Matrix Storage Formats
Exam 1 Problem Six

Lawson Busch

March 6, 2018

Abstract

This paper begins by discussing three methods used to store sparse matrices in a computer, the Dictionary of Keys, List of Lists, and Compressed Row Storage methods. It breaks down each method in detail, discussing how it works, as well as its strengths and weaknesses as an implementation. Then all of the methods are compared as a whole. Finally, example code is given for matrix vector multiplication using the Compressed Row Storage demonstrating its effectiveness in performing matrix vector multiplication for sparse matrices.

1 Introduction

The storage of sparse matrices is an important problem to understand and implement, as proper implementation of sparse matrices allows matrix computations on sparse matrices to be sped up significantly. There are multiple ways to implement the storage of sparse matrices in a computer, each of which have their own distinct advantages and disadvantages that enable certain operations to be optimized. Understanding which operations are optimized by which sparse matrix storage implementation is crucial to selecting the proper sparse matrix storage implementation to use. In the next section, I will discuss three such implementations, their advantages and disadvantages, and directly compare each of them.

2 Storage Formats

2.1 Dictionary Of Keys

Dictionary of Keys is a simple but effective way to store a sparse matrix. It stores the a matrix M as a dictionary, where the keys of the dictionary are tuples of the for (r, c), where r is the row index of element and c is the column index of the element. The actual value stored at any arbitrary key (r, c) is the value at $M[r, c]$. If a position $(r1, c1)$ is not found in the dictionary, it is assumed to hold a value of 0.

The Dictionary of Keys is an example of sparse matrix storage that is good for constructing sparse matrices but does not actually speed up any matrix operations. This is due to the nature of dictionaries, where elements can be added, changed, or looked up in $O(1)$ time. Thus you can easily add or change elements of the dictionary, but cannot iterate over the elements in order (since dictionaries are not stored in lexicographical order).

As a result, Dictionary of Keys is often used to create, store, and update sparse matrices but is seldom used for matrix operations as it does not speed up matrix operations at all for sparse matrices. This is because it still needs to check for every possible row and column key for matrix vector multiplication, still requiring $O(N^2)$ operations for matrix vector multiplication.

2.2 Compressed Row Storage

The Compressed Row Storage format of storing a sparse matrix, M, stores as matrix as a series of three arrays where the first array, A, contains all the nonzero values of the matrix, the second array, B, contains

the number of elements in a row, and the third matrix, C contains the column indices of each nonzero element in the matrix. If N is the number of nonzero entries in the matrix M, the A will contain N elements, and stores all the non zero elements in M in row order (left to right, then top to bottom). The array B will have $m+1$ elements, where m is the number of rows in the matrix. The values of B are defined recursively, where $B[0] = 0$ and for any i, $B[i] = B[i-1] + b$, where b is the number of elements in the $i-1$ row of M. Thus the number at position i in the array B will be the total number of elements in the matrix M found in the previous $i-1$ rows. Finally, the array C contains the column index of every nonzero element in M, and thus is N elements long.

The advantages of Compressed Row Storage are that it allows for fast matrix vector multiplication, since only N elements need to be multiplied as well having fast row access, as it stores the exact location of each nonzero entry in the matrix. This allows the computer to pull only the non zero values stored in each row and multiply them directly with the corresponding element in the vector in matrix vector multiplication (since this position is stored in matrix C). Thus matrix vector multiplication can be done on $O(N)$, where N is the number of nonzero elements stored in the matrix.

However, a disadvantage of storing your data in Compressed Row Storage is that it takes more memory to store, since it is storing information about your matrices in three separate arrays. As a result, the Compressed Row Storage is only efficient (in terms of space) when the number of nonzero elements, N satisfies the following equation $N < (r(c-1)-1)/2)$, where r is the number of rows in M and c is the number of columns in M.

2.3 List of Lists

Similar to the Dictionary of Keys, List of Lists is another method for storing sparse matrices that is good for constructing sparse matrices but not particularly good for speeding up matrix operations. Lists of Lists stores a matrix as a series of lists, one for each row that is sorted in row order. Within the lists, tuples are stored for the nonzero elements of the matrix. These tuples are stored in the form (i, v), where i is the column index for the element and v is the value of the element. These tuples are sorted within each list by their i values.

