatives are usually unavailable. For example the function for which the derivatives are sought is available in the form of a "black-box" i.e. given an input vector, the function gives an output vector which represents the function evaluated at the input vector.

2.2.1 Gradient

Let $f : \mathbb{R}^n \mapsto \mathbb{R}$ be a twice continuously differentiable function. The gradient of function $f(x_1, x_2, x_3, \ldots, x_n)$ is denoted by,

$$\nabla f = \frac{\partial f}{\partial x_1} e_1 + \cdots + \frac{\partial f}{\partial x_n} e_n = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{bmatrix} \tag{2.5}$$

where $\frac{\partial f}{\partial x_i}$ are the partial derivatives of f and the e_i is the ith unit coordinate vector, that is, the vector whose elements are all 0 except for 1 in the ith position.

Hessian matrix is a square matrix of second order partial derivatives of a scalar-valued function f.

$$H = \begin{bmatrix} \frac{\partial f}{\partial x_1}\left(\frac{\partial f}{\partial x_1}\right) & \cdots & \frac{\partial f}{\partial x_1}\left(\frac{\partial f}{\partial x_n}\right) \\ \vdots & \ddots & \vdots \\ \frac{\partial f}{\partial x_n}\left(\frac{\partial f}{\partial x_1}\right) & \cdots & \frac{\partial f}{\partial x_n}\left(\frac{\partial f}{\partial x_n}\right) \end{bmatrix} = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \cdots & \frac{\partial^2 f}{\partial x_1 x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n x_1} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix} \tag{2.6}$$

The second-derivative is independent of the order in which derivatives are taken. Hence, $H(i, j) = H(j, i)$ for every pair (i, j) and the Hessian is a symmetric matrix.

For example, let $f(x_1, x_2) = x_1^2 + 3x_1x_2^2 + 5x_2$

The gradient is,

$$\nabla f(x_1, x_2) = \begin{bmatrix} 2x_1 + 3x_2^2 \\ 6x_1x_2 + 5 \end{bmatrix}$$

The Hessian matrix is

$$H(x_1,x_2) = \begin{bmatrix} 2 & 6x_2 \\ 6x_2 & 6x_1 \end{bmatrix}$$

2.2.2 Finite Difference Approximation

Finite differencing (FD) is an approach to the calculation of approximate derivatives whose motivation comes from Taylor's theorem. For instance, we can get an approximation of the partial derivative of f with respect to the i^{th} variable x_i using the forward difference formula,

$$\frac{\partial f}{\partial x_i} \approx \frac{f(x+\varepsilon e_i) - f(x)}{\varepsilon} \tag{2.7}$$

where $i = 1,2,...,n$, e_i is the ith unit coordinate vector and $\varepsilon > 0$ is small. Assuming f has already been evaluated, we can estimate the partial derivatives in the ith column of matrix A through the additional function evaluation $f(x+\varepsilon e_i)$. Then, we will need n extra function evaluations to determine A if the sparsity information is not exploited.

Finite difference approximation is easy to implement. To obtain an approximation to the derivatives, we just need call the subroutine that implements the mathematical function without accessing the function code. But it is prone to numerical instability. If ε is too large, then approximation may not be accurate due to truncation error. Also if ε is too small, then the accuracy is compromised due to rounding errors. Truncation error is the error that is caused by neglecting the higher order terms in mathematical series and approximating it to a finite sum. Rounding error is caused by performing arithmetic operations in fixed precisions. For detail information on errors in FD method we refer to [33].

2.2.3 Approximating the Hessian

It is possible to obtain the Hessian matrix by applying finite difference approximation techniques for the vector function ∇f. In many algorithms, each iteration requires only the Hessian-vector product $\nabla^2 f(x)p$, for a given vector p instead the knowledge of the full

Hessian matrix. We can obtain an approximation to this matrix-vector product by Taylor's theorem. When second derivatives of f exist and are Lipschitz continuous near x, we have

$$\nabla^2 f(x)p \approx \frac{\nabla f(x+\varepsilon p) - \nabla f(x)}{\varepsilon} \tag{2.8}$$

For the case in which even gradients are not available, we can use Taylor's theorem once again for approximating the Hessian that use only function values [33]. For the sparse Hessian matrix, any estimate of the element $\nabla^2 f(x_{ij}) = \frac{\partial^2 f(x)}{\partial x_i \partial x_j}$ is also an estimate of its symmetric counter part $\nabla^2 f(x_{ji})$ because of symmetry. By exploiting symmetry, it is possible to estimate the entire Hessian by evaluating ∇f and using the formula 2.8.

2.2.4 Automatic Differentiation

Automatic differentiation (AD) is a chain rule based technique used for computing derivatives of a function with respect to the given arguments without incurring truncation error [19]. According to chain rule, if f is a function of the vector $y \in \mathbb{R}^m$, which in turn is a function of the vector $x \in \mathbb{R}^n$, we can write the derivative of f with respect to x as follows:

$$\nabla_x f(y(x)) = \sum_{i=1}^{m} \frac{\partial f}{\partial y_i} \nabla y_i(x) \tag{2.9}$$

As an example, consider the function

$$f = x_1^2 \sin(x_3) + e^{x_1 x_2} \tag{2.10}$$

The evaluation of this function can be broken down into its elementary operations. Figure 2.1 shows the ordering associated with these operations. For instance, the multiplication $x_4 * x_5$ can follow the multiplication of $x_1 * x_1$.

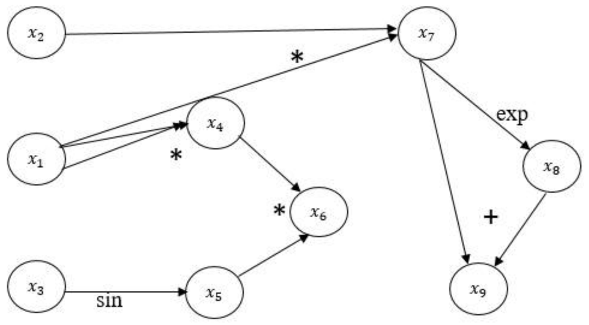

Figure 2.1: Computational Graph

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

$$x_4 = x_1 * x_1,$$

$$x_5 = sin x_3,$$

$$x_6 = x_4 * x_5$$

$$x_7 = x_1 * x_2$$

$$x_8 = e^{x_7}$$

$$x_9 = x_7 + x_8$$

Here, $x_4, x_5, \ldots$ distinguished from the *independent variables* x_1, x_2, x_3, contain the results of intermediate computations. The final node x_9 contains the function value $f(x)$. For an directed arc from i to j, node j is the *child* of node i and node i is the *parent* of node j. Any node can be evaluated when the values of all its parents are known, so computation flows through the graph from left to right. 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, x_2 and x_3. There are two basic modes of automatic differentiation: the *forward* and *reverse* modes.

2.2.5 Forward Mode

In forward mode, intermediate partial derivatives are accumulated in the same order as the function values are computed. By the chain rule, the gradient of each intermediate variables $x_k, k = 4, 5, \ldots, 9$ of the function f above with respect to three independent variables x_1, x_2, x_3 can be written as

$$\nabla x_k = \sum_{j=1}^{3} \frac{\partial x_k}{\partial x_j} \Phi_k \tag{2.11}$$

where $x_k = \Phi_k(x_1, x_2, x_3)$. We can find the corresponding value of ∇x_k as soon as the value of x_k at any node is known. For instance, to calculate ∇x_7, using the chain rule we can write

$$\nabla x_7 = \tfrac{\partial x_7}{\partial x_1} \nabla x_1 + \tfrac{\partial x_7}{\partial x_2} \nabla x_2 = x_2 \nabla x_1 + x_1 \nabla x_2$$

assuming the values of $x_1, \nabla x_1, x_2$ and ∇x_2 are known. Applying 2.11, we obtain

$$\nabla x_7 = \frac{\partial x_7}{\partial x_1} \left(\sum_{j=1}^{3} \frac{\partial x_1}{\partial x_j} \Phi_1 \right) + \frac{\partial x_7}{\partial x_2} \left(\sum_{j=1}^{3} \frac{\partial x_2}{\partial x_j} \Phi_2 \right) \tag{2.12}$$

As the derivatives are computed and combined with other derivatives via the chain rule in the evaluation steps of forward mode the computational complexity of one pass is proportional to the complexity of the original code.

2.2.6 Reverse Mode

The reverse mode recovers the partial derivatives of f with respect to each independent variable x_i by performing a *reverse sweep* of the computational graph after the evaluation of f is complete. The gradient vector ∇f can be assembled from the partial derivatives at the end of this process. For any node i, the partial derivative $\frac{\partial f}{\partial x_i}$ can be built up from the partial derivatives $\frac{\partial f}{\partial x_j}$ corresponding to its child nodes j according to the following formula:

$$\frac{\partial f}{\partial x_i} = \sum_{\text{j a child of i}} \frac{\partial f}{\partial x_j} \frac{\partial x_j}{\partial x_i} \tag{2.13}$$

Once $\frac{\partial f}{\partial x_i}$ is known for a child node, it is ready to contribute a term to the summation for each of its parent nodes according to 2.13. The process continues in this way until all nodes are known.The accumulation of adjoint derivatives are computed in reverse order as the function values are computed.

Forward mode is more efficient than reverse mode for functions $f : \mathbb{R}^n \to \mathbb{R}^m$ with m much larger than n as only n passes are necessary, where reverse accumulation is more efficient than forward accumulation with n much larger than m as only m passes are necessary.

The Newton's method algorithm of section 2.1 requires the evaluation of the second derivative matrix of the function and solution of the Newton equations for the Newton step s_k at each iteration. The jth column of the Hessian $\nabla^2 f(x)$ can be approximated using the formula (2.8), by setting the vector $p = \varepsilon e_j, j = 1, 2, \ldots, n$. But then, each evaluation of the Hessian will require n gradient evaluations. This can be computationally prohibitive if n is large and the function is highly nonlinear (e.g. containing components that are transcendentals). Additionally, as n grows large, the method becomes impractical from data storage viewpoint. However, many large problems have sparse Hessians. In such cases, only the nonzero entries are explicitly stored and it is possible to significantly reduce the number of gradient evaluations by exploiting known sparsity and the symmetry of the Hessian.

