Sparse Matrices in Numerical Mathematics

Miroslav Tůma

Faculty of Mathematics and Physics

Charles University

mirektuma@karlin.mff.cuni.cz

Praha, September 18, 2022

Outline

Sparse vector in a computer

Example

Consider the sparse row vector $v \in \mathbb{R}^8$

$$v = \begin{pmatrix} 1. & -2. & 0. & -3. & 0. & 5. & 3. & 0. \end{pmatrix}. \quad (1)$$

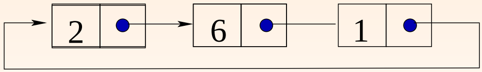
The real array `valV` that stores the nonzero values and corresponding integer array of their indices `indV` are of length $|\mathcal{S}\{v\}| = 5$ and are as follows:

Subscripts	1	2	3	4	5
`valV`	1.	$-2.$	$-3.$	5.	3.
`indV`	1	2	4	6	7

Sparse vectors and matrices in a computer

Sparse vector in a computer

- Alternatively, a linked list can be used.
- linked list - based format: stores matrix rows/columns as items connected by pointers
- linked lists can be cyclic, one-way, two-way
- A figure for demonstration, only values (not their indices) are shown



- rows/columns embedded into a larger array: emulated dynamic behavior

Sparsity

Sparse vector in a computer

- Linked list can be embedded into a large array.

Example

Two possible ways of storing the sparse vector using linked lists.

Subscripts	1	2	3	4	5
Values	1.	$-2.$	$-3.$	5.	3.
Indices	1	2	4	6	7
Links	2	3	4	5	0
Header	1				

Subscripts	1	2	3	4	5
Values	5.	3.	1.	$-2.$	$-3.$
Indices	6	7	1	2	4
Links	2	0	4	5	1
Header	3				

Sparse vectors and matrices in a computer

- Reasons for using linked lists: straightforward adds and removes.

Example

On the left, an entry -4 has been added in position 5. On the right, an entry -2 in position 2 has been removed. $*$ indicates the entry is not accessed. The links that have changed are in bold.

Subscripts	1	2	3	4	5	6
Values	1.	$-2.$	$-3.$	5.	3.	$-4.$
Indices	1	2	4	6	7	5
Links	2	3	4	5	6	0
Header	1					

Subscripts	1	2	3	4	5
Values	1.	$*$	$-3.$	5.	3.
Indices	1	$*$	4	6	7
Links	3	$*$	4	5	0
Header	1				

Sparse vectors and matrices in a computer

Sparse matrix storage

- coordinate (or triplet format: the individual entries of A are held as triplets (i, j, a_{ij}), where i is the row index and j is the column index of the entry $a_{ij} \neq 0$. (dynamic storage format)
- CSR (Compressed Sparse Row) format. The column indices of the entries of A held by rows in an integer array (which we will call `colindA`) of length $nz(A)$, with those in row 1 followed by those in row 2, and so on (with no space between rows). Sorted or unsorted. (static storage format)
- CSC (Compressed Sparse Columns): analogously by columns instead of rows.
- If A is symmetric, only the lower (or upper) triangular part is generally stored.
- Possible to store only $\mathcal{S}\{A\}$.

Sparse vectors and matrices in a computer

Sparse matrix in the coordinate format

- Example matrix $A \in \mathbb{R}^{5\times 5}$

$$\begin{array}{c c} & \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} & \begin{pmatrix} 3. & & & -2. & \\ & 1. & & & 4. \\ -1. & & 3. & & 1. \\ & & & 1. & \\ & 7. & & & 6. \end{pmatrix} \end{array}. \tag{2}$$

Example

Subscripts	1	2	3	4	5	6	7	8	9	10
`rowindA`	3	2	3	4	1	1	2	5	3	5
`colindA`	3	2	1	4	4	1	5	5	5	2
`valA`	3.	1.	-1.	1.	-2.	3.	4.	6.	1.	7.

Sparse vectors and matrices in a computer

Sparse matrix stored using linked lists

- Easy adding and deleting entries is possible if t linked lists are used: the matrix held as a collection of columns, each in a linked list. `colA_head` holds header pointers.

