CsegGraph: Column Segment Graph Generator [⋆]

Shahadat Hossain

Department of Mathematics and Computer Science, University of Lethbridge, Alberta, Canada

Zhenshuan Zhang

Department of Mathematics and Computer Science, University of Lethbridge, Alberta, Canada

Abstract

A graph generator associated with the determination of mathematical derivatives is described. The graph coloring instances are obtained as intersection graphs $G_\Pi(A)$ of $m \times n$ sparse pattern matrix A with row partition Π. The size of the graph is dependent on the row partition; the number of vertices can be varied between the number of columns (using single block row partition Π_1) and the number of nonzero entries of A (using m block row partition Π_m). The chromatic number of the generated graph instances satisfy $\chi(G(A)) = \chi(G_{\Pi_1}(A)) \leq \chi(G_{\Pi_m}(A))$.

Key words: Column Segment Graph, Sparse Matrix, Mathematical Derivative

[⋆] Supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) Discovery Grant.

Email addresses: `hossain@cs.uleth.ca` (Shahadat Hossain), `zhenshuan.zhang@uleth.ca` (Zhenshuan Zhang).

URL: `http://www.cs.uleth.ca/~hossain` (Shahadat Hossain).

1 Introduction

Graph coloring problems arise in variety of scientific applications and are one of the widely studied class of problems in graph theory. Applications where the underlying problem is modelled by graphs arise, for example, in scheduling and partitioning problems, matrix determination problems [2,5], and register allocation problems. Unfortunately, determining whether or not an arbitrary graph is 3-colorable is NP-complete. The availability of suitable benchmark test problems is therefore an important component in the design and testing of effective algorithms for graph coloring and related problems. The main purpose of this paper is to describe the software implementation of graph coloring test instances described in [6]. The remainder of this paper is organized in five sections. Section 2 provides a brief introduction to the coloring problem we are concerned with. In section 3, an algorithmic description of the column segment graph instances is given. Section 4 contains instructions for using our graph generator. In section 5 we present a graph generator based on a given partition of the edges of an undirected graph and section 6 concludes the paper.

2 Description of the Coloring Problem

A *Graph* $G = (V,E)$ is a set V of *vertices* and a set E of *edges*. An edge $e \in E$ is denoted by an unordered pair $\{u,v\}$ which connects vertices u and v, $u,v \in V$. A graph G is said to be a *complete* graph or *clique* if there is an edge between every pair of distinct vertices. In this paper multiple edges between a pair of vertices are considered as a single edge. A *p-coloring* of the vertices of G is a function $\Phi : V \to \{1,2,\cdots,p\}$ such that $\{u,v\} \in E$ implies $\Phi(u) \neq \Phi(v)$. *The chromatic* *number* $\chi(G)$ of G is the smallest p for which it has a *p-coloring*. An *optimal coloring* is a *p-coloring* with $p = \chi(G)$.

Given an $m \times n$ matrix A, the *intersection graph* of the columns of A is denoted by $G(A) = (V, E)$ where corresponding to column j of A, written $A(:, j), j = 1, 2, \ldots, n$, there is a vertex $v_j \in V$ and $\{v_j, v_l\} \in E$ if and only if columns $A(:, j)$ and $A(:, l), l \neq j$ have nonzero elements in the same row position.

Let Π be a partition of $\{1, 2, \ldots, m\}$ yielding $w_1, w_2, \ldots, w_{\tilde{i}}, \ldots, w_{\tilde{q}}$ where $w_{\tilde{i}}$ contains the row indices that constitute block $\tilde{i}$ written $A(w_{\tilde{i}}, :) \in R^{m_{\tilde{i}} \times n}$, $\tilde{i} = 1, 2, \ldots, \tilde{q}$. A segment of column j in block $\tilde{i}$ of A denoted by $A(w_{\tilde{i}}, j), \tilde{i} = 1, 2, \ldots, \tilde{q}$ is called a *column segment*.

