sparse matrices and graphs

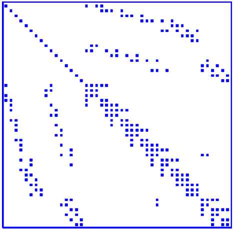
L. Olson

Department of Computer Science
University of Illinois at Urbana-Champaign

objectives

- Convert a graph into a *sparse* matrix
- Go over a few sparse matrix storage formats
- Give an example of lower memory benefits
- Give an example of computational complexity benefits

sparse matrices



- Vague definition: matrix with few nonzero entries
- For all practical purposes: an $m \times n$ matrix is sparse if it has $\mathcal{O}(\min(m, n))$ nonzero entries.
- This means roughly a constant number of nonzero entries per row and column

sparse matrices

- Other definitions use a slow growth of nonzero entries with respect to n or m.
- Wilkinson's Definition: "..matrices that allow special techniques to take advantage of the large number of zero elements." (J. Wilkinson)"
- A few applications which lead to sparse matrices: Structural Engineering, Computational Fluid Dynamics, Reservoir simulation, Electrical Networks, optimization, data analysis, information retrieval (LSI), circuit simulation, device simulation, ...

sparse matrices: the goal

- To perform standard matrix computations economically i.e., without storing the zeros of the matrix.
- For typical Finite Element /Finite difference matrices, number of nonzero elements is $\mathcal{O}(n)$.

Example

To add two square dense matrices of size n requires $\mathcal{O}(n^2)$ operations. To add two sparse matrices A and B requires $\mathcal{O}(nnz(A) + nnz(B))$ where $nnz(X) =$ number of nonzero elements of a matrix X.

remark

A^{-1} is usually dense, but L and U in the LU factorization may be reasonably sparse (if a good technique is used).

goal

- Principle goal: *solve*
$$Ax = b$$
where $A \in \mathbb{R}^{n\times n}$, $x, b \in \mathbb{R}^n$
- Assumption: A is very sparse
- General approach: iteratively improve the solution
- Given x_0, ultimate "correction" is
$$x_1 = x_0 + e_0$$
where $e_0 = x - x_0$, thus $Ae_0 = Ax - Ax_0$,
- or
$$x_1 = x_0 + A^{-1} r_0$$
where $r_0 = b - Ax_0$

goal

- Principle difficulty: how do we “approximate” $A^{-1}r$ or reformulate the iteration?
- One simple idea:

$$x_1 = x_0 + \alpha r_0$$

- operation is inexpensive if r_0 is inexpensive
- requires very fast sparse mat-vec (matrix-vector multiply) Ax_0

sparse matrices

- So how do we store A?
- Fast mat-vec is certainly important; also ask
  - what type of access (rows, cols, diag, etc)?
  - dynamic allocation?
  - transpose needed?
  - inherent structure?
- Unlike dense methods, not a lot of standards for iterative
  - dense BLAS have been long accepted
  - sparse BLAS still iterating
- Even data structures for dense storage not as obvious
- Sparse operations have low operation/memory reference ratio

popular storage structures

DNS	Dense	**ELL**	Ellpack-Itpack
BND	Linpack Banded	**DIA**	Diagonal
COO	Coordinate	**BSR**	Block Sparse Row
CSR	Compressed Sparse Row	**SSK**	Symmetric Skyline
CSC	Compressed Sparse Column	**BSR**	Nonsymmetric Skyline
MSR	Modified CSR	**JAD**	Jagged Diagonal
LIL	Linked List		

note: CSR = CRS, CCS = CSC, SSK = SKS in some references

dns

$$A = \begin{bmatrix} 1.0 & 2.0 & 3.0 \\ 4.0 & 5.0 & 6.0 \\ 7.0 & 8.0 & 9.0 \end{bmatrix}$$

$$AA = \begin{bmatrix} 3 & 3 & 1.0 & 2.0 & 3.0 & 4.0 & 5.0 & 6.0 & 7.0 & 8.0 & 9.0 \end{bmatrix}$$

