Using Graph Theory to Control Fill-in for Sparse Matrix Reduction to RREF over Fields of non-zero characteristic

. . .

Joint work: G. Bard and R. Miller

Outline

- Introduction to Sparse Matrices over $\mathbb{C}, \mathbb{R}, \mathbb{Q}$.
- Overview of Graph Theoretic Methods of Matrix Factoring: $\mathbb{C}, \mathbb{R}, \mathbb{Q}$
- What breaks over characteristic $\neq 0$?
- Graph Theory Terminology.
- Core Idea: The Damage Formula.
- Generation One: The Basic Algorithm.
- Changes for Generation Two: Co-Pivots.
- Experimental Results are missing right now.

Sparse Matrices over $\mathbb{C}, \mathbb{R}, \mathbb{Q}$

- Occur in too many applications to list.
- Can be structured or otherwise.
- "Most entries" are zero.
- The "content", denoted c, of a matrix is the number of non-zero entries.
- $\beta = c/mn$ is the density of an $m \times n$ matrix.
- β is the probability that a random element is non-zero.
- Typically $10^{-3} < \beta < 10^{-1}$.

The Shadow!

- The shadow of a matrix A is a matrix S with
$$S_{ij} = \begin{cases} 1 & A_{ij} \neq 0 \\ 0 & A_{ij} = 0 \end{cases}$$
- We simply erase the non-zero entries and replace them with 1.
- The shadow graph of a square matrix A is the directed graph (digraph) that has adjacency matrix equal to the shadow of A.
- This means there is one vertex for each row and column, and we draw an edge from v_x to v_y if and only if $A_{xy} \neq 0$.

- If the original matrix is rectangular, then just let $|V| = \max(m, n)$, because the storage cost of a graph is proportional to $|E|$, and $|V|$ does not matter much.

What is Fill-in?

- If you have a sparse matrix, and perform Gaussian Elimination in the high-school way, then
- It will become dense VERY quickly.
- Even with heuristics like "take the lowest weight row possible" at each step, it still becomes dense 1/2 way through or so, maybe earlier.
- Since a sparse matrix can have a dense inverse, your computer might not have enough memory to perform the Gaussian Elimination.
- Therefore, controlling this process "fill-in" is critical.

Philosophy

- In order to understand why we do what we do over char $\neq 0 \ldots$
- $\ldots$ it becomes necessary to understand the char $= 0$ case.
- For sparse matrices, solving $Ax = b$ is almost always done as a Cholesky Factorization. (to be explained later).
- D. J. Rose in 1972 noticed that performing one Cholesky step is identical to a particular graph theoretic operation on the shadow-graph of A.

History

- D. J. Rose in 1972 noticed that performing one Cholesky step is identical to a particular graph theoretic operation on the shadow-graph of A.
- Using a simple greedy-algorithm approach, he found a way to sequence the steps of a Cholesky factorization so as to minimize fill-in. This is the “min-degree” algorithm, and many papers have been written about it.
- This won’t work over characteristic $\neq 0$, for reasons we will get to shortly.

Matrix Factorizations

- Solving $A\vec{x} = \vec{b}$ is usually a cubic time or $n^{2.807}$ time operation in practice, but. . .
- If A is upper-triangular, lower-triangular, a permutation matrix, an orthogonal matrix, or a diagonal matrix (just as examples) then one can solve $A\vec{x} = \vec{b}$ in quadratic time or better.
- Therefore, it makes sense to factor A into a product of matrices of that type.

Examples of Factorizations

- Common Factorizations include
- $A = LUP$
- $A = QR$
- $A = LDL^T$
- $PAP^{-1} = LL^T$ Cholesky Factorization (the fastest).

Cholesky Factorization

- If $PAP^{-T} = LL^T$ then since LL^T is symmetric and square, so must A be also.
- Note $P^T = P^{-1}$.
- Turns out such a factorization exists iff A is positive semi-definite.
- This means that $Q_A(\vec{x}) = \vec{x}^T A\vec{x}$, the quadratic form derived from A, is never negative for any vector x. (There are other definitions).
- For both the dense and sparse cases, this is usually the fastest factorization.

- Developed by a WWI French artillery officer so that he could factor matrices quickly during combat conditions.

Limitations of the Cholesky

- So, A must be symmetric, therefore square, as well as positive semi-definite!
- For reasons of physics, or sometimes mathematical reasons, e.g. The Method of Least Squares, it will be positive semi-definite.
- What if it isn't?
- If A is square and non-singular, then $A^T A$ will be symmetric, positive semi-definite!
- Provided that A has a trivial null-space, then $A^T A$ will be square, symmetric, positive semi-definite, even if A is rectangular!