Definition 2.1 *Structurally orthogonal column segment*

- `(Same Column)`

  Column segments $A(w_{\tilde{i}}, j)$ and $A(w_{\tilde{k}}, j)$, $\tilde{i} \neq \tilde{k}$ are *structurally orthogonal*
- `(Same Row Block)`

  Column segments $A(w_{\tilde{i}}, j)$ and $A(w_{\tilde{i}}, l)$, $j \neq l$ are *structurally orthogonal* if they do not have nonzero entries in the same row position.
- `(Different)`

  Column segments $A(w_{\tilde{i}}, j)$ and $A(w_{\tilde{k}}, l)$, $\tilde{i} \neq \tilde{k}$ and $j \neq l$ are *structurally orthogonal* if

  · $A(w_{\tilde{i}}, j)$ and $A(w_{\tilde{i}}, l)$ are *structurally orthogonal* and

  · $A(w_{\tilde{k}}, j)$ and $A(w_{\tilde{k}}, l)$ are *structurally orthogonal*

An *orthogonal partition of column segments* is a mapping

$$\kappa : \{(\tilde{i}, j) : 1 \leq \tilde{i} \leq \tilde{q}, 1 \leq j \leq n\} \rightarrow \{1, \ldots, p\}.$$

where column segments in each group are structurally orthogonal.

Definition 2.2 Given matrix A and row q-partition Π, the *column-segment graph* associated with A under partition Π is a graph $G_\Pi(A) = (V,E)$ where vertex $v_{\tilde{i}j} \in V$ corresponds to the column segment $A(w_{\tilde{i}}, j)$ not identically 0, and $\{v_{\tilde{i}j}, v_{\tilde{k}l}\} \in E$ $1 \leq \tilde{i}, \tilde{k} \leq \tilde{q}, 1 \leq j, l \leq n$ if and only if column segments $A(w_{\tilde{i}}, j)$ and $A(w_{\tilde{k}}, l)$ are not structurally orthogonal.

The problem of determining Jacobian matrices using column segments can be stated as the following graph problem.

Theorem 1 *[5] Φ is a coloring of $G_\Pi(A)$ if and only if Φ induces a orthogonal partition κ of the column segments of A.*

3 The Column Segment Graph Generator

In this section we describe an algorithm for constructing *column segment graph* $G_\Pi(A)$ associated with a $m \times n$ matrix A and a row partition Π. Furthermore, we describe the column segment matrix A_Π associated with the given row partition.

Let Π be a row partition of matrix A that partitions the rows into blocks $A_1, A_2, \ldots, A_k$ (See Fig. 1(a)). Denote the intersection graph corresponding to $A_{\tilde{i}}$ by $G(A_{\tilde{i}}), \tilde{i} = 1, 2, \ldots \tilde{q}$. The construction of A_Π involves two phases. In the first phase, blocks $A_{\tilde{i}}$, $\tilde{i} = 1, 2, \ldots, \tilde{q}$ are placed successively in left to right fashion (see Fig. 1(b)) such that each nonzero column segment is mapped to a unique column of A_Π. In other words, for every nonzero column segment of A, a column is created in A_Π where all the entries are zero except that the column segment is copied in the matching row positions. This situation is illustrated in the top part of Fig. 1(b). In the second phase of

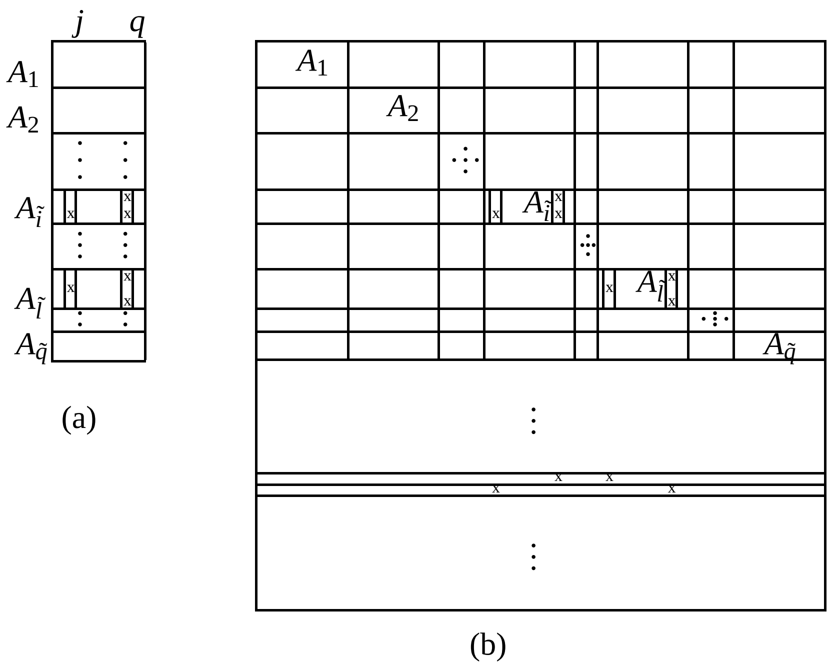

Fig. 1. (a): Matrix A is partitioned into $\tilde{q}$ blocks (b): Column segment matrix corresponding to the partition.