Example

Subscripts	1	2	3	4	5	6	7	8	9	10
`rowindA`	3	2	3	4	1	1	2	5	3	5
`valA`	3.	1.	-1.	1.	-2.	3.	4.	6.	1.	7.
`link`	0	10	0	0	4	3	9	0	8	0
`colA_head`	6	2	1	5	7					

If we consider column 4, then $\texttt{colA\_head}(4) = 5$, $\texttt{rowindA}(5) = 1$ and $\texttt{valA}(5) = -2.$, so the first entry in column 4 is $a_{1,4} = -2.$. Next, $\texttt{link}(5) = 4$, $\texttt{rowindA}(4) = 4$ and $\texttt{valA}(4) = 1.$, so the next entry in column 4 is $a_{4,4} = 1.$.

Sparse vectors and matrices in a computer

Sparse matrix in the CSR format

- CSR format represents A as follows. Here the entries within each row are in order of increasing column index.

Example

Subscripts	1	2	3	4	5	6	7	8	9	10
`rowptrA`	1	3	5	8	9	11				
`colindA`	1	4	2	5	1	3	5	4	2	5
`valA`	3.	-2.	1.	4.	-1.	3.	1.	1.	7.	6.

- In our codes we often use: ia: rowptrA, ja: colindA, aa: valindA

Sparse vectors and matrices in a computer

Sparse matrix: static versuis dynamic formats

- dynamic data structures:
  - – more flexible but this flexibility might not be needed
  - – difficult to vectorize
  - – difficult to keep spatial locality
  - – used preferably for storing vectors
- static data structures:
  - – ad-hoc insertions/deletions should be avoided (better algorithms)
  - – much simpler to vectorize
  - – efficient access to rows/columns

Sparse vectors and matrices in a computer

Simulating dynamic storage format

- A disadvantage of linked list storage: prohibits the fast access to rows (or columns) of the matrix. And this is needed!
- Simulated dynamism of storage schemes: storage format with some additional elbow space for new non zero entries of A is needed.
- Often the case in approximate factorizations where new non zero entries can be added and/or removed and it is hard to predict the necessary space in advance.
- In this case, the elbow space can embed new non zeros.
- The format is called the DS format.

Sparse vectors and matrices in a computer

Sparse matrix: DS formats

- Consider again the sparse matrix $A \in \mathbb{R}^{5\times 5}$ (2). The DS format represents A as follows.

Example

Subscripts	1	2	3	4	5	6	7	8	9	10	11	12	13	14
`rowptrA`	1	5	8	12	14									
`colindA`	1	4			2	5		1	3	5		4		2
`valAR`	3.	-2.			1.	4.		-1.	3.		1.		1.	7.
`rowlength`	2	2	3	1	2									
`colptrA`	1	4	6	9	12									
`rowindA`	1	3		2	5	3			1	4		2	3	5
`valAC`	3.	-1.		1.	7.	3.			-2.	1.		4.	1.	6.
`collength`	2	2	1	2	3									

Sparse vectors and matrices in a computer

Sparse matrix: DS formats

- It can happen that the free space between row and/or column segments disappears throughout a computational algorithm. Then the DS format must be reorganized.
- In particular, a row segment can be moved to the end of the arrays `valAR` and `colindA` implying also a corresponding update in `rowptrA`. The space where the row i originally resided is then denoted as free.
- If there is no free space at the end of the arrays `valAR` and `colindA`, a compression of the row segments or full reallocation should be done.
- While the DS format seems to be complicated, it can be extremely useful in some cases. Surprisingly efficient if the amount of changes is limited as it often is in approximate factorizations.

Block formats

- Blocked formats may be used to accelerate multiplication between a sparse matrix and a dense vector.
- The Variable Block Row (VBR) format groups together similar adjacent rows and columns.
- The data structure of the VBR format uses six arrays. Integer arrays `rptr` and `cptr` hold the index of the first row in each block row and the index of the first column in each block column, respectively. In many cases, the block row and column partitionings are conformal and only one of these arrays is needed. The real array `valA` contains the entries of the matrix block-by-block in column-major order. The integer array `indx` holds pointers to the beginning of each block entry within `valA`. The index array `bindx` holds the block column indices of the block entries of the matrix and, finally, the integer array `bptr` holds pointers to the start of each row block in `bindx`.