2.3 Efficient Determination of Sparse and Structured Derivatives

In this section we review methods that exploit sparsity and/or symmetry for efficient determination of Hessian matrices. The central idea shared by these methods is the notion of structural orthogonality and the combinatorial problem of grouping of columns and/or rows.

Given the sparsity structure of a matrix $A \in \mathbb{R}^{m\times n}$, matrix partitioning problem is to obtain vectors $d_1, d_2, \ldots, d_p$ so that the elements of the given matrix A are uniquely determined from the products $Ad_1, Ad_2, \ldots, Ad_p$ with p as small as possible. This can be achieved by partitioning the columns into structurally orthogonal columns i.e. no two columns from the same group have non-zeros in the same row position. Then, the non-zeros in each group can be determined by one finite differencing or by the forward mode of automatic differentiation.

Curtis, Powell and Reid [13] were the first to address this problem. They noted that sparsity of the Jacobian matrices can be exploited if the matrix can be partitioned into groups such that columns in each group are structurally orthogonal to each other. The appropriate notion of structural independence and the corresponding partitioning problem depends on whether the derivative matrix to be computed is symmetric or unsymmetric, whether the evaluation scheme employed is direct or substitution-based, and whether a unidirectional (1d) partition or a bidirectional (2d) partition is used.

2.3.1 Direct or Indirect Partition

In direct determination method, each nonzero element of a derivative matrix A can be read-off from some row of a matrix-vector product or a vector-matrix product without any further arithmetic operation. Using the direct evaluation scheme, the partitioning problem can be posed as to find a $n \times p$ matrix S or $q \times m$ matrix W^T so that the product AS or W^TA can be used to define a diagonal system of equations where the unknowns are the nonzero entries of A and the right-hand side is the result obtained by p FD calculation or p passes of the forward mode of AD for matrix-vector product AS and q passes of the reverse mode of AD for vector-matrix product W^TA. The following example will demonstrate direct determination method. Let

$$A = \begin{bmatrix} a_{11} & a_{12} & 0 & a_{14} & 0 & 0 \\ 0 & 0 & a_{23} & a_{24} & a_{25} & 0 \\ a_{31} & 0 & 0 & a_{34} & 0 & 0 \\ 0 & a_{42} & 0 & 0 & a_{45} & 0 \\ 0 & a_{52} & 0 & a_{54} & 0 & a_{56} \\ 0 & 0 & a_{63} & 0 & 0 & a_{66} \\ a_{71} & 0 & a_{73} & 0 & 0 & 0 \end{bmatrix} \quad S = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

We can determine all the non-zeros of the matrix A by using only three matrix-vector products in column computation. On the other hand, four vector-matrix products will be required for a row computation if we consider

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

From this example we can see that column computation is more efficient than the row computation in terms of the number of AD passes required to completely determine the matrix. In a substitution method the unknown elements of the matrix A are determined by solving a triangular system of equations i.e. the ordering of the non-zeros of A is such that every nonzero is determined using formerly computed values. Substitution method usually require fewer number of function evaluation or AD passes but is subject to numerical instability [23].

Let us demonstrate this method with the help of an example from [23]. Let

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

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

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

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. The non-zeros of the other two rows of A can be found in the similar way. For this instance, direct method will require three AD passes, while allowing substitution as shown above, the nonzero entries can be determined with two AD passes.

Powell and Toint [34] have used substitution method for computing Hessian matrices based on acyclic coloring. Coleman and Moré [10] have showed that star coloring models the Hessian matrix determination problem via direct method. The acyclic coloring model has been used by Coleman and Cai [7] to represent the problem in a substitution-method. Hossain and Steihaug [23] have applied a variant of substitution method for computing a Jacobian matrix which works in two steps. First, it finds a partition of the columns into groups of structurally orthogonal columns and then it merges the pair of successive groups to get a column grouping that allows overlaps.

2.3.2 Unidirectional or Bidirectional Partition

A partitioning scheme in which either the columns or the rows are partitioned into structurally orthogonal groups is known as *unidirectional partitioning*.

Matrix A can be partitioned into two column groups such that all the non-zeros of A can be obtained from the product AS as shown in Figure 2.2.

$$A = \begin{bmatrix} a_{11} & 0 & 0 & 0 & 0 \\ a_{21} & a_{22} & 0 & 0 & 0 \\ a_{31} & 0 & a_{33} & 0 & 0 \\ a_{41} & 0 & 0 & a_{44} & 0 \\ a_{51} & 0 & 0 & 0 & a_{55} \end{bmatrix} \quad S = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \end{bmatrix}$$

Figure 2.2: Column Partitioning

In Figure 2.3, we see that by partitioning the matrix A into two row groups, we can obtain all the non-zeros of A from the product $W^T A$.

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ 0 & a_{22} & 0 & 0 & 0 \\ 0 & 0 & a_{33} & 0 & 0 \\ 0 & 0 & 0 & a_{44} & 0 \\ 0 & 0 & 0 & 0 & a_{55} \end{bmatrix} \quad W^T = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Figure 2.3: Row Partitioning

For a sparse matrix A, if seed matrices $S \in \mathbb{R}^{n \times p}$ and $W \in \mathbb{R}^{m \times q}$ can be obtained such that all the non-zeros of A can be determined uniquely from the products $B = AS$ and $C^T = W^T A$, then the resulting partitioning is known as *bidirectional partitioning*.

Considering Figure 2.4, we notice that unidirectional partitioning (either row or column) will require at least 5 groups. But if we determine row 1 and column 1 separately and collect the remaining non-zeros in one column(row) group then we require only 3 groups.

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

Figure 2.4: Bidirectional Partitioning

For Hessian matrices, there is no advantage in considering a bidirectional partition due to symmetry.

2.3.3 Symmetric or Nonsymmetric Partition

Powell and Toint [34] extended the approach of Curtis, Powell and Reid in 1979 for Hessian matrices. They showed that the number of function evaluations can be reduced further by exploiting symmetry in addition to exploiting sparsity using structurally orthogonal partitions in case of Hessian matrix estimation using finite differences.

Graph coloring have been used for efficient computation of Hessians. Mostly used models are *star* and *acyclic* coloring. In a star coloring, every pair of adjacent vertices receives distinct colors (a distance-1 coloring), and every path of four vertices uses at least three colors. In an acyclic coloring, every pair of adjacent vertices receives distinct colors, and every cycle uses at least three colors. Let

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{21} & a_{22} & 0 & 0 & 0 \\ a_{31} & 0 & a_{33} & 0 & 0 \\ a_{41} & 0 & 0 & a_{44} & 0 \\ a_{51} & 0 & 0 & 0 & a_{55} \end{bmatrix} \quad S = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \end{bmatrix}$$

With the knowledge of sparsity and exploiting symmetry, the non-zeros of A can be read off from B.

$$B = \begin{bmatrix} a_{11} & a_{12} + a_{13} + a_{14} + a_{15} \\ a_{21} & a_{22} \\ a_{31} & a_{33} \\ a_{41} & a_{44} \\ a_{51} & a_{55} \end{bmatrix}$$

Chapter 3

Graph Models for Derivative Matrix Determination

In this chapter we give graph notations followed by the methods used to represent a matrix partitioning problem into graph coloring problems. We also discuss various graph models and the pattern graph model we used in this thesis.

3.1 Graph Terminologies

A *graph* G is an ordered pair (V,E), where V is a finite and nonempty set of *vertices* and E is a set of *edges*. In an *undirected* graph, an edge is an unordered pair of vertices whereas for a *directed* graph the pair of vertices is ordered, i.e. an edge (u,v) corresponds to the arc from u to v. A graph allowing multiple edges between two vertices is called *multigraph*. Self-loop or *loop* is an edge between a vertex and itself. An undirected graph is known as a *simple* graph if it contains no multiple edges or loops. We have considered simple undirected graphs throughout this thesis.

Two vertices u and v are *adjacent* if $(u,v) \in E$, otherwise they are called *nonadjacent*. If vertex u is one of edge e's endpoints, u is incident to e. The *degree* of a vertex v, denoted by $deg(v)$ is the number of edges incident on it.

A *path* of length l is a sequence of distinct vertices $v_1, v_2, \ldots, v_{l+1}$ in G such that v_i is adjacent to v_{i+1}, for $1 \le i \le l$. If the shortest path length between two distinct vertices is at most k then they are said to be *distance-k* neighbors.

A *subgraph* $G' = (V', E')$ of a graph G is a graph for which $V' \subseteq V$ and $E' \subseteq E$. For any set of vertices $V' \subseteq V$, the graph *induced* by V' is the subgraph of G whose vertex set is V' and whose edges are the edges of G with both endpoints in V'. The graph induced by V' is denoted by $G[V']$.

3.2 Graph Coloring

Graph coloring is an assignment of colors or labels to the vertices of the graph such that no two adjacent vertices receive the same color.

A *p-coloring* of a graph G 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 requires only $\chi(G)$ colors to color all the vertices of G is known as *optimal coloring*.

A *Multi-coloring* is an assignment where each vertex can be assigned with more than one color. A distance-k coloring of a graph is a *p-coloring* whenever u and v are distance-k neighbors. Lin and Skiena [28] proved that the distance-k coloring problem is NP-hard for every $k \geq 1$.

Both the star and the acyclic coloring has been proved to be NP-hard by Coleman and Moré [10] and Coleman and Cai [7] respectively. Johnson [26] showed that the approximation ratio of the first approximation algorithm for distance-1 coloring is $O(n\frac{1}{logn})$ where n is the number of vertices. According to [3], distance-1 coloring can not be approximated within $O(n^{1/7-\varepsilon})$ for any $\varepsilon > 0$, unless $P = NP$.

3.2.1 Graph Coloring Methods

The vertex coloring problem (VCP) is known to belong to the class of problems called NP-hard problems [31]. The VCP problem arises in numerous practical applications and the problem instances are large such that optimal coloring is impractical. On the other hand, VCP and its many variants are being actively and vigorously researched for their theoretical properties and their real-life applications.