construction, restrictions are enforced on column segments that are not structurally orthogonal in order to prevent them from being grouped together. Consider column segments $A(w_{\tilde{i},j})$ and $A(w_{\tilde{i},q})$ in $A_{\tilde{i}}$. If there are nonzero entries in the same row position in $A(w_{\tilde{i},j})$ and $A(w_{\tilde{i},q})$ then they are not orthogonal implying that $A(w_{\tilde{i},j})$ is not orthogonal to column segments $A(w_{\tilde{p}})$ for all $\tilde{p} \neq \tilde{i}$. Consequently, $A(w_{\tilde{i},j})$ cannot be grouped together with any of the segments $A(w_{\tilde{p},q})$. Similarly, $A(w_{\tilde{i},q})$ is not orthogonal to columns $A(w_{\tilde{p},j})$ for all $\tilde{p} \neq \tilde{i}$. To enforce these restrictions we simply introduce two new rows in A_Π, one containing nonzero entries in the column positions mapped by the column segments $A(w_{\tilde{i},j})$ and $A(w_{\tilde{p},q})$ and the other containing nonzero entries in the column positions mapped by the column segments $A(w_{\tilde{i},q})$ and $A(w_{\tilde{p},j})$ for all $\tilde{p} \neq \tilde{i}$. This is done for every pair of dependent column segments in $A_{\tilde{i}}$, $\tilde{i} = 1, 2, \ldots, \tilde{q}$ (see Fig. 1(b)). To see this dependency restriction in terms of graphs, consider vertices $v_{\tilde{i}j}$ and $v_{\tilde{i}q}$ in $G(A_{\tilde{i}})$. For each such edge we define edges between vertex $v_{\tilde{i}j}$ and vertices $v_{\tilde{p}q}$ from $G(A_{\tilde{p}})$ for $\tilde{p} \neq \tilde{i}$. Similarly, vertex $v_{\tilde{i}q}$ is connected with the vertices $v_{\tilde{p}j}$ from $G(A_{\tilde{p}})$ for $\tilde{p} \neq \tilde{i}$. This situation is illustrated in Fig. 2. In Fig. 1(a) the matrix is partitioned into $\tilde{q}$ blocks denoted by

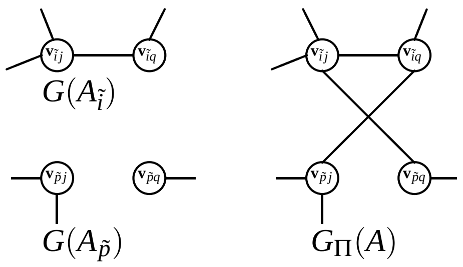

Fig. 2. Graph $G_{\Pi}(A)$ before and after the insertion of edges due to the edge $\{v_{\tilde{i}j}, v_{\tilde{i}q}\}$ in $G(A_i)$.

$A_1, A_2, \ldots, A_{\tilde{q}}$. In Fig. 1(b) placement of each of the blocks $A_1, A_2, \ldots, A_{\tilde{q}}$ is shown. Column segments $A(w_{\tilde{i},j})$ and $A(w_{\tilde{i},q})$ are not orthogonal, and hence two rows containing nonzero entries in the appropriate columns are introduced.

From the procedure for column segment matrix construction described above we have the following result.

Theorem 2 *[5] $G(A_{\Pi})$ is isomorphic to $G_{\Pi}(A)$.*

An upper bound on the size of the column segment matrix and graph is easily obtained from its construction. For a $\tilde{q}$ block partition, the number of columns $n' \leq \rho \leq n * \tilde{q}$ where ρ is the number of nonzero elements in A. The number of rows $m' \leq m + (\tilde{q} - 1) \sum_{i=1}^{m} \rho_i(\rho_i - 1)$ where ρ_i is the number of nonzero in the ith row of A. In practice, however, the numbers n' and m' are smaller due to many zero column segments and repeated edges between pair of distinct vertices.

