Grammar-Based Representations of Large Sparse Binary Matrices

by

Jingyun Bian

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2016

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Large sparse matrices representation is a fundamental problem in big data processing and analysis. In some applications dealing with large sparse matrices, the I/O of these sparse matrices is the bottleneck of the whole system. To reduce the requirement of memory bandwidth in this scenario, it is important to develop alternative compact representations of large sparse matrices, while facilitating, if possible, matrix operations.

In this thesis, we propose two grammar based methods to compactly represent a sparse binary matrix with the capability of random accessing an element in the matrix. The first approach combines dimension coding (proposed by Yang[12]) with one of raster scan or Hilbert scan, where the so-called directionless grammar is applied. With the power of scanning, dimension coding's capability of representing 1-D sparse signals can be extended to 2-D sparse matrices. This approach inherits the random accessibility of dimension coding. In the second approach, we will introduce a new concept called Context-free Bipartite Grammar (CFBG) and present a framework wherein large sparse binary matrices can be represented by CFBG. Similar to the traditional concept of Context-free Grammar (CFG), a CFBG consists of a set of production rules. Unlike CFGs, however, the right member of each production rule in a CFBG is a labeled bipartite graph with each edge labeled either as a variable or terminal symbol. As the right hand side of a production rule is an ordered edge set, CFBG is also directionless. Two bipartite grammar transforms, a Sequential D-Neighborhood Pairing Transform (SNPT) and an Iterative Pairing Transform (IPT), are further presented to convert any binary matrix into a CFBG representing it.

Experiments show that compared with popular sparse matrix storage methods such as compressed row storage and quadtree, grammar-based sparse binary matrix representations can reduce the storage requirement of sparse matrices significantly (by a factor of as much as 70).

Acknowledgements

I would like to thank my supervisor, Prof. En-hui Yang, for his guidance and support. He is a very good exemplar of researcher and always inspires me to think more logically and precisely.

I would like to thank Prof. Derek Rayside and Prof. Mohamed Oussama Damen for being the committee members and providing me valuable comments and suggestions.

I would also like to thank all my colleagues in the multimedia communications lab for their continuous support and friendship: Yi Shan, Jiasen Xu, Xulai Cao, Hossam Amer, Jeffrey Erbrecht, Jin Meng and Xiang Yu. I would like to thank visiting scholars Xiangwen Wang and Jing He for their support and encouragement in my research.

Last but not least, I would like to thank my family members and my girl friend for their understanding and support.

Table of Contents

List of Tables viii

List of Figures ix

1 Introduction 1

1.1 Research Motivation and Problem Description 1

1.2 Research Contributions 1

1.3 Thesis Organization 2

2 Background 4

2.1 Sparse Matrix Representation 4

2.1.1 Coordinate Storage(COO) 5

2.1.2 Compressed Row Storage(CRS) 6

2.1.3 Quadtree 6

2.1.4 Other representations 7

2.2 JBIG 9

2.2.1 Resolution Reduction algorithm 9

2.2.2 Typical Prediction 9

2.2.3 Compression algorithm 10

2.3 Context-free Grammar 11

2.4 Grammar-based code 13

2.5 Summary 16

3 Sparse Binary Matrix Representation via Dimension Coding **17**
3.1 Dimension Coding . . . 17
3.1.1 Overview . . . 17
3.1.2 Directionless Grammar . . . 19
3.1.3 Incremental Directionless Grammar Transform . . . 22
3.1.4 Grammar encoding . . . 24
3.1.5 Random access decoder . . . 25
3.2 Scan Methods . . . 26
3.2.1 Raster Scan . . . 27
3.2.2 Hilbert Scan . . . 27
3.3 Sparse Binary Matrix coder via Dimension coding . . . 29

4 Binary Matrix Representation via Bipartite Gramamr Based Coding **31**
4.1 Overview . . . 31
4.2 Correspondence between binary matrix and bipartite . . . 33
4.3 Context-free Bipartite Grammar . . . 33
4.3.1 Graph Operations . . . 34
4.3.2 Context-free Bipartite Grammar . . . 35
4.4 Bipartite Grammar Encoding . . . 38
4.4.1 Canonical CFBG . . . 38
4.4.2 Grammar encoder . . . 40
4.5 Parallel Random Access . . . 42
4.6 Bipartite Grammar Transform . . . 44
4.6.1 Sequential D-Neighborhood Pairing Transform . . . 45
4.6.2 Iterative Pairing Transform . . . 47
4.6.3 Implementation of IPT . . . 47

5 Experimental Results **52**

5.1 Source and Descriptions of Testing Matrices 52

5.2 Overall Result 53

5.3 Detailed Result of Bipartite Grammar Coding 58

6 Conclusion and Future Work **61**

6.1 Conclusion 61

6.2 Future Work 62

6.2.1 Locality and Random Accessibility 62

6.2.2 Potential Capability of Pattern Discovery 62

6.2.3 Support more operations 62

References **63**

List of Tables

5.1 Statistics of testing matrices 53
5.2 Description of testing matrices 54
5.3 Space complexity of different representations 55
5.4 Experimental results including JBIG 58
5.5 Statistics of Context-free Bipartite Grammar (CFBG) 59

List of Figures

2.1 (a) Sample Matrix A. (b) COO representation of matrix A. 5

2.2 Compressed Row Storage (CRS) representation of the matrix in 2.1a . . . 6

2.3 Quadtree examples. A submatrix will be divided if its size is larger than 1×1(each non-empty leaf node corresponding to one non-zero elements). Four numbers inside the tree node are corresponding to its four children. 0 denotes the child is empty while 1 denotes non-empty. (a) Matrix A. (b) Quadtree representation of A. (c) Matrix B. (d) Quadtree representation of B. 8

2.4 Resolution reduction weights 10

2.5 Context templates in lowest layer (a) Two-line template. (b) Three-line template. 11

2.6 Illustration of a grammar-based code. 13

3.1 Structure of a dimension coder 18

3.2 Structure of a dimension encoder 18

3.3 Illustration of raster scan 27

3.4 Illustration of Hilbert scan. (a) 2×2 (b) 4×4 (c) 8×8 28

3.5 Structure of a 2-D dimension coder 30

4.1 Structure of a bipartite grammar coder 32

4.2 (a) A bipartite graph B; (b) the biadjacency matrix of B 33

4.3 Operations over B in Fig. 4.2a with solid edges labeled as δ and dash edges labeled as v_1: (a) subgraph B_3 induced by $\{(2,2)(3,2)(3,3)\}$; (b) bipartite graph $B_3-(3,2)$ indicated by label v_1; (c) graph B' obtained by subtracting B_3 from B at edge $(3,2)$; (d) graph obtained by adding B_3 to B' at edge $(3,2)$. Adding $B_3-(3,2)$ to B' at edge $(3,2)$ gets B back. 36

4.4 The derivation tree of CFBG in Exmp. 11 38

4.5 Structure of a bipartite grammar encoder 38

4.6 Matrix represented by G 40

4.7 (a) Illustration of scanning. (b) Sort left end point of cord. 49

5.1 Structure of four matrices where CFBG is better. (a) bmw7st_1 (b) cegb2919 (c) cage12 (d) FEM_3D_thermal2 56

5.2 Structure of matrix Chem97Zt 57

5.3 Structure of matrix roadNet-CA 57

Chapter 1

Introduction

1.1 Research Motivation and Problem Description

Large sparse matrices appear a lot in big data analytics (such as natural language processing, consumer data analysis, etc) as well as many scientific and engineering applications. In some applications, the I/O of these sparse matrices is the bottleneck of the whole system. Due to their sheer sizes, it is practically infeasible to represent and operate large sparse matrices in terms of the standard two dimensional array structure.

Therefore, it is important to develop alternative compact representations of large sparse matrices while facilitating, if possible, matrix operations, pattern discovery, and coding directly over these compact representations. In this thesis, we will try to obtain compact representations of binary matrices. These representations should not only require less storage but also support matrix operations, to be specific, random accessing any element in the matrix.

1.2 Research Contributions

In this thesis, we will discuss the problem of representing a large sparse binary matrix so that the representation requires less storage and, meanwhile, supports random access of this matrix.

We will inherit the idea of using context-free grammar to represent 1-D string, which has already been proved successful in the series of papers of grammar-based code proposed by Yang and Kieffer[20, 16]. We will propose two extensions of grammar-based coding.

The first approach combines dimension coding proposed by Yang[12], the target of which is to represent 1-D sparse signal, with a scan method to represent 2-D matrices. This approach naturally has the random accessibility provided by dimension coding. The grammar used in [12] is called directionless grammar which is an extended context-free grammar on an ordered index set.

The second approach uses newly proposed Context-free Bipartite Grammar (CFBG). CFBG, like directionless grammar in [12], is directionless since it is a context-free grammar on the ordered edge set of a bipartite graph. We will introduce the definition of CFBG and provide an algorithm to further encode the CFBG into a vector representation. Compared with the previous approach, the representations through this approach does not have as strong random access capability as the dimension coding provides. Therefore, we will call the random access on the CFBG based representation "Parallel random access". We will also provide two algorithms called grammar transformation which generate a CFBG representing a binary matrix.

By doing experiments on various matrices from real applications, we will show that the grammar-based representation greatly outperforms the state-of-art sparse matrix representation methods such as Compressed Row Storage (CRS)(by a factor of as much as 70) in the sense of space complexity. We will also show that when the density of the matrix is not extremely low and the distribution of the non-zero elements in the matrix is far from random, the power of grammar can be fully utilized by CFBG. However, since the scan method is not so adaptive to the distribution of non-zero elements, the performance of dimension coding with scanning is not as steady as bipartite grammar-based coding.

1.3 Thesis Organization

The other parts of the thesis are organized as follows: In the second chapter, we will first review some state-of-arts sparse matrix representation methods and then give the definition of Context-free Grammar, which plays a vital role in so-called grammar-based code, which is introduced right after CFG. In the third chapter, we will first introduce the dimension code proposed by Yang[12] to represent 1-D sparse signal, where the CFG is extended to so-called directionless grammar. Then we will introduce two scan methods, which combined with dimension coding, can be used to represent 2-D sparse matrices. In the fourth chapter, we further extended the grammar to Context-free Bipartite Grammar, which has a true 2-D nature and does not need to work together with scan method. The bipartite grammar-based coding is introduced right after the newly proposed grammar. In the fifth chapter, we will show some experimental results that illustrate the comparison

between different representation methods. Finally, we will conclude the paper in chapter 6 and discuss some future work.

Chapter 2

Background

2.1 Sparse Matrix Representation

Sparse matrices are matrices that contain only a small number of non-zero elements. As the percentage of non-zero elements in a matrix gets lower and lower, there is a turning point where the traditional method in which a matrix is stored as a 2-D array is no longer efficient since most operations with zeros are trivial. As a result, it is more efficient to store only the non-zero elements in addition to some indexing scheme.

One main issue in sparse matrix storage is the reduction of space requirement. We are concerned about two types of storage: "primary" where actual non-zero matrix elements are stored; "overhead" where all indexing information are stored. When the matrix is binary, primary storage can be omitted or replaced by a number indicating the number of non-zero elements.

Besides the memory requirement, another issue we need to take into consideration is the computational cost. In most cases, representing a matrix is not the ultimate goal. When some computation needs to be done on the matrix, it is required that using the representation will not sacrifice the complexity of the computation too much.

Example 1 Suppose we first represent a $n \times m$ binary matrix by 1. encoding it with some entropy encoder, for example JBIG, or 2. traditional 2-D array. Then we query whether the element at row r and column c is zero or not.

Compare the time complexity to query based on each representation:

Representation 1 $\Omega(nm)$ as the decoding process is needed beforehand.

Representation 2 $O(1)$.

We can see that, although entropy codeword has good compression efficiency, the high time complexity of following operations may prevent it from being a feasible representation.

We will briefly introduce some popular sparse matrix representations in the following pages. During the discussion, the index of the array starts from 1, N_{nz} denotes the number of non-zero elements, n, m denote the height and width of the matrix respectively.

2.1.1 Coordinate Storage(COO)

Coordinate Storage (COO) is the most intuitive format where a list of triple (r, c, v) is stored. Here r denotes the row index, c denotes the column index, and v denotes the value of the non-zero element located at row r and column c. Note that COO does not impose any order of the triples, however, raster scan order is usually applied to facilitate element look-up. Fig. 2.1 shows an example of COO representation. Each column in Tab. 2.1b represent a triple. We may conclude that the storage requirement of COO is $3N_{\text{nz}} + 3$ entries.

6	0	9	0	0	4	0	0
0	0	0	0	0	4	0	0
0	0	0	0	0	0	0	0
0	0	3	5	8	0	0	0
0	0	0	0	6	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	4	3	0
0	0	0	0	0	0	2	2
0	0	0	0	0	0	0	0

(a)

v	6	9	4	4	3	5	8	6	4	3	2	2
r	1	1	1	2	4	4	4	5	7	7	8	8
c	1	3	6	6	3	4	5	5	6	7	7	8

(b)

Figure 2.1: (a) Sample Matrix A. (b) COO representation of matrix A.

2.1.2 Compressed Row Storage(CRS)

CRS format may be the most popular representation of sparse matrices. It has been in use since at least the mid-1960s. In CRS, three arrays are needed:

value It contains the non-zero elements in raster scan order.

column_index It contains the column positions of the corresponding element in *value*.

row_pointer If *value* and *column_index* are partitioned into runs, each run contains all the non-zero elements in the same row. *row_pointer*[i] contains the pointer to the non-zero element where the run of row i starts.

Fig. 2.2 shows an example of CRS where $N_{\text{nz}} = 12$, row $3, 6$ and 9 contain no non-zero element. If row i and all the following rows have no non-zero element, row_pointer[i] will be set to $N_{\text{nz}} + 1$(row_pointer[9] in Fig. 2.2). Set row_pointer[$n + 1$] to $N_{\text{nz}} + 1$ for the convenience of following discussions. Run of row i starts with element with index row_pointer[i] and ends with row_pointer[$i + 1$] $- 1$. row_pointer[i] $>$ row_pointer[$i + 1$] $- 1$ denotes row i is empty(row $3, 6$ and 9 in Fig. 2.2).

We can conclude that CRS requires $2N_{\text{nz}} + n$ entries to represent the matrix. Compared with COO, when there are on average at least one non-zero elements in each row, in other words $n < N_{\text{nz}}$, CRS is better than COO.

value	6	9	4	4	3	5	8	6	4	3	2	2
column_index	1	3	6	6	3	4	5	5	6	7	7	8
row_index	1	4	5	5	8	9	9	11	13			

Figure 2.2: CRS representation of the matrix in 2.1a

2.1.3 Quadtree

A quadtree[11] is a tree data structure where each internal node has exactly four children. Quadtrees are often used to partition a 2D space by dividing it into 4 quadrants recursively.

When applied to represent a matrix, the quadtree structure has larger control and data overhead compared to standard formats such as COO and CRS. Therefore, some modifications are needed[15]:

- Empty submatrices that contain no non-zero elements are represented by the NULL pointer.
- When the size of submatrix is within some threshold, stop dividing.
- Modified versions of COO and CRS are used to represent non-empty nodes. Here modified means all coordinates are expressed relatively to the beginning of the submatrix (node).

Fig. 2.3 shows two matrices A, B and their quadtree representations. Both A, B have 4 non-zero elements. However, quadtree representation of A requires 6 more tree nodes than that of B. We can see from this example that when non-zero elements are distributed as several clusters, quadtree representation tends to have good performance.

To further reduce the overhead introduced by 4 pointers in each node, all the nodes can be stored in an array in Breadth-first search (BFS) order. In this case, only 4 flags indicating whether the child is empty are needed instead of 4 pointers. Note that applying this storage saving method also means the loss of capability to get access to an element without traverse the array from left to right.

2.1.4 Other representations

There are some other sparse matrix representations which we will concisely introduce in the following paragraphs.

Compressed Column Storage (CCS) is the transpose of CRS. CCS stores row indices and column pointers and is better than CRS in some programming language, such as Fortune, where matrices are stored in a column-wise manner.

Compressed Diagnal Storage (CDS)[2] is used particularly when the matrix is banded. CDS is not suitable for general usage as a few elements exceed the diagonal band will result in storing a lot of zeros.

Jagged Diagnoal Storage (JD)[10] is tailored for sparse matrix-vector multiplications. Non-zero elements are first shifted to the left leaving zeros to the right. Each row of the new matrix is then permuted so that the number of non-zero elements of each row is in decreasing order. Finally, the matrix we get from the previous step will be represented in a column-wise manner. Transposed Jagged Diagnal Storage (TJD) is a variant of JD. In TJD, all non-zero elements are shifted to the top instead of left. As TJD does not need the permutation step, it can save the storage used to indicate the permutation in JD.

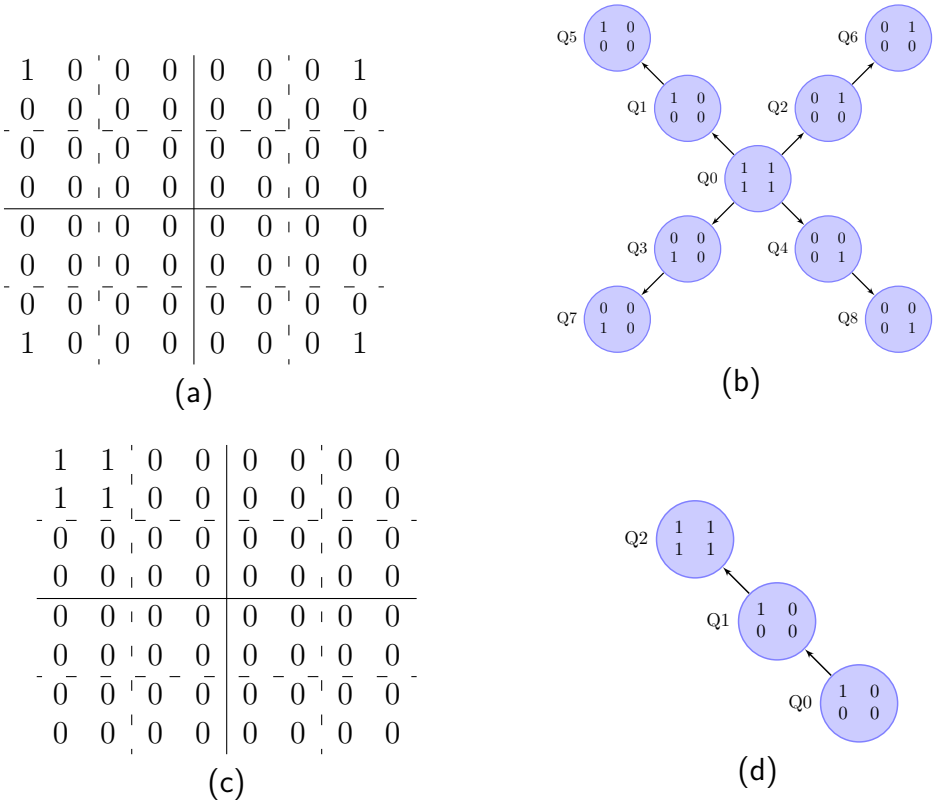


Figure 2.3: Quadtree examples. A submatrix will be divided if its size is larger than 1×1(each non-empty leaf node corresponding to one non-zero elements). Four numbers inside the tree node are corresponding to its four children. 0 denotes the child is empty while 1 denotes non-empty. (a) Matrix A. (b) Quadtree representation of A. (c) Matrix B. (d) Quadtree representation of B.