Like Dictionary of Keys, List of Lists is good for creating and updating sparse matrices, as you can simply add elements to the list and resort them or iterate over only a few elements to update an element stored in the List of Lists. However, since List of Lists can be iterated over lexicographically making it better than Dictionary of Keys for matrix operations. However, since you still have to iterate over a series of N lists to perform matrix operations, List of Lists is still not considered an efficient way to implement matrix operations for sparse matrices. Thus, like Dictionary of Keys, it is often used to construct sparse matrices but not used to perform matrix operations.

2.4 Comparison

Overall, both the Dictionary of Keys and the List of Lists are both good for creating and storing sparse matrices. The Dictionary of Keys is particularly good at storing matrices, as it only requires one piece of information to be stored per nonzero element of the sparse matrix. The list of lists requires slightly more storage space, as it requires that you hold at least N lists, where N is the number of rows in the sparse matrix, each of which contain some number of elements. However, because you can iterate over the List of Lists lexicographically, it is more efficient than the dictionary when it comes to matrix operations, as you do not have to check all possible $R * C$ elements in the List of Lists, but you have to check all of the possible elements in the Dictionary of Keys.

On the other end of the spectrum is Compressed Row Storage, which is not super efficient when it comes to storage of the sparse matrix, as it stores three separate pieces of information about each element, but is extremely efficient when it comes to performing matrix vector multiplication, as you can directly pull out nonzero elements of the sparse matrix for multiplication, thus avoiding any multiplication of a 0 element of

the sparse matrix with the vector. Because you can avoid any 0 elements as you iterate over the lists in the Compressed Row Storage, you only have to do N multiplications, where N is the number of nonzero elements in a matrix M.

In summary, both the Dictionary of Keys and List of Lists method of storing sparse matrices are great for creating and storing matrices, with the Dictionary of Keys being slightly better for updating entries. However, neither method is particularly good for performing matrix operations, such as matrix multiplication. This is where a method like the Compressed Row Storage comes in. This method has its limitations to its efficiency depending on storage size, as discussed in its section. However, it is far more efficient when it comes to performing operations such as matrix vector multiplication.

3 Matrix Vector Multiplication for Compressed Row Storage

Below is psuedocode for matrix vector multiplication using the compressed row storage method. In this pseudocode N is the number of nonzero elements in the matrix M, x is the input vector to be multiplied, and *result* is the vector resulting from the multiplication of M and x. B is the array keeping track of the number of elements in each row of M. A is the array holding the values in row order of the matrix M, and C is the array holding the columnar positions of the values in A.

```
1  result = new float[size(x)]
2  for(k=0; k<N; k++) {
3    result[k] = 0;     #This instantiates the result vector as all 0s
4  }
5
6  for(i = 0; i < N; i++) {  #Iterates once for every element
7    for(j = B[i]; j < B[i+1]; j++) {  #Sets j to iterate over the number of elements in the
         row
8        result[i] = result[i] + A[j] * x[C[j]]; #Multiplies corresponding values in A and x
9    }
10 }
```

This code works by iterating over the number of elements in each row of the array B. It then moves forward in both the A and C arrays by the number of elements in the $B[i]$ row, adding their corresponding multiplication's with x to the result for the i^{th} row in M.

4 Conclusion

Each of the implementations for the storage of sparse matrices discussed in this paper have their own distinct advantages and disadvantages. It is important to fully comprehend each of their unique attributes, as it enables the correct selection of implementation for a problem. Both Dictionary of Keys and List of Lists are good implementations for creating and updating sparse matrices. Further, neither implementation is good for matrix operations such as matrix vector multiplication. In contrast, the Compressed Row Storage method is great for implementing matrix vector multiplication, but is not as storage or update efficient as either the Dictionary of Keys or the List of Lists method. In conclusion, each of these three methods has their own strengths and weaknesses, and should be used according to the operations likely to be performed on the stored sparse matrix M.

5 Citations

References

[1] Working with Sparse Matrices. mathcs.emory.edu, `http://www.mathcs.emory.edu/~cheung/Courses/561/Syllabus/3-C/sparse.html`

[2] Jack Dongarra, netlib.org, Compressed Row Storage, 1995. http://netlib.org/linalg/html_templates/node91.html#SECTION00931100000000000000

[3] Sparse Matrix. wikipedia.org, https://en.wikipedia.org/wiki/Sparse_matrix

[4] Dictionary of Keys Format (DOK), scipy-lectures.org, https://www.scipy-lectures.org/advanced/scipy_sparse/dok_matrix.html

[5] List of Lists Format (LIL), scipy-lectures.org, http://www.scipy-lectures.org/advanced/scipy_sparse/lil_matrix.html