4 Using the Graph Generator

Our graph generator implements column segment graph instances. The software uses SparseLib++v.1.5d [3], a collection of C++ sparse matrix classes that can read and convert between a number of standard sparse matrix data structures e.g., coordinate, compressed column, and compressed row format which are also supported by Harwell-Boeing test matrix collection [4].

The C++ source code is provided in two directories:

(1) The directory named `col_seg_graph` contains C++ code implementing the column-segment matrix and the associated graph from a given sparse matrix. The directory also contains several utility functions and a Makefile that can be used to generate the executables.

(2) The directory named `SparseLib++` contains the sparse matrix library described in [3]. `SparseLib++` provides support for Harwell-Boeing and Matrix Market [1] sparse matrix exchange formats.

4.1 Column Segment Graph

The function for defining column segment graph has the following prototype declaration.

```
Coord_Mat_double& extend( const Coord_Mat_double& A,
                          const vector<int>& perm,
                          const vector<int>& part,
                          const char* fileName );
```

Given input matrix A in `Coord_Mat_double` format [1], input vector `perm` representing a permutation of the row indices of A, input vector `part` representing a row partition of A, and character string `fileName`, `extend` returns the resulting column segment matrix in `Coord_Mat_double` format as function return value, and writes the associated column segment graph in the file `fileName`.

Users can make (by running the command `make`) the executable program named `extend` which can be executed to generate column segment graph and the associated matrix.

```
 Usage: extend TESTFILE [OUT_FILE (Optional)]
```

Information such as row partition, permutation, and the location of the input matrix are provided in an input text file `TESTFILE`. Argument `OUT_FILE_NAME` stores the column segment graph and is optional; if not provided the graph is written to the standard output. The resulting column segment matrix is stored in a text file called `inputMatrixFileName_ext.mtx` where `inputMatrixFileName.mtx` is the name of the input matrix in Matrix Market exchange format.

The format of `TESTFILE` is described below.

Comment lines: The hash sign (#) at the beginning of a line marks that line as comment. The comments can only appear at the beginning of the file (i.e., before permutation and partition data) and cannot be interleaved with data.

The partition size is a single integer that specifies the number of row blocks in the partition.

Partition lines: Following the comment lines and partition size, commences the specification of the row partition given by listing the index of the first row of each row block: $r_1\ r_2\ \ldots\ r_{nblks+1}$ where $nblks$ denotes the number of row blocks in the partition. The first row block in the partition consists of rows $r_1, \ldots, r_2 - 1$, the second row block in the partition consists of rows $r_2, \ldots, r_3 - 1$ and so on. Since a permutation of the rows can also be specified (explained next) the indices r_1, r_2 etc. are given in increasing order with $r_1 = 1$ and $r_{nblks} + 1 = m + 1$ where m denotes the number of rows in the input matrix. An m block partition can also be indicated by writing the negative of the integer m+1.

Permutation Lines: It is possible to specify a permutation of rows in specifying the row partition. This allows for the rows in a row block not necessarily be consecutive. For example, if we have 4 rows and the row partition is 1 3 5 and the permutation is given as 3 1 2 4 then first block consists of rows with indices 3 1 and second block consists of rows with indices 2 4. If no permutation need to be specified then the negative of the number of rows m is written in the permutations lines.

Input Matrix file: This last line specifies the name of the file (full path name) containing the input matrix.

4.2 Examples

Example test file (`TESTFILE`) Example 1:

```
# File t1.input
# Input matrix has 4 columns and 4 rows
# Row 2-partition with permutation
    2
    1 3 5
    2 3 1 4
    input_dir/test1.mtx
```