Block Based Compression Storage (BBCS)[26, 24] is a block-based format which is also designed for matrix vector multiplications under hardware support. In BBCS, a matrix is first partitioned into Vertical Block (VB)s. Each VB is then stored as a sequence of 6-tuple entries. Because the index values associated with each non-zero element are confined within the VB boundary, BBCS can have lower memory overhead and bandwidth requirement.

Hierarchical Sparse Matrix Storage (HiSM)[25][7] is the format that retains the advantages of BBCS while alleviating its two drawbacks: (1) a large number of indexed memory accesses, (2) inflexibility of change the content of the matrix. In HiSM, the matrix is divided hierarchically into several levels where the lowest level contains the actual value of

the non-zero elements and the higher levels contains pointers to the non-empty blocks of one level lower.

2.2 JBIG

JBIG[1] is a lossless image compression standard from the Joint Bi-level Image Experts Group. JBIG supports both sequential and progressive encoding methods. In sequential encoding, an image is encoded from the top to bottom and from left to right. In progressive encoding, series of multiple-resolution versions of the same image is stored within a single JBIG data stream. We will call the image with lowest resolution "lowest layer" and other images "differential layer". Note that using the progressive transmission of JBIG usually incurs a small overhead in bit-rate.

We will describe JBIG in three aspects: resolution reduction algorithm, prediction and compression algorithm.

2.2.1 Resolution Reduction algorithm

The resolution reduction algorithm is used to obtain the lower resolution version of an image. The original image is first divided into 2×2 blocks, and one lower-resolution pixel is assigned to each of these blocks. The low-resolution pixels is determined in the normal raster scan order.

In Fig. 2.4, a circle denotes a pixel in lower-resolution layer while a square denotes a pixel in higher-resolution layer. The question mark denotes the pixel that is being processed. The same convention will be used in the following figures. As we can see in Fig. 2.4, a weighted mean of 12 pixels (3 from lower-resolution layer and 9 from higher-resolution layer) is calculated. Only if the mean value is greater than or equal to 5, the pixel will be set to 1.

2.2.2 Typical Prediction

The typical prediction is designed to speed up the coding process since the encoding of a typical line is skipped. Meanwhile, it also provides some coding gain.There are two different typical predictions in JBIG, which are used in different layers.

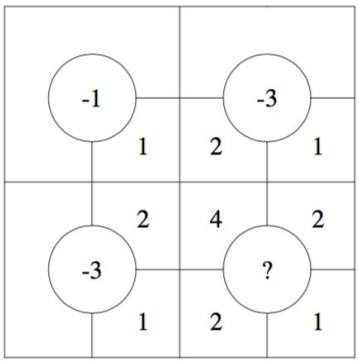


Figure 2.4: Resolution reduction weights

In the differential layer typical prediction, a low-resolution pixel is said "not typical" if (1) this pixel and all the pixels in its 8-neighborhood are the same color (2) one or more of the four high-resolution pixels associated with it differs from this common color. A given low-resolution line is "not typical" if it contains any not typical pixels. If a line composed of only typical pixels, a flag will be generated to denote the decoder that the encoding of the whole line is skipped.

In the lowest layer typical prediction, a line is said to be "typical" only if it is exactly the same as the line above it.

2.2.3 Compression algorithm

The compression algorithm is a context-based arithmetic encoding. The arithmetic coder applied in JBIG is Q-coder[23] developed by IBM and refined by Mitsubishi. The context templates are chosen according to the layers.

The lowest resolution layer uses a two- or three- line template of the pixels, shown in Fig. 2.2.3. The pixel to be coded is marked "?", while the pixels to be used for the template are marked "X" or "A". The "A" pixel can be thought of as a floating member of the neighborhood whose position can be changed. As 10 neighbor pixels make up the context shown in Fig. 2.2.3, 1024 arithmetic coder in parallel is needed.

The differential layer context template comprises 6 neighbor pixels from the same layer and 4 pixels from the lower layer. Based on the relative position of the pixel to be coded and the lower-resolution pixel assigned to it, we may have 4 different phases as shown in

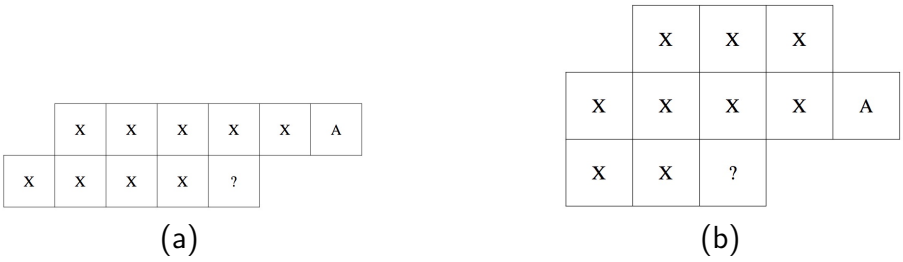
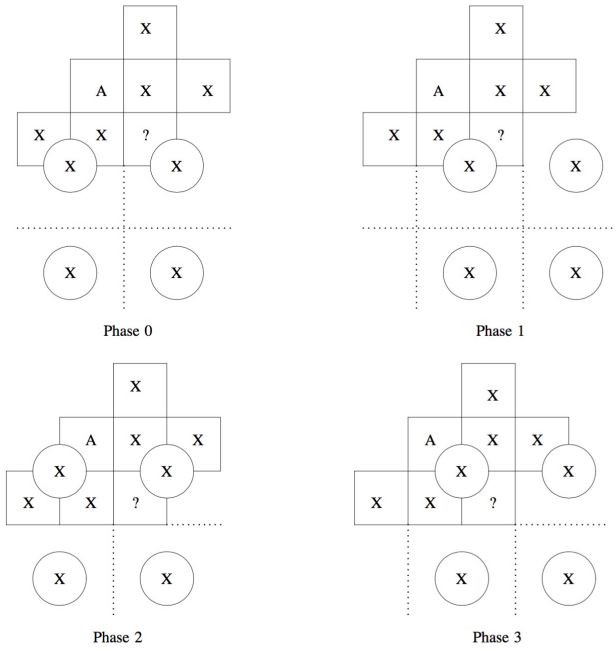


Figure 2.5: Context templates in lowest layer (a) Two-line template. (b) Three-line template.

Fig. 2.2.3. Because there are 10 pixels in the differential-layer templates and 2 bits to describe the phase, there are 4096 different possible contexts.



2.3 Context-free Grammar

Context-free Grammar (CFG)[14] is a certain type of formal grammar that describes all possible strings in a given formal language. If $\mathcal{A}$ is an alphabet with cardinality greater

than or equal to 2, let $\mathcal{A}^+$ be the set of all finite strings of positive length from $\mathcal{A}$. Let $\mathcal{A}^*$ be $\mathcal{A}^+ \bigcup \{\epsilon\}$, where ϵ denotes the empty string. A CFG can be defined as a quadruple $G = (V, \Sigma, R, S)$, where

- V is a finite nonempty set whose elements are called variables,
- Σ is another finite nonempty set disjoint from V, whose elements are called terminal symbols,
- R is a finite relation from V to $(V \cup \Sigma)^*$, and
- $S \in V$ is a special variable called the start variable.

A production rule in R is formalized mathematically as a pair $(\alpha, \beta) \in R$. However, instead of using ordered pair notation, production rules are usually written using an arrow operator: $\alpha \to \beta$. It is also common to list all right-hand sides for the same left-hand side on the same line, using "|" to separate them. Note that, in CFG, production rules can be one-to-one, one-to-many, or one-to-none. These production rules can be applied regardless of context.

For any strings $u, v \in (V \cup \Sigma)^*$, we say u directly yields v, written as $u \Rightarrow v$, if $\exists (\alpha, \beta) \in R$ with $\alpha \in V$ and $u_1, u_2 \in (V \cup \Sigma)^*$ such that $u = u_1 \alpha u_2$ and $v = u_1 \beta u_2$. We can see that v is the result of applying the production rule $\alpha \to \beta$ to u. We say u yields v, written as $u \overset{*}{\Rightarrow} v$ if $\exists u_1, \cdots, u_k$ where $k \geq 1$ such that $u = u_1 \Rightarrow u_2 \Rightarrow \cdots \Rightarrow u_k = v$.

The language of a grammar G is a set of string defined as follows: $L(G) = \{\omega \in \Sigma^* : S \overset{*}{\Rightarrow} \omega\}$

A derivation of a string for a grammar is a sequence of production rule applications that transforms the start symbol into that string. A derivation proves that the string belongs to the grammar's language.

Example 2 Consider CFG $G = (\{S\}, \{a, b, c\}, R, S)$ with production rules:

$$S \to aS | abc$$

S directly yields aS, written as $S \Rightarrow aS$. First apply the first production rule then the second, we can get S yields $aabc$, written as $S \overset{*}{\Rightarrow} aabc$. The derivation of string $aaabc$ is $S \Rightarrow aS \Rightarrow aaS \Rightarrow aaabc$. Thus, $aaabc \in L(G)$. If we define string multiplication as concatenation: $u \cdot v = uv$ where u, v are two strings. We may concludes that $L(G) = \{a^n bc : n \in \mathcal{N}^+\}$.

2.4 Grammar-based code

There are some previous works that utilize CFG for lossless data compression. In general, two approaches have been used. In one approach[3, 19], a fixed CFG is used, known to both encoder and decoder, such that its language contains all of the data strings that are to be compressed. To compress a particular data string, one then compresses the derivation tree. In the other approach[6, 22], a unique CFG G_x is assigned to each possible input string x, so that $L(G_x) = \{x\}$. The encoder transmit code bits to the decoder so that G_x can be reconstructed, from which x can be derived. The grammar-based code we will introduce here applies the second approach.

Grammar-based code was developed by Kieffer and Yang in a series of papers [20, 9, 16]. As shown in Figure 2.6, to compress a data object x, a grammar-based code first transforms x into a CFG G, from which x can be fully recovered, and then compresses x indirectly by lossless encoding G.

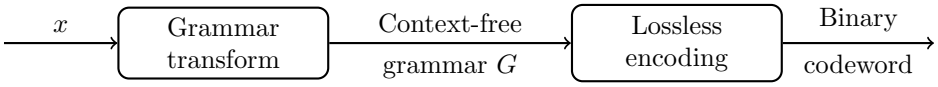


Figure 2.6: Illustration of a grammar-based code.

Clearly, CFGs play a vital role in grammar-based code. Following [20, 9], the CFG used in grammar-based code is a quadruple $G = (V, \Sigma, R, S)$ defined as follows, which is slightly different from the normal definition in Section 2.3:

- V is a finite nonempty set whose elements are called variables,
- Σ is another finite nonempty set disjoint from V, whose elements are called terminal symbols,
- R is a mapping from V to $(V \cup \Sigma)^+$, and
- $S \in V$ is a special variable called the start variable.

Note that in this definition

1. one-to-many production rules are not allowed. Thus, the production rule of varialbe v is unique and can be written as $v \to R(v)$.
2. empty production rules are not allowed.

3. many-to-one production rules are allowed but are actually useless. If $\exists u, v \in V$ with $R(u) = R(v)$ we can simply replace all appearance of v by u resulting a grammar G' that $L(G') = L(G)$. Keep doing this replacement, we can finally get a grammar with no many-to-one production rules.

Given a CFG G, start with S and replace in parallel each variables v in $R(S)$ by $R(v)$. We then get another string from $V \cup \Sigma$. If we keep doing this parallel replacement, one of the two following will hold:

P1 after finitely many parallel replacement steps, we get a string x from Σ; or

P2 the parallel replacement procedure never ends because the string so obtained at each step always contains a variable $v \in V$.

In grammar-base coding, we are interested only in G for which the parallel replacement procedure ends up with State P1 and every production rule $v \to R(v)$ is used at least once in the whole replacement procedure. Such a CFG G is called an admissible context-free grammar, and the string x from Σ obtained at State P1 is said to be represented by G or the start variable S. Since in this case, each production rule is used at least once, all other variables v in V represent substrings of x.

Example 3 In this example, we will illustrate above discussions. Let $\Sigma = \{0, 1\}$ and $V = \{v_0, v_1, v_2, v_3\}$ with v_0 designated as the start variable. The set of production rules below then defines an admissible CFG G:

$$\begin{aligned} v_0 &\to 0v_3v_2v_1v_2v_310 \\ v_1 &\to 01 \\ v_2 &\to v_11 \\ v_3 &\to v_1v_2 \end{aligned}$$

Start with v_0 and perform the parallel replacement:

$$\begin{aligned} v_0 &\stackrel{*}{\Rightarrow} 0v_3v_2v_1v_2v_310 \\ &\stackrel{*}{\Rightarrow} 0v_1v_2v_1101v_11v_1v_210 \\ &\stackrel{*}{\Rightarrow} 001v_110110101101v_1110 \\ &\stackrel{*}{\Rightarrow} 001011011010110101110 \end{aligned}$$

The CFG G or v_0 represents $x = 001011011010110101110$ with v_1, v_2, v_2 representing substrings 01, 011, and 01011, respectively.

A grammar transform converts any sequence $x \in \Sigma^+$ into an admissible grammar G, written as $x \to G$.

In [5, 4], a grammar transform called SEQUITUR algorithm was proposed. This algorithm maintains a dictionary of the two-symbol pairs. Each time append one symbol to the end of the current string. Based on the last two symbols, update the dictionary or replace last two symbols with a variable. During the SEQUITUR algorithm, these two properties hold: (1) no pair of adjacent symbols appears more than once in the grammar, (2) every rule is used more than once.

Define the size $|G|$ of G as the total length of its production rules, i.e., $|G| = \sum_{v \in V} |R(v)|$. We say $x \to G$ is asymptotically compact if $|G|/|x| \to 0$ as $|x| \to \infty$. Lempel-Ziv Grammar Transform[28] and Bisection Grammar Transform are two examples of asymptotically compact grammar transform. It is showed in [20] that a grammar-based code with an underlying asymptotically compact grammar transform is universal.

Define an admissible grammar to be irreducible if

(1) Each variable $v \in V - \{S\}$ appears at least twice in the right-hand sides of production rules.

(2) There is no non-overlapping repeated pattern of length ≥ 2

(3) Each distinct variable represents a distinct sequence

If a grammar transforma converts $x \in \Sigma^+$ into an irreducible grammar, we say the grammar transform is irreducible. It is proved in [20][9] that any grammar based code with an underlying irreducible grammar transform is universal and can outperform asymptotically any finite state code. Also in [9], an irreducible grammar transform called greedy grammar transform was proposed, based on which a universal lossless compression algorithm called YK-algorithm was then proposed. YK-algorithm utilize the power of both arithmetic coding and string matching to outperform standard 1-D compression algorithms.

Motivated by the great efficiency of Grammar-based code, some work has been done to generalize Grammar-Based code to 2-D. Yang and Guo[27] combined 1-D Grammar-based code with scanning and prediction. They got comparable or better result than JBIG[1]. However, as this method serialize 2-D data into 1-D sequence, it lacks 2-D nature. Multi-level pattern matching(MPM) code as a special case of Grammar-Based code is proposed by Kieffer, Yang, Nelson and Cosman[16]. Several years later, Jia and Yang[18] generalize MPM code to 2-D MPM code to compress bi-level image. However, 2-D MPM code is a block-based code and suffers from so-called 2-D boundary effect. With the power

of context modeling, 2-D boundary effect can be alleviated. It is shown that satisfying a mild condition, the context-dependent 2-D MPM code has an $O(1/\log n)$ worst case redundancy[18].

2.5 Summary

In this chapter, we first reviewed some state-of-art sparse matrix representations including COO, CRS and quadtree, which will be used as the benchmarks in the experiments. Then we introduce the well-known definition of Context-free Grammar, which is applied in the grammar based code to compress 1-D string. The idea of CFG and the structure of grammar based code is introduced as well in this chapter and will be inherited by the dimension coding and bipartite grammar based coding proposed in the following chapters.

Chapter 3

Sparse Binary Matrix Representation via Dimension Coding

In this chapter, we will use Dimension Coding, which is proposed by Yang in [12] to represent a sparse 1-D signal, along with a scanning method, which converts 2-D data into 1-D sequence, to represent sparse binary matrices.

The Dimension Coding will be introduced in Sec. 3.1. Two scan methods, namely raster scan and Hilbert scan, will be introduced in Sec. 3.2. The new structure of coder that utilize both dimension coding and a scan method to represent sparse binary matrix will be shown in Sec. 3.3.

3.1 Dimension Coding

3.1.1 Overview

Dimension coding is proposed by Yang in [12] to represent sparse signal while facilitating computation over the compressed signal. Consider a sparse signal as a vector $\boldsymbol{x} = [x(0), x(1), \cdots x(n-1)]$, with density $\alpha = \frac{\omega(\boldsymbol{x})}{n} \ll 1$, where $\omega(\boldsymbol{x})$ denotes the number of non-zero elements in $\boldsymbol{x}$. The compressed dimension representation of $\boldsymbol{x}$ is another vector $\boldsymbol{r} = [r(0), r(1), \cdots, r(m-1)]$ from which $\boldsymbol{x}$ can be fully recovered. The structure of a dimension coder is shown in Fig. 3.1. The mapping from $\boldsymbol{x}$ to $\boldsymbol{r}$ is called the dimension encoder. The mapping from $\boldsymbol{r}$ and l to $x(l)$ is called the random access decoder whose

time complexity is a polynomial time of $\log \omega(\boldsymbol{x})$, say $O(\log^2 \omega(\boldsymbol{x}))$. The mapping from $\boldsymbol{r}$ to $\boldsymbol{x}$ is called the recovery decoder whose time complexity is linear to $\omega(\boldsymbol{x})$.