There are heuristic techniques as well as exact methods to color the vertices of the graph. We have used heuristic approach to solve the partitioning problem as it yields a result in polynomial time. We also want to measure how our heuristic based algorithm differs from the exact coloring techniques. The following sections will provide a short description of both the techniques.

Exact Methods

Algorithms that give optimal solution for the problem are called *exact* methods. No other algorithm can guarantee better solution than this. Exact methods are hard and often not solvable in polynomial time. For example, the algorithm developed by Brélaz [4] works by subdividing the problem into subproblems. Each subproblem corresponds to a partial coloring of the graph. Depending on the number of colors used in the partial coloring and the upper bound on the needed number of colors subproblems are updated. Mehrotra and Trick [30] proposed a Branch and Price algorithm based on the VCP-Set Covering formulation (VCP-SC). According to VCP-SC, VCP can be formulated through the following model:

$$min \sum_{s \in S} x_s, \tag{3.1}$$

$$\sum_{s \in S: i \in s} x_s \geq 1 \quad i \in V, \tag{3.2}$$

$$x_s \in \{0,1\} \quad s \in S. \tag{3.3}$$

where a binary variable x_s is associated with each independent set (column) $s \in S$ having value 1 iff the vertices of s receive the same color. The objective function 3.1 minimizes the total number of independent sets (number of colors) used. Constraint 3.2 state that every vertex i in the graph must belong to at least one independent set (i.e., must receive at least one color) where constraint 3.3 impose variables x_s to be binary.

Heuristic Methods

Algorithms which give solution more quickly than classical methods but do not guarantee an optimal solution of the problem are called *heuristic* or *inexact* methods.The performance measurement for these methods is usually done by benchmarking i.e. measuring the quality of performance on different sets of inputs. The weakness of this performance measuring is that it is difficult to predict the results of arbitrary sets of inputs. The first heuristic approaches were mostly based on greedy algorithms. Greedy algorithm applies some technique to choose the next vertex to color and the color to use while coloring the vertices sequentially of the graph. For instance, in the greedy version of DSATUR algorithm by Brélaz [4], vertex v is chosen for which $|d_{color}(v)|$ is maximum and colored with the first available color at each iteration having the initial graph uncolored. Here, $d_{color}(v)$ for an uncolored vertex is defined as $d_{color}(v) = \{\Phi(v')|(v,v') \in E$ and v' is colored$\}$. *Recursive Largest First* algorithm proposed by Leighton [27] colors the vertices, one class at a time, using a greedy approach. Culberson and Luo proposed the *Iterated Greedy Algorithm* [12]. It iteratively colors the graph by means of the sequential algorithm, in such a way that at each iteration the number of used colors is not increased.
Surveys by Galinier and Hertz [14] and by Chiarandini et al. [6] review the algorithms for the VCP based on local search. Local search requires a set of candidate solutions corresponding to partial colorings or complete colorings, a neighborhood relation and solution evaluating function. Local search algorithms for the VCP can be partitioned in three families: fixed k-complete colorings, fixed k-partial colorings and variable k-complete colorings [6].

3.3 Graph Formulation of Matrix Partitioning Problem

3.3.1 Graph Models for Matrix Partitioning Problem

Graphs are used to model and solve many combinatorial problems. It is very crucial to choose a graph type which can accurately represent the underlying problem and enable the efficient design and computer implementation of relevant algorithms. Several graph models for partitioning computational tasks in a parallel computing environment are considered in [5]. There may arise some problem while using standard graph model for partitioning problem like optimization of an inaccurate cost metric and missing information e.g., unsymmetric dependencies. These can be circumvented by employing more versatile models such as hypergraphs and bipartite graphs [5]. We will review different graph representations of the sparsity structure of matrices and the motivation towards our choice of graph model for the Hessian matrix determination problem in the following subsections.

3.3.2 Adjacency Graph

The adjacency graph of A is $G(A) = (V,E)$ where there is a vertex $v_j \in V$ for each column $j = 1,2,\ldots,n$ of $A(:,j)$ and an edge $(v_i,v_j) \in E$ corresponds to each nonzero a_{ij} in A for $1 \le i \le n, 1 \le j \le n, i \ne j$. Graph $G(A)$ exploits the symmetry in A as both nonzero entries a_{ij} and a_{ji} are represented by a single edge (v_i,v_j). Figure 3.1 shows a symmetric matrix and the associated adjacency graph in Figure 3.2.

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{21} & a_{22} & . & . & . \\ a_{31} & . & a_{33} & . & . \\ a_{41} & . & . & a_{44} & . \\ a_{51} & . & . & . & a_{55} \end{bmatrix}$$

Figure 3.1: Sparse Symmetric Matrix

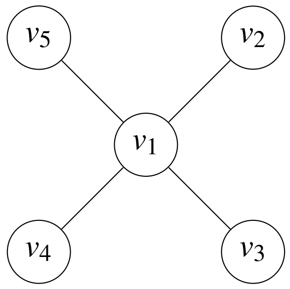

Figure 3.2: Adjacency Graph

3.3.3 Bipartite Graph

The bipartite graph of matrix A denoted by $G_b(A) = (V_c \cup V_r, E)$ where there is a column vertex $v_j \in V_c$ for each column $j = 1, 2, \ldots, n$ of $A(:, j)$ and a row vertex $v_i \in V_r$ for each row $i = 1, 2, \ldots, m$ of $A(i, :)$;$(v_i, v_j) \in E$ if and only if $a_{ij} \neq 0$. For example, Figure 3.3 shows the bipartite graph representation of the matrix from 3.1.

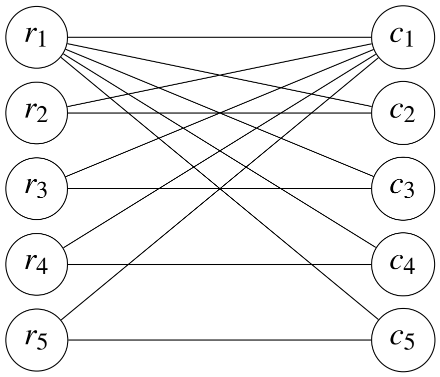

Figure 3.3: Bipartite Graph

3.3.4 Column Intersection Graph

The column intersection graph of matrix A denoted by $G_I(A) = (V, E)$ is where there is a vertex $v_j \in V$ for each column $j = 1, 2, \ldots, n$ of $A(:, j)$ and $(v_j, v_l) \in E, j \neq l$ if and only if $a_{ij} \neq 0$ and $a_{il} \neq 0$ for some index i. For example, Figure 3.4 shows the column intersection graph with the corresponding matrix at Figure 3.1.

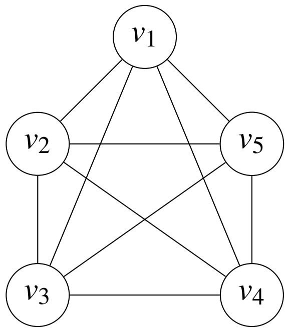

Figure 3.4: Column Intersection Graph

3.3.5 Pattern Graph

A lateral neighbor of v_{ij} is $v_{ij'}$ such that the difference $j' - j$ is the smallest if $j' > j$ or such that the difference $j - j'$ is the smallest if $j > j'$ among all such indices j' in row i. A vertical neighbor of v_{ij} is its lateral neighbor in A^T. The pattern graph associated with matrix A is $G_{\mathcal{P}}(A) = (V, E)$, where $V = \{v_{ij} \mid a_{ij} \neq 0, i = 1, 2, \ldots, m, j = 1, 2, \ldots, n\}$ and $\{v_{ij}, v_{i'j'}\} \in E$ if a_{ij} and$a_{i'j'}$ are lateral or vertical neighbors.
Figure 3.5 shows the pattern graph representation of the above mentioned symmetric matrix.

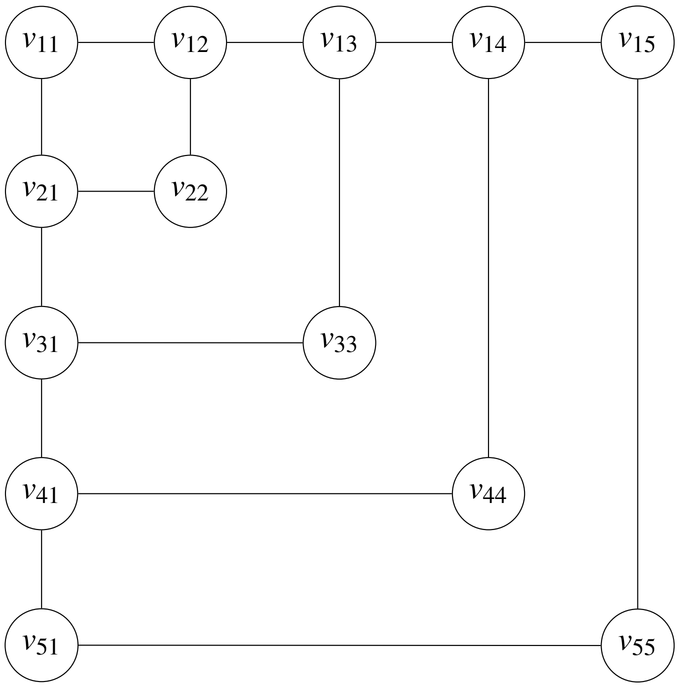

Figure 3.5: Pattern Graph

3.4 Comparative Study of Graph Models for Matrix Partitioning Problem

3.4.1 Graph Models

Coleman and More [10] modeled the matrix partitioning problem as a distance-1 graph coloring problem. They used column intersection graph to represent the matrix. For the

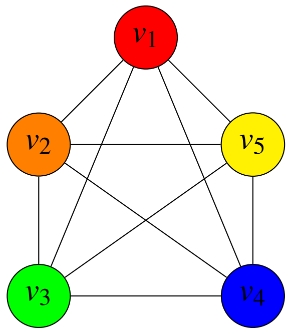

Figure 3.6: Distance-1 coloring of Column Intersection Graph

column intersection graph of (3.4) the distance-1 coloring will require 5 colors as shown in Figure 3.6. In their proposed algorithm, they ordered the vertices of the column intersection graph using some ordering heuristics and then applied the algorithm on these ordered vertices [8].