The first 3 lines are comments. The fourth line says that the row partition defines two blocks. The fifth line specifies the two-block row partition: the first block starting at row 1 and ending at row 2, the second starting at row 3 and ending at row 4. There are 4 rows in the input matrix. The sixth line says that a row permutation is provided: rows 2 and 3 of the input matrix constitute the first block and the rows 1 and 4 constitute the second block. The seventh line specifies that the input matrix is contained in file `test1.mtx` in the directory `input_dir`. The suffix `mtx` indicates that the input sparse matrix is provided in Matrix Market format. The input matrix can also be provided in Harwell-Boeing format (`test1.p[rsu][ae]`). Note that a Harwell-Boeing pattern matrix has a three letter suffix in which the first one is the character `p`, the second is one of the characters `r,s`, or `u`, and the third character is either an `a` or an `e`. (The current implementation only supports matrix market exchange format for output of column segment matrix.) The content of the input matrix `test1.mtx` is shown below.

```
%%MatrixMarket matrix coordinate pattern general
4 4 8
1 1
1 2
2 1
2 3
3 2
3 3
4 3
4 4
```

Then the command

```
 extend test1.input test1_ext.graph
```

will create the files `test1_ext.mtx` which contains the column segment matrix and `test1_ext.graph` contains the column segment graph corresponding to the given row partition.

Content of `test1_ext.mtx`:

```
%%MatrixMarket matrix coordinate pattern general
% Generated by writeMM()
11 7 22
  1     1
  1     3
  2     2
  2     3
  3     4
  3     5
  4     6
  4     7
  5     1
  5     6
  6     3
  6     4
  7     2
  7     6
  8     3
  8     5
  9     4
```

```
   9      2
  10      5
  10      1
  11      7
  11      3
```

Content of `test1_ext.graph`

```
   1      3
   2      3
   4      5
   6      7
   1      6
   3      4
   2      6
   3      5
   4      2
   5      1
   7      3


 7 11
```

Figure 3 displays the column intersection graph and the column segment graph of the example contained in `test1.mtx` (A). An edge of the original graph $G(A)$ is indicated by a solid line. The dashed lines in the column segment graph denote edges introduced to enforce dependency. Note that in Figure 3 $G_\Pi(A)$ has seven vertices and the indices of these vertices are indicated in parentheses. In the graph file `test1_ext.graph` the edges of the column segment graph are given as pairs of

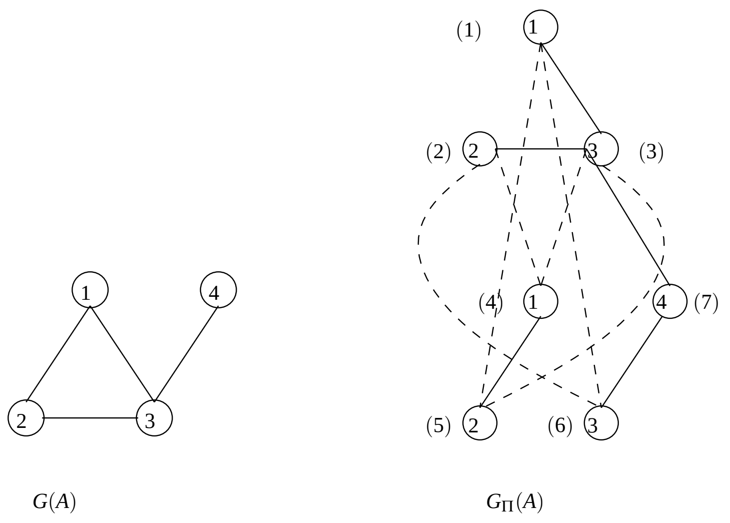

Fig. 3. The column intersection graph $G(A)$ and the column segment graph $G_{\Pi}(A)$ for the test matrix A provided in test1.mtx

indices one edge per line in all but the last line. The last line contains the number of vertices (7) and the number of edges (11) of the column segment graph. The column segment matrix contained in file `test1_ext.graph` is described using matrix market format.

Example test files (`TESTFILE`) Example 2:

```
# File t2.input
# Input matrix has 4 columns and 4 rows
# Row 4-partition and no permutation
    4
    -5
    -4
    input_dir/test2.mtx
```

The first 3 lines are comments. The fourth line says that this row partition has four blocks: each row constitutes a block. The input matrix has the same dimension as in Example 1. The fifth line says that no row permutation is given which is indicated by the negative of the $m+1$ where $m=4$ denotes the number of rows of the input matrix. The sixth line says that no row permutation is provided. The last line specifies that the input matrix is contained in file `test2.mtx` in the directory `input_dir`.