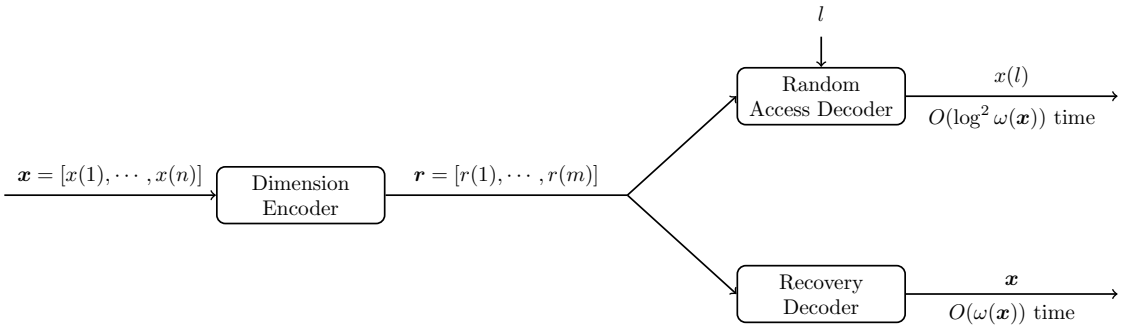


Figure 3.1: Structure of a dimension coder

Let $\mathcal{P}(\boldsymbol{x})$ denote the set of position of non-zero elements in $\boldsymbol{x}$, i.e.

$$\mathcal{P}(\boldsymbol{x}) = \{l : x(l) \neq 0,\ 0 \leq l \leq n-1\}$$

. As this paper mainly focuses on binary matrix representation, we will only introduce how to represent $\mathcal{P}(\boldsymbol{x})$. Regarding how to represent the actual values of non-zero elements along with the positions, please refer to [12].

The dimension encoder in Fig. 3.1 share a similar structure with grammar-based encoder shown in Fig. 2.6. To compress a sparse signal $\boldsymbol{x}$, we first transform $\boldsymbol{x}$ into a directionless grammar G, from which $\boldsymbol{x}$ can be fully recovered, and then encode $\boldsymbol{x}$ indirectly by encode G into $\boldsymbol{r}$. According to the structure of dimension coder, in the following parts of this section, we will first introduce the directionless grammar then explain the Incremental Directionless Grammar Transform (IDGT) with the grammar encoder, and finally, take a look at the random access decoder.

During the following discussions, let $\mathcal{N}$ denote the set of all integers and $\mathcal{N}^+$ denote the set of all positive integers. Let $\mathcal{X}_n$ denotes the set of all vectors of length n. The notation $|A|$ denotes the dimension of A if A is a vector, the size of A if A is a set or a grammar.

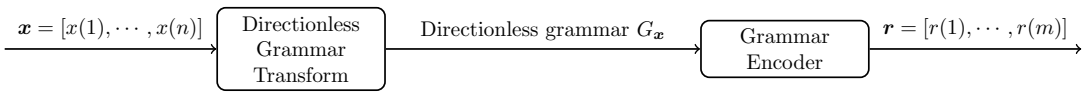


Figure 3.2: Structure of a dimension encoder

3.1.2 Directionless Grammar

Let $\mathcal{V} = \{v_0, v_1, v_2, \cdots\}$ be a countable infinite set disjoint with $\mathcal{N}$. Elements in $\mathcal{V}$ are called variables. Let Σ be a nonempty finite set disjoint with both $\mathcal{N}$ and $\mathcal{V}$. Elements in Σ are called terminal symbols. In the following discussion, we will use a singleton terminal symbol set, i.e. $\Sigma = \{\delta\}$.

A labeled set is a pair (S, L) such that S is a set and L is a labeling function which assigns a label $L(y)$ to each element $y \in S$, i.e.

$$(S, L) = \{(y, L(y)) : y \in S,\ L(y) \in V \cup \Sigma\}$$

. For each element $(i, L(i)) \in (S, L)$, integer i is called the position of the element while $L(i)$ is called the label of the element. We define a special labeling function L_δ so that it only take value from Σ, i.e. $L_\delta(y) = \delta$ for all $y \in \mathcal{N}$. For any set $U \subset \mathcal{V} \cup \Sigma$, let $\mathcal{S}(U)$ denote the set of all finite labeled sets (S, L) with $S \subset \mathcal{N}$ and L taking values over U.

We define two labeled sets (S_1, L_1) and (S_2, L_2) to be disjoint if S_1 and S_2 are disjoint.

We define an ordered set is a pair $(S, \leq)$ so that S is a set and $\leq$ is a transitive binary relation defined on S under which any pair of elements in S are comparable. For the sparse signal $\boldsymbol{x}$, $\mathcal{P}(\boldsymbol{x})$ along with the natural order of integers is an ordered set.

For $S \subset \mathcal{N}$ and $i \in \mathcal{N}$, define $S + i = \{j + i :\ j \in S\}$. We say $S + i$ is a translation of S by i. We can now generalize the definition of translation to labeled set. (S_2, L_2) is a translation of (S_1, L_1) by i if $S_1 + i = S_2$ and $L_1(j) = L_2(j + i)$ for every $j \in S_1$. In this case, we write $(S_2, L_2) = (S_1, L_1) + i$.

Let (S_1, L_1) and (S_2, L_2) be two subsets of (S, L), we say (S_1, L_1) and (S_2, L_2) are repeated subsets of (S, L) if they are disjoint and $(S_1, L_1) = (S_2, L_2) + i$ for some integer i.

We will now introduce two operations on labeled set which allow us to contract or expand a labeled set.

Labeled Set Subtraction with Trace Let (S_1, L_1) be a subset of (S, L), subtracting (S_1, L_1) from (S, L) with trace at $i \in S_1$ means that we first subtract (S_1, L_1) from (S, L) and then insert a new labeled element (i, v) into (S, L), where $v \in \mathcal{V}$ is a new variable which has never appeared in (S, L) before and is designated to represent the labeled set $(S_1, L_1) - i$.

Labeled Set Addition with Trace Let (S_1, L_1) and (S_2, L_2) be two labeled sets, suppose that $S_1 + i,\ i \in S_2$ is disjoint with $S_2 - \{i\}$. Adding (S_1, L_1) into (S_2, L_2)

with trace $i \in S_2$ means that we first delete the element $(i, L_2(i))$ from (S_2, L_2) and then insert all elements in $(S_1, L_1) + i$ into (S_2, L_2). Denote this operation by $(S_1, L_1) + (S_2, L_2)_i$. To be specific, we have

$$(S_1, L_1) + (S_2, L_2)_i = ((S_1, L_1) + i) \cup ((S_2, L_2) - \{(i, L_2(i))\})$$

if $i \in S_2$ and $(S_1 + i) \cap (S_2 - \{i\}) = \emptyset$. Otherwise, this operation is not defined.

Example 4 Consider the labeled set (S, L) given by

$$(S, L) = \{(1, \delta), (2, v_1), (8, \delta), (7, v_2), (14, v_1), (20, \delta)\}$$

We can see that it contains two disjoint subsets $\{(2, v_1), (8, \delta)\}$ and $\{(14, v_1), (20, \delta)\}$ which are a repetition of each other. Subtracting these two disjoint subsets from (S, L) with trace at $i = 2$ and $i = 14$ respectively, we can get

$$(S_1, L_1) = \{(1, \delta), (2, v_3), (7, v_2), (14, v_3)\}$$

with v_3 represents

$$(S_2, L_2) = \{(0, v_1), (6, \delta)\}$$

. Then add (S_2, L_2) to (S_1, L_1) with trace $i = 2$:

$$(S_3, L_3) = (S_2, L_2) + (S_1, L_1)_{i=2} = \{(1, \delta), (2, v_1), (7, v_2), (8, \delta), (14, v_3)\}$$

. If add S_2, L_2 to (S_3, L_3) with trace $i = 14$, we will get (S, L). If add S_2, L_2 to (S_3, L_3) with trace $i = 1$, the addition fails since it cause a collision at position 7.

Let $V \subset \mathcal{V}$ be a finite set containing v_0, given V and $\Sigma = \{\delta\}$, a directionless grammar with variable set V, terminal set Σ, and starting variable v_0 is a mapping G from V to $\mathcal{S}(V \cup \Sigma)$ such that

1. For any $v \in V$, $|G(v)| \geq 2$.
2. For any $v \neq v_0$, the labeled set $G(v)$ contains an element whose position is 0, i.e. $(0, u)$ for some $u \in (V \cup \Sigma)$

For each $v \in V$, $v \to G(v)$ is called the production rule corresponding to v and $G(v)$ is called the right hand side or right member of production rule. Give V, Σ, v_0 a directionless grammar G can be explicitly specified by its set of production rules $\{v \to G(v) : \ v \in V\}$.

Start with v_0 and for each element (i, v), $v \in V$ in $G(v_0)$, add $G(v)$ into $G(v_0)$ with trace i. One of the following situations will happen:

(a) One of the labeled set addition with trace fails since it is undefined.

(b) A collision happens in the sense that a new position j is inserted at least twice by different additions at different i.

(c) None of the above two happens, we get another labeled set.

If one of first two situations happens, declare a failure and stop. Otherwise, keep doing the addition process, then one of the followings will hold:

(1) The process terminates and we get a labeled set (S^*, L_δ).

(2) The process never ends because there always are at least one variable labeled elements in the labeled set.

If the process terminates with no failure reported and each $G(v),\ v \neq v_0$ is added at least once, during the whole process of addition, then G is said admissible.

Let G be an admissible grammar with variable set V, G is said to be localized if

1. each variable $v \in V$ other than v_0 appears at least twice in the right members of G, i.e. in $\{G(u) :\ u \in V\}$.
2. each variable $v \in V$ represents a distinct set under translation, i.e. the set represented by one variable can not be a translation of the set represented by another variable.
3. for each $v \in V$ and any two elements $(i, u_1), (j, u_2) \in G(v)$ where u_1 and u_2 are variables, if $i < j$, then the largest integer in the subset represented by (i, u_1) is less than the smallest integer in the subset represented by (j, u_2).

Let S_1 be a finite subset of $S \subset \mathcal{N}$, define

$$i_{min}(S_1) = \min\{i :\ i \in S_1\}$$

and

$$i_{max}(S_1) = \max\{i :\ i \in S_1\}$$

S_1 is said to be consecutive within S if the following argument holds:

$$\text{For each } j \in S, \text{ if } i_{min}(S_1) \leq j \leq i_{max}(S_1) \text{ then } j \in S_1$$

Admissible directionless grammar G with variable set V is said regular if it is localized and for each $v \in V$ and any element $(i, u) \in G(v)$, where u is a variable, the subset represented by (i, j) is consecutive within the set represented by v.

3.1.3 Incremental Directionless Grammar Transform

Given $x \in \mathcal{X}_n$, start with the trivial grammar

$$v_0 \to (\mathcal{P}(\boldsymbol{x}), L_\delta) \tag{3.1}$$

Sort elements $(i, L_\delta(i))$ in $(\mathcal{P}(\boldsymbol{x}), L_\delta)$ in the increasing order of position i:

$$(\mathcal{P}(\boldsymbol{x}), L_\delta) = \{(i_j, L_\delta(i_j))\}_{j=1}^{\omega(\boldsymbol{x})} \text{ where } i_1 < i_2 < \cdots < i_{\omega(\boldsymbol{x})}$$

Partition $(\mathcal{P}(\boldsymbol{x}), L_\delta)$ sequentially, in the increasing order of position i, into disjoint non-empty subsets $S_1, S_2, \cdots, S_t$ such that

(a) $S_1 = \{(i_1, L_\delta(i_1))\}$, and

(b) for $k > 1$,

$$S_k = \{(i_j, L_\delta(i_j))\}_{j=j(k)}^{l}$$

where

$$j(k) = 1 + \sum_{f=1}^{k-1} |S_f|$$

and l is the smallest integer such that translation of $\{(i_j, L_\delta(i, j))\}_{j=j(k)}^{l}$ have not appeared in $S_1, S_2, \cdots, S_{k-1}$ if such an integer exists, and equal to $\omega(\boldsymbol{x})$ otherwise.

The above partition share the same spirit of the Lempel-Ziv incremental parsing of strings [28] and is referred to as the incremental partition. It is not hard to verify that the incremental partition has the following properties:

1. All subsets $S_1, S_2, \cdots, S_t$ except, possibly, S_t, are distinct under translation.
2. Each distinct (under translation) subset S_k in $\{S_1, S_2, \cdots, S_t\}$ with $|S_k| \geq 3$ is equal to the union of a translation of S_i and $(i_l, L_\delta(i_l))$ for some $i < k$.

Rewrite the production rule in (3.1) as

$$v_0 \to \{S_1, S_2, \cdots, S_t\} \tag{3.2}$$

For $1 \leq k \leq t$, let p_k be the smallest position in S_k, let $v(S_k)$ be δ if $|S_k| = 1$ and $\{(i_j - j_{j(k)}, L_\delta(i_j))\}_{j=j(k)}^{l}$ if $S_k = \{(i_j, L_\delta(i, j))\}_{j=j(k)}^{l}$ with $|S_k| \geq 2$. Now, S_k can be uniquely represented by $(p_k, v(S_k))$.

Replace S_k in (3.2) by $(p_k, v(S_k))$, we can get

$$v_0 \to \{(p_1, v(S_1)), (p_2, v(S_2)), \cdots, (p_t, v(S_t))\} \tag{3.3}$$

Treat $\{v(S_k) \ : |S_k| \geq 2, 2 \leq k \leq t\}$ as the variable set and write $v(S_k)$ as

$$v(S_k) \to \{(i_j - i_{j(k)}, L_\delta(i_j))\}_{j=j(k)}^{l} \tag{3.4}$$

if $|S_k| = 2$ and as

$$v(S_k) \to \{(0, v(S_i)), (i_l - i_{j(k)}, L_\delta(i_l))\} \tag{3.5}$$

if $|S_k| \geq 3$. In (3.5), S_k is equal to the union of a translation of S_i for some $i < k$ and $\{i_l, L_\delta(i_l)\}$, which we can see from the second property of incremental partition.

Now (3.3) to (3.5) defines a directionless grammar $\hat{G}$. Prune $\hat{G}$ by expanding each element labeled with a variable that appears only once so that every variable other than v_0 appears at least twice on the right-hand side of all the production rules. Then remove all the useless variables and relabel the remaining variables. The resulting directionless grammar represents $\mathcal{P}(\boldsymbol{x})$ and is denoted by $G_{\boldsymbol{x}}$. The mapping from $\boldsymbol{x}$ to $G_{\boldsymbol{x}}$ is called the IDGT. It is easy to see that the grammar G_x obtained from IDGT is regular.

Example 5 In this example, we will illustrate the IDGT process. We choose a sparse signal $\boldsymbol{x}$ with

$$\mathcal{P}(\boldsymbol{x}) = \{3, 5, 7, 10, 13, 15, 17, 20, 24, 27, 31, 33\}$$

Incremental partition $(\mathcal{P}(\boldsymbol{x}), L_\delta)$, we get

$$\begin{aligned} S_1 &= \{(3, \delta)\} \\ S_2 &= \{(5, \delta), (7, \delta)\} \\ S_3 &= \{(10, \delta), (13, \delta)\} \\ S_4 &= \{(15, \delta), (17, \delta), (20, \delta)\} \\ S_5 &= \{(24, \delta), (27, \delta), (31, \delta)\} \\ S_6 &= \{(33, \delta)\} \end{aligned}$$

Note that S_4 is the union of $S_2 + 10$ and $\{(20, \delta)\}$, S_5 is the union of $S_3 + 14$ and $\{(31, \delta)\}$. The directionless grammar $\hat{G}$ is:

$$\begin{aligned} v_0 &\to \{(3, \delta), (5, v_1), (10, v_2), (15, v_3), (24, v_4), (33, \delta)\} \\ v_1 &\to \{(0, \delta), (2, \delta)\} \\ v_2 &\to \{(0, \delta), (3, \delta)\} \\ v_3 &\to \{(0, v_1), (5, \delta)\} \\ v_4 &\to \{(0, v_2), (7, \delta)\} \end{aligned}$$

We can see that v_3 and v_4 appear only once on the right hand of $\hat{G}$. Further prune $\hat{G}$ gives G_x:

$$\begin{aligned} v_0 &\rightarrow \{(3,\delta),(5,v_1),(10,v_2),(15,v_1),(20,\delta),(24,v_2),(31,\delta),(33,\delta)\} \\ v_1 &\rightarrow \{(0,\delta),(2,\delta)\} \\ v_2 &\rightarrow \{(0,\delta),(3,\delta)\} \end{aligned} \tag{3.6}$$

3.1.4 Grammar encoding

After a directionless grammar $G_{\boldsymbol{x}}$ is obtained through a directionless grammar transform, it needs to be further encoded to vector $\boldsymbol{r}$. Here we only consider the directionless grammar generated by IDGT, for the grammar encoding of a general directionless grammar, please refer to [12].

Suppose $\hat{G}$ is the grammar defined in (3.3) to (3.5) and prune $\hat{G}$ we get G. From the second property of incremental partition in Sec. 3.1.3 we can easily concludes that all the variable in $\hat{G}$ except v_0 are of size 2. Let u, v be two variables that $u \neq v_0,\ v \neq v_0$ and $\hat{G}(u)$ contains an element labeled by v. The production rule of u is of the following form:

$$u \rightarrow \{(0,v),(\Delta_u,\delta)\}$$

Where Δ_u is some positive integer. Since v represents a partition of $\boldsymbol{x}$ and must have already appeared in $G(v_0)$. Hence, pruning will not expand $(0,v)$ in $\hat{G}(u)$. As a result, pruning will only expand variable labeled element in $\hat{G}(v_0)$. As a result, the following property of G holds:

(1) for any $v \neq v_0 \in V$, $|G(v)| = 2$.

(2) $G(v) = \{(0,l),(\Delta_v,\delta)\}$ where $l \in V \cup \Sigma$ and $\Delta_v \in \mathcal{N}^+$

Having the above property, we can now start to encode the directionless grammar obtained by IDGT.

STEP 1 All the variable labeled elements in $G(v_0)$ are sorted in increasing order by their positions and then followed by terminal symbol labeled elements sorted again by their positions in increasing order. Let $|G(v_0)|_1$ denotes the number of terminal symbol labeled elements in $G(v_0)$ while $|G(v_0)|_0$ denotes the number of variable labeled elements in $G(v_0)$.

STEP 2 Set $r(1) = 2|G(v_0)|_0$, $r(2) = r(1) + |G(v_0)|_1$ and $r(3) = r(2) + 2(|V| - 1)$

STEP 3 Follow the order we get in STEP 1, append all elements in $G(v_0)$ to $\boldsymbol{r}$

(1) For a variable labeled element (Δ, v), append Δ then v.

(2) For a terminal symbol labeled elements (Δ, δ), only Δ is appended.

STEP 4 For variable $v \in \{v_1, v_2, \cdots, v_{|V|-1}\}$, $G(v) = \{(0, l), (\Delta_v, \delta)\}$ first append l then Δ_v to $\boldsymbol{r}$.

STEP 5 Replace all the terminal symbol δ in $\boldsymbol{r}$ by 0, and variable v_t by t.

It is easy to show that if each number in $\boldsymbol{r}$ is represented by fixed number of bis, then $\boldsymbol{r}$ is a prefix code.