- simple
- row-wise
- easy blocked formats

COO

$$A = \begin{bmatrix} 1 & 0 & 0 & 2 & 0 \\ 3 & 4 & 0 & 5 & 0 \\ 6 & 0 & 7 & 8 & 9 \\ 0 & 0 & 10 & 11 & 0 \\ 0 & 0 & 0 & 0 & 12 \end{bmatrix}$$

$$\begin{array}{lllllllllllllll} AA & = & [& 12.0 & 9.0 & 7.0 & 5.0 & 1.0 & 2.0 & 11.0 & 3.0 & 6.0 & 4.0 & 8.0 & 10.0 &] \\ JR & = & [& 5 & 3 & 3 & 2 & 1 & 1 & 4 & 2 & 3 & 2 & 3 & 4 &] \\ JC & = & [& 5 & 5 & 3 & 4 & 1 & 4 & 4 & 1 & 1 & 2 & 4 & 3 &] \end{array}$$

- simple, often used for entry

csr

$$A = \begin{bmatrix} 1 & 0 & 0 & 2 & 0 \\ 3 & 4 & 0 & 5 & 0 \\ 6 & 0 & 7 & 8 & 9 \\ 0 & 0 & 10 & 11 & 0 \\ 0 & 0 & 0 & 0 & 12 \end{bmatrix}$$

$$\begin{array}{lllllllllllllll} AA & = & [& 1.0 & 2.0 & 3.0 & 4.0 & 5.0 & 6.0 & 7.0 & 8.0 & 9.0 & 10.0 & 11.0 & 12.0 &] \\ JA & = & [& 1 & 4 & 1 & 2 & 4 & 1 & 3 & 4 & 5 & 3 & 4 & 5 &] \\ IA & = & [& 1 & 3 & 6 & 10 & 12 & 13 &] \end{array}$$

- Length of AA and JA is nnz; length of IA is $n+1$
- $IA(j)$ gives the index (offset) to the beginning of row j in AA and JA (one origin due to Fortran)
- no structure, fast row access, slow column access
- related: CSC, MSR

msr

$$A = \begin{bmatrix} 1 & 0 & 0 & 2 & 0 \\ 3 & 4 & 0 & 5 & 0 \\ 6 & 0 & 7 & 8 & 9 \\ 0 & 0 & 10 & 11 & 0 \\ 0 & 0 & 0 & 0 & 12 \end{bmatrix}$$

$$\begin{array}{lllllllllllll} AA = & [1.0 & 4.0 & 7.0 & 11.0 & 12.0 & * & 2.0 & 3.0 & 5.0 & 6.0 & 8.0 & 9.0 & 10.0] \\ JA = & [7 & 8 & 10 & 13 & 14 & 14 & 4 & 1 & 4 & 1 & 4 & 5 & 3] \end{array}$$

- places importance on diagonal (often nonzero and accessed frequently)
- first n entries are the diag
- $n + 1$ is empty
- rest of AA are the nondiagonal entries
- first $n + 1$ entries in JA give the index (offset) of the beginning of the row (the IA of CSR is in this JA)
- rest of JA are the columns indices

dia

$$A = \begin{bmatrix} 1 & 0 & 2 & 0 & 0 \\ 3 & 4 & 0 & 5 & 0 \\ 0 & 6 & 7 & 0 & 8 \\ 0 & 0 & 9 & 10 & 0 \\ 0 & 0 & 0 & 11 & 12 \end{bmatrix} \quad DIAG = \begin{bmatrix} * & 1.0 & 2.0 \\ 3.0 & 4.0 & 5.0 \\ 6.0 & 7.0 & 8.0 \\ 9.0 & 10.0 & * \\ 11.0 & 12.0 & * \end{bmatrix} \quad IOFF = \begin{bmatrix} -1 & 0 & 2 \end{bmatrix}$$

- need to know the offset structure
- some entries will always be empty

try it...

$$A = \begin{bmatrix} 7 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 0 & 0 & 0 \\ 0 & 2 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 0 & 6 & 4 \end{bmatrix}$$

- CSR
- COO

example

$$A = \begin{bmatrix} 7 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 0 & 0 & 0 \\ 0 & 2 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 0 & 6 & 4 \end{bmatrix}$$

i	IA	JA	AA
1	2	2	1
2	3	4	2
3	4	5	5
4	2	3	2
5	5	6	4
6	1	1	7
7	5	5	6
8	3	2	2

COO

i	IA	JA	AA
1	1	1	7
2	2	2	1
3	4	3	2
4	6	2	2
5	7	4	2
6	9	5	5
7	-	5	6
8	-	6	4

CSR

sparse matrix-vector multiply

$z = Ax$, $A_{m\times n}$, $x_{n\times 1}$, $z_{m\times 1}$

```
input A, x
z = 0
for i = 1 to m
   for col = A(i,:)
      z(i) = z(i) + A(i,col)x(col)
   end
end
```

sparse matrix-vector multiply

$z = Ax$, $A_{m \times n}$, $x_{n \times 1}$, $z_{m \times 1}$

```
DO I=1, m
  Z(I)=0
  K1 = IA(I)
  K2 = IA(I+1)-1
  DO J=K1, K2
    z(I) = z(I) + A(J)*x(JA(J))
  ENDDO
ENDDO
```