In 1983, McCormick [29] introduced a distance-2 graph coloring model. He used adjacency graph representation of the underlying symmetric matrix which requires that in every path of three vertices in the graph, all of them will receive distinct colors. Figure 3.7 shows

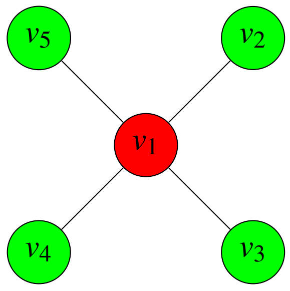

Figure 3.7: Graph coloring of Adjacency Graph

the graph coloring of the adjacency graph from Figure 3.2.

Hossain and Steighaug [22], Coleman and Verma [11] independently proposed bipartite graph model. Coleman and Verma suggested that it is sufficient to partition subsets of rows and columns such that A is determined directly. The vertices not involved in the determination of non-zeros are assigned the *neutral color zero*. Hossain and Steihaug [22] have used bipartite model to determine Jacobian matrices with two-sided compression or bidirectional determination. For both unidirectional and bidirectional partitioning, bipartite graph model has been used in [17]. Figure (3.8) shows the distance-2 coloring of (3.3). It is called

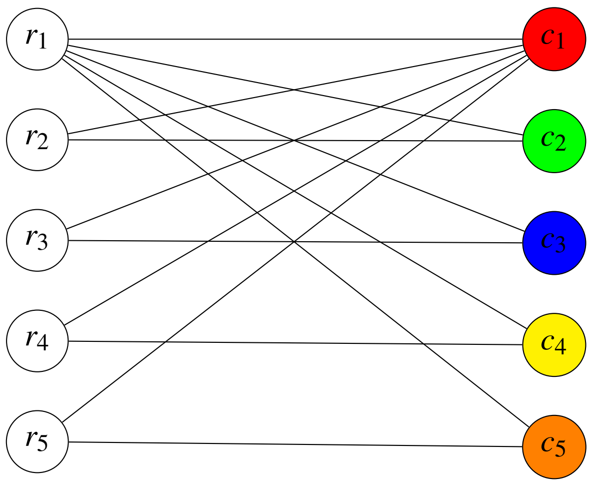

Figure 3.8: Distance-2 coloring of Bipartite Graph

partial distance-2 coloring since the row vertex set is left uncolored [17]. Hossain and Steihaug [24] have introduced pattern graph model to determine Jacobian matrices. They have suggested the pattern graph as a unifying framework for methods that exploit sparsity by matrix compression: unidirectional, bidirectional, or a combination of the two. We extend the pattern graph model to incorporate symmetry. The graph coloring of pattern graph associated with a symmetric matrix will be discussed in next chapter.
In sections 3.4.2 through 3.4.5, we elaborate these graph models in terms of size, representation, implementation and versatility.

3.4.2 Graph size

The size of the graph is an important metric to consider while designing the data structure for making the matrix storage space efficient. It depends on the sparsity structure of the given matrix A. For adjacency graph, $|V| = n$, where n is the number of vertices in A and $|E| = \frac{1}{2}(nnz(A) - n)$ with nnz as the number of non-zeros while for bipartite graph $G_b(A)$ we have $|V| = m + n$ and $|E| = nnz$. Here, m is the number of row vertices and n is the number of column vertices in $G_b(A)$. Let ρ_i be the number of nonzero element in row i. Then the size of the column intersection graph can be written as, $|V| = n$, $|E| = O\left(\sum_{i=1}^{n} \rho_i^2\right)$. For pattern graph $|V| = nnz$ and $|E| = 2nnz - m - n$.

3.4.3 Representation

An important consideration in choosing a graph for sparse matrix determination problem is to preserve the structure and sparsity of the matrix. While transforming from graph into matrix representations, loss of not-so-obvious structural information in the original matrix may result indiscrepancies in the solution. As an example, column intersection graph of a sparse arrow-head matrix contains all possible edges. Arrow-head matrix is the matrix where except for the main diagonal, *n*th column and *n*th row, all entries all identically zero. Adjacency graph can only be used for symmetric matrices. Bipartite and pattern graph are equivalent in representation of the matrix regardless of symmetry.

3.4.4 Implementation

The efficiency of graph algorithms is critically dependent on the data structures used to represent the graphs. Adjacency list and adjacency matrix are two graph data structures that are commonly considered in textbook presentation of graph algorithms [35]. More specifically, for sparse graphs, adjacency list representation of graphs are implemented using pointer-based linked lists. Many graph operations are characterized by irregular access to data where data movement through deep memory hierarchy dominates the running time. The prevalence of multi-core architectures in the recent years with nonuniform memory access (NUMA) have rendered the data access optimization efforts even more challenging. An exciting and promising line of research that has been considered in the recent years is to express and implement basic graph operations with sparse linear algebra [18, 20, 24]. The pattern graph model makes the connection between sparse derivative matrix determination and its associated graph problem more explicit such that the underlying computation can be made structured and cache-friendly [21].

A data structure consisting two arrays: *colind* and *rowptr* seems an efficient one for matrix representation where *colind* stores the column indices of the nonzero entries row-by-row and the later contains the index of the first nonzero element of each row of matrix A stored in *colind* array. Except this two, sparsity pattern of A^T is explicitly stored in analogous arrays *rowind* and *colptr*. The storage meets the design strategies for sparse linear algebra implementation [18]. In the column intersection graph distance-1 neighbors $v_{j'}$ of vertex v_j where $\Phi(v_{j'}) \neq \Phi(v_j)$ are computed as $j' = colind(ind_j)$ where $ind_j = rowptr(i) : rowptr(i+1)-1, i = rowind(ind_i), ind_i = colptr(j) : colptr(j+1)-1$. For bipartite graph, distance-2 neighbors $v_{j'}$ of vertex v_j are checked assuming j a column index. For the pattern graph, paths of our interest are $v_{ij} \sim v_{ij'}, j \neq j'$, $v_{ij} \sim v_{ij'} \sim v_{i'j'}, i \neq i', j \neq j'$ and $v_{ij} \sim v_{i'j} \sim v_{i'j'}, i \neq i', j \neq j'$ for v_{ij} corresponding to the nonzero entry a_{ij} where '$\sim$' representing a path between two vertices in $G_{\mathcal{P}}$..

3.4.5 Versatility

Column intersection graph has been used for unidirectional partitioning of sparse Jacobian matrix determination in [9, 17, 25, 32]. For bidirectional partitioning this graph model requires the concept of neutral color while for bipartite graph model, one set of vertices (column or row vertices) are left out [17]. Hossain and Steihaug [24] have showed that pattern graph can be used for both unidirectional and bidirectional Jacobian matrix partitioning problems.

In this thesis we are concerned with the Hessian sparse matrix determination with the knowledge of sparsity. We have studied the available graph models for matrix determination problems and chosen the pattern graph to represent our problem after careful consideration of each of the criteria mentioned above. Adjacency graph is only applicable to symmetric matrices with nonzero diagonal. Though for the bipartite graph and the pattern graph the size of the graph is proportional to the number of nonzero elements in sparse matrix A, bipartite graph contains extraneous information in unidirectional partitioning of the given matrix. The main disadvantage of column intersection graph is that while translating the matrix problem to the graph problem due to connecting edge between each pair of distinct vertices sparsity structure of the matrix is lost. Also to apply ordering and partitioning algorithms, explicit construction of $G_I(A)$ will need $\Theta(n^2)$ effort and $\Theta(n^2)$ space to store the graph [24]. Pattern graph is structurally close to the actual computer representation of the associated sparse matrix. Both unidirectional [13] and bidirectional [11, 22] as well as column-segments [25, 32] computation of the graph operations can be expressed in an uniform manner in pattern graph representation. The details of this can be found in [24]. It also allows to express the computational cost of graph operations in terms of nonzero unknowns to be determined. Thus, pattern graph representation of the input sparse matrix enables the same asymptotic computational cost as indicated by its graph abstraction. Note that one of the main objectives in this thesis is to extend the pattern graph model to enable symmetric determination.

Chapter 4

Determination of Hessian Matrix

In this chapter we give the problem definition in a formal way, describe our main result connecting symmetric direct cover and coloring, and a new algorithm for sparse Hessian matrix determination.

4.1 Problem Definition

4.1.1 Compression-Reconstruction for matrix determination

The general problem of sparse derivative matrix determination problem can be viewed as a compression - reconstruction process where compression is involved with minimizing the number of matrix-vector products p and the reconstruction phase is responsible for solving for the non-zero unknowns while maintaining numerical stability and efficiency. Without loss of generality, we assume that $\rho_i \leq p$ for all i. The compression-reconstruction framework consists of two main algorithmic stages:

1. **Seeding or Compression**. Obtain $S \in \mathbb{R}^{n \times p}$ and compute $B = AS$ using FD or forward AD.

2. **Harvesting or Reconstruction**. Determine the nonzero elements of A row-by-row:

   a. Identify the reduced seed matrix $S_i \in \mathbb{R}^{\rho_i \times p}$ for $A(i, \mathcal{J}_i)$ where $\mathcal{J}_i$ denotes a vector containing the column indices of the nonzero entries in row i of A,

$$S_i = S(\mathcal{J}_i, :).$$

b. Solve for the ρ_i unknown elements of $A(i,:)$,

$$A(i,\mathcal{J}_i)S_i = B(i,:).$$

For Hessian matrix determination problem, if a nonzero entry a_{ij} of A is unknown then so is a_{ji} due to symmetry such that any determination method needs to determine only one of a_{ij} and a_{ji}. A symmetric determination method is called *direct determination* if there is a seed matrix $S \in \{0,1\}^{n\times p}$ such that each unknown a_{ij} is determined *directly* i.e., there is an index k such that $a_{ij} = b_{ik}$ or $a_{ji} = b_{jk}$ in the matrix equation $AS = B$ in which the compressed matrix B is computed via AD or FD. A direct determination in which the number of columns p of S is minimum is said to be *optimal*.

4.1.2 Symmetric Direct Cover

We consider the matrix equation

$$AS = B$$

Let $\mathcal{J}_i$ and I_j, respectively, be vectors containing the column indices of the nonzero entries in row i and row j of matrix A. Also let $\widehat{S_i} \equiv S(\mathcal{J}_i,:)$ and $\widehat{S_j} \equiv S(I_j,:)$ denote the submatrix of matrix S associated with the nonzero entries in row i and row j of matrix A respectively.