Example 6 Revisit the directionless grammar G_x in (3.6). After STEP 1, we have the following grammar:

$$\begin{aligned} v_0 &\rightarrow \{(5, v_1), (10, v_2), (15, v_1), (24, v_2), (3, \delta), (20, \delta), (31, \delta), (33, \delta)\} \\ v_1 &\rightarrow \{(0, \delta), (2, \delta)\} \\ v_2 &\rightarrow \{(0, \delta), (3, \delta)\} \end{aligned}$$

And we have $|G(v_0)|_0 = 4$, $|G(v_0)|_1 = 4$ and $|V| = 3$. Hence $r(1) = 8$, $r(2) = 12$, $r(3) = 16$. The final vector $\boldsymbol{r}$ is:

$$\boldsymbol{r} = [8, 12, 16, 5, 1, 10, 2, 15, 1, 24, 2, 3, 20, 31, 33, 0, 2, 0, 3] \tag{3.7}$$

3.1.5 Random access decoder

As in Sec. 3.1.4, here we only introduce the random access decoder based on IDGT, for the description of a general random access decoder, please refer to [12]. Given a vector $\boldsymbol{r}$ generated by the algorithm in Sec. 3.1.4 and $l \in \mathcal{N}^+$ denotes the position in $\boldsymbol{x}$ we want to access. The output of random access decoder is a boolean value

$$S(l) = \begin{cases} 1 & l \in \mathcal{P}(\boldsymbol{x}) \\ 0 & \text{Otherwise} \end{cases}$$

Let $\boldsymbol{r}[a; b]$, where $a \leq b$, denote a sub-vector from position a to position b of r:

$$\boldsymbol{r}[a;\ b] = [r(a), r(a+1), \cdots, r(b)]$$

The random access decode algorithm under IDGT shows as follows:

STEP 1 Binary search l in $\boldsymbol{r}[r(1)+4;\ r(2)+3]$. If l can be found return 1.

STEP 2 If $r(1)=0$ or $l<r(4)$ return 0.

STEP 3 Binary search for $r(p)$ in $\boldsymbol{r}[4;\ r(1)+3]$ so that p is even and $r(p)\leq l$. If $r(p)=l$ return 1. Otherwise, set $v=r(p+1)$ and update the value of l to $l-r(p)$.

STEP 4 If $r(r(2)+2v+3)=l$ return 1.

STEP 5 If $r(r(2)+2v+2)=0$ return 0. Otherwise, set v to $r(r(2)+2v+2)$ and goto STEP 4.

Example 7 In this example, we will use the vector $\boldsymbol{r}$ given by (3.7) in Exmp. 6.

$l_1=31$**:** In STEP 1, 31 can be found in $r[12;\ 16]$ and the algorithm returns 1.

$l_2=10$**:** In STEP 2, $r(6)=10$ will be found in the binary search and the algorithm gives 1.

$l_3=13$**:** $p=6$ will be found in the binary search in STEP 3. v is then set to 2 and l_3 is set to $13-r(6)=3$. We have $r(2)+2v+3=19$. Because $r(19)=l_3$, it gives 1.

$l_4=9$**:** $p=4$ will be given in STEP 3. Then $v=1$ and l_4 is set to 4. We have $r(2)+2v+3=17$. Since $r(17)\neq 4$, we will go to STEP 5. $r(16)$ is found 0, in this case, the algorithm terminates and gives 0.

Regarding the time complexity of random access decoder, the binary search applied in STEP 1 and STEP 3 is $O(\log\omega(\boldsymbol{x}))$. Let $d(\boldsymbol{x})$ be the depth of the derivation tree of $G_{\boldsymbol{x}}$. STEP 4 and STEP 5 will iterate at most $d(\boldsymbol{x})$ times, which results in $O(d(\boldsymbol{x}))$ time. The time complexity in total is $O(\omega(\boldsymbol{x})+d(\boldsymbol{x}))$. The author of [12] shows that for most $\boldsymbol{x}$, $d(\boldsymbol{x})=O(\omega(\boldsymbol{x}))$. Therefore, we can conclude that the time complexity of random access decoder for most $\boldsymbol{x}$ is $O(\omega(\boldsymbol{x}))$.

3.2 Scan Methods

A scan method is a one-to-one mapping that transforms a 2-dimensional matrix M into 1-dimensional vector $\boldsymbol{s}$. In this section, we will first introduce the raster scan which is the most intuitive and straightforward scan method and then the Hilbert scan which is capable of preserving the locality of points.

3.2.1 Raster Scan

In the raster scan method, the whole matrix is scanned in a top-to-bottom and left-to-right manner. To be specific, suppose all indices start from 0, given a $n \times m$ matrix $M_{n \times m}$ and an element at position (x, y), after the raster scan, this element will have index $xm+y$ in resulted vector $\boldsymbol{s}$. Given an index d in $\boldsymbol{s}$, the corresponding x, y can be obtained by $x = \lfloor d/m \rfloor$, and $y = d \ \% \ m$. here "%" denotes modular.

Fig. 3.3 shows the raster scan on a 4×8 matrix. Note that the scan curve is not continuous. When the scan reaches the right-most element in a row, it will jump to the left-most element of next row. The jumps are denoted by dashed lines in Fig. 3.3.

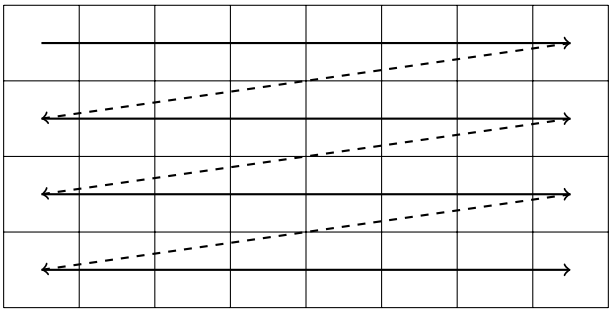


Figure 3.3: Illustration of raster scan

3.2.2 Hilbert Scan

There are various scan methods that focus on maintaining the locality of 2-D points. Scan methods using space filling curve is typical examples of such methods among which Hilbert scan is the most popular one[21].

Hilbert scan requires the matrix to be a square matrix with the length of both sides being a power of 2. In this case, before the scanning, we first pad 0s into the right and bottom of the matrix until it becomes a square matrix with the height equal to 2^k, $k \in \mathcal{N}^+$. Note that unlike image compression, the padding here will not cause any trouble since all the non-zero elements have the same coordinates after padding while the zero elements are ignored in the later representation of the sparse matrix. Fig. 3.2.2 shows how Hilbert scan works on 2×2, 4×4 and 8×8. We can see that the scan curve of Hilbert scan is continuous.

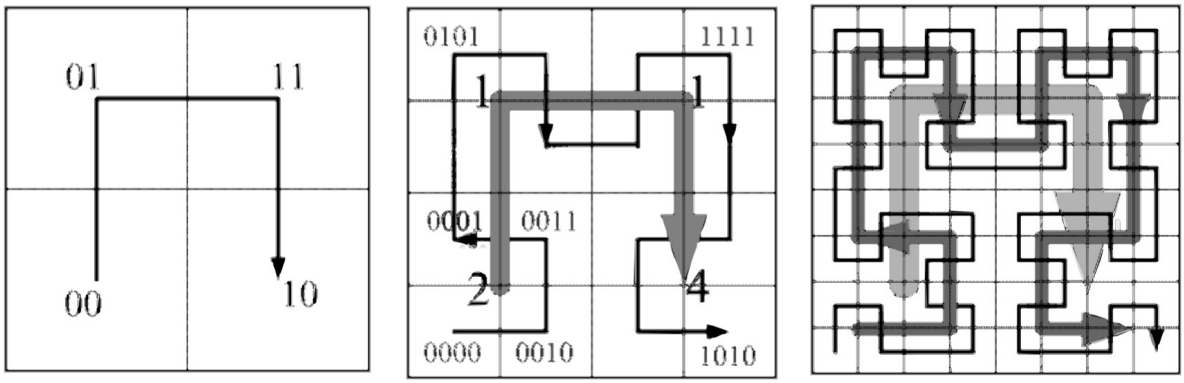


Figure 3.4: Illustration of Hilbert scan. (a) 2×2 (b) 4×4 (c) 8×8

Let (x, y) be the coordinates of a point in the $n \times n$ matrix $M_{n \times n}$, where n is some power of 2, and d be the length along the scan curve when it reaches that point. We say Hilbert scan can preserve locality in the sense that, given two points in the matrix, say, (x_1, y_1) with curve length d_1 and (x_2, y_2) with curve length d_2, if the difference between d_1 and d_2 is small then the distance between (x_1, y_1) and (x_2, y_2) is also small. Note that the converse is not always true. There will sometimes be points where the (x, y) coordinates are close but their d values are far apart (For example, $(2, 1)$ and $(2, 2)$ in 3.4b). Compared with Hilbert scan, raster scan does not have the above property. Think of $(0, 7)$ and $(1, 0)$ in Fig. 3.3. These two points are consecutive elements in the vector after scan. However, they are far away from each other in the matrix.

When representing a sparse binary matrix, it makes more sense to have algorithms to map (x, y) to d and vice versa. The following algorithm provides mappings in both directions, meanwhile, it is implemented with iteration rather than recursion. In function XY2D, D2XY and ROT, all the arguments of these functions are passed by reference i.e. the change of the arguments in the functions is still valid when the function ends. Also, "$\ll$" denotes bit-wise left-shift, "$\gg$" denotes bit-wise right-shift, "&" is bit-wise and, "$\oplus$" is bit-wise xor. Function SWAP simply swap the values of its two arguments.

We can see that the function ROT runs in constant time. The while loop in XY2D executes at most $\lceil \log_2 n \rceil$ times, so does the while loop in D2XY. As a result, conversion between (x, y) and d takes $O(\log n)$ time, where n is the height of the square matrix.

```
XY2D(n, x, y)
1  r_x, r_y, s = n ≫ 1, d = 2
2  while s > 0
3        r_x = (x & s) > 0
4        r_y = (y & s) > 0
5        d = d + s^2 · ((3 · r_x) ⊕ r_y)
6        ROT(s, x, y, r_x, r_y)
7        s = s ≫ 1
8  return d
```

```
D2XY(n, d, x, y)
 1  r_x, r_y, s = 1, t = d
 2  while s < n
 3        r_x = 1 & (t ≫ 1)
 4        r_y = 1 & (t ⊕ r_x)
 5        ROT(s, x, y, r_x, r_y)
 6        x = x + s · r_x
 7        y = y + s · r_y
 8        t = t ≫ 2
 9        s = s ≪ 1
10  return (x, y)
```

```
ROT(n, x, y, r_x, r_y)
1  if r_y == 0
2        if r_x == 1
3              x = n − 1 − x
4              y = n − 1 − y
5        SWAP(x, y)
```

3.3 Sparse Binary Matrix coder via Dimension coding

Let the $n \times m$ sparse binary matrix to be represented is $M_{n\times m}$. $\mathcal{P}(M_{n\times m})$ denotes the set of the positions of all non-zero elements in $M_{n\times m}$. In this case, given n and m, $M_{n\times m}$ is equivalent to $\mathcal{P}(M_{n\times m})$. As in sparse matrix representation, zero elements are usually ignored, we will use $\mathcal{P}(M_{n\times m})$ as our input. $\omega(M_{n\times m})$ denote the number of non-zero

elements in $M_{n\times m}$. Recall the $\mathcal{P}(\boldsymbol{x})$ is the set of positions of non-zero elements in vector $\boldsymbol{x}$. A scan method s can be specified by a pair of function $xy2d_s(x,y)$ and $d2xy_s(d)$ that are the inverse function of each other. T_{xy2d} and T_{d2xy} are the time complexity of $xy2d$ and $d2xy$.

The structure of the sparse binary matrix coder via dimension coding called 2-D dimension coder is shown in Fig. 3.5.

Note that the random access capability of dimension coding is still maintained in Fig. 3.5, however, the time complexity of recovery decoder may, possibly, if some high complexity scan method is applied, increase. If raster scan is applied, where the mappings $xy2d$ and $d2xy$ are $O(1)$, 2-D dimension coder will have the same time complexity as dimension coder in both random access decoder and recovery decoder. If Hilbert scan is applied, where the time complexity of convert between 1-D and 2-D is $\log n$, the random access decoder will then have time complexity $O(\log(\omega(M_{n\times m}))+\log n)\leq O(\log nm+\log n)=O(\log n)$, while the recovery decoder has time complexity $O(\omega(M_{n\times m})\cdot\log n)$.

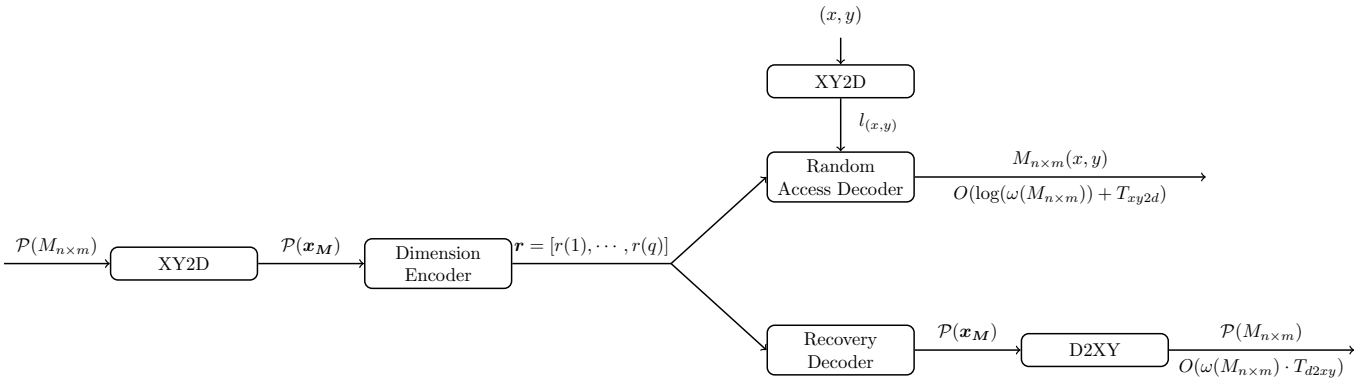


Figure 3.5: Structure of a 2-D dimension coder

Chapter 4

Binary Matrix Representation via Bipartite Gramamr Based Coding

4.1 Overview

Recall that in CFG defined in grammar based coding, each $R(v)$, $v \in V$ represents a string from $(V \cup \Sigma)^+$, hence, CFGs defined here as well as in computation theory [13] are direction oriented. In other words, entries in $R(v)$ are ordered from left to right; replacement of v by $R(v)$ in the parallel replacement procedure follows the same order constraint; and re-ordering entries in $R(v)$ would result in different CFGs representing different x. This is further illustrated in the following example.

Example 8 Let us revisit the CFG G in Example 3, Re-ordering some entries in $R(v_0)$ and $R(v_3)$ in G, we get a new CFG G':

$$\begin{aligned} v_0 &\rightarrow v_2v_3v_1v_2v_3100 \\ v_1 &\rightarrow 01 \\ v_2 &\rightarrow v_11 \\ v_3 &\rightarrow v_2v_1 \end{aligned}$$

G' or v_0 now represents 011011010101101101100 with v_1, v_2, v_3 representing substrings 01, 011, and 01101, respectively.

When grammar-based coding was initially formulated in [20], CFGs were used to represent only strings from Σ. Later on, grammar-based coding and CFGs were also extended

to images and binary trees in limited settings [18], [17]. However, the direction oriented constraint in CFGs makes them inapplicable to images and graphs in general.

Regarding the directionless grammar which we have used in dimension coding in the previous chapter, even though it can overcome the direction oriented constraint, to represent a 2-D matrix, it needs to be used together with a scan method, which is not capable of preserving all the neighborhood of an element. For example, even if we use Hilbert scan, which is thought better to preserve the neighborhood. Hilbert scan works in a quadrant by quadrant manner, the block boundary still exist and the relationship between elements across this boundary can not be preserved.

In this chapter, we will introduce the new concept of CFBG for compactly representing sparse matrices. In contrast with CFGs and directionless grammar, CFBGs are directionless with their variables capable of representing any subgraph of a given bipartite graph, which gives CFBG true 2-D nature and, as a result, makes it very suitable for representing the grammar.

The structure of coder that applies CFBG is shown in Fig. 4.1. Similar to the process of dimension coding, a matrix $M_{n\times m}$ is first transformed into a CFBG, which is then represented by a vector $\boldsymbol{r}$.

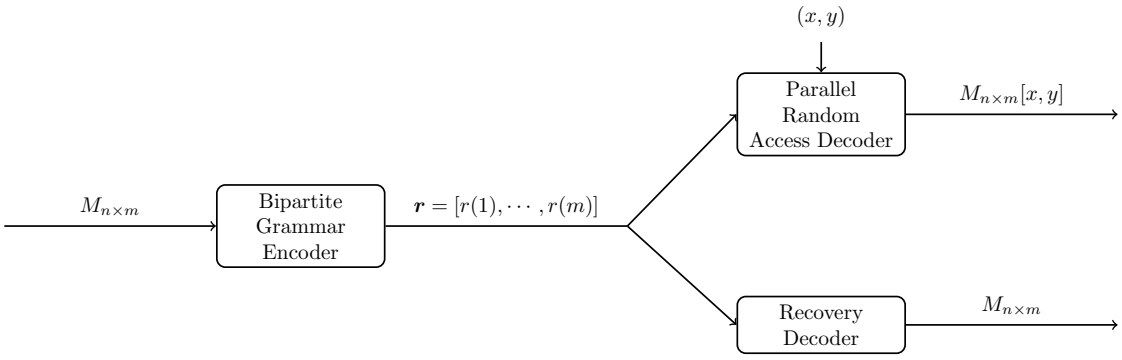


Figure 4.1: Structure of a bipartite grammar coder

In the following discussion of this chapter, we will first show how to identify a matrix by a bipartite graph in Sec. 4.2, then give the definition of CFBG in Sec. 4.3. Sec. 4.4 shows how to convert a general CFBG into its canonical form and then represent it by $\boldsymbol{r}$. After $\boldsymbol{r}$ is obtained, Sec. 4.5 shows how to parallel random access an element of matrix through $\boldsymbol{r}$. Finally, in Sec. 4.6, we will propose two grammar transformations.

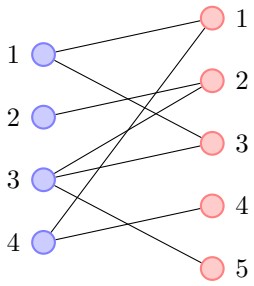
4.2 Correspondence between binary matrix and bipartite

Consider a bipartite graph $B = (X, Y, E)$, where X and Y are two disjoint vertex sets, and $E \subset X \times Y$ is the set of edges. Without loss of generality, we assume that both X and Y are subsets of $\mathcal{N}$ with the understanding that $i \in X$ and $i \in Y$ are two distinct vertices with the former coming from X and the latter from Y. We define $I(n)$ to be the set of all integers from 1 to n, i.e. $I(n) = \{1, 2, \cdots, n\}$.