4.3 Utilities

The graph generator is packaged with a number of utility programs for user convenience. The executable `showmat` displays small (with less than approximately 30 columns) pattern matrices on to the terminal screen.

```
Usage: showmat { -m MFILE | -t TESTFILE }
            -m: MFILE is a matrix in Matrix Market or
            Harwell-Boeing exchange format
            -t: TESTFILE is an input file as in extend
```

With option `-m` the argument is expected to be a file that describes a sparse pattern matrix in either Matrix Market or Harwell-Boeing format. Option -t, on the other hand, allows users to specify a `TESTFILE` (see the usage of `extend` command).

Since the column segment graph output by `extend` does not conform to the input format of any particular graph coloring application we provide Perl scripts to format the column segment graph. The script `ToDSaturFmt.pl` converts the column segment graph to the input format of `DSATUR` graph coloring implementation by Michael Trick [7].

```
 Usage:Perl ToDSaturFmt.pl InputFile OutPutFile
       InputFile is a ASCII file describing
       column segment graph (output of extend function)
       OutPutFile is a ASCII file describing Column Segment
       graph in the input format for DSATUR implementation by
       Michael Trick.
```

The script `ToDIMACS.pl` converts the column segment graph to the input format of DIMACS challenge.

```
 Usage:Perl ToDIMACS.pl InputFile OutPutFile
       InputFile is a ASCII file describing
       column segment graph (output of extend function)
       OutPutFile is a ASCII file describing Column Segment
       graph in the input format for DIMACS challenge.
```

5 A Graph Theoretic Approach

The graph generator of the preceding section is based on row partition of sparse pattern matrices. The generated graph instances are described by listing the edges (undirected) followed by the number of vertices and edges of the graph. An instance generator can also be described in purely graph-theoretic terms. Let $G = (V,E)$ be an undirected graph with $|V| = n > 0$ vertices and $|E| = m > 0$ edges. Let Π be a partition[1] of the edges of E into subsets $E_1, E_2, \ldots, E_{\tilde{q}}$. Define graphs $G_1 = (V_1,E_1), G_2 = (V_2,E_2), \ldots, G_{\tilde{q}} = (V_{\tilde{q}},E_{\tilde{q}})$ where $V_{\tilde{i}} = \{v \in V|$ there is an edge $e \in E_{\tilde{i}}$ which is incident on $v\}, \tilde{i} = 1,2,\ldots,\tilde{q}$. A construction similar to the one shown

[1]
$$E = \bigcup_{\tilde{i}} E_{\tilde{i}},\ \ E_{\tilde{i}} \cap E_{\tilde{j}} = \emptyset \text{ whenever } \tilde{i} \neq \tilde{j}, \text{ and } E_{\tilde{i}} \neq \emptyset, \tilde{i} = 1,2,\ldots,\tilde{q}.$$

in Figure 2 can be used to introduce new edges to incorporate dependency information among the subgraphs $G_{\tilde{i}}, \tilde{i} = 1, 2, \ldots, \tilde{q}$.

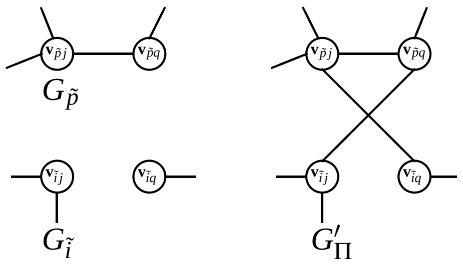

Fig. 4. Graph G'_{Π} before and after the insertion of edges due to the edge $\{v_{\tilde{p}j}, v_{\tilde{p}q}\}$ in $G_{\tilde{p}}$.

In Figure 4 the subgraph $G_{\tilde{p}}$ contains edge $\{v_{\tilde{p}j}, v_{\tilde{p}q}\}$. This dependency is incorporated in the graph G'_{Π} by defining the "dependency edges" $\{v_{\tilde{p}j}, v_{\tilde{i}q}\}$ and $\{v_{\tilde{p}q}, v_{\tilde{i}j}\}$ for each subgraph $G_{\tilde{i}}, \tilde{i} = 1, 2, \ldots, \tilde{q}$. Let $\widehat{E}$ be the set of such dependency edges. Then the resulting graph

$$G'_{\Pi} = (V', E')$$

where

$$V' = \bigcup_{\tilde{i}} V_{\tilde{i}}, \text{ and } E' = \left(\bigcup_{\tilde{i}} E_{\tilde{i}}\right) \bigcup \widehat{E}, \tilde{i} = 1, 2, \ldots, \tilde{q}$$

satisfies $\chi(G'_{\Pi}) \leq \chi(G)$.