- Even if A has a null-space, this can be handled.

General Recipe over $\mathbb{C}, \mathbb{R}, \mathbb{Q}$

To solve $A\vec{x}_1 = \vec{b}_1, A\vec{x}_2 = \vec{b}_2, \ldots, A\vec{x}_\ell = \vec{b}_\ell$, do:

- Calculate $A^T A$.
- Factor $A^T A = P^{-1} L L^T P$. (The Cholesky).
- For $i = 1 to \ell$ do
  - Solve $P^{-1}\vec{m}_1 = \vec{b}_i$
  - Solve $L\vec{m}_2 = \vec{m}_1$
  - Solve $L^T\vec{m}_3 = \vec{m}_2$
  - Solve $P\vec{x}_i = \vec{m}_3$

What breaks over Characteristic$\neq 0$?

- The whole above procedure is predicated on the fact that

  Nullspace(A) = Nullspace ($A^T A$)

- For characteristic $\neq 0$ this is false.
- We can only say Nullspace(A) $\subset$ Nullspace ($A^T A$)
- Not to mention it is hard to determine the equivalent notion of positive semi-definite because $\vec{x}^T A \vec{x} \geq 0$ requires a notion of $\geq$, which does not exist in finite characteristic.
- Also, over $\mathbb{C}, \mathbb{R}, \mathbb{Q}$, no one ever developed any other approaches, since the Cholesky is so very fast in the sparse case.

And now we'll do it my way!

Graph Theoretic Terminology

- Let $G = V, E$ be a directed graph or digraph.
- This means that if there is an edge from v_i to v_j, then there is not necessarily an edge from v_j to v_i.
- We say, for an edge from v_x to v_y that
- v_x is a parent of v_y and
- v_y is a child of v_x
- Not only can you have many, one, or no parents/children, we allow self-loops (edges from v_x to v_x and so you can be your own parent/child.

What does this really mean?

- The set of vertices that are parents of v_y would be all those v_x with an edge v_x, v_y.
- More simply, it would be each row x, such that there is a non-zero entry in column y.
- Parent set = a column.
- The set of vertices that are children of v_x would be all those v_y with an edge v_x, v_y.
- More simply, it would be each column y, such that there is a non-zero entry in row x.
- Child set = a row.

Other Notions

- The content of the matrix is the number of edges.
- Fill-in is an increase in the number of edges.
- A self-loop is a main-diagonal element.
- A childless vertex is an empty row.
- A parentless vertex is an empty column.

Warm-Up: Adding two Rows

- Suppose we add two rows, e.g. row x to row z, and store the answer in row z.
- An entry A_{zy} of row z is non-zero after this if either A_{xy} was non-zero, or if A_{zy} was non-zero.
  - Of course, if $A_{xy} = -A_{zy}$ then this is false, but unless we force this, we assume it will not happen accidentally.
  - (Very false over $\mathbb{GF}(2)$, but true with probability equal to the size of the field, in general).
  - This is the "no accidental cancellations" assumption, very common in this topic.

So let's make that assumption

- An entry A_{zy} of row z is non-zero after this if either A_{xy} was non-zero, or if A_{zy} was non-zero.
- This means that y will be a child of z after this operation if either y was a child of x or y was a child of z.
- More plainly, we insert the set of children of x to the set of children of z.
- The number of new elements is $|\text{children}(v_x)| - |\text{children}(v_x) \cap \text{children}(v_z)|$
- We call the (net) number of new edges, i.e. number added minus number deleted, the "damage" of an action.

On the Set Intersection

- We will need to calculate this: $|\text{children}(v_x)| - |\text{children}(v_x) \cap \text{children}(v_z)|$ extremely often.
- This was the cause of much grief!
- At first we approximated this as: $|\text{children}(v_x) \cap \text{children}(v_z)| = 0$, that was bad.
- In Gaussian Elimination, you wouldn't add row z to row x unless they both had a non-zero in the "pivot column". Thus the intersection is at least one.
- Then we tried $|\text{children}(v_x) \cap \text{children}(v_z)| = 0$.
- That's still not quite enough!

Randomly Distributed Intersection

- If we assume that the ones are randomly distributed, then we can calculate the expected value of the intersection. (This is our second assumption).

- ... but, ... there are no ones to the left of column i after the ith iteration. So, what we need is a notion of "active submatrix density."