When $X = I(n)$ and $Y = I(m)$ for some $n, m \in \mathcal{N}^+$, the biadjacency matrix of $B = (X, Y, E)$ is an $n \times m$ binary matrix $M_{n\times m}$ defined as

$$M_{n\times m}[i, j] = \begin{cases} 1 & \text{if } (i, j) \in E \\ 0 & \text{Otherwise} \end{cases} \tag{4.1}$$

With this correspondence, one can identify any $n \times m$ binary matrix $M_{n\times m}$ with a bipartite graph $B = (I(n), I(m), E)$, the biadjacency matrix of which is equal to $M_{n\times m}$. As a result, rather than represent the original matrix $M_{n\times m}$, we can now represent the bipartite graph B. Fig. 4.2 illustrates the correspondence between B and $M_{n\times m}$.



(a)

	1	2	3	4	5
1	1	0	1	0	0
2	0	1	0	0	0
3	0	1	1	0	1
4	1	0	0	1	0

(b)

Figure 4.2: (a) A bipartite graph B; (b) the biadjacency matrix of B

4.3 Context-free Bipartite Grammar

The same as in the previous discussions, let $V = \{v_0, v_1, \cdots\}$ be a finite set of variables where v_0 denotes the starting variable and singleton $\Sigma = \{\delta\}$ be the set of terminal symbol.

We define a *labeled bipartite graph* is a quadruple $A = (X, Y, E, L)$, where X, Y, and E have the same meanings as in a bipartite graph B, and $L : E \to V \cup \Sigma$ assigns a label $L(e)$ to each edge $e \in E$. L_δ is a special labeling function that takes value only from Σ, i.e. $\forall e \in E,\ L_\delta(e) = \delta$.

A bipartite graph B that identify $M_{n\times m}$ can also be regarded as a labeled bipartite graph with all edges labeled as δ. In this case, we say the labeled bipartite graph $B = (I(n), I(m), E, L_\delta)$ identify the matrix $M_{n\times m}$.

We define a labeled edge as a triple $(x, y, L(x, y))$, where $x \in X, y \in Y$. The pair (x, y) is called the position of the edge while $L(x, y)$ is called the label of the edge. If $L(x, y) \in V$ we say the edge is variable labeled. Otherwise, i.e. $L(x, y) \in \Sigma$, we say the edge is terminal symbol labeled.

We say a labeled bipartite graph $A_s = (X_s, Y_s, E_s, L_s)$ is a *subgraph* of $A = (X, Y, E, L)$ if (1) $X_s \subseteq X$ (2) $Y_s \subseteq Y$ (3) $E_s \subseteq E$ and (4) $L_s(x, y) = L(x, y)$ for any $(x, y) \in E_s$.

The subgraph A_s is said to be *vertex-induced* if $E_s = \{(x, y) : x \in X_s, y \in Y_s$ and $(x, y) \in E\}$, and *edge-induced* if $X_s = \{x : (x, y) \in E_s$ for some $y\}$ and $Y_s = \{y : (x, y) \in E_s$ for some $x\}$.

Two subgraphs A_s and $\hat{A}_s$ of $A = (X, Y, E, L)$ are said *disjoint* if the edge sets of A_s and $\hat{A}_s$ are disjoint.

When $X = I(n)$ and $Y = I(m)$ for some $n, m \in \mathcal{N}^+$, the biadjacency matrix of A is defined in a manner similar to (4.1) except that $M_{n\times m}[i, j] = L(i, j)$, i.e.

$$M_{n\times m}[i, j] = \begin{cases} L(i, j) & \text{if } (i, j) \in E \\ 0 & \text{Otherwise} \end{cases} \tag{4.2}$$

Example 9 In the bipartite graph $B = (I(4), I(5), E, L_\delta)$ shown in Fig. 4.2a. Consider three subgraphs: B_1 which induced by edge set $\{(1, 1), (2, 2), (1, 3)\}$, B_2 which induced by edge set $\{(3, 3), (4, 4), (3, 5)\}$ and B_3 which induced by edge set $\{(2, 2), (3, 2), (3, 3)\}$.

B_1 and B_2 are disjoint subgraphs of B. B_3 and B_1 are not disjoint since $(2, 2)$ is the common edge in both edge sets.

4.3.1 Graph Operations

In CFBG, we start with a bipartite graph and create, through graph operations, many labeled bipartite graphs.

Graph Translation Let $A_1 = (X_1, Y_1, E_1, L_1)$ and $A_2 = (X_2, Y_2, E_2, L_2)$ be two labeled bipartite graphs. A_2 is said to be a translation of A_1 if there exists (i, j) such that $X_2 = X_1 + i$, $Y_2 = Y_1 + j$, $E_2 = \{(k+i, l+j) : (k, l) \in E_1\}$, and $L_2(k+i, l+j) = L_1(k, l)$ for any $(k, l) \in E_1$, in which case we write A_2 as $A_2 = A_1 + (i, j)$.

With the translation, we can further define two disjoint subgraphs A_s and $\hat{A}_s$ to be a *repetition* of each other (i.e., equivalent) if $\hat{A}_s = A_s + (i, j)$ for some (i, j). In Exmp. 9, B_1 and B_2 are repetition of each other.

Graph Subtraction Let $A_s = (X_s, Y_s, E_s, L_s)$ be a subgraph of A. Let (k, l) be an edge in E_s. By subtracting A_s from A at edge (k, l), we mean deleting all edges in $E_s - \{(k, l)\}$ from A and then changing the label of edge (k, l) to a new label which never appeared before in A. The new label represents the graph $A_s - (k, l)$.

If $A_s + (i, j)$ is a subgraph of A, disjoint with A_s, by subtracting A_s and its repetition $A_s + (i, j)$ simultaneously from A at edge (k, l), we mean subtracting A_s and $A_s + (i, j)$ from A at edges (k, l) and $(k + i, l + j)$, respectively, and changing the label of edges (k, l) and $(k + i, l + j)$ to the same new label which never appeared before in A. The new label represents the graph $A_s - (k, l)$. The same principle applies to subtracting more than two repetitions.

Graph Addition Let A_1 and A_2 be two labeled bipartite graphs. Let (k, l) be an edge in A_2. By adding A_1 to A_2 at edge (k, l), we mean first deleting edge (k, l) from A_2 and then inserting all edges in $A_1 + (k, l)$ along with their labels into A_2 with edges in $A_1 + (k, l)$ overwriting any pre-existing edges in A_2 whenever duplication occurs.

Example 10 Fig. 4.3 illustrates graph subtraction and addition for the bipartite graph B shown in Fig. 4.2.

4.3.2 Context-free Bipartite Grammar

When the vertex sets are known, a labeled bipartite graph A can be conveniently identified with the set consisting of all its edges, called the edge set of A. For example, the graph B' in Fig. 4.3c can be identified with its edge set $\{(1, 1, \delta), (1, 3, \delta), (3, 2, v_1), (3, 5, \delta), (4, 1, \delta), (4, 4, \delta)\}$. For any set Ω, denote the set of all possible finite edge sets with labels taking values over Ω by $\mathcal{B}(\Omega)$.

We define a CFBG is a quadruple $G = (V, \Sigma, R, v_0)$, where

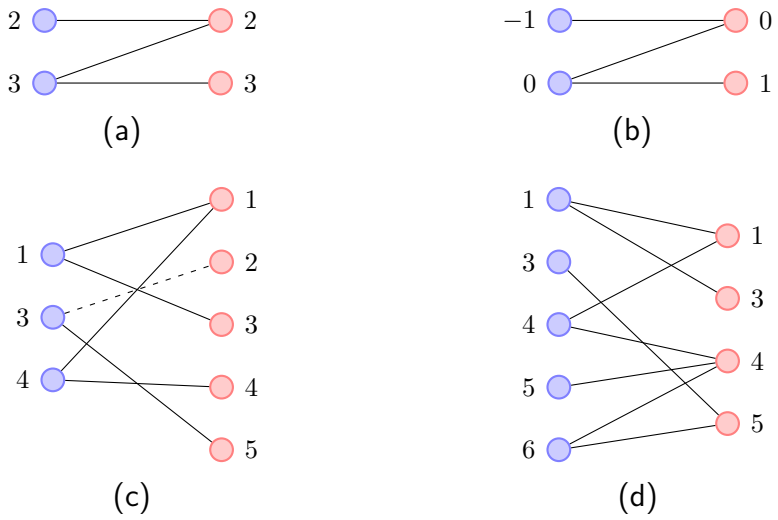


Figure 4.3: Operations over B in Fig. 4.2a with solid edges labeled as δ and dash edges labeled as v_1: (a) subgraph B_3 induced by $\{(2,2)(3,2)(3,3)\}$; (b) bipartite graph $B_3-(3,2)$ indicated by label v_1; (c) graph B' obtained by subtracting B_3 from B at edge $(3,2)$; (d) graph obtained by adding B_3 to B' at edge $(3,2)$. Adding $B_3-(3,2)$ to B' at edge $(3,2)$ gets B back.

- V is a finite nonempty set whose elements are called variables, and $v_0 \in V$ is a special variable called the start variable;
- Σ is another finite nonempty set disjoint from V, whose elements are called terminal symbols; and
- R is a mapping from V to $\mathcal{B}(V \cup \Sigma)$ satisfying that $R(v)$ contains the edge $(0,0)$ for any $v \neq v_0$.

The relationship $v \to R(v)$ is once again called the production rule corresponding to v and $R(v)$ is the right hand side or right member of the production rule. Note that each $R(v)$ is a labeled bipartite graph.

Start with v_0 and add in parallel, for each edge (x, y, v) in $R(v_0)$ with variable label v, $R(v)$ to $R(v_0)$ at edge (x, y). If there is no collision, i.e., no parallel edges would be inserted, we then get another edge set in $\mathcal{B}(V \cup \Sigma)$. Otherwise, a failure should be reported and the parallel addition ends. If we keep doing this parallel addition and no collision would occur, then one of the following holds:

(1) We arrive at an edge set in $\mathcal{B}(\Sigma)$ after finitely many steps

(2) The procedure never ends.

The CFBG G is said admissible if there is no collision in every step of parallel addition, each $R(v)$ is added at least once, and we finally arrive at an edge set in $\mathcal{B}(\Sigma)$; in this case, G or v_0 is said to represent the graph corresponding to the final edge set in $\mathcal{B}(\Sigma)$ or equivalently its biadjacency matrix.

Example 11 Let $\Sigma = \{\delta\}$ and $V = \{v_0, v_1, v_2\}$. The following production rules give an admissible CFBG:

$$\begin{aligned} v_0 &\to \{(1,1,v_1),(3,3,v_1),(4,1,v_2)\} \\ v_1 &\to \{(0,0,\delta),(1,1,v_2)\} \\ v_2 &\to \{(0,0,\delta),(-1,1,\delta)\} \end{aligned}$$

Start with v_0 and perform parallel addition:

$$\begin{aligned} v_0 &\to \{(1,1,\delta),(2,2,v_2),(3,3,\delta),(4,4,v_2), \\ &\quad (4,1,\delta),(3,2,\delta)\} \\ &\to \{(1,1,\delta),(2,2,\delta),(1,3,\delta),(3,3,\delta), \\ &\quad (4,4,\delta),(3,5,\delta),(4,1,\delta),(3,2,\delta)\} \end{aligned}$$

It can be verified that G represents the graph B in Fig. 4.2a G also indicates that B contains a repeated subgraph which appears at edges $(1,1)$ and $(3,3)$ and is a translation of the graph represented by v_1.

Given an admissible CFBG G, define the size of G as $|G| = \sum_{v\in V} |R(v)| - (|V| - 1)$. Since each $R(v)$ in G is a set, there is no order among the set elements of $R(v)$ and hence G is directionless. It is this directionlessness that allows CFBGs to provide compact representations for graphs in $\mathcal{B}(\Sigma)$ by capturing repeated subgraphs of any kind. In Exmp. 11, $|G| = 5$ whereas the graph B represented by G has 8 edges.

Given an admissible CFBG G representing $B \in \mathcal{B}(\Sigma)$, let $\{B_j\}_{j=0}^{d}$ be the sequence of edge set obtained during the parallel expansion process of G, where $B_0 = G(v_0)$ and $B_d = B$. We can associate a tree called derivation tree with the repeated parallel expansion process. The root of the tree is labeled with v_0 and other nodes correspond one to one to elements of the edge set. The derivation tree of CFBG shown in Exmp. 11 is shown in Fig. 4.3.2. The value d, which indicates the number of iterations in the repeated expansion process, is called the depth of derivation tree.

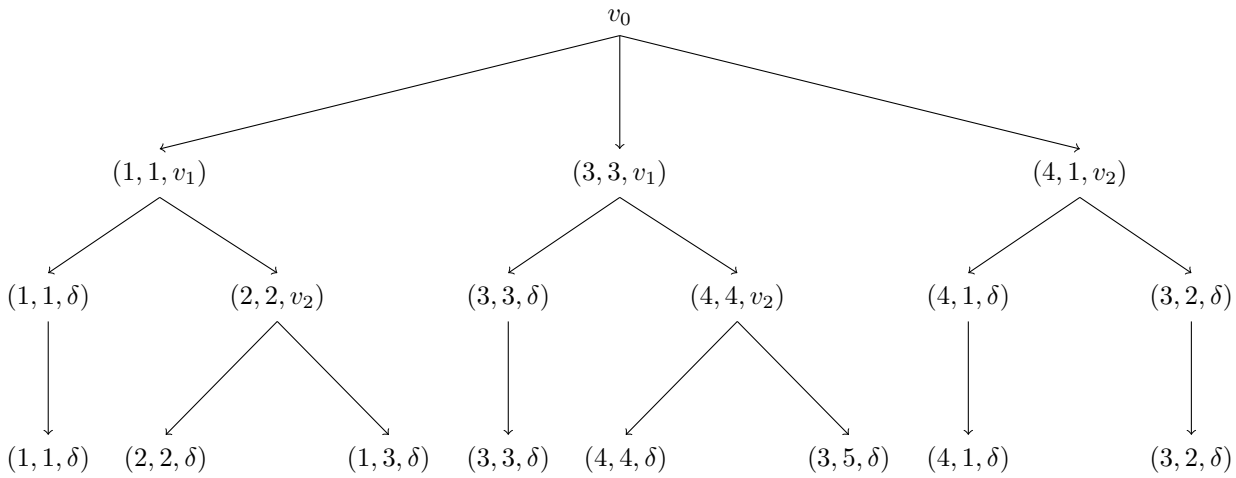


Figure 4.4: The derivation tree of CFBG in Exmp. 11

4.4 Bipartite Grammar Encoding

The structure of the block "Bipartite Grammar Encoder" in Fig. 4.1 is shown in 4.5, once we get a admissible CFBG $G_{M_{n\times m}}$ represents $B_{M_{n\times m}} \in \mathcal{B}$, through grammar transform, we feed $G_{M_{n\times m}}$ to grammar encoder to get final representation vector $\boldsymbol{r}$.

In the grammar encoder, the CFBG is first rewritten into its canonical form $\hat{G}$ and then encoded into $\boldsymbol{r}$.

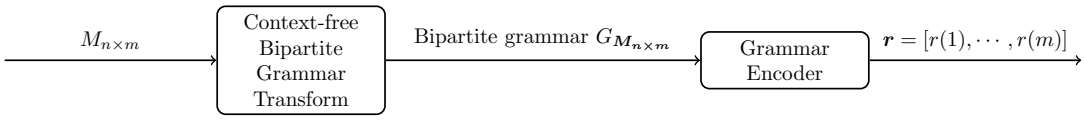


Figure 4.5: Structure of a bipartite grammar encoder

4.4.1 Canonical CFBG

Given an admissible CFBG G is said canonical if

(1) The variable set of G is $V = \{v_0\} \cup \{v_2, v_4, v_6, \cdots, v_{2i}\} \cup \{v_1, v_3, v_5, \cdots, v_{2j-1}\}$ for some $i \geq 0$ and $j \geq 0$, where $i = 0$ or $j = 0$ implies the corresponding set is empty.

(2) $G(v_0)$ has following properties:

   (a) The variable labeled edges are before the terminal symbol labeled edges.

   (b) The variable labeled edges are of raster scan order, i.e. first ordered by x then by y.

   (c) The terminal symbol labeled edges are of raster scan order.

(3) for each variable $v_{2t},\ 1 \leq t \leq i$, $G(v_{2t})$ is of size 2 and the position of its first edge is $(0,0)$.

(4) for each variable $v_{2t-1},\ 1 \leq t \leq j$, $G(v_{2t-1})$ has following properties:

   (a) The edge with position $(0,0)$ is the first edge.

   (b) The edges that have same type of label as $(0,0)$ are after $(0,0)$ and before the edges with different type of label from $(0,0)$. i.e. if $(0,0)$ is variable labeled, then all the variable labeled edge comes before all the terminal symbol labeled edge. If $(0,0)$ is terminal symbol labeled, then the terminal symbol labeled edges are before variable labeled edges.

   (c) Among variable labeled edges,possibly except $(0,0)$, the edges are listed in raster scan order, so do terminal symbol labeled edges, again possibly excep $(0,0)$.

From an admissible CFBG G, we can always get a canonical CFBG $\hat{G}$ by renaming the variable and reordering each edge set $R(v)$, $v \in V$, see Exmp. 12.

Example 12 Consider the following CFBG G representing $B \in \mathcal{B}(\Sigma)$ that identify the matrix shown in Fig. 4.6.

$$
\begin{aligned}
v_0 &\rightarrow \{(1,1,v_3),(1,8,\delta),(2,6,v_1),(4,6,\delta),\\
&\quad\ (4,1,v_2),(6,5,\delta),(7,1,v_3)\}\\
v_1 &\rightarrow \{(0,0,\delta),(1,0,\delta)\}\\
v_2 &\rightarrow \{(0,0,\delta),(0,1,v_1),(1,2,\delta)\}\\
v_3 &\rightarrow \{(0,0,v_2),(0,3,v_1)\}
\end{aligned}
$$

1	1	0	1	0	0	0	1
0	1	1	1	0	1	0	0
0	0	0	0	0	1	0	0
1	1	0	0	0	1	0	0
0	1	1	0	0	0	0	0
0	0	0	0	1	0	0	0
1	1	0	1	0	0	0	0
0	1	1	1	0	0	0	0

Figure 4.6: Matrix represented by G

Rename the variable v_1 to v_2, v_3 to v_4 and v_2 to v_1. Then reorder the edge set of each variable, we can get $\hat{G}$:

$$\begin{aligned} v_0 &\rightarrow \{(1,1,v_4),(2,6,v_2),(4,1,v_1),(7,1,v_4) \\ &\quad\ (1,8,\delta),(4,6,\delta),(6,5,\delta),\} \\ v_1 &\rightarrow \{(0,0,\delta),(1,2,\delta),(0,1,v_2)\} \\ v_2 &\rightarrow \{(0,0,\delta),(1,0,\delta)\} \\ v_4 &\rightarrow \{(0,0,v_1),(0,3,v_2)\} \end{aligned}$$

4.4.2 Grammar encoder

We now use following algorithm to encode canonical CFBG $\hat{G}$ into vector $\boldsymbol{r}$:

STEP 1: Traverse the production rules of $\hat{G}$, edge by edge, in the following order: $\hat{G}(v_0)\ \hat{G}(v_2)\ \cdots\ \hat{G}(v_{2i})\ \hat{G}(v_1)\ \hat{G}(v_3)\ \cdots\ \hat{G}(v_{2j-1})$. During the traversal, for each edge traversed, record its edge information. The edge information is defined as follows:

(1) If the edge is in $\hat{G}(v_0)$, its edge information is both its position and variable label if that element is variable labeled, and only its position if that edge is terminal symbol labeled.