Definition 4.1. An unknown a_{ij} is said to be *covered* by the seed matrix S if a_{ij} or a_{ji} can be uniquely solved in

$$A(i,\mathcal{J}_i)\widehat{S_i} = B(i,:) \text{ or } A(j,\mathcal{J}_j)\widehat{S_j} = B(j,:) \text{ respectively}$$

Definition 4.2. Matrix S is said to constitute a *cover* for A if each $a_{ij} = a_{ji} \neq 0$ is covered. A cover is a *direct cover* if A can be determined directly from the cover.

Consider the pattern graph $G_{\mathcal{P}}$ associated with matrix A where $A = A^T$. To determine the symmetric direct cover we have defined the neighbor of a vertex v_{ij} of $G_{\mathcal{P}}$ in a different way.

Definition 4.3. The neighbors of v_{ij} consist of $\{v_{ij'}|v_{ij} \sim v_{ij'} j \neq j'\} \cup \{v_{i'j'}|v_{ij} \sim v_{ij'} \sim v_{i'j'}, j \neq j', i \neq i'\} \cup \{v_{i'j'}|v_{ij} \sim v_{i'j} \sim v_{i'j'}, j \neq j', i \neq i'\}$.

Here '$\sim$' represents a path between two vertices in $G_{\mathcal{P}}$. So the degree of v_{ij} can be represented by $deg(v_{ij}) = \left|N(v_{ij})\right|, N(v_{ij})$ is the set of neighbors of v_{ij}.
Figure 4.2 shows the associated pattern graph of sparse Hessian matrix from Figure 4.1 pointing out all the neighbors of a blue circled vertex by red bordered circles.

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & . & . & a_{16} \\ a_{21} & a_{22} & a_{23} & a_{24} & . & . \\ a_{31} & a_{32} & a_{33} & . & a_{35} & . \\ . & a_{42} & . & a_{44} & a_{45} & a_{46} \\ . & . & a_{53} & a_{54} & a_{55} & a_{56} \\ a_{61} & . & . & a_{64} & a_{65} & a_{66} \end{bmatrix}$$

Figure 4.1: Sparse Hessian Matrix

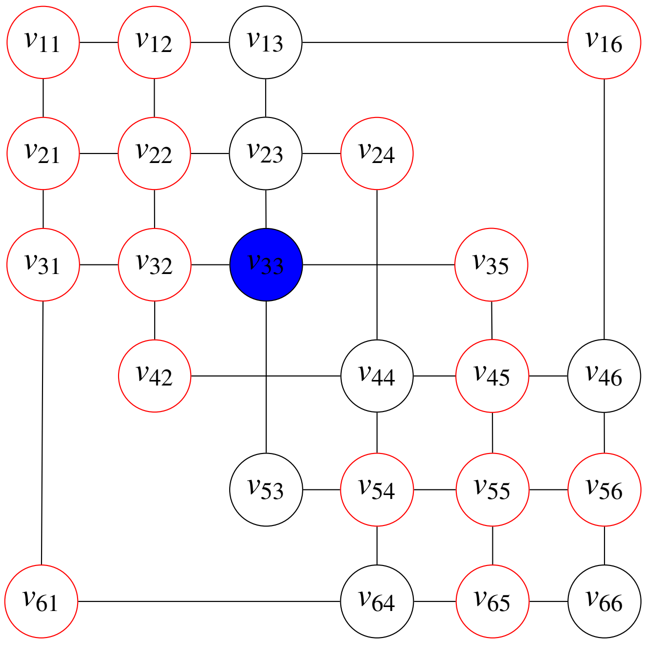

Figure 4.2: Pattern graph

Definition 4.4. Let $\mathscr{P}$ be a power set of $\{1, 2, \ldots, p\}$. Mapping $\Phi : V \mapsto \mathscr{P}$ is a generalized $p-$Multi-coloring (generalized p-MC) of pattern graph $G_{\mathcal{P}}(A)$ if for each vertex v_{ij} we have

EITHER

1.

$$\Phi(v_{ij}) \setminus \bigcup_{\{v_{ij'}|v_{ij}\sim v_{ij'}, j'\neq j\}} \Phi(v_{ij'}) \neq \emptyset$$

OR

2.

$$\Phi(v_{ji}) \setminus \bigcup_{\{v_{ji'}|v_{ji}\sim v_{ji'}, i'\neq i\}} \Phi(v_{ji'}) \neq \emptyset$$

Theorem 4.5. *Mapping $\Phi : V \mapsto \mathscr{P}(\{1,2,\dots,p\})$ yields a direct cover for the Hessian matrix A if and only if Φ is a generalized $p-$Multi-coloring of $G_{\mathcal{P}}(A)$.*

Proof. Let the mapping Φ yield a direct cover for A that does not satisfy conditions 1 and 2. Then for each $k \in \Phi(v_{ij})$ there is an index $j' \neq j$ such that $k \in \Phi(v_{ij'})$ and for each $k' \in \Phi(v_{ji})$ there is an index $i' \neq i$ such that $k' \in \Phi(v_{ji'})$. Then,

$$b_{ik} = (e_i^T A)(Se_k) = A(i,:)\left(\sum_{\{j''|k\in\Phi(v_{ij''})\}} e_{j''}\right) = \sum_{\{j''|k\in\Phi(v_{ij''})\}} a_{ij''} \neq a_{ij},$$

since the index set j'' contains at least two elements. Similarly, it can be shown that $b_{jk'} \neq a_{ji}$. Then, S does not constitute a direct cover implying that Φ does not yield a direct cover - a contradiction.

To establish the converse consider the seed matrix S defined by Φ. We want to show that for each unknown $a_{ij} \neq 0$, we have $a_{ij} = b_{ik}$ or $a_{ji} = b_{jk'}$ in $AS = B$. Since Φ must satisfy at least one of the two conditions above assume, without loss of generality, that there is an index $k \in 1,2,\dots,p$ such that $k \in \Phi(v_{ij}) \setminus \bigcup_{v_{ij'}|j'\neq j} \Phi(v_{ij'})$. We claim that the kth column of $\widehat{S}_i, \widehat{S}_i e_k$ is the jth coordinate vector e_j giving $b_{ik} = (e_i^T A)e_j = a_{ij}$. This is clearly the case since for $a_{ij'} \neq 0$, we have $k \notin \Phi(v_{ij'})$ implying that

$$\widehat{S}_i(l,k) = \begin{cases} 1 & \text{if l = j} \\ 0 & \text{otherwise.} \end{cases}$$

Therefore, $b_{ik} = (e_i^T A)e_j = a_{ij}$. $\square$

The associated adjacency graph of the symmetric matrix above is shown in Figure 4.3. Here, each pair of distinct vertices are connected by a path of length 2. It can be verified

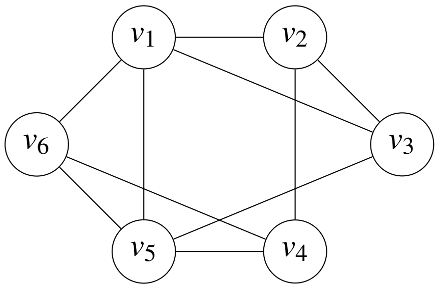

Figure 4.3: Adjacency Graph

that any star coloring will require more than 4 colors for this adjacency graph. For example, if the set of colors is $\{R,G,B,Y,O\}$, then the colors of the corresponding graph can be shown as Figure 4.4 based on the star coloring proposed in [34]. The star coloring algo-

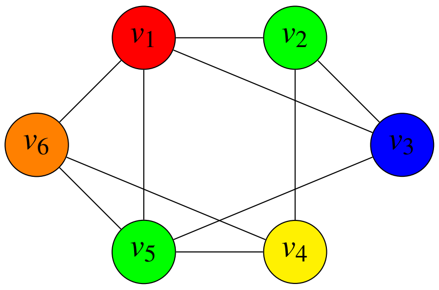

Figure 4.4: Star coloring of [34]

rithm from [17] also gives 5 colors as shown in Figure 4.5.

Now consider the associated pattern graph. If we use 4-Multi-Coloring of the graph as Figure 4.6 taking the colors $\{R,G,B,O\}$, then by Theorem 4.5 the associated matrix of the graph can be directly determined. In order to construct the seed matrix S, we use $\Phi(v_{ij}) \setminus \bigcup_{\{v_{ij'} | v_{ij} \sim v_{ij'}, j' \neq j\}} \Phi(v_{ij'})$ to represent the color of each vertex v_{ij}. For example, v_{24} can be represented by the color R as $\Phi(v_{24}) - \Phi(\{v_{21}, v_{22}, v_{23}\}) = \{R\}$. We can represent all the vertices using only one color in similar way shown in Figure 4.7. The identically colored vertices can be grouped together which contribute 1 in the associated rows of a

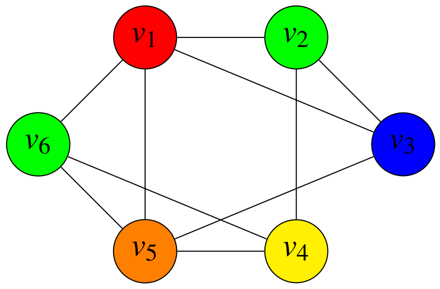

Figure 4.5: Star coloring of [17]

$\Phi(v_{11}) = \{R,B\}$	$\Phi(v_{42}) = \{O\}$
$\Phi(v_{12}) = \{O\}$	$\Phi(v_{44}) = \{R,B\}$
$\Phi(v_{13}) = \{B\}$	$\Phi(v_{45}) = \{R,G\}$
$\Phi(v_{16}) = \{R,O\}$	$\Phi(v_{46}) = \{R,O\}$
$\Phi(v_{21}) = \{G\}$	$\Phi(v_{53}) = \{B\}$
$\Phi(v_{22}) = \{O\}$	$\Phi(v_{54}) = \{R,B\}$
$\Phi(v_{23}) = \{B\}$	$\Phi(v_{55}) = \{R,G\}$
$\Phi(v_{24}) = \{R,B\}$	$\Phi(v_{56}) = \{R,O\}$
$\Phi(v_{31}) = \{R,B\}$	$\Phi(v_{61}) = \{R,B\}$
$\Phi(v_{32}) = \{O\}$	$\Phi(v_{64}) = \{R,B\}$
$\Phi(v_{33}) = \{B\}$	$\Phi(v_{65}) = \{R,G\}$
$\Phi(v_{35}) = \{R,G\}$	$\Phi(v_{66}) = \{R,O\}$

Figure 4.6: 4-Multi-Coloring of pattern graph

column and 0 otherwise in S. For instance, as v_{16}, v_{24} and v_{35} of column 4, 5 and 6 in A share the same color R, the corresponding rows 4, 5 and 6 will have 1 in a column group of S. The unknowns of such group can be determined with one matrix-vector product each giving 4 matrix-vector products. The uncolored vertices are known by symmetry. Thus the matrix A is completely determined with only 4 matrix vector products. The associated seed matrix has the structure of Figure 4.8.