- The active submatrix is from $(1, i)$ to (m, n). There should be $i - 1$ non-zeroes outside that area, and if the matrix has content c then $c - i + 1$ non-zeroes inside it. Thus the "β" of the active submatrix is:

$$\alpha = \frac{c - i + 1}{[m][n - i + 1]} = \frac{\beta - (i - 1)/mn}{1 - (i - 1)/n} \approx \frac{\beta}{1 - (i - 1)/n}$$

- And then α^2 is the probability of an entry in the active part of the row being non-zero for both row x and row z.
- Therefore the intersection has expected size $\alpha^2(n - i + 1)$.
- But we know there is a shared non-zero element, so $\alpha^2(n - i) + 1$.
- If that is the size of the overlap, then the damage is clearly

$$|\text{children}(v_x)| - \alpha^2(n - i) - 1$$

How Does that Help?

- The damaging of adding row x to row z is:

$$|\text{children}(v_x)| - \alpha^2(n-i) - 1$$

- How about pivoting on A_{xy}. What does that mean?
  - Multiply row x by the scalar A_{xy}^{-1} to force $A_{xy} = 1$.
  - For any $A_{zy} \neq 0$ with $z \neq x$ do
    - Add row z to row x.

The Damage of Pivoting

- If we pivot on A_{xy} then there will be a row-add for each non-zeor in column y, minus 1 for the pivot row itself which doesn't get added.
- This is $|\text{parents}(v_y)| - 1$ row-adds.
- Then we have $\left(|\text{children}(v_x)| - \alpha^2(n-i) - 1\right)(|\text{parents}(v_y)| - 1)$ new edges.
- Ah, we said no accidental cancelations but the deliberate ones? All of column y will go to only one non-zero element.
- Thus $(|\text{parents}(v_y)| - 1)$ edges are deleted, and so we have a net effect of

$$\left(|\text{children}(v_x)| - \alpha^2(n-i) - 2\right)(|\text{parents}(v_y)| - 1)$$

Damage of Pivoting

- Then we are left with

$$\Big(|\text{children}(v_x)| - \alpha^2(n-i) - 2\Big)\,(|\text{parents}(v_y)| - 1)$$

- This is the damage of pivoting on A_{xy}.
- Note it can be positive, zero, or negative.

How to Choose a Pivot?

- This is a fairly easy computation, but it would be long to compute it for each edge in the graph.
- For A_{xy} to be a pivot:
  - $A_{xy} \neq 0$ or there must be an edge from v_x to v_y, or v_y is a child of v_x.
  - Nothing in row x must have been used as a pivot before.
  - Nothing in column y must have been used as a pivot before.
- Maintain a linked list of unused parents, and unused children.
- Delete as you use vertices.

Example

- Suppose the number of unused-parents $<$ the number of unused-children:
- For each unused-parent v_x do
  - Does it have any children that are on the list: unused-children?
  - If not: delete it from unused-parents.
  - If so: among the children on the unused-children list, take the one v_y with the fewest parents.
  - Mark the choice A_{xy} with the damage:
    $\left(|\text{children}(v_x)| - \alpha^2(n-i) - 2\right)(|\text{parents}(v_y)| - 1)$

Inner Loop

- Therefore we do that for each unused-parent. If the number of unused children is smaller, we can swap parents/children in the pseudocode and make an identical list.
- This gives us a list of “candidate” pivots, and their damages.
- Ah, but we had to do some non-trivial computing to get here.
- So we want the fewest number of loop runs possible!

Co-Pivots

- Suppose two pivot rows had non-overlapping column support. (i.e. they never both had a one in the same column).
- Alternatively suppose two pivot columns had non-overlapping row support. (i.e. they never had a one in the same row).
- Thus for two potential pivots $A_{x1,y1}$ and $A_{x2,y2}$ if either:
  - The rows x_1 and x_2 are disjoint (i.e. the children of x_1 and the children of x_2 are disjoint as sets).
  - OR The columns y_1 and y_2 are disjoint (i.e. the parents of y_1 and the parents of y_2 are disjoint as sets).
  - Then you can pivot on A_{x_1,y_1} and A_{x_2,y_2} at the same time, or in either order, and they won't interfere with each other.

The Algorithm

- Each parent or child vertex nominates a parent-child pair as a pivot, with a damage score.
- Sort those pivots by order of damage, lowest first. (some are negative).
- Enqueue the lowest damage pivot vertex.
  - For each remaining pivot:
  - Will it interfere with any of the enqueued pivots?
  - If not, enqueue it.
- Then update the graph based on these pivots.