Sparse matrix: DS formats

Example

Consider the sparse matrix $A \in \mathbb{R}^{8\times 8}$

$$
\begin{array}{c c}
 & \begin{array}{cccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{array} \\
\begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \end{array} &
\begin{pmatrix}
1. & 2. & & & & 3. & & \\
4. & 5. & & & & 6. & & \\
 & & 7. & 8. & 9. & 10. & & \\
11. & 12. & & & & & 15. & 16. \\
 & 13. & & & & & 17. & \\
14. & & & & & & & 18. \\
 & & 19. & & 20. & & & \\
 & & 21. & 22. & & & &
\end{pmatrix}
\end{array}.
$$

Here the row blocks comprise rows 1:2, 3, 4:6 and 7:8. The column blocks comprise columns 1:2, 3:5, 6, 7:8. The VBR format stores A as follows.

Sparse matrix: DS formats

Example

Subscripts	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
`rptr`	1	3	4	7	9												
`cptr`	1	3	6	7	9												
`valA`	1.	4.	2.	5.	3.	6.	7.	8.	9.	10.	11.	14.	12.	13.	15.	17.	16.
`indx`	1	5	7	10	11	15	19										
`bindx`	1	3	2	3	1	4	2										
`bptr`	1	3	5	7													

Sparse matrices and data structures

Matmats in CSR/CSC

1) CSR - CSC

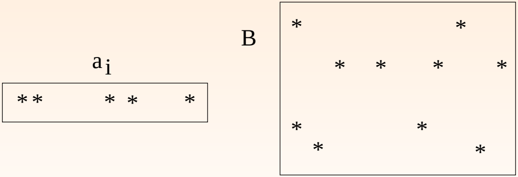
$$C = AB, A = \begin{pmatrix} a_1 \\ \vdots \\ a_m \end{pmatrix}, B = \left(b_1, \ldots, b_n\right), C = (c_{ij}) \tag{3}$$

- Each entry c_{ij} computed as a product of a compressed row of A and compressed column of B
- Not clear whether the result c_{ij} is nonzero
- Consequently: $O(n^3)$ operations, not useful for sparse matrices.

Matmats in CSR/CSC

2) CSR - CSR

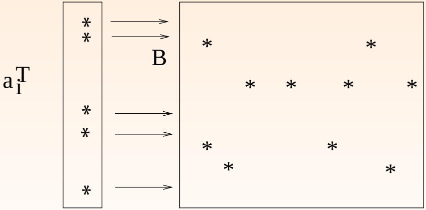
$$C = AB, A = \begin{pmatrix} a_1 \\ \vdots \\ a_m \end{pmatrix}, B = \begin{pmatrix} b_1 \\ \vdots \\ b_n \end{pmatrix}, C = (c_{ij}) \tag{4}$$



Matmats in CSR/CSC

2) CSR - CSR

$$C = AB, A = \begin{pmatrix} a_1 \\ \vdots \\ a_m \end{pmatrix}, B = \begin{pmatrix} b_1 \\ \vdots \\ b_n \end{pmatrix}, C = (c_{ij}) \tag{5}$$



Sparse matrices and data structures

Matmats in CSR/CSC

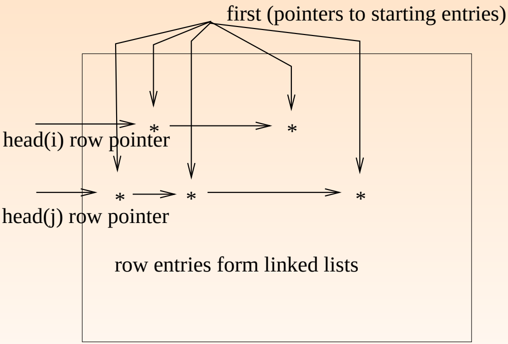
3) CSC - CSR

$$C = AB, A = \left(a_1, \ldots, a_m\right), B = \begin{pmatrix} b_1 \\ \vdots \\ b_n \end{pmatrix}, C = (c_{ij}) \tag{6}$$

- How one can store A by CSC and pass it by rows?
- Pointers to first entries in columns: (array $first$)
- First test: nonzero in the first row $\rightarrow$ move one step down, add next nonzero into the list $value(next)$
- Complexity: $O(nonzeros) + O(n)$

Matmats in CSR/CSC

3) CSC - CSR



Based on forming virtual rows in A