This example demonstrates that using the pattern graph model, sparsity and symmetry can be exploited more effectively. Specifically, the definition of cover allows for Hessian matrices with the most general sparsity patterns including instances with sparse diagonal. We remark that the adjacency graph representation of a symmetric matrix sparsity pattern does

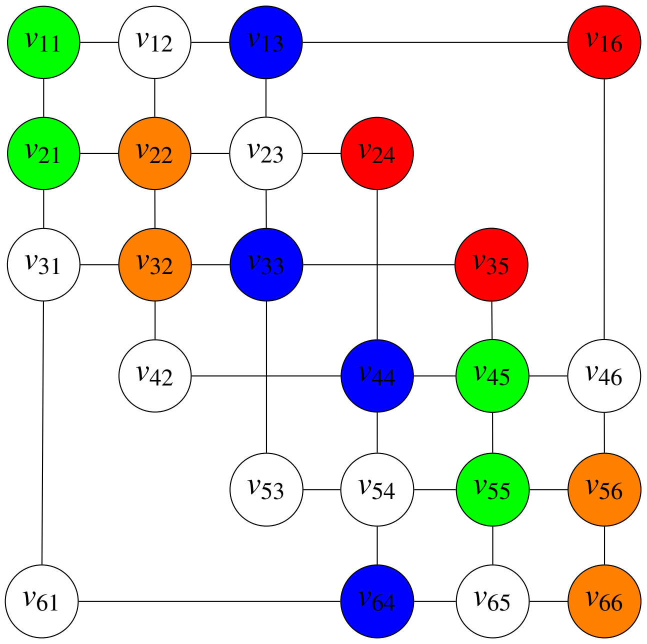

Figure 4.7: Pattern graph coloring

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

Figure 4.8: Seed Matrix

not define the sparsity of the diagonal elements.

4.2 Algorithm for Symmetric Direct Cover

In this section, we will describe our algorithm for determining the symmetric direct cover S for Hessian Matrix H. We have used the term *group* which consists of the column indices that can be grouped together to form a vector d of the seed matrix S. As we have relaxed *structurally orthogonal* property, a column may belong to more than one *group*. For example, there is no column that does not share non-zeros at same row in Figure 4.1. Thus structurally orthogonal partitioning of the matrix will require 6 matrix-vector product which we have reduced to 4 by relaxing this property. From Figure 4.8, we can see that the first group consists of column 4, 5 and 6.

Throughout the algorithm 1, we maintain three sets of information *forbiddenSet*, *fullCoveredSet* and *coveredSetInGroup*. The *forbiddenSet*($\mathcal{F}$) contains the unknown vertices (uncovered / undetermined nonzero element of the matrix) that cannot be included in the same group, *coveredSetInGroup*($\mathcal{D}$) is a set of vertices directly covered in each iteration of the outer **while** loop. Within the outer loop, we initialize $\mathcal{D}$ with no elements in it, construct the group and update *coveredSet*($\mathcal{C}$), set of all the covered vertices by combining the elements of $\mathcal{D}$ and $\mathcal{C}$ in line 16. $\overline{\mathcal{C}}$ denotes the complement of $\mathcal{C}$.

Algorithm 1: Symmetric Direct Cover determination of H

```
   Input  : Sparse Hessian $H$
   Output: number of groups $numgroup$, direct cover $C$
1  $numgroup \leftarrow 0$
2  $\mathcal{C} \leftarrow \emptyset$
3  while $\overline{\mathcal{C}} \neq \emptyset$ do
4      $\mathcal{D} \leftarrow \emptyset$
5      $\mathcal{F} \leftarrow \emptyset$
6      Let $\alpha \in \overline{\mathcal{C}}$ be an unknown such that $deg_{\overline{\mathcal{C}}}(\alpha)$ is maximum
7      $\mathcal{D} \leftarrow \mathcal{D} \cup \{\alpha\}$
8      $\mathcal{F} \leftarrow \mathcal{F} \cup (N(\alpha) \cap \overline{\mathcal{C}})$
9      while $\overline{\mathcal{C}} \setminus \mathcal{F} \neq \emptyset$ do
10         Let $\beta \in \overline{\mathcal{C}} \setminus \mathcal{D}$ be an unknown such that $deg_{\mathcal{F}}(\beta)$ is maximum
11         $\mathcal{D} \leftarrow \mathcal{D} \cup \{\beta\}$
12         $\mathcal{F} \leftarrow \mathcal{F} \cup (N(\beta) \cap (\overline{\mathcal{C}} \setminus \mathcal{D}))$
13     $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}^T$
14     $numgroup \leftarrow numgroup + 1$
15     Assign the elements of $\mathcal{D}$ group number $numgroup$
16     $\mathcal{C} \leftarrow \mathcal{C} \cup \mathcal{D}$
```

We choose the first uncovered vertex α with maximum degree (maximum number of neighbors in the associated pattern graph) to be in $\mathcal{D}$ in line 6. Then we update the *forbidenSet* such a way that for the next vertex to be chosen is anything but not a neighbor of α as well as covered. We proceed to add as many vertices as possible in $\mathcal{D}$ until there is none left. The choice of those vertices is based on an attribute *forbiddenCount*($deg_{\mathcal{F}}(\beta)$) : size of $\mathcal{F}$

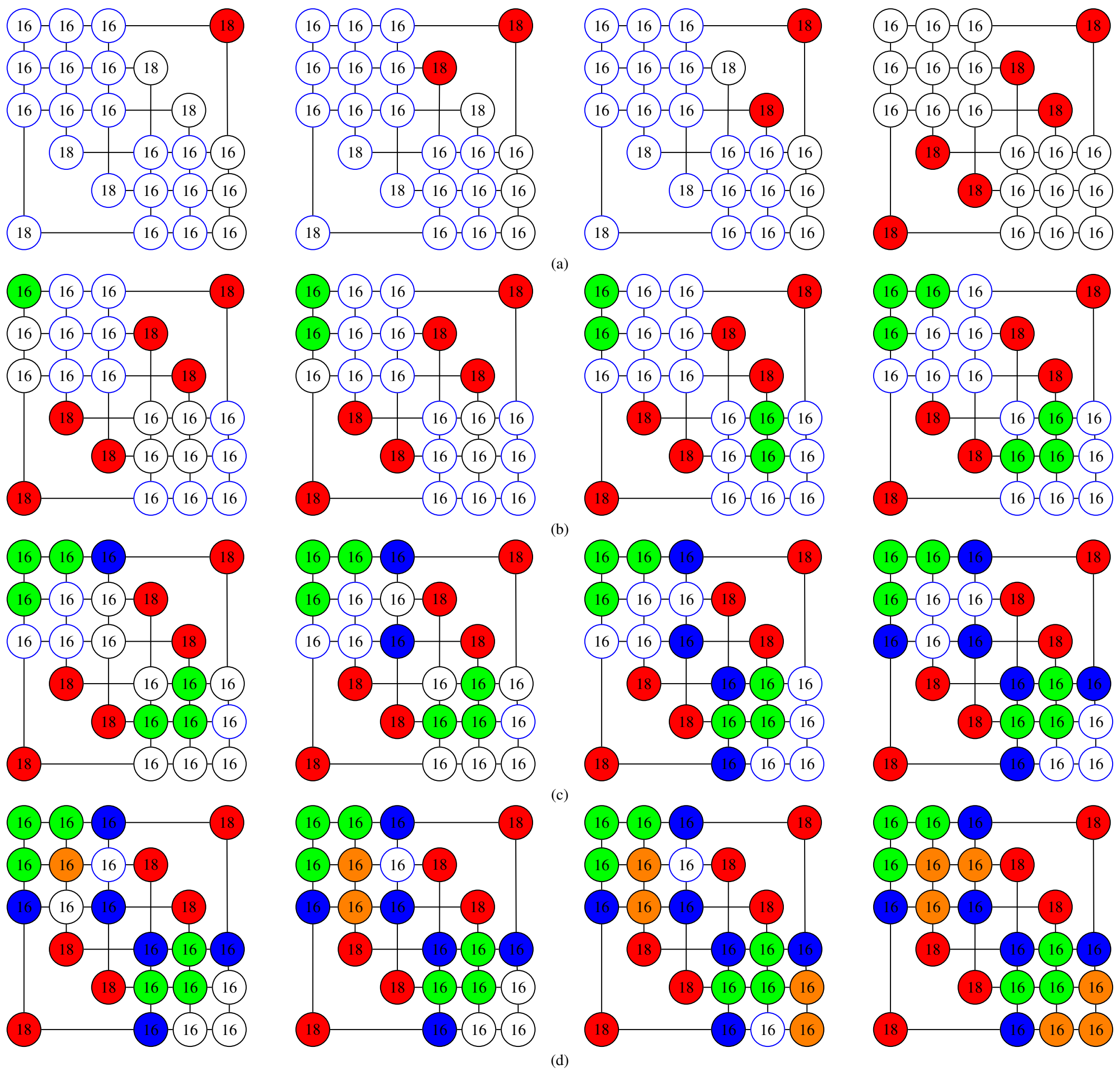

Figure 4.9: Visualizing steps of Algorithm 1

after adding β in it where $\beta \notin N(\alpha)$ and $\beta \notin \mathcal{D}$. We can write $deg_{\mathcal{F}}(\beta)$ as $|N(\beta) \cap \mathcal{F}|$. We select the one which contributes the maximum *forbiddenCount* in line 10. When there are no more unknowns left, we update $\mathcal{C}$ and again start doing these steps with empty $\mathcal{D}$. For calculating *forbiddenCount* we have used the algorithm 2 which returns the count for an unknown vertex v.