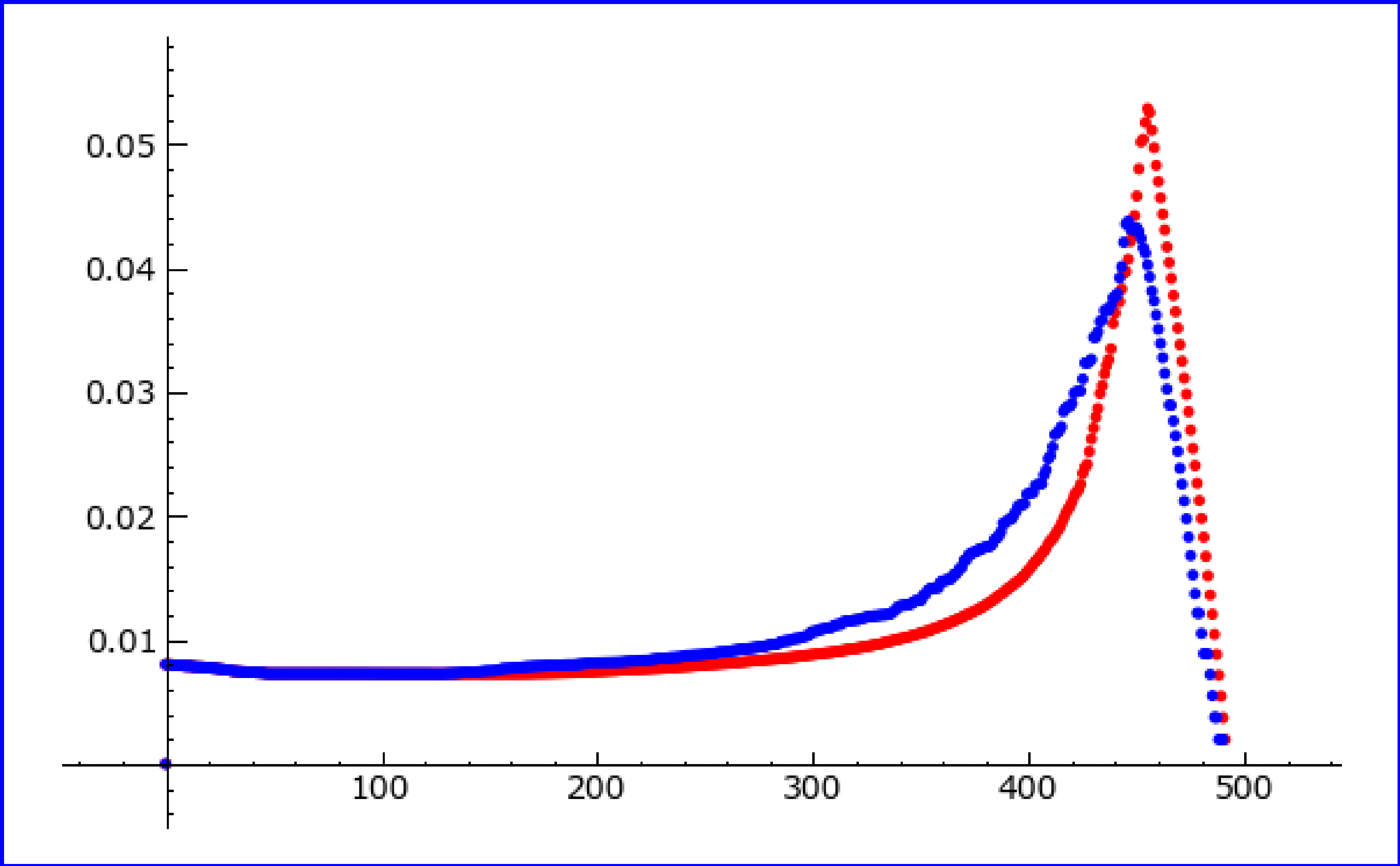
What does Update Mean?

- This we perform exactly, not approximately.
- Suppose we pivot on A_{xy}
- For each parent of v_y (call it v_z), add the children of v_x to the children of v_z.
- Then remove v_y from the children of v_z.
- All those new children of v_z also get v_z added as one of their parents.
- Finally remove v_z as a parent of v_y.
- Provided there are no accidental cancellations, this is an EX-ACT update of the graph.

One Last Innovation

- Once a row or column becomes dense, it is unlikely to become sparse again.
- Also, if a row is dense (a vertex with many children) or a column is dense (a vertex with many parents) it is unlikely to be chosen as pivot-parent or pivot-child respectively.
- Therefore, if the number of children of v_x is greater than $10\sqrt{\max(m,n)}$ or some other arbitrary threshold, then delete it from the unused-parents list.
- If the number of parents of v_y is greater than $10\sqrt{\max(m,n)}$ or some other arbitrary threshold, then delete it from the unused-child list.
- These are called procrastinator nodes.

Experimental Results Coming Soon!



0.05
0.04
0.03
0.02
0.01
100
200
300
400
500

Thank you, that is all!