- $\mathcal{O}(nnz)$
- marches down the rows
- very cheap

sparse matrix-matrix multiply

- ways to optimize ("SMPP", Douglas, Bank)

$Z = AB$, $A_{m \times n}$, $B_{n \times p}$, $z_{m \times p}$

```
for i = 1 to m
  for j = 1 to n
    Z(i,j) = dot(A(i,:), B(:,j))
  end
end
return Z
```

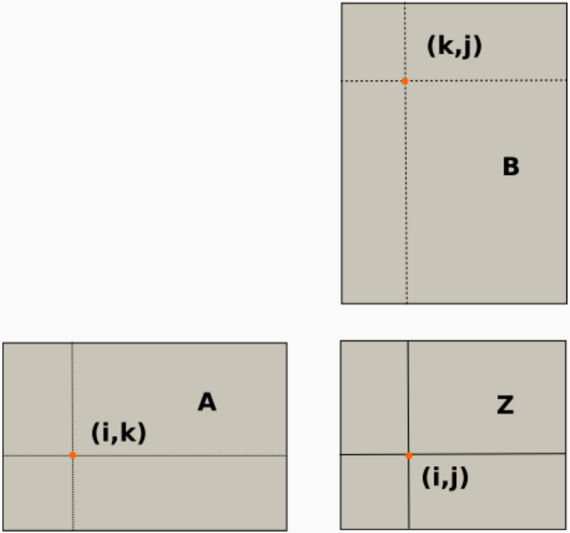
- obvious problem: column selection of B is expensive for CSR
- not-so-obvious problem: Z is sparse(!!), but the algorithm doesn't account for this.

sparse matrix-matrix multiply

$Z = AB$, $A_{m \times n}$, $B_{n \times p}$, $z_{m \times p}$

```
1 Z=0
2 for i = 1 to m
3   for colA = A(i,:)
4     for colB = A(colA,:)
5       Z(i,colB)+ = A(i,colA) · B(colA,colB)
6     end
7   end
8 end
9 return Z
```

- only marches down rows
- only computes nonzero entries in Z (aside from fortuitous subtractions)
- line 5 will do and insert into Z. Two options:
  1. precompute sparsity of Z in CSR
  2. use LIL for Z



(k,j)
B
A
(i,k)
Z
(i,j)

some python

$$A = \begin{bmatrix} 7 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 0 & 0 & 0 \\ 0 & 2 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 0 & 6 & 4 \end{bmatrix}$$

i	IA	JA	AA
1	2	2	1
2	3	4	2
3	4	5	5
4	2	3	2
5	5	6	4
6	1	1	7
7	5	5	6
8	3	2	2

COO

```python
from scipy import sparse
from numpy import array
IA=array([1,2,3,1,4,0,4,2])
JA=array([1,3,4,2,5,0,4,1])
V=array([1,2,5,2,4,7,6,2])

A=sparse.coo_matrix((V,(IA,JA)),shape=(5,6))
```

some python

From COO to CSC:

```
from scipy import sparse
from numpy import array
import pprint
IA=array([1,2,3,1,4,0,4,2])
JA=array([1,3,4,2,5,0,4,1])
V=array([1,2,5,2,4,7,6,2])

A=sparse.coo_matrix((V,(IA,JA)),shape=(5,6)).tocsr()
```

Nonzeros:

```
print(A.nnz)
```

To full and view:

```
B=A.todense()
pprint.pprint(B)
```

simple matrix iterations

- Solve

$$Ax = b$$

- Assumption: A is very sparse
- Let $A = N + M$, then

$$\begin{aligned} Ax &= b \\ (N + M)x &= b \\ Nx &= b - Mx \end{aligned}$$

- Make this into an iteration:

$$\begin{aligned} Nx_k &= b - Mx_{k-1} \\ x_k &= N^{-1}(b - Mx_{k-1}) \end{aligned}$$

- Careful choice of N and M can give effective methods
- More powerful iterative methods exist