(2) If the edge is the first edge in $\hat{G}(v),\ v \neq v_0$, its edge information is only its label(variable or terminal symbol).

(3) If the edge is the second edge in $\hat{G}(2t),\ 1 \leq t \leq i$, its edge information is both its position and label.

(4) If the edge is in $\hat{G}(2t-1),\ 1 \le t \le j$, but not the first edge of $\hat{G}(2t-1)$, the edge information is both its position and variable label if that edge is variable labeled, and only its position if that edge is terminal symbol labeled.

Denote the resulting vector by $\hat{r}_1(G)$

Example 13 Apply STEP 1 to the canonical CFBG we get in Exmp. 12, the resulting $\hat{r}_1(G)$ is:

$$\begin{aligned}\hat{r}_1(G) &= [1,1,v_4,2,6,v_2,4,1,v_1,7,1,v_4,1,8,4,6,6,5,\\ &\quad \delta,1,0,\delta,v_1,0,3,v_2,\delta,1,2,0,1,v_2]\end{aligned}$$

STEP 2: Replace the terminal symbol δ by 0 and each variable v_t by t in $\hat{r}_1(\hat{G})$, the resulting vector is our desired first vector $r_1(\hat{G})$.

Example 14 Apply STEP 2 to $\hat{r}_1(\hat{G})$ we get in Exmp. 13 yields:

$$\begin{aligned}r_1(\hat{G}) &= [1,1,4,2,6,2,4,1,1,7,1,4,1,8,4,6,6,5,\\ &\quad 0,1,0,0,1,0,3,2,0,1,2,0,1,2]\end{aligned}$$

STEP 3: For each variable $v \in V$, let $|\hat{G}(v)|_0$ denote the number of edges in $\hat{G}(v)$ that is terminal symbol labeled and $|\hat{G}(v)|_1$ denote the number of edges that is variable labeled. For each $1 \le t \le j$, let indicator variable $I(\hat{G}(v_{2t-1}))$ be 1 if the first element in $\hat{G}(v_{2t-1})$ is variable labeled and 0 otherwise. Our desired second vector $r_2(\hat{G}) = [r_2(0), r_2(1), r_2(2), \cdots, r_2(2j+2)]$ is constructed recursively as follows:

$$r_2(0) = 3|\hat{G}(v_0)|_1 \tag{4.3}$$
$$r_2(1) = r_2(0) + 2|\hat{G}(v_0)|_0 \tag{4.4}$$
$$r_2(2) = r_2(1) + 4i; \tag{4.5}$$

and for $t = 1, 2, \cdots, j$

$$\begin{aligned}r_2(2t+1) &= r_2(2t)\\ &\quad + \begin{cases} 3|\hat{G}(v_{2t-1})|_1 - 2 & \text{if } I(\hat{G}(v_{2t-1})) = 1\\ 2|\hat{G}(v_{2t-1})|_0 - 1 & \text{if } I(\hat{G}(v_{2t-1})) = 0 \end{cases}\end{aligned} \tag{4.6}$$

and

$$\begin{aligned} r_2(2t+2) &= r_2(2t+1) \\ &+ \begin{cases} 2|\hat{G}(v_{2t-1})|_0 & \text{if } I(\hat{G}(v_{2t-1})) = 1 \\ 3|\hat{G}(v_{2t-1})|_1 & \text{if } I(\hat{G}(v_{2t-1})) = 0 \end{cases} \end{aligned} \tag{4.7}$$

We can see that $r_2(\hat{G})$ indicates the partition of $r_1(\hat{G})$, which helps to reconstruct the canonical CFBG $\hat{G}$.

Example 15 Apply STEP 3 to the canonical CFBG we get in Exmp. 12 yields:

$$r_2(\hat{G}) = [12, 18, 26, 29, 32]$$

STEP 4: The vector representation of G

$$\boldsymbol{r}(\hat{G}) = [|r_2(\hat{G})|, r_2(\hat{G}), r_1(\hat{G})] \tag{4.8}$$

It is easy to see that $\hat{G} \to \boldsymbol{r}(\hat{G})$ is a one to one mapping. And the set $\{r(\hat{G})\}$ is a prefix set.

Example 16 The final vector representation of the canonical CFBG we get in Exmp. 12 is:

$$\begin{aligned} \boldsymbol{r}(\hat{G}) = [&5, 12, 18, 26, 29, 32 \\ &1, 1, 4, 2, 6, 2, 4, 1, 1, 7, 1, 4, 1, 8, 4, 6, 6, 5, \\ &0, 1, 0, 0, 1, 0, 3, 2, 0, 1, 2, 0, 1, 2] \end{aligned}$$

4.5 Parallel Random Access

After first rewriting into canonical form and then encoding the CFBG G which represents matrix $M_{n\times m}$ into $\boldsymbol{r}$, we can now have the capability to get access to the elements in $M_{n\times m}$ through $\boldsymbol{r}$.

The vector $\boldsymbol{r} = [|\boldsymbol{r_2}|, \boldsymbol{r_2}, \boldsymbol{r_1}]$ is self-delimiting, we can determine where $\boldsymbol{r_2}$ and $\boldsymbol{r_1}$ starts and ends in $\boldsymbol{r}$ in constant time. Therefore, in what follows, the input vector $\boldsymbol{r}$ will be regarded as two input vectors $\boldsymbol{r_2}$ and $\boldsymbol{r_1}$.

Further more, for each variable $v \in \hat{G}$, we can in constant time, by the following algorithm, get the sub-vector $\boldsymbol{r_v}$ of $\boldsymbol{r_1}$, which stores all the edge informations of $\hat{G}(v)$. Again, the variable set V of $\hat{G}$ is $\{v_0\} \cup \{v_2, v_4, \cdots, v_{2i}\} \cup \{v_1, v_3, \cdots, v_{2j-1}\}$, where $i = 0$ or $j = 0$ denotes the corresponding set is empty and $\boldsymbol{r}[a; b] = [r(a), a(a+1), \cdots, r(b)]$ for $1 \leq a \leq b \leq |\boldsymbol{r}|$.

1. If $v = v_0$, $\boldsymbol{r_v}$ starts at 1 and end at $r_2(2)$ i.e. $\boldsymbol{r_v} = \boldsymbol{r_1}[1; r_2(2)]$, where
   (a) $\boldsymbol{r_1}[1; r_2(1)]$ stores the edge information of variable labeled edges, and
   (b) $\boldsymbol{r_1}[r_2(1)+1; r_2(2)]$ stores the edge information of terminal symbol labeled edges.
2. If $v = v_{2t}$, $1 \leq t \leq i$, $\boldsymbol{r_v} = \boldsymbol{r_1}[r_2(2) + 4t - 3; r_2(2) + 4t]$, where
   (a) $r_1(r_2(2) + 4t - 3)$ denotes the label of edge $(0, 0)$, and
   (b) $\boldsymbol{r_1}[r_2(2) + 4t - 2; r_2(2) + 4t]$ stores the edge information of the other edge.
3. If $v = v_{2t-1}$, $1 \leq t \leq j$, $\boldsymbol{r_v} = \boldsymbol{r_1}[r_2(2t + 1) + 1; r_2(2t + 3)]$, where
   (a) $r_1(r_2(2t + 1) + 1)$ denote the label of edge $(0, 0)$,
   (b) $\boldsymbol{r_1}[r_2(2t + 1) + 2; r_2(2t + 2)]$ stores the edge information of edges which have same type of label as $(0, 0)$, and
   (c) $\boldsymbol{r_1}[r_2(2t + 2) + 1; r_2(2t + 3)]$ stores the edge information of edges which have different type of label from $(0, 0)$.

With the correspondence of edge information and edge in $\hat{G}(v)$, we can state the random access algorithm in terms of $\hat{G}$ rather than $\boldsymbol{r}$. The random access algorithm ACCESS(x, y, v), where $x, y \in \mathcal{N}$ and $v \in V$, is shown in the following pseudo-code. ACCESS(x, y, v_0) will return $M_{n\times m}[x, y]$. 1 and 0 are also regarded as boolean value **True** and **False**, and vice versa.

ACCESS(x, y, v)

```
1  foreach terminal symbol labeled edge (x_1, y_1, δ) ∈ Ĝ(v)
2      if (x == x_1 and y == y_1) then return True
3  foreach variable labeled edge (x_1, y_1, v_1) ∈ Ĝ(v)
4      if ACCESS(x − x_1, y − y_1, v_1) then return True
5  return False
```

As the terminal symbol labeled edges in $\hat{G}(v)$ are sorted in raster scan order, the functionality of the loop from line 1 to line 2 can be done by a binary search among all the terminal labeled edges. Hence, the time complexity is $O(|\hat{G}(v)|)$.

However, unlike the random access in dimension coding, where the locality of grammar makes it possible to directly go to the variable labeled element containing the query point, given x, y, we can not tell which variable labeled edge to go to. In this case, we have to check each variable labeled edge sequentially resulting in high time complexity.

If the multi-threading feature is allowed and assume each thread takes care of on of the function calls of ACCESS in line 4. These threads can run in parallel and if one of them returns **True**, returns **True**, otherwise return **False**. In this scenario, we have $O(d(\hat{G}))$ time complexity, where $d(\hat{G})$ is the depth of derivation tree of CFBG $\hat{G}$. This is the reason why we say the random access is parallel random access.

4.6 Bipartite Grammar Transform

For any $n \times m$ binary matrix $M_{n\times m}$, let $\{M_{n\times m}\}$ denote the edge set of its corresponding bipartite graph. For example, if $M_{n\times m}$ is the matrix shown in Fig. 4.2b, then $\{M_{n\times m}\} = \{(1,1,\delta),(2,2,\delta),(1,3,\delta),(3,3,\delta),(4,4,\delta),(3,5,\delta),(4,1,\delta),(3,2,\delta)\}$. A bipartite grammar transform is a mapping which assigns to each binary matrix $M_{n\times m}$ an admissible CFBG $G_{M_{n\times m}}$ that represents $\{M_{n\times m}\}$.

Start with the trivial CFBG consisting of the single production rule $v_0 \to \{M_{n\times m}\}$. Find a repeated subgraph of $\{M_{n\times m}\}$, and simultaneously subtract all repetitions of that subgraph from $\{M_{n\times m}\}$. Then we get a new CFBG with two production rules $\{v_0 \to R(v_0); v_1 \to R(v_1)\}$, where $R(v_0)$ is the updated $\{M_{n\times m}\}$, v_1 is the new label introduced in the subtraction, and $R(v_1)$ is the edge set of the graph represented by v_1. Repeating the step for the resulting set of edge sets, one would get better and better CFBGs representing $\{M_{n\times m}\}$ in general.

There are many possible bipartite grammar transforms providing reasonably compact representations for $M_{n\times m}$. Among them is the smallest bipartite grammar transform that assigns to $M_{n\times m}$ an admissible CFBG $G_{M_{n\times m}}$ with the smallest size.

The CFBG in Exmp. 11 is the smallest CFBG for the matrix shown in Fig. 4.2b. In general, however, we conjecture that finding the smallest CFBG for $M_{n\times m}$ is NP hard. In the next sections, we will instead present two bipartite grammar transforms with low complexity.

4.6.1 Sequential D-Neighborhood Pairing Transform

With reference to $M_{n\times m}$ as a 2-D array, we define the distance between two edges (i,j) and (k,l) as Manhattan distance, i.e. $|i-k|+|j-l|$ and arrange edge in $\{M_{n\times m}\}$ in the raster scan order of $M_{n\times m}$ (from top to bottom and left to right): $(x_1,y_1,\delta),(x_2,y_2,\delta),\cdots$. Denote the set consisting of the first i edge vectors by $\{M_{n\times m}\}^i$.

Fix $D>0$. Sequential D-Neighborhood Pairing Transform (SNPT) recursively constructs an admissible CFBG G_i representing $\{M_{n\times m}\}^i$ for $i=1,2,\cdots,|\{M_{n\times m}\}|$.

STEP 1: G_1 consists of the single production rule $v_0\rightarrow\{M_{n\times m}\}^1$.

STEP 2: Update $R(v_0)$ in G_i, $i\geq 1$ into $R(v_0)\cup\{(x_{i+1},y_{i+1},\delta)\}$. Then we get a CFBG G'_{i+1} representing $\{M_{n\times m}\}^{(i+1)}$.

STEP 3: Find a repeated subgraph of size 2 with its two edges within the distance D in $R(v_0)$ of G'_{i+1}, if any, and then subtract the repetitions of that subgraph from $R(v_0)$ of G'_{i+1}. We get a new CFBG G''_{i+1} in which $R(v_0)$ is the updated $R(v_0)$ in G'_{i+1} and which includes a new production rule $v\rightarrow R(v)$ with v being the newly introduced label in the subtraction and $R(v)$ the edge vector set of the graph represented by v.

STEP 4: Repeat STEP 3 for the newly updated $R(v_0)$ until there is no repeated subgraph of size 2 with its two edges within the distance D in $R(v_0)$. The resulting CFBG is then G_{i+1}.

STEP 5: Repeat STEP 2, for $1\leq i<|\{M_{n\times m}\}|$. Once $G_{|\{M_{n\times m}\}|}$ is obtained, further prune it by removing, through graph addition, any variable appearing only once in the union of all $R(v)$, $v\in V$, and deleting the production rule corresponding to that variable. The resulting CFBG is $G_{M_{n\times m}}$ produced by SNPT for representing $M_{n\times m}$.

Example 17 We will revisit the matrix $M_{n\times m}$ shown in Fig. 4.2b in this example. The ordered edge set of labeled CFBG B is:

$$\{M_{n\times m}\}=\{(1,1,\delta),(1,3,\delta),(2,2,\delta),(3,2,\delta),(3,3,\delta),(3,5,\delta),(4,1,\delta),(4,4,\delta),\}$$

Run the SNPT on this matrix with $D=2$.

1. When the SNPT starts, G_1:

$$v_0\rightarrow\{(1,1,\delta)\}$$

2. G'_2:

$$v_0 \rightarrow \{(1,1,\delta),(1,3,\delta)\}$$

There is no repetition of subgraphs in $G'_2(v_0)$, hence, $G_2 = G'_2$.

3. Similar to G_2, $G_3 = G'_3$:

$$v_0 \rightarrow \{(1,1,\delta),(1,3,\delta),(2,2,\delta)\}$$

4. Similar to previous steps, $G_4 = G'_4$:

$$v_0 \rightarrow \{(1,1,\delta),(1,3,\delta),(2,2,\delta),(3,2,\delta)\}$$

5. $G_5 = G'_5$:

$$v_0 \rightarrow \{(1,1,\delta),(1,3,\delta),(2,2,\delta),(3,2,\delta),(3,3,\delta)\}$$

Note that subgraphs $\{(1,1,\delta),(2,2,\delta)\}$ and $\{(2,2,\delta),(3,3,\delta)\}$ are not repetition of each other since they are not disjoint.

6. G'_6:

$$v_0 \rightarrow \{(1,1,\delta),(1,3,\delta),(2,2,\delta),(3,2,\delta),(3,3,\delta),(3,5,\delta)\}$$

The subgraphs $\{(1,1,\delta),(1,3,\delta)\}$ and $\{(3,3,\delta),(3,5,\delta)\}$ are repetition of each other. The distance between $(1,1)$ and $(1,3)$ is $2 \leq D$. By subtracting them from $G'_6(v_0)$ and adding another variable v_1, which represents $\{(1,1,\delta),(1,3,\delta)\} - (1,1)$, we get G''_6 :

$$\begin{aligned} v_0 &\rightarrow \{(1,1,v_1),(2,2,\delta),(3,2,\delta),(3,3,v_1)\} \\ v_1 &\rightarrow \{(0,0,\delta),(0,2,\delta)\} \end{aligned}$$

There is no more repetition of subgraphs. Therefore, $G_6 = G''_6$.

7. G'_7:

$$\begin{aligned} v_0 &\rightarrow \{(1,1,v_1),(2,2,\delta),(3,2,\delta),(3,3,v_1),(4,1,\delta)\} \\ v_1 &\rightarrow \{(0,0,\delta),(0,2,\delta)\} \end{aligned}$$

There's no repetition of subgraphs. Therefore, $G_7 = G'_7$.

8. G_8':

$$\begin{aligned} v_0 &\rightarrow \{(1,1,v_1),(2,2,\delta),(3,2,\delta),(3,3,v_1),(4,1,\delta),(4,4,\delta)\} \\ v_1 &\rightarrow \{(0,0,\delta),(0,2,\delta)\} \end{aligned}$$

Subgraphs $\{(1,1,v_1),(2,2,\delta)\}$ and $\{(3,3,v_1),(4,4,\delta)\}$ are repetition of each other and distance between $(1,1)$ and $(2,2)$ is $2 \leq D$. By subtracting these two repeated subgraphs and adding a new variable v_2, which represents $\{(1,1,v_1),(2,2,\delta)\}-(1,1)$, we get G_8'':

$$\begin{aligned} v_0 &\rightarrow \{(1,1,v_2),(3,2,\delta),(3,3,v_2),(4,1,\delta)\} \\ v_1 &\rightarrow \{(0,0,\delta),(0,2,\delta)\} \\ v_2 &\rightarrow \{(0,0,v_1),(1,1,\delta)\} \end{aligned}$$

There is no repetition of subgraphs. We get $G_{|M_{n\times m}|} = G_8 = G_8''$

9. Now we prune G_8. We can see that variable v_1 appears only once among the union of all right hand sides of production rules $\{G_{|M_{n\times m}|}(v) : \ v \in V\}$. Delete v_1 along with its production rule and rename the remaining variables, we can get our final grammar:

$$\begin{aligned} v_0 &\rightarrow \{(1,1,v_1),(3,2,\delta),(3,3,v_1),(4,1,\delta)\} \\ v_1 &\rightarrow \{(0,0,\delta),(0,2,\delta),(1,1,\delta)\} \end{aligned}$$

4.6.2 Iterative Pairing Transform

When D is large, the complexity of SNPT is still high. To overcome this, we can iteratively run SNPT, starting with a small D and gradually increasing D at each iteration, as shown below in Iterative Pairing Transform (IPT).