Algorithm 2: Find forbiddenCount of v

```
1 $count \leftarrow 0$ ;
2 for each element $i \in \mathcal{F}$ do
3     for each element $j \in N(v)$ do
4         if $i == j$ then
5             $count++$ ;
6 return $count$ ;
```

4.3 Data Structure

It is very important to choose a data structure which can take advantage of sparse structure for storing and manipulating sparse matrices on a computer. As a sparse matrix has many zeros which remain unused so using two-dimensional array to store the matrix will cause wastage of memory. Depending on the number and distribution of the nonzero entries, different data structures are used. We have used Compressed Storage as accessing each nonzero of the matrix becomes more faster and easier which is an important step in our algorithm to find neighbors as well as the *fobiddenCount*.

4.3.1 Compressed Storage

A compressed storage is proposed in [20] which represents a matrix by three one dimensional arrays, that contain nonzero values, the extents of rows, and column indices. It takes only $3 \times nnz(A) + m + n + 2$ memory locations. The total data structure is divided into two parts: Compressed Row Storage (CRS) and Compressed Column Storage (CCS).

Compressed Column Storage

In compressed column storage, a row index is stored for each value in array *row_ind*, and column pointers are stored in *col_ptr*. Row indices of each column j can be found in between $row\_ind[col\_ptr[j]]$ and $row\_ind[col\_ptr[j+1]-1]$. The following Figure 4.10 shows the CCS data structure of the matrix of Figure 4.1.

row_ind

1	2	3	6	1	2	3	4	1	2	3	5	2	4	5	6	3	4	5	6	1	4	5	6

col_ptr

1	5	9	13	17	21	25

Figure 4.10: CCS data structure of sparse matrix

Compressed Row Storage

Like Compressed Column Storage, CRS uses array *row_ptr* to represent starting index of each row and array *col_ind* to store column indices of each row. Column indices of each row i can be found in between $col\_ind[row\_ptr[i]]$ and $col\_ind[row\_ptr[i+1]-1]$. Figure (4.11) shows the CRS data structure of same matrix.

col_ind

1	2	3	6	1	2	3	4	1	2	3	5	2	4	5	6	3	4	5	6	1	4	5	6

row_ptr

1	5	9	13	17	21	25

Figure 4.11: CRS data structure of sparse matrix

Chapter 5

Numerical Experiments

In this chapter, we present numerical results of coloring algorithm proposed in this thesis on practical instances. First, we give details of test data sets and the environment on which we run our experiments. For the purpose of comparison we also include sequential and star coloring results.

5.1 Test Data Sets

We have considered multiple groups of matrices collected from Matrix Market Collection [2]. The name of the matrix is under the column labeled Matrix, the columns labeled *n* is used to represent the number of columns and rows (square matrix) and *nnz* is used to represent the total number of non-zeros in the matrix. Table 5.1 is the **CANNES** collection which are finite-element structure problems in aircraft design.

Table 5.1: Matrix Data Set - 1

Matrix	*n*	*nnz*
can___62	62	218
can__256	256	2916
can__268	268	3082
can__292	292	2540
can__634	634	7228
can__715	715	6665

Matrix	*n*	*nnz*
can_1054	1054	12196
can_1072	1072	12444

The non-zero structure of can__715 is shown in Figure 5.1.

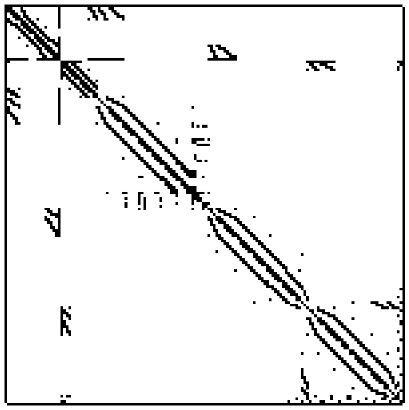

Figure 5.1: Sparse Matrix, Name:can__715, Dimensions: 715 $\times$ 715, 6665 nonzero elements shown in black, Source: [2]

Table 5.2 is on **DWT** consisting of 30 matrices collected from various US military and NASA users of NASA's structural engineering package NASTRAN for use as a benchmark collection for variable bandwidth reordering heuristics.

Table 5.2: Matrix Data Set - 2

Matrix	*n*	*nnz*
dwt___59	59	267
dwt___66	66	320
dwt___72	72	222
dwt___87	87	541
dwt__162	162	1182
dwt__193	193	3493

Matrix	*n*	*nnz*
dwt__198	198	1392
dwt__209	209	1743
dwt__221	221	1629
dwt__234	234	834
dwt__245	245	1461
dwt__307	307	2523
dwt__310	310	2448
dwt__346	346	3226
dwt__361	361	2953
dwt__419	419	3563
dwt__492	492	3156
dwt__503	503	6027
dwt__512	512	3502
dwt__592	592	5104
dwt__607	607	5131
dwt__758	758	5994
dwt__869	869	7285
dwt__878	878	7448
dwt__918	918	7384
dwt__992	992	16744
dwt_1005	1005	8621
dwt_1007	1007	8575
dwt_1242	1242	10426

Matrix	*n*	*nnz*
dwt_2680	2680	25026

Figure 5.2 shows the non-zero structure of dwt__419.

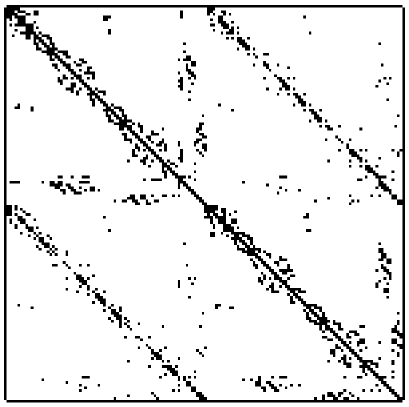

Figure 5.2: Sparse Matrix, Name:dwt__419, Dimensions: 419 × 419, 3563 nonzero elements shown in black, Source: [2]

LSHAPE: Matrices on finite-element model problems are shown in Table 5.3.

Table 5.3: Matrix Data Set - 3

Matrix	*n*	*nnz*
lshp_265	265	1009
lshp_406	406	1561
lshp_577	577	2233
lshp_778	778	3025
lshp1009	1009	3927
lshp1270	1270	4969
lshp1561	1561	6121
lshp1882	1882	7393
lshp2233	2233	8785

Matrix	*n*	*nnz*
lshp2614	2614	10297
lshp3025	3025	11929
lshp3466	3466	13681

The non-zero structure of all the sparse matrix of **LSHAPE** collection is quite similar. One of them is shown shown in Figure 5.3.

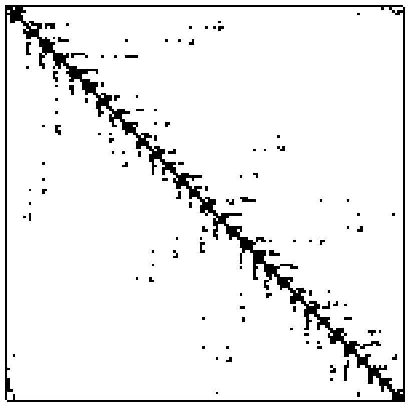

Figure 5.3: Sparse Matrix, Name:lshp1561, Dimensions: 1561 $\times$ 1561, 6121 nonzero elements shown in black, Source: [2]

Table 5.4 is the **JAGMESH** collection which are also finite-element model problems.

Table 5.4: Matrix Data Set - 4

Matrix	*n*	*nnz*
jagmesh1	936	3600
jagmesh2	1009	3837
jagmesh3	1089	4225
jagmesh4	1440	5472
jagmesh5	1180	4465
jagmesh6	1377	5185

Matrix	*n*	*nnz*
jagmesh7	1138	4294
jagmesh8	1141	4303
jagmesh9	1349	5225

Figure 5.4 shows the non-zero structure of a sparse matrix from **JAGMESH** collection.

5.2 Test Environment

Numerical Experiments with data sets in Table (5.1, 5.2, 5.3, 5.4) were done on an AMD PC with AMD Opteron(tm) Processor 4284, 16 GB RAM 64 bit Linux.

5.3 Test Results

We use C++ for the implementation. We compare our results with two software packages: DSJM [20] and ColPack [1].

DSJM represents the matrix using sparse data structure mentioned in section 4.3.1 of the previous chapter and applies sequential coloring algorithm to color the vertices in some chosen order which is more efficient. Sequential coloring is a greedy approach which assigns the smallest available (colors not been assigned to $N(v)$) color to the vertex v in each iteration. The ordering methods used in DSJM are: Largest-First Ordering (LFO), Smallest-Last Ordering (SLO), Incidence-Degree Ordering (IDO), Saturation-Degree Ordering (SDO), Recursive-Largest-First (RLF) and a hybrid approach MRLF-SLO based on RLF and SLO. In our comparison we consider RLF as it provides better result than the other for partitioning the columns into structurally orthogonal groups. In RLF, vertex set V is partitioned into $V_1, V_2, \ldots, V_p$ independent sets and a partition with p number of column groups is formed. A largest degree vertex is chosen from V_i in the induced graph, and adja-

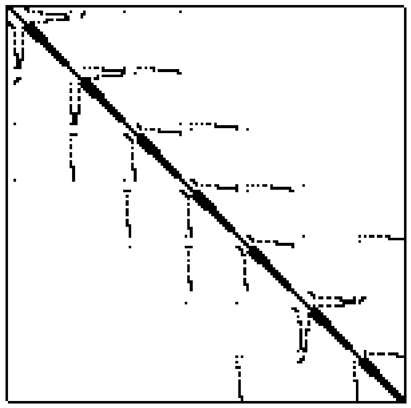

Figure 5.4: Sparse Matrix, Name:jagmesh9, Dimensions: 1349 $\times$ 1349, 5225 nonzero elements shown in black, Source: [2]

cent vertices of v_1 are added to the inadmissible set U. RLF proceeds to add vertices v_k in V_i which has the largest number of adjacent vertices in the set U keeping the neighbors of v_k in U. ColPack has implementations of various algorithms for graph coloring and recovery. Depending on the derivative matrix to be computed and the recovery scheme, it has several variations of the coloring models. For example, for direct Hessian recovery ColPack implements star coloring algorithm and acyclic coloring for the substitution recovery. We are only concerned about the direct recovery.