Denote by $G'_{\Pi}(A) = (V', E')$ the "extended graph" of $G(A)$ (with $\tilde{q}$ block row partition Π) which is obtained via a simple modification of definition 2.2 where corresponding to each column segment, including the identical zero column segment, there is a vertex in V'. Let $G_{\Pi}(A) = (V, E)$ be the column segment graph under the same row partition Π. Then we have the following result that $G_{\Pi}(A)$ is p-colorable if and only $G'_{\Pi}(A)$ is p-colorable. To see this suppose Φ be a p-coloring of $G_{\Pi}(A)$. We show that Φ can be modified to construct a new p-coloring Φ' for $G'_{\Pi}(A)$. First, let $\Phi'(v_{\tilde{i}j}) = \Phi(v_{\tilde{i}j})$ corresponding to nonzero column segments $A(w_{\tilde{i}}, j), \tilde{i} = 1, 2, \ldots, \tilde{q}$ and $j = 1, 2, \ldots, n$. Let $A(w_{\tilde{p}}, q)$ be any column segment which is identical zero. Then $\{v_{\tilde{i}j}, v_{\tilde{p}q}\} \in E'$ implies that $\tilde{i} \neq \tilde{p}$ and $j \neq q$. Consequently, $A(w_{\tilde{i}}, j)$ and $A(w_{\tilde{i}}, q)$ must be nonzero and the corresponding vertices are included in V. We set $\Phi'(v_{\tilde{p}q}) = \Phi(v_{\tilde{i}q})$. Since $\Phi(v_{\tilde{i}j}) \neq \Phi(v_{\tilde{i}q})$ the coloring of $v_{\tilde{p}q}$ is valid. Since $G_\Pi(A) \subset G'_\Pi(A)$, Φ is a p-coloring of $G'_\Pi(A)$ implies that Φ is a p-coloring of $G_\Pi(A)$. Therefore, $G_\Pi(A)$ is p-colorable if and only $G'_\Pi(A)$ is p-colorable.

6 Concluding Remarks

In this paper we have described a graph instance generator based on intersection graphs of row partitioned sparse pattern matrices. We have also outlined a procedure for defining graph instances based on edge partition of a given input graph. That the generated instances are highly structured and the size of the generated instances can be varied easily make them convenient for use as test sets for combinatorial problems such as graph coloring.

7 Availability

The source code of the software described in this paper can be obtained by contacting the first author at `shahadat.hossain@uleth.ca`.

Acknowledgements

We are grateful to Professor Trond Steihaug for his careful reading of an early draft of the manuscript and providing many helpful comments.

References

[1] R. F. Boisvert, R. Pozo, K. Remington, R. Barrett, and J. J. Dongarra. The Matrix Market: A web resource for test matrix collections. In R. F. Boisvert, editor, *Quality of Numerical Software, Assessment and Enhancement*, pages 125–137, London, 1997. Chapman and Hall.

[2] T. F. Coleman and J. J. Moré. Estimation of sparse Jacobian matrices and graph coloring problems. *SIAM J. Numer. Anal.*, 20(1):187–209, 1983.

[3] J. Dongarra, A. Lumsdaine, R. Pozo, and K. Remington. A sparse matrix library in c++ for high performance architectures. In *Proceedings of the Second Object Oriented Numerics Conference*, pages 214–218, 1994.

[4] I. S. Duff, R. G. Grimes, and J. G. Lewis. Sparse matrix test problems. *ACM Transactions on Mathematical Software*, 15(1):1–14, 1989.

[5] S. Hossain and T. Steihaug. Optimal Direct Determination of Sparse Jacobian Matrices. Technical Report 254, Department of Informatics, University of Bergen, Norway, October 2003.

[6] S. Hossain and T. Steihaug. Graph Coloring in the Estimation of Sparse Derivative Matrices: Instances and Applications. Technical report, Department of Mathematics and Computer Science, University of Lethbridge, Alberta, Canada, March 2004.

[7] A. Mehrotra and M. A. Trick. A column generation approach for graph coloring. *INFORMS Journal on Computing*, 8:344–354, 1996.