Let $0 = D_0 < D_1 < D_2 < \cdots < D_K = D$ and $\{M^{(0)}\} = \{M_{n\times m}\}$. For $i = 1, 2, \cdots, K$, run SNPT with D_i on $\{M^{(i-1)}\}$ without the final pruning step in SNPT, and denote $R(v_0)$ in $G_{|\{M^{(i-1)}\}|}$ by $\{M^{(i)}\}$. At the end of iterations, prune the grammar as in SNPT. The resulting grammar is $G_{M_{n\times m}}$ produced by IPT for representing $M_{n\times m}$.

4.6.3 Implementation of IPT

To better illustrate the implementation of IPT, when talking about $\hat{G}(v_0)$ of some canonical CFBG $\hat{G}$, we will use its biadjacency matrix, i.e. edge (x, y, v) will be represented

by an element that has value v and locates on row x and column y in the biadjacency matrix. Again, we use Manhattan distance as our distance metric.

Viewed as a set of edges, a bipartite graph is not sequential compared to a 1-D data sequence. As a result, the number of possible subgraphs is exponential regarding to the size of the bipartite graph while the number of sub-strings is polynomial regrading to the length of sequence, which also means to find repetitions of sub-graphs is much harder than to find repeated sub-string as in 1-D grammar based code.

As a result, we set the following constraints, which result in the two grammar transforms discussed in Sec. 4.6:

1. We only look for subgraph of size 2, which corresponding to a pair of elements in the biadjacency matrix.

2. These two elements are within some distance D

To find all the pairs of elements within distance D, we raster scan the biadjacency matrix. As shown in Fig. 4.7a, during the scan, for each non-zero element we draw a "circle" with radius D and a horizontal scan line that touches that element.

Finding all other elements, distances from which to element (r, c) are within D, is equivalent to finding all the cords, which are the result of cutting previously drawed circles by the scan line touches (r, c), that contains (r, c). As shown in Fig. 4.7a, where the current scan line is the blue one, the cord of black circle contains (r, c) as well as the red circle, which means the black point and red point are within D from (r, c). To find these cords, we can, for each cord whose left end point lies in interval $[c - D, c]$, check its right end point. In this case, if we can order the cords by its left end point, finding all pairs within distance D will be much easier.

As shown in Fig. 4.7b, let s denote the position of the scan line, and (s, x) denote the left end of a cord. To sort all the cords by x, we can use the equation $(c-x)+(s-r) = D$ which gives $x = c - r + s - D$. If the scan line is fixed, s and D are constant, sorting cord by x value is equivalent to sort by $c - r$, i.e. sort the origin of the circle corresponding to the cord by the difference of its column positions and row position. Therefore, it is possible for us to store only the set of origins, again ordered by the difference of column and row, instead of maintaining the set of cords, whose ending points are changing during the scan.

Note that in the description of IPT in Sec. 4.6, each D_i, $i \geq 1$ can be chosen in arbitary mannar. However, in the actual implementation, we chose $D_1 = 1$ and $D_i = 2D_{i-1}$ for $i \geq 2$. In this case, when D gets larger and larger, the biadjacency matrix gets sparser and sparser, which not only accelerate the scan but also do good to the locality of subgraphs.

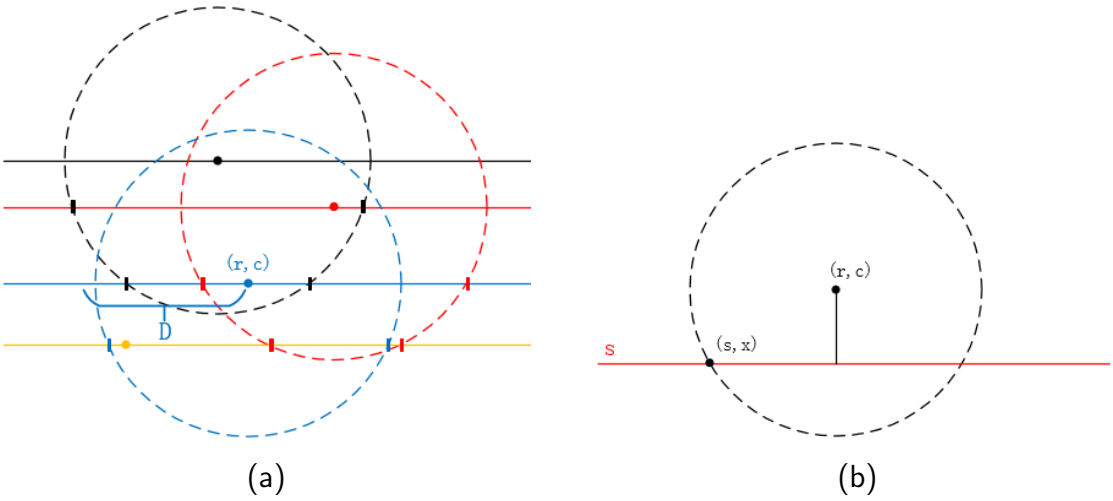


Figure 4.7: (a) Illustration of scanning. (b) Sort left end point of cord.

To get more compact CFBG, we set the following constraints:

1. no pair of elements within distance D in the biadjacency matrix appears more than once
2. every production rule is used more than once

The first constraint is satisfied by maintaining a dictionary that contains all pair of elements within distance D. The second constraint will be satisfied after pruning the grammar.

Three maps will be maintained during the execution:

1. map from element position to element label $\mathcal{L} : (r, c) \to N$
2. map from variable candidate to its position of appearance $\mathcal{C} : (\Delta_r, \Delta_c, v_1, v_2) \to (p_1, p_2)$. $\mathcal{C}$ is cleared at the beginning of each round.
3. map from variable to its index $\mathcal{V} : (\Delta_r, \Delta_c, v_1, v_2) \to N$

Map $\mathcal{C}$ act as a dictionary, once we find a pair that is already in the dictionary and does not overlap with current pair, we subtract these two subgraphs(pairs) from $\hat{G}(v_0)$, and then add a new member to $\mathcal{V}$ representing these two subgraphs.

In the following pseudo-code, repetitions of subgraphs will be replaced by an edge at the position of lower-right edge rather than the upper-left edge for the convenience of implementation, which does not affect the correctness of the algorithm.

Procedure REMOVE and CHANGE-LABEL are used to update map $\mathcal{C}$ and $\mathcal{L}$. When called on element p, they will change all the pairs that contain p in the dictionary. Procedure PRUNE prune the CFBG represented by $\mathcal{V}$. Before pruning, all variables in $\mathcal{V}$ except v_0 are of size 2. PRUNE expand the edge labeled by variable v_i, which appears only once, until all variable in $\mathcal{V}$ appear at least twice. PRUNE also relabel the variables.

```
GRAMMAR-TRANSFORM(R)
    // R represent the number of rounds
1   L = ∅, V = ∅
    // use variable count to count the number of variables
2   count = 1
3   for i = 0 to R
4       D = 2^i
5       SCAN-MATRIX(D)
6   G = PRUNE(V)
7   return G
```

```
SCAN-MATRIX(D)
 1  C = ∅
 2  I = ∅ // I is set of origins
 3  P = domain of L
 4  Sort P first by row then by column
 5  forall p ∈ P
 6      s = p.r
 7      forall q ∈ I and q.c − q.r ∈ [p.c − s, p.c − s + D]
        // in other words,q.x ∈ [p.c − D, p.c]
 8          if s − q.r > D
            // scan line is too far from point q
 9              remove q from P
10          elseif DIST(q, p) ≤ D
11              PROCESS-PAIR(q, p)
12      I = I ⋃ {p}
```

```
PROCESS-PAIR(p_3, p_4)
 1  t = (p_4.r − p_3.r,  p_4.c − p_3.c,  L(p_3),  L(p_4))
 2  if t ∈ domain of V
 3        REMOVE(p_3)
 4        CHANGE-LABEL(p_4, V(t))
 5  elseif t ∈ domain of C
 6        if p_2 == p_3 continue // overlap
 7        (p_1, p_2) = C(t)
 8        V(t) = count^{++}
 9        domain of C = domain of C − {t}
10        REMOVE(p_1), REMOVE(p_3)
11        CHANGE-LABEL(p_2, V(t)), CHANGE-LABEL(p_4, V(t))
    else
12        C(t) = (p_3, p_4)
```

Chapter 5

Experimental Results

5.1 Source and Descriptions of Testing Matrices

The testing matrices we use are all taken from The University of Florida Sparse Matrix Collection[8], which is a large and actively growing set of sparse matrices that arise in real applications. The collection is widely used in order to develop and evaluate the performance of sparse matrix algorithms. The collection allows for robust and repeatable experiments: robust because performance results with artificially-generated matrices can be misleading, and repeatable because these matrices are made available publicly in many formats. We chose 21 matrices from this matrix collection, whose statistics are shown in Tab. 5.1 and descriptions are shown in Tab. 5.2.

In Tab. 5.1, the first column is the name of the matrix, according to which the matrix can be found in [8]. The fourth column is the number of non-zero elements. Density is calculated by $\frac{\#\text{ of NZ}}{\text{height}\times\text{width}}$. The last column shows the symmetricity of the matrix. Note that if a matrix is symmetric, only lower triangular half of it is kept. As we can see in Tab. 5.1, these matrices are of different size, the smallest of which is 1005 by 1005 matrix dwt_1005 while the largest of which is 9.85×10^6 by 9.85×10^6 matrix wb-edu. Also, these matrices are of different shape, some of them are square matrices like amazon0302, others are rectangular such as wheel_601. The number of non-zero elements and the density of selected matrices also covers a wide range.

As we can see in Tab. 5.2, these matrices cover a wide rage of domains, include those from problems with underlying 2-D or 3-D geometry, such as thermodynamics, and those that do not have such geometry, such as networks and graphs.

Table 5.1: Statistics of testing matrices

name	height	width	# of NZ	density	sym
amazon0302	2.62E+05	2.62E+05	1.23E+06	1.80E-05	no
amazon0312	4.01E+05	4.01E+05	3.20E+06	1.99E-05	no
bmw7st_1	1.41E+05	1.41E+05	3.74E+06	1.87E-04	yes
cage12	1.30E+05	1.30E+05	2.03E+06	1.20E-04	yes
cegb2919	2.92E+03	2.92E+03	1.62E+05	1.90E-02	yes
Chem97Zt	2.54E+03	3.10E+04	6.20E+04	7.87E-04	no
dwt_1005	1.01E+03	1.01E+03	4.81E+03	4.77E-03	yes
enron	6.92E+04	6.92E+04	2.76E+05	5.76E-05	no
FEM_3D_thermal2	1.48E+05	1.48E+05	3.49E+06	1.60E-04	yes
flower_8_4	5.51E+04	1.25E+05	3.75E+05	5.43E-05	no
Freescale1	3.43E+06	3.43E+06	1.89E+07	1.61E-06	no
gupta2	6.21E+04	6.21E+04	2.16E+06	5.60E-04	yes
IMDB	4.28E+05	8.96E+05	3.78E+06	9.85E-06	no
pf2177	9.73E+03	1.02E+04	3.10E+04	3.13E-04	no
roadNet-CA	1.97E+06	1.97E+06	2.77E+06	7.12E-07	yes
roadNet-PA	1.09E+06	1.09E+06	1.54E+06	1.30E-06	yes
roadNet-TX	1.39E+06	1.39E+06	1.92E+06	9.90E-07	yes
thread	2.97E+04	2.97E+04	2.25E+06	2.54E-03	yes
wb-cs-stanford	9.91E+03	9.91E+03	3.69E+04	3.75E-04	no
wb-edu	9.85E+06	9.85E+06	5.72E+07	5.90E-07	no
wheel_601	9.02E+05	7.24E+05	2.17E+06	3.33E-06	no

5.2 Overall Result

In Tab. 5.3, we compared three grammar based representations, including bipartite grammar based coding shown in column CFBG, dimension coding with raster scan shown in column Dim_RS and dimension coding with Hilbert scan shown in column Dim_HS, with 2 state-of-art sparse matrix representation method, COO and CRS and quadtree representation. The metric we use here is the number of 32-bit data entries needed in each representation.

In our quadtree implementation, the tile size was set to 256; each non-leaf node stores four addresses of its children while each leaf node stores its data in either COO or CRS format. Since each index within a leaf node does not exceed 256, we can use one byte to

Table 5.2: Description of testing matrices

name	description
amazon0302	Amazon product co-purchasing network from March 2 2003
amazon0312	Amazon product co-purchasing network from March 12 2003
bmw7st_1	stiffness matrix
cage12	DNA electrophoresis, 12 monomers in polymer. A. van Heukelum, Utrecht U
cegb2919	finite element problem. 3-dimensional cylinder with flange
Chem97Zt	Mixed-effects model from D. Bates, Univ. Wisc.
dwt_1005	symmetric connection table from dtnsrdc, washington
enron	Laboratory for Web Algorithmics (LAW) Enron email network
FEM_3D_thermal2	FEM 3D nonlinear thermal problem, 8-node bricks as volume elements
flower_8_4	Combinatorial optimization as polynomial eqns, Susan Margulies, UC Davis
Freescale1	circuit problem from K. Gullapalli, Freescale Semiconductor
gupta2	Linear programming matrix (A*A'), Anshul Gupta, anshul@watson.ibm.com
IMDB	Pajek network: IMDB movie/actor network, www.imdb.com
pf2177	linear programming problem, C. Meszaros test set
roadNet-CA	Road network of California
roadNet-PA	Road network of Pennsylvania
roadNet-TX	Road network of Texas
thread	DNV-Ex 7 : Threaded connector/contact problem-1999-01-17
wb-cs-stanford	Stanford CS web, A(i,j)=1 if page i links to page j (2001)
wb-edu	*.edu web pages, A(i,j)=1 if page i links to page j (2001)
wheel_601	Combinatorial optimization as polynomial eqns, Susan Margulies, UC Davis

represent it, which counts only for 1/4 data entry. Note that if this special calculation is not applied, i.e. we count each index within a leaf node by 1 data entry, the quadtree can not have better performance than COO or CRS since each non-zero element still needs to be represented by either COO or CRS in the leaf nodes.

In our actual implementation of IPT, the following optimization is applied: same as in grammar encoding of bipartite grammar based coder, $R(v_0)$ is first divided into two disjoint subgraphs: one consisting of all terminal symbol labeled edges in $R(v_0)$, and the other consisting of all the rest in $R(v_0)$. The biadjacency matrices of the two subgraphs are then represented separately by COO or CRS depending on which representation is more succinct. (If COO is adopted, then each variable labeled edge in $R(S)$ is represented by three data entries, and each terminal symbol labeled edge in $R(v_0)$ is represented by two

Table 5.3: Space complexity of different representations

name	COO	CRS	QUAD	CFBG	Dim_RS	Dim_HS
amazon0302	2.47E+06	1.50E+06	1.55E+06	1.54E+06	1.29E+06	1.18E+06
amazon0312	6.40E+06	3.60E+06	5.33E+06	3.12E+06	3.04E+06	2.77E+06
bmw7st_1	7.48E+06	3.88E+06	1.37E+06	1.29E+05	9.60E+05	5.24E+05
cage12	4.07E+06	2.16E+06	9.60E+05	2.55E+05	1.70E+06	7.37E+05
cegb2919	3.24E+05	1.65E+05	4.58E+04	3.20E+03	3.04E+04	2.63E+04
Chem97Zt	1.24E+05	6.46E+04	1.89E+04	7.63E+03	6.48E+03	2.38E+04
dwt_1005	9.63E+03	5.82E+03	1.95E+03	1.44E+03	3.11E+03	3.25E+03
enron	5.52E+05	3.45E+05	2.03E+05	2.38E+05	1.70E+05	1.95E+05
FEM_3D_thermal2	6.98E+06	3.64E+06	1.16E+06	6.71E+04	1.89E+05	5.34E+05
flower_8_4	7.51E+05	4.30E+05	2.68E+05	1.42E+05	3.06E+05	3.58E+05
Freescale1	3.78E+07	2.23E+07	1.05E+07	2.18E+06	1.66E+07	5.07E+06
gupta2	4.31E+06	2.22E+06	6.40E+05	6.37E+05	7.24E+05	9.73E+05
IMDB	7.56E+06	4.21E+06	1.70E+07	4.67E+06	4.07E+06	4.20E+06
pf2177	6.20E+04	4.07E+04	1.65E+04	2.08E+04	3.10E+04	2.35E+04
roadNet-CA	5.53E+06	4.74E+06	1.92E+06	2.84E+06	2.28E+06	2.15E+06
roadNet-PA	3.08E+06	2.63E+06	1.22E+06	1.69E+06	1.33E+06	1.26E+06
roadNet-TX	3.84E+06	3.32E+06	1.33E+06	2.10E+06	1.64E+06	1.56E+06
thread	4.50E+06	2.28E+06	6.93E+05	3.17E+04	3.75E+05	4.16E+05
wb-cs-stanford	7.37E+04	4.68E+04	1.90E+04	2.08E+04	2.60E+04	2.26E+04
wb-edu	1.14E+08	6.70E+07	3.25E+07	1.59E+07	2.54E+07	1.99E+07
wheel_601	4.34E+06	3.07E+06	1.03E+07	2.80E+06	1.14E+05	2.11E+06

data entries, i.e., its edge position.)

It is clear from Tab. 5.3 that the grammar based codings provide, in general, significantly more succinct representations than CRS and quadtree, especially when the density of matrix is not extremely low and the distribution of non-zero elements in the matrix is not very random. For example, in the matrix thread, CRS is 70 times larger than CFBG and quadtree is 20 times larger.

In Tab. 5.3, among three grammar based representations, we color the best results by green and worst results by red. We can see bipartite grammar based representation performs better than the other two in the majority of cases. In some cases where bipartite grammar is better, the difference is huge, for example in FEM_3D_thermal2, CFBG is near 3 times better than Dim_RS and almost 8 times better than Dim_HS. Meanwhile, when

CFBG performs not so well, the difference between CFBG and the best representation is not so significant, such as in amazon0312 and roadNet-CA.