Table 5.5, 5.6, 5.7 and 5.8 illustrates the coloring results found in our proposed algorithm under the column Pattern Graph, DSJM and ColPack along with the maximum number of non-zeros in any row denoted by ρ_{max}. It is a lower bound on the number of groups in a structurally orthogonal partition. The results from our numerical testing are organized in two sections. In the first, we demonstrate the advantage of utilizing symmetry in addition to sparsity. In the second section, we compare our algorithm for pattern graph multi-coloring with ColPack star coloring.

Table 5.5: Graph coloring for data set 1

Matrix	ρ_{max}	**Pattern Graph**	DSJM
can___62	7	5	7

Matrix	ρ_{max}	**Pattern Graph**	DSJM
can__256	83	20	83
can__268	37	18	37
can__292	35	13	35
can__634	28	20	28
can__715	105	14	105
can_1054	35	20	35
can_1072	35	22	35

Table 5.5 displays the coloring results for CANNES problems. We compare the results of DSJM and Algorithm 1 of this thesis. DSJM ignores symmetry while determining the number of colors i.e. number of groups required to directly determine the input matrix. From the Table it is evident that exploiting symmetry combined with sparsity gives better coloring than when symmetry is ignored. In particular, on can__715, exploiting symmetry achieves more than 7-fold decrease in the number of matrix-vector products. On all the test sets exploiting symmetry always showed significant improvement in the number of matrix-vector products.

Table 5.6: Graph coloring for data set 2

Matrix	ρ_{max}	**Pattern Graph**	ColPack
dwt___59	6	6	6
dwt___66	6	5	5
dwt___72	5	5	4
dwt___87	13	12	11
dwt__162	9	9	9
dwt__193	30	31	27

Matrix	ρ_{max}	**Pattern Graph**	ColPack
dwt__198	12	11	10
dwt__209	17	13	13
dwt__221	12	11	11
dwt__234	10	8	6
dwt__245	13	11	12
dwt__307	9	10	10
dwt__310	11	10	10
dwt__346	19	17	18
dwt__361	9	9	11
dwt__419	13	12	13
dwt__492	11	11	10
dwt__503	25	22	21
dwt__512	15	15	13
dwt__592	15	12	14
dwt__607	14	14	12
dwt__758	11	10	10
dwt__869	14	12	12
dwt__878	10	10	11
dwt__918	13	13	12
dwt__992	18	19	20
dwt_1005	27	20	20
dwt_1007	10	10	11
dwt_1242	12	13	13

Matrix	ρ_{max}	**Pattern Graph**	ColPack
dwt_2680	19	16	16
Total		377	371

Table 5.7: Graph coloring for data set 3

Matrix	ρ_{max}	**Pattern Graph**	ColPack
jagmesh1	7	8	8
jagmesh2	7	8	8
jagmesh3	7	7	9
jagmesh4	7	8	8
jagmesh5	7	9	8
jagmesh6	7	8	9
jagmesh7	7	8	8
jagmesh8	7	8	9
jagmesh9	7	8	9
Total		72	76

Table 5.8: Graph coloring for data set 4

Matrix	ρ_{max}	**Pattern Graph**	ColPack
lshp_265	7	8	8
lshp_406	7	9	9
lshp_577	7	8	9
lshp_778	7	8	8
lshp1009	7	8	8

Matrix	ρ_{max}	**Pattern Graph**	ColPack
lshp1270	7	8	9
lshp1561	7	8	9
lshp1882	7	9	9
lshp2233	7	9	8
lshp2614	7	9	9
lshp3025	7	9	9
lshp3466	7	9	9
Total		102	104

Table 5.6, 5.7 and 5.8 illustrates the difference between the number of colors by Algorithm 1 and ColPack. Each of the these tables also shows the total number of colors. On the test suite **DWT** of Table 5.7, ColPack gives marginally better (fewer) coloring. On the other hand, our algorithm performs slightly better on the test suite **LSHAPE** and **JAGMESH**. Both the star coloring used in ColPack and Algorithm 1 presented in this thesis are based on Greedy heuristic. It is inconclusive as to which method is better based on the test instances. On the other hand, it is evident that the multi-coloring approach with pattern graph model is more general compared to the star coloring using adjacency graph model. The pattern graph model allows for a strictly larger class of instances (matrices) than the adjacency graph. Moreover, the notion of symmetric direct cover more accurately captures the determination ability in the sense of matrix-vector products. This is clearly demonstrated by the example sparsity structure in page 36. It can be shown that no star coloring of a Hessian matrix having this sparsity pattern can be determined with fewer than 5 matrix-vector products.

Chapter 6

Conclusion and Future works

6.1 Conclusion

For large-scale problems, it is preferable to take advantage of the sparsity structure and the symmetry of the Hessian matrix to reduce number of function evaluations. In this thesis,

- We have presented a generalized model for the symmetry-exploiting direct determination methods for sparse Hessian matrices. Our model uses pattern graphs to represent the sparsity structure of Hessian matrices.

- We have proposed a new heuristic for the direct determination. We have run our heuristic on a set of large-scale practical test instances and found promising numerical results.

6.2 Future Direction

For extending the scope of this thesis, we would like to give the following suggestions:

- Our algorithm tries to minimize the number of groups required to determine the Hessian matrix. Though it provides better result on some instances but the implementation still needs to be efficient.

- A parallel implementation of our algorithm on shared memory symmetric multiprocessor system seems very promising research direction.

Bibliography

[1] A graph coloring algorithm package. https://github.com/CSCsw/ColPack. Accessed: 2016-06-20.

[2] The matrix market project. `http://math.nist.gov/MatrixMarket/`. Accessed: 2016-06-20.

[3] Mihir Bellare, Oded Goldreich, and Madhu Sudan. Free bits, pcps, and nonapproximability—towards tight results. *SIAM Journal on Computing*, 27(3):804–915, 1998.

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

[5] Tamara G. Kolda Bruce Hendrickson. Graph partitioning models for parallel computing. *Parallel Computing - Special issue on graph partioning and parallel computing*, 26(12):1519–1534, 2000.

[6] Marco Chiarandini, Irina Dumitrescu, and Thomas Stützle. Stochastic local search algorithms for the graph colouring problem. *Handbook of approximation algorithms and metaheuristics*, pages 63–1, 2007.

[7] T. F. Coleman and J.-Y. Cai. The cyclic cloring problem and estimation of sparse Hessian matrices. *SIAM Journal on Algebraic Discrete Methods*, (7):221–235, 1986.

[8] 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.

[9] 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.

[10] Thomas F Coleman and Jorge J Moré. Estimation of sparse Hessian matrices and graph coloring problems. *Mathematical Programming*, 28(3):243–270, 1984.

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

[12] 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.

[13] 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.

[14] Philippe Galinier and Alain Hertz. A survey of local search methods for graph coloring. *Computers & Operations Research*, 33(9):2547–2562, 2006.

[15] Assefaw H Gebremedhin, Arijit Tarafdar, Fredrik Manne, and Alex Pothen. New acyclic and star coloring algorithms with application to computing Hessians. *SIAM Journal on Scientific Computing*, 29(3):1042–1072, 2007.

[16] Assefaw H Gebremedhin, Arijit Tarafdar, Alex Pothen, and Andrea Walther. Efficient computation of sparse Hessians using coloring and automatic differentiation. *INFORMS Journal on Computing*, 21(2):209–223, 2009.

[17] 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.

[18] John R Gilbert, Steve Reinhardt, and Viral B Shah. High-performance graph algorithms from parallel sparse matrices. In *International Workshop on Applied Parallel Computing*, pages 260–269. Springer, 2006.

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

[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 *Proceedings of the 5th International Conference on Mathematical Software, ICMS*, volume 9725 of *LNCS*, pages 275–283, Berlin,Germany, 2016. Springer.

[21] Shahadat Hossain and Nasrin Hakim Mithila. Pattern graph for sparse Hessian matrix determination. In *The 7^{th} International Conference on Algorithmic Differentiation*, pages 78–81. SIAM, 2016.

[22] Shahadat Hossain and Trond Steihaug. Computing a sparse Jacobian matrix by rows and columns. *Optimization Methods and Software*, 10(1):33–48, 1998.

[23] 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 Newyork, 2002.

[24] Shahadat Hossain and Trond Steihaug. Graph models and their efficient implementation for sparse Jacobian matrix determination. *Discrete Applied Mathematics*, 161(12):1747–1754, 2013.

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

[26] David S Johnson. Worst case behavior of graph coloring algorithms. In *Proc. 5th SE Conf. on Combinatorics, Graph Theory and Computing*, pages 513–528, 1974.

[27] 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.

[28] Yaw-Ling Lin and Steven S Skiena. Algorithms for square roots of graphs. *SIAM Journal on Discrete Mathematics*, 8(1):99–118, 1995.

[29] S Thomas McCormick. Optimal approximation of sparse Hessians and its equivalence to a graph coloring problem. *Mathematical Programming*, 26(2):153–171, 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] R Garey Michael and S Johnson David. Computers and intractability: a guide to the theory of np-completeness. *WH Free. Co., San Fr*, 1979.

[32] Garry N Newsam and John D Ramsdell. Estimation of sparse Jacobian matrices. *SIAM Journal on Algebraic Discrete Methods*, 4(3):404–418, 1983.

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

[34] MJD Powell and Ph L Toint. On the estimation of sparse Hessian matrices. *SIAM Journal on Numerical Analysis*, 16(6):1060–1074, 1979.

[35] Jeremy P Spinrad. *Efficient Graph Representations.: The Fields Institute for Research in Mathematical Sciences.*, volume 19. American Mathematical Soc., 2003.