Off-diagonal low-rank preconditioner for PageRank problems

Zhao-Li Shen

E-mail: z.shen@rug.nl

Supervisor: Dr. Bruno Carpentieri

Instituut voor Wiskunde en Informatica, Rijksuniversiteit Groningen

June 26, 2015

Background

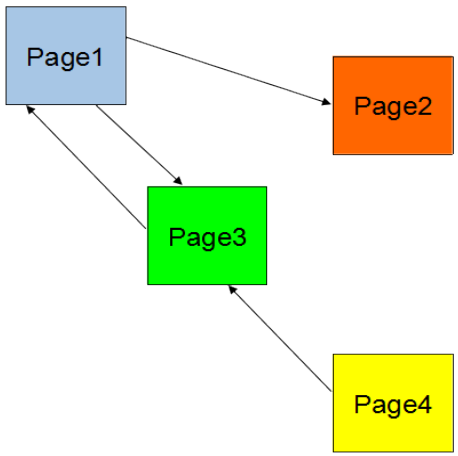
PageRank problem: ranking pages based on their adjacency structure.

university of groningen

Background

PageRank problem: ranking pages based on their adjacency structure.

The structure of a network can be represented by a adjacency matrix G. If the j_{th} page points to the i_{th} page, $g_{ij} = 1$, otherwise, $g_{ij} = 0$.

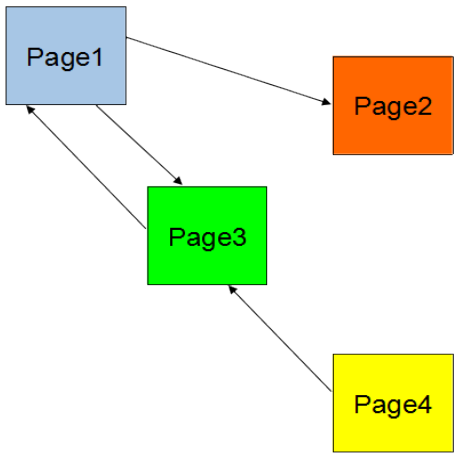


university of groningen

Background

PageRank problem: ranking pages based on their adjacency structure.

The structure of a network can be represented by a adjacency matrix G. If the j_{th} page points to the i_{th} page, $g_{ij} = 1$, otherwise, $g_{ij} = 0$.



$$\longrightarrow \qquad G = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

university of groningen

Background

Basic idea:

1) Consider a random walk on this structure
2) Use the stationary distribution to rank pages.

university of groningen

Background

Basic idea:

1) Consider a random walk on this structure

2) Use the stationary distribution to rank pages.

The transition probability from page j to page i is defined as the reciprocal of page j's outlink amount.

$$\text{transition probability matrix } P = \begin{cases} p_{ij} = \frac{1}{\sum_{k=1}^{n} g_{kj}}, & \text{if } g_{ij} = 1 \\ p_{ij} = 0, & otherwise \end{cases} \tag{1}$$

The stationary distribution x satisfies:

$$Px = x,\ x > 0,\ ||x||_1 = 1 \tag{2}$$

To guarantee a unique solution of system (2), P must satisfy Perron-Frobenius(P.F.) theorem.

university of groningen

Background

[**P.F. Theorem**] *If A is an irreducible non-negative $n \times n$ matrix with spectral radius $\rho(A) = r$, there is a unique positive eigenvector of eigenvalue r up to scale.*

As P may be reducible, system (2) is usually modified as

$$(\alpha P + (1-\alpha)ve^T)x = x, \tag{3}$$

to get irreducibility, where v is a distribution over all pages, e is a vector with all ones and $0 < \alpha < 1$ is a teleportation paramete.