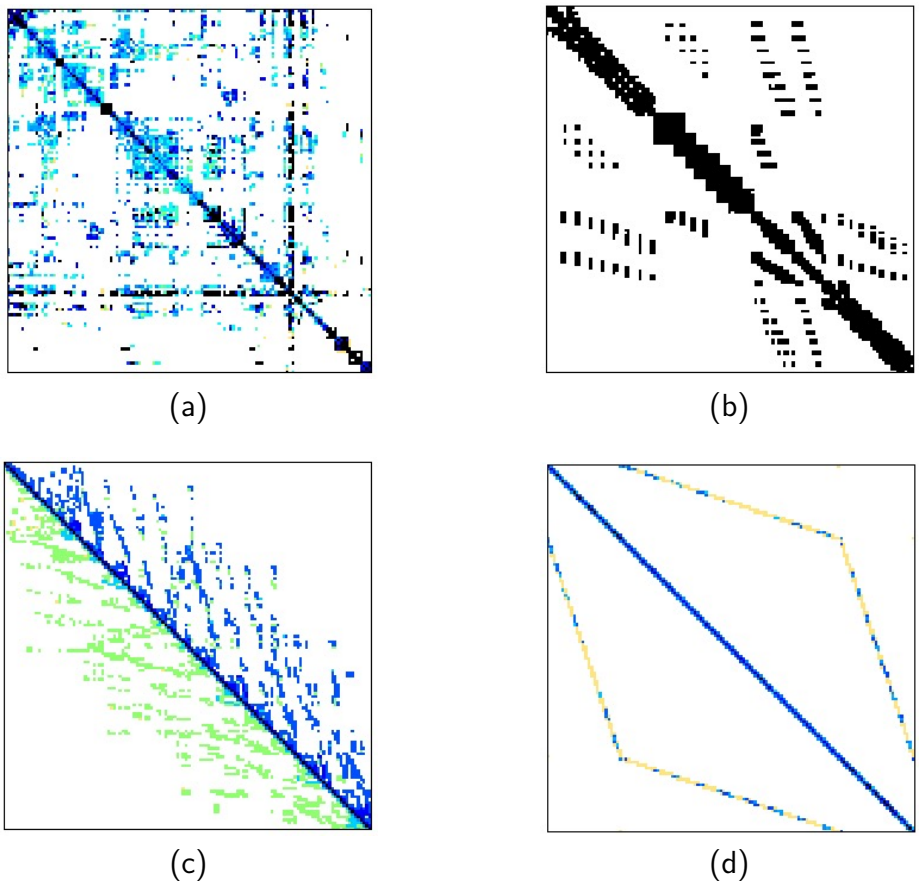


Figure 5.1: Structure of four matrices where CFBG is better. (a) bmw7st_1 (b) cegb2919 (c) cage12 (d) FEM_3D_thermal2

In Fig. 5.1, we show the structures of 4 matrices where CFBG is better. Note that all of them are symmetric and hence only lower triangular half is kept. These and following figures of the structure of matrices are all taken from [8]. We can see a lot of repeated patterns in each of these four matrices. By introducing variables representing the subgraphs corresponding to these repeated patterns, they can be captured well in CFBG, which results in succinct representations.

However, the scan methods do not have the capability to get adapted to these patterns. After the scanning, a pattern in original matrix is easily get separated apart in

the resulting vector, which makes it harder for the following dimension coding to give compact representation. In matrix cegb2919 shown in Fig. 5.1b, where the patterns of non-zeros are rectangular blocks, Hilbert scan is more suitable than raster scan. In matrix FEM_3D_thermal2 shown in Fig. 5.1d, both raster scan and Hilbert scan can not preserve the patterns shown as skew lines, which result in worse performance than CFBG. In matrix Chem97Zt shown in Fig. 5.2, the non-zero elements distribute in an echelon style, where there are a lot of horizontal lines. In this case, raster scan naturally fits these patterns performs much better than Hilbert scan. Note that in this matrix, CFBG is slightly worse than Dim_RS but still outperforms Dim_HS by a factor of 3.

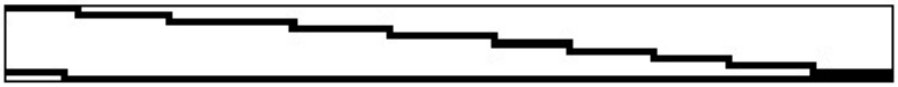


Figure 5.2: Structure of matrix Chem97Zt

Fig. 5.3 shows one example that grammar based representations do not perform very well. We can see the distribution of non-zero elements is quite random compared to matrices in Fig. 5.1.

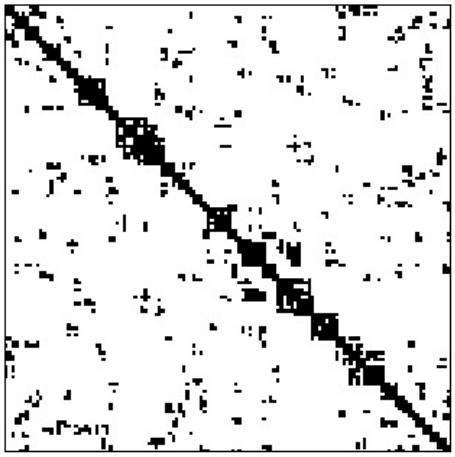


Figure 5.3: Structure of matrix roadNet-CA

We can also treat sparse binary matrices as binary images and use binary image encoding method JBIG to produce another representation. We only do so to relatively smaller matrices, since JBIG needs to arithmetic encode every pixel in the image, which takes a

very long time on large matrices. Also, the representation produced by JBIG can not support any operation, since the representation is not comprehensible without first decoding it. In out experiments, we run JBIG in non-progressive mode, since it requires slightly less storage. The comparison between grammar based representations and JBIG is shown in Tab. 5.4. We can see that grammar based representations sometimes are even more compact than JBIG, for example, in matrix flower_8_4.

Table 5.4: Experimental results including JBIG

name	# of NZ	density	QUAD	CFBG	Dim_RS	Dim_HS	JBIG
dwt_1005	4.81E+03	4.77E-03	1.95E+03	1.44E+03	3.11E+03	3.25E+03	1.33E+03
Chem97Zt	6.20E+04	7.87E-04	1.89E+04	7.63E+03	6.48E+03	2.38E+04	2.13E+03
cegb2919	1.62E+05	1.90E-02	4.58E+04	3.20E+03	3.04E+04	2.63E+04	3.52E+03
wb-cs-stanford	3.69E+04	3.75E-04	1.90E+04	2.08E+04	2.60E+04	2.26E+04	1.69E+04
pf2177	3.10E+04	3.13E-04	1.65E+04	2.08E+04	3.10E+04	2.35E+04	2.48E+04
thread	2.25E+06	2.54E-03	6.93E+05	3.17E+04	3.75E+05	4.16E+05	1.47E+05
enron	2.76E+05	5.76E-05	2.03E+05	2.38E+05	1.70E+05	1.95E+05	2.08E+05
flower_8_4	3.75E+05	5.43E-05	2.68E+05	1.42E+05	3.06E+05	3.58E+05	6.69E+05
gupta2	2.16E+06	5.60E-04	6.40E+05	6.37E+05	7.24E+05	9.73E+05	9.13E+05

5.3 Detailed Result of Bipartite Grammar Coding

We list some statistics of the CFBG based representations of the same set of testing matrices in Tab. 5.5. The second column, again, is the number of non-zero elements in the matrix, which is also the number of edges in $G(v_0)$ before grammar transform. The column $|V|$ is the total number of variables in the grammar. The column $|G(v_0)|$ denotes the size of $G(v_0)$ after grammar transform. $|G|$ denotes the total size of CFBG. The column $|V_{2t}|$ denotes the number of even variables, i.e. the number of the variables of size 2, except, possibly, v_0. The last two columns denote the representation of terminal labeled edges and variable labeled edges in $G(v_0)$.

From Tab. 5.5, in the matrices where CFBG can not provide huge gain, for example, amazon0302,

1. The size of $G(v_0)$ is only slightly reduced by the grammar transform. Drop from 1.23×10^6 to 5.88×10^5 in amazon0302.

Table 5.5: Statistics of CFBG

name	# of NZ	$\|V\|$	$\|G(v_0)\|$	$\|G\|$	$\|G(v_0)\|_\delta$	$\|V_{2t}\|$	$G(v_0)_\delta$	$G(v_0)_v$
amazon0302	1.23E+06	3.08E+04	5.88E+05	6.50E+05	2.79E+05	3.08E+04	CRS	CRS
amazon0312	3.20E+06	5.42E+04	1.39E+06	1.50E+06	6.75E+05	5.40E+04	CRS	CRS
bmw7st_1	3.74E+06	6.62E+03	3.29E+04	4.72E+04	8.00E+01	6.01E+03	COO	COO
cage12	2.03E+06	1.76E+04	3.75E+04	9.51E+04	7.98E+02	1.18E+04	COO	COO
cegb2919	1.62E+05	2.63E+02	4.19E+02	1.21E+03	0.00E+00	3.20E+03	COO	COO
Chem97Zt	6.20E+04	4.20E+02	1.97E+03	2.82E+03	1.50E+01	4.06E+02	COO	COO
dwt_1005	4.81E+03	1.35E+02	1.91E+02	5.59E+02	1.20E+01	9.80E+01	COO	COO
enron	2.76E+05	9.81E+03	6.99E+04	8.96E+04	1.12E+04	9.73E+03	COO	COO
FEM_3D_thermal2	3.49E+06	2.26E+02	2.85E+03	2.25E+04	0.00E+00	1.05E+02	COO	COO
flower_8_4	3.75E+05	1.06E+04	2.15E+04	5.36E+04	1.42E+03	6.83E+03	COO	COO
Freescale1	1.89E+07	8.06E+04	6.13E+05	7.97E+05	5.41E+04	7.52E+04	COO	COO
gupta2	2.16E+06	4.31E+04	9.61E+04	2.48E+05	1.35E+03	3.09E+04	COO	CRS
IMDB	3.78E+06	2.34E+04	3.05E+06	3.09E+06	2.37E+06	2.34E+04	CRS	CRS
pf2177	3.10E+04	1.23E+03	4.50E+03	7.79E+03	2.57E+02	1.09E+03	COO	COO
roadNet-CA	2.77E+06	9.37E+04	8.51E+05	1.04E+06	9.13E+04	9.36E+04	COO	COO
roadNet-PA	1.54E+06	5.83E+04	5.02E+05	6.18E+05	5.19E+04	5.82E+04	COO	COO
roadNet-TX	1.92E+06	7.22E+04	6.23E+05	7.68E+05	5.82E+04	7.21E+04	COO	COO
thread	2.25E+06	2.53E+03	4.11E+03	1.20E+04	0.00E+00	1.68E+03	COO	COO
wb-cs-stanford	3.69E+04	1.15E+03	5.18E+03	7.92E+03	8.58E+02	1.01E+03	COO	COO
wb-edu	5.72E+07	6.13E+05	4.15E+06	5.83E+06	3.94E+05	5.35E+05	COO	COO
wheel_601	2.17E+06	4.70E+02	1.17E+06	1.17E+06	6.97E+05	4.57E+02	COO	COO

2. There are still a large number of terminal symbol labeled edges in $G(v_0)$ after the transform. In amazon0302, around half of the edges in $G(v_0)$ are terminal symbol labeled.

3. And most of variables excluding v_0 is of size 2, i.e. the difference between $|V|$ and $|V_{2t}|$ is small. In amazon0302, this difference is negligible.

On the opposite side, in the matrices where CFBG greatly outperforms other methods, such as matrix thread:

1. The number of edges in $G(v_0)$ is greatly reduced. In thread, drop from 2.25×10^6 to

4.11×10^3, which is 500 times smaller.

2. Very few terminal symbol labeled edges remains in $G(v_0)$ after grammar transformation. In thread, this number is negligible.

3. A lot of variables have sizes larger than 2. Around $\frac{2}{5}$ of the variables have sizes larger than 2 in thread.

Compare these two cases, we can have the conclusion that when the density of the matrix is not extremely low and the distribution of non-zero elements are far from random, in which case more repeated subgraphs can be found, the bipartite grammar based representation can achieve better result in the sense of storage requirement.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

In this thesis, we discussed the problem of representing a large sparse binary matrix so that the representation requires less storage and, meanwhile, supports operations on this matrix. Here, by supporting operations, we only considered getting access to one element in the matrix, which is the most fundamental operation. Followed the idea of context-free grammar, which has already been applied in representing 1-D data sequences and proved successful, we proposed two grammar based representations of sparse binary matrix. The first one is based on so-called directionless grammar, which resulted in dimension coding that can represent 1-D sparse signals. We combined dimension coding with raster scan or Hilbert scan to get a representation of a 2-D matrix. This representation directly inherits the random accessibility from dimension coding. The second one is based on a new called concept Context-free Bipartite Grammar (CFBG), which we proposed in this thesis. Compared with directionless grammar, CFBG is also directionless in the sense that each variable represents a bipartite graph, which can be viewed as a set of edges. However, CFBG has true 2-D nature. By doing experiments on various matrices from real applications, we showed that the grammar based representation greatly outperforms the state-of-art methods COO, CRS and quadtree. We also showed that when the density of the matrix is not extremely low and the distribution of the non-zero elements in the matrix is far from random, the power of grammar can be fully utilized by CFBG. However, since the scan method is not adaptive to the distribution of non-zero elements, even if the directionless grammar is very powerful, the performances are not stable.

6.2 Future Work

6.2.1 Locality and Random Accessibility

From the discussion in Chapter 4, we can see that CFBG does not have a well-defined "locality" concept, which is vital for random accessing in dimension coding.

However, as we use the IPT as the grammar transform, it requires the distance of the two labeled edges represented by a variable does not exceed some threshold. With this property, it is possible to define some concepts similar to "locality" to speed up the random access decoder.

6.2.2 Potential Capability of Pattern Discovery

When we use a CFBG G to represent a matrix M which is represented by $B \in \mathcal{B}(\Sigma)$, for each variable $v \neq v_0$, $G(v)$ represents a subgraph of B. This subgraph can also be thought of a pattern, i.e. subset of elements in M. The list of all $G(v)$s can also be thought of a list of repeated patterns in the matrix. According to the meaning of each dimension of the matrix, it is possible to utilize the grammar to help with the analysis of the matrix.

6.2.3 Support more operations

In this thesis, we only consider the operation of random access an element located at specific row and column. There are other common operations on the matrix such as matrix-vector multiplication. There exist some representations focusing on accelerating these operations. It is also possible to extend the grammar or tune the grammar transformation to support various operations.

References

[1] Progressive bi-level image compression. Technical Report ISO/IEC International Standard 11544, ISO/IEC, 1992.

[2] Richard Barrett, Michael W Berry, Tony F Chan, James Demmel, June Donato, Jack Dongarra, Victor Eijkhout, Roldan Pozo, Charles Romine, and Henk Van der Vorst. *Templates for the solution of linear systems: building blocks for iterative methods*, volume 43. Siam, 1994.

[3] Robert D Cameron. Source encoding using syntactic information source models. *IEEE Transactions on Information Theory*, 34(4):843–850, 1988.

[4] C.Nevill-Manning and I.Witten. Compression and explanation using hierarchical grammars. *Comput.J.*, pages 103–116, 1997.

[5] C.Nevill-Manning and I.Witten. Identifying hierarchical structure in sequences: A linear-time algorithm. *J.Artificial Intell. Res.*, pages 67–82, 1997.

[6] Craig M Cook, Azriel Rosenfeld, and Alan R Aronson. Grammatical inference by hill climbing. *Information Sciences*, 10(2):59–80, 1976.

[7] P. Tvrdik D. langr, I. Simecek and T. Dytrych. Adaptive-blocking hierarchical storage format for sparse matrix. pages 545–551, Wroclaw, Poland, Jul. 2012. FedCSIS.

[8] T. A. Davis and Y. Hu. The university of florida sparse matrix collection. *ACM Transactions on Mathematical Software*, (1):1–25, 2011. Available: http://www.cise.ufl.edu/research/sparse/matrices.

[9] E.-H.Yang and J. C. Kieffer. Efficient universal lossless data compression algorithms based on a greedy sequential grammar transform—part i: Without context model. *IEEE Trans. Info. Theory*, (3):755–777, May 2000.

[10] Anand Ekambaram and Eurípides Montagne. An alternative compressed storage format for sparse matrices. In *International Symposium on Computer and Information Sciences*, pages 196–203. Springer, 2003.

[11] Raphael Finkel and J.L. Bentley. Quad trees: A data structure for retrieval on composite keys. *Acta Informatica*, (1):1–9, 1974.

[12] E. h. Yang. Dimension coding—part one: Original sparse signals. Submitted to IEEE Trans. Inf. Theory.

[13] J. Hopcroft and J. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, Reading, MA, 1979.

[14] John E Hopcroft. *Introduction to Automata Theory, Languages and Computation: For VTU, 3/e.* Pearson Education India, 1979.

[15] D. langr I. Simecek and P. Tvrdik. Minimal quadtree format for compression of sparse matrices storage. pages 359–364, Timisoara, Romania, 2012. roc. 14th Int. Symp. Symbolic & Numeric Algorithms for Scientific Computing.

[16] G. J. Nelson J. C. Kieffer, E. H. Yang and P. Cosman. Universal lossless compression via multilevel pattern matching. *IEEE Trans. Info. Theory*, (4):1227–1245, Jul. 2000.

[17] E.-H. Yang J. Zhang and J. C. Kieffer. A universal grammar-based code for lossless compression of binary trees. *IEEE Trans. Inf. Theory*, (3):1373–1386, Mar. 2014.

[18] Y. Jia and E.-H. Yang. Context-dependent multilevel pattern matching for lossless image compression. *IEEE Trans. Info. Theory*, (12):3169–3184, Dec. 2003.

[19] Eiji Kawaguchi and Tsutomu Endo. On a method of binary-picture representation and its application to data compression. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, (1):27–35, 1980.

[20] J. C. Kieffer and E.-H. Yang. Grammar based codes: A new class of universal lossless source codes. *IEEE Trans. Info. Theory*, (3):737–754, May 2000.

[21] Bongki Moon, Hosagrahar V Jagadish, Christos Faloutsos, and Joel H. Saltz. Analysis of the clustering properties of the hilbert space-filling curve. *IEEE Transactions on knowledge and data engineering*, 13(1):124–141, 2001.

[22] Craig G. Nevill-Manning and Ian H. Witten. Identifying hierarchical strcture in sequences: A linear-time algorithm. *J. Artif. Intell. Res.(JAIR)*, 7:67–82, 1997.

[23] William B. Pennebaker, Joan L. Mitchell, GG Langdon, and Ronald B Arps. An overview of the basic principles of the q-coder adaptive binary arithmetic coder. *IBM Journal of research and development*, 32(6):717–726, 1988.

[24] Pyrrhos Stathis, Sorin Cotofana, and Stamatis Vassiliadis. Sparse matrix vector multiplication evaluation using the bbcs scheme. In *in Proc. of 8th Panhellenic Conference on Informatics*. Citeseer, 2001.

[25] Pyrrhos Stathis, Stamatis Vassiliadis, and Sorin Cotofana. A hierarchical sparse matrix storage format for vector processors. In *Parallel and Distributed Processing Symposium, 2003. Proceedings. International*, pages 8–pp. IEEE, 2003.

[26] Stamatis Vassiliadis, Sorin Cotofana, and Pyrrhos Stathis. Block based compression storage expected performance. In *High Performance Computing Systems and Applications*, pages 389–406. Springer, 2002.

[27] E.-H. Yang and J. Guo. Lossless image coding via one-dimensional grammar based codes. pages 966–972, Beijing, China, Aug. 2000. Proc. of the 16th IFIP World Computer Congress | 2000 International Conference on Communication Technology.

[28] Jacob Ziv and Abraham Lempel. Compression of individual sequences via variable-rate coding. *IEEE transactions on Information Theory*, 24(5):530–536, 1978.