Since $e^T x = 1$, system (3) can be reformulated as the final system:

$$(I - \alpha P)x = (1-\alpha)v. \tag{4}$$

$$\longrightarrow Ax = b,$$

where $A = I - \alpha P$ and $b = (1-\alpha)v$

university of groningen

Off-diagonal low-rank property

From the definition of P: If $g_{ij} = 1$, $p_{ij} = \frac{1}{\sum_{k=1}^{n} g_{kj}}$, else $p_{ij} = 0$,

Property 1 Nonzero elements of columns all have the same value.

$$G = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \end{pmatrix} \longrightarrow P = \begin{pmatrix} \frac{1}{4} & 0 & \frac{1}{3} & 0 \\ \frac{1}{4} & \frac{1}{3} & \frac{1}{3} & 0 \\ \frac{1}{4} & \frac{1}{3} & \frac{1}{3} & 0 \\ \frac{1}{4} & \frac{1}{3} & 0 & 1 \end{pmatrix}$$

This property gives a benefit: rows are same if their patterns are same.

university of groningen

Off-diagonal low-rank property

From the definition of P: If $g_{ij} = 1$, $p_{ij} = \frac{1}{\sum_{k=1}^{n} g_{kj}}$, else $p_{ij} = 0$,

Property 1 Nonzero elements of columns all have the same value.

$$G = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \end{pmatrix} \longrightarrow I - \alpha P = \begin{pmatrix} 1-\alpha\frac{1}{4} & 0 & -\frac{1}{3} & 0 \\ -\frac{1}{4} & 1-\alpha\frac{1}{3} & -\frac{1}{3} & 0 \\ -\frac{1}{4} & -\frac{1}{3} & 1-\alpha\frac{1}{3} & 0 \\ -\frac{1}{4} & -\frac{1}{3} & 0 & 1-\alpha 1 \end{pmatrix}$$

This property gives a benefit: rows are same if their patterns are same.

The final coefficient matrix A keeps this property except the diagonal elements, since $A = (I - \alpha P)$ destroy this property on diagonal elements.

Our object: construct efficient preconditioner for A by exploiting this property.

university of groningen

Off-diagonal low-rank property

With a block partition dividing A into diagonal part D and off-diagonal part B_{off},

$$A = \begin{pmatrix} \boxed{D_1} & off_{12} & off_{13} \\ off_{21} & \boxed{D_2} & off_{23} \\ off_{31} & off_{23} & \boxed{D_2} \end{pmatrix} \rightarrow D = \begin{pmatrix} D_1 & & \\ & D_2 & \\ & & D_3 \end{pmatrix}, B_{off} = \begin{pmatrix} & off_{12} & off_{13} \\ off_{21} & & off_{23} \\ off_{31} & off_{32} & \end{pmatrix}$$

where B_{off} has some same row patterns, we can easily implement low-rank factorization of B_{off}.

$$B_{off} = \begin{pmatrix} 0 & 0 & * & * & * & * \\ 0 & 0 & * & * & * & * \\ 0 & * & 0 & 0 & 0 & * \\ 0 & * & 0 & 0 & 0 & * \\ * & 0 & 0 & * & 0 & 0 \\ * & 0 & 0 & * & 0 & 0 \end{pmatrix} \rightarrow H = \begin{pmatrix} 0 & 0 & * & * & * & * \\ 0 & * & 0 & 0 & 0 & * \\ * & 0 & 0 & * & 0 & 0 \end{pmatrix}, F = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix}$$

university of groningen

Off-diagonal low-rank property

With a block partition dividing A into diagonal part D and off-diagonal part B_{off},

$$A = \begin{pmatrix} \boxed{D_1} & off_{12} & off_{13} \\ off_{21} & \boxed{D_2} & off_{23} \\ off_{31} & off_{23} & \boxed{D_2} \end{pmatrix} \rightarrow D = \begin{pmatrix} D_1 & & \\ & D_2 & \\ & & D_3 \end{pmatrix}, B_{off} = \begin{pmatrix} & off_{12} & off_{13} \\ off_{21} & & off_{23} \\ off_{31} & off_{32} & \end{pmatrix}$$

where B_{off} has some same row patterns, we can easily implement low-rank factorization of B_{off}.

$$B_{off} = \begin{pmatrix} 0 & 0 & * & * & * & * \\ 0 & 0 & * & * & * & * \\ 0 & * & 0 & 0 & 0 & * \\ 0 & * & 0 & 0 & 0 & * \\ * & 0 & 0 & * & 0 & 0 \\ * & 0 & 0 & * & 0 & 0 \end{pmatrix} \rightarrow H = \begin{pmatrix} 0 & 0 & * & * & * & * \\ 0 & * & 0 & 0 & 0 & * \\ * & 0 & 0 & * & 0 & 0 \end{pmatrix}, F = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix}$$

$$B_{off} = F * H,$$

$$\rightarrow A = D + F * H$$

university of groningen

Off-diagonal low-rank property

With a block partition dividing A into diagonal part D and off-diagonal part B_{off},

$$A = \begin{pmatrix} \boxed{D_1} & off_{12} & off_{13} \\ off_{21} & \boxed{D_2} & off_{23} \\ off_{31} & off_{23} & \boxed{D_2} \end{pmatrix} \rightarrow D = \begin{pmatrix} D_1 & & \\ & D_2 & \\ & & D_3 \end{pmatrix}, B_{off} = \begin{pmatrix} & off_{12} & off_{13} \\ off_{21} & & off_{23} \\ off_{31} & off_{32} & \end{pmatrix}$$

where B_{off} has some same row patterns, we can easily implement low-rank factorization of B_{off}.

$$B_{off} = \begin{pmatrix} 0 & 0 & * & * & * & * \\ 0 & 0 & * & * & * & * \\ 0 & * & 0 & 0 & 0 & * \\ 0 & * & 0 & 0 & 0 & * \\ * & 0 & 0 & * & 0 & 0 \\ * & 0 & 0 & * & 0 & 0 \end{pmatrix} \rightarrow H = \begin{pmatrix} 0 & 0 & * & * & * & * \\ 0 & * & 0 & 0 & 0 & * \\ * & 0 & 0 & * & 0 & 0 \end{pmatrix}, F = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix}$$

$$B_{off} = F * H,$$

$$\rightarrow A = D + F * H$$

Then we can construct a preconditioner as

$$A^{-1} = D^{-1}(I - F(I + HD^{-1}F)^{-1}HD^{-1}). \quad (5)$$

based on Woodbury formulation.

Off-diagonal low-rank property

- same rows in B_{off} are compressed into one row of H
- F is an index matrix which can be represented by an integer vector

We define the total compression factor (tcf) as

$$tcf = nnz(B_{off}) - nnz(H) \tag{6}$$

If B_{off} has many same row patterns and the same row patterns are relatively dense, we can get a large tcf.

$\longrightarrow$ Expect memory and computing time saving.

university of groningen

Exploit Internet structure

Does there exist a partition guarantee such a B_{off} which brings large *tcf*?

The web internet has a block structure according to the internet address.

$$\underbrace{\text{www.rug.nl}}_{1_{\text{st}}\ \text{level}}\underbrace{/\text{education}}_{2_{\text{nd}}\ \text{level}}\underbrace{/\text{find}-\text{out}-\text{more}}_{3_{\text{rd}}\ \text{level}} \qquad /\text{parents}/$$

several levels
each level has domains like
"parents" and "find-out-more".

- Pages has more connections to the other pages in the same domain at each level
- Pages in a domain may get some similar behaviour on connecting with other domains

These characters guarantee a possibility to generate such a partition with large *tcf*.

university of groningen

How to generate it?

Try to get as large *tcf* as possible

We only look at the sparsity pattern matrix S of A. Denote set R as $\{1, 2 \ldots n\}$.

For a group N, the off-diagonal part between this group is defined as $S(N, R \backslash N)$.

$$
\begin{array}{c} \\ i \\ j \\ k \\ l \end{array}
\begin{array}{c}
\begin{array}{cccccccccccc} 1 & 2 & \cdots & i & \cdots & j & \cdots & k & \cdots & l & \cdots & n \end{array} \\
\begin{pmatrix}
* & * & \cdots & * & \cdots & * & \cdots & * & \cdots & * & \cdots & * \\
* & * & \cdots & * & \cdots & * & \cdots & * & \cdots & * & \cdots & * \\
* & * & \cdots & * & \cdots & * & \cdots & * & \cdots & * & \cdots & * \\
* & * & \cdots & * & \cdots & * & \cdots & * & \cdots & * & \cdots & *
\end{pmatrix}
\end{array}
$$

blue represents the off-diagonal part, black represents the diagonal part

We call the row patterns of this off-diagonal part as off-diagonal patterns of N
Basic idea:

gather rows if their off-diagonal patterns are the same.

university of groningen

Try to get as large *tcf* as possible

First we permute rows of S in order of decreasing density.
Initially set the group of row i as: $i.group = \{i\}$.

Algorithm 1

1) **for** $i = 1 : n$, if row i is unmarked
2) **for** $j = (i + 1) : n$, if row j is unmarked
3) **if** the off-diagonal patterns between row j and the rows in $i.group$ are same.
4) $i.group \leftarrow \{i.group \cup j\}$; mark row j; back to 1) still at current i.

output: block ordering

Example.1

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \end{array} \begin{array}{c} \begin{array}{ccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \end{array} \\ \begin{pmatrix} 0 & * & * & * & * & * & * & * & 0 \\ * & 0 & 0 & * & * & * & * & * & 0 \\ 0 & * & 0 & * & * & * & * & * & 0 \\ * & 0 & 0 & * & * & * & * & 0 & * \\ * & 0 & 0 & * & * & * & * & 0 & * \\ * & 0 & 0 & * & * & * & * & 0 & * \\ * & 0 & 0 & * & * & * & * & 0 & * \\ 0 & 0 & * & 0 & 0 & 0 & 0 & * & * \\ 0 & 0 & * & 0 & 0 & 0 & 0 & * & 0 \end{pmatrix} \end{array}$$

university of groningen

Try to get as large *tcf* as possible

This process will result in a blocked matrix.

- All the off-diagonal blocks have rank 1 or 0.
- The *tcf* is 11.

We found that it doesn't achieve the largest *tcf*

For example, consider the group corresponding to rows {4-7}

Example.1

	1	2	3	4	5	6	7	8	9
1	0	*	*	*	*	*	*	*	0
2	*	0	0	*	*	*	*	*	0
3	0	*	0	*	*	*	*	*	0
4	0	0	0	*	*	*	*	0	0
5	0	0	0	*	*	*	*	0	0
6	0	0	0	*	*	*	*	0	0
7	0	0	0	*	*	*	*	0	0
8	0	0	*	0	0	0	0	*	*
9	0	0	*	0	0	0	0	*	0

university of groningen

Try to get as large *tcf* as possible

$$
\begin{array}{c}
4 \\ 5 \\ 6 \\ 7
\end{array}
\begin{pmatrix}
0 & 0 & 0 & \boxed{*} & * & * & * & 0 & 0 \\
0 & 0 & 0 & * & \boxed{*} & * & * & 0 & 0 \\
0 & 0 & 0 & * & * & \boxed{*} & * & 0 & 0 \\
0 & 0 & 0 & * & * & * & \boxed{*} & 0 & 0
\end{pmatrix}
$$

These rows have same patterns, leading them all in a group finally.
During this process, the compression factor reaches a local maximum after gathering row 5 to row 4 as a group, then it will be decreased through the following process of adding row 6, 7.

We add a criterion: each gathering must increase the *tcf*. Then row 6 and 7 will form another group instead of being added to {4,5}

$$
\begin{array}{c}
 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9
\end{array}
\begin{array}{c}
\begin{array}{ccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \end{array} \\
\left(\begin{array}{ccc|cc|cc|cc}
0 & * & * & * & * & * & * & * & 0 \\
* & 0 & 0 & * & * & * & * & * & 0 \\
0 & * & 0 & * & * & * & * & * & 0 \\
\hline
0 & 0 & 0 & * & * & * & * & 0 & 0 \\
0 & 0 & 0 & * & * & * & * & 0 & 0 \\
\hline
0 & 0 & 0 & * & * & * & * & 0 & 0 \\
0 & 0 & 0 & * & * & * & * & 0 & 0 \\
\hline
0 & 0 & * & 0 & 0 & 0 & 0 & * & * \\
0 & 0 & * & 0 & 0 & 0 & 0 & * & 0
\end{array}\right)
\end{array}
$$

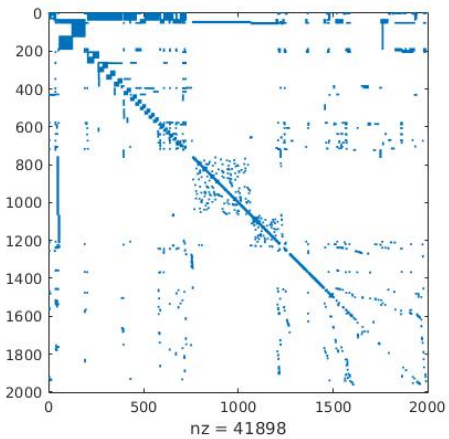
with $tcf = 15$ which is larger than the previous $tcf = 11$.

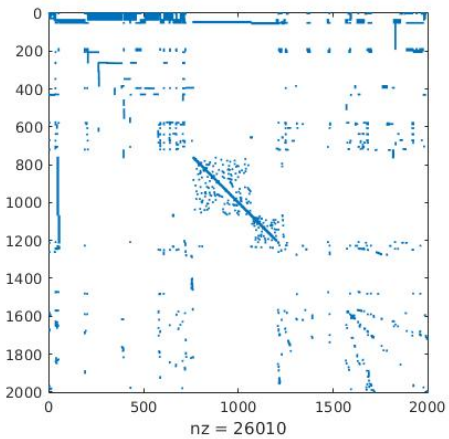
university of groningen

Try to get as large *tcf* as possible

We take a small web crawl of .in domain with size=2000, nnz=59,528.
Show the total compression factor(*tcf*) we can get,

and the difference between with and with out the criterion.





the off-diagonal part(B_{off}) with and without the criterion.

We can see more compressible blocks in B_{off} with this criterion.

methed	p	*tcf*	$\frac{tcf}{nnz}$	inne_size
`Ag1 with criterion`	1008	28801	48.38%	970
`Ag1 without criterion`	857	13812	23.20%	817

university of groningen

p is the partition number, *inner_size* is denoted as the row size of *H*.

Gathering blocks to decrease the size of the inner-system

Our preconditioner: $M \approx A^{-1} = D^{-1}(I - F(I + HD^{-1}F)^{-1}HD^{-1})$.

Application of M requires: inverse D and the inner-system $Inn = (I + HD^{-1}F)$

The previously-discussed partition method doesn't guarantee a satisfied performance of M, which will be shown latter.

We thought the unsatisfactory performance is caused by the large Inn, because Inn is usually much more dense than A.

large $Inn \longrightarrow$ large memory and time for computing $Inn^{-1}v$ when using ilu factorization

university of groningen

Gathering blocks to decrease the size of the inner-system

Our preconditioner: $M \approx A^{-1} = D^{-1}(I - F(I + HD^{-1}F)^{-1}HD^{-1})$.

Application of M requires: inverse D and the inner-system $Inn = (I + HD^{-1}F)$

The previously-discussed partition method doesn't guarantee a satisfied performance of M, which will be shown latter.

We thought the unsatisfactory performance is caused by the large Inn, because Inn is usually much more dense than A.

large $Inn \longrightarrow$ large memory and time for computing $Inn^{-1}v$ when using ilu factorization

Control the size of Inn below a user-defined upper bound.

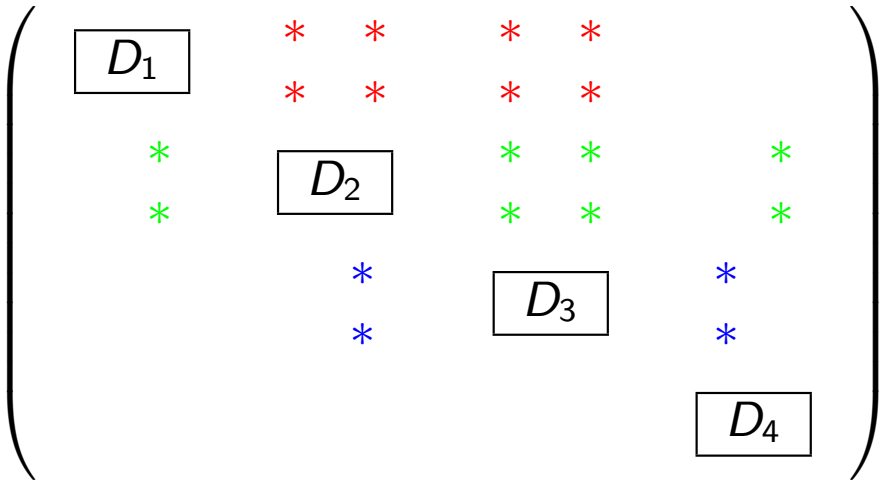
university of groningen

Gathering blocks to decrease the size of the inner-system

As $Inn = (I + HD^{-1}F)$, the size of the Inn is equal to the row amount in H which is equal to the amount of different row patterns in B_{off}.

After the previously-discussed process, each group of rows must have the same off-diagonal patterns.

Example.2



contribute 1 to the size

contribute 1 to the size

contribute 1 to the size

contribute 0 to the size

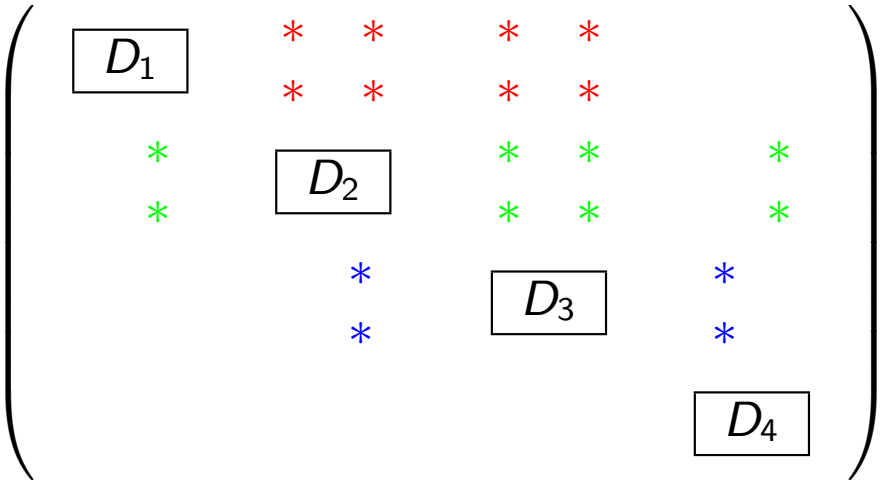
Each group's off-diagonal part contributes 0 or 1 to the size of Inn. We call the groups which have contribution to the size of Inn as contribution groups.

university of groningen

Gathering blocks to decrease the size of the inner-system

We consider to decrease the size of inner-system by making some group's contribution from 1 to 0.

If the contribution of a group is 1, there must be some nonzero off-diagonal blocks at this block row and define the corresponding groups as the row neighbours.



row neighbours : $2, 3$

row neighbours : $1, 3, 4$

row neighbours : $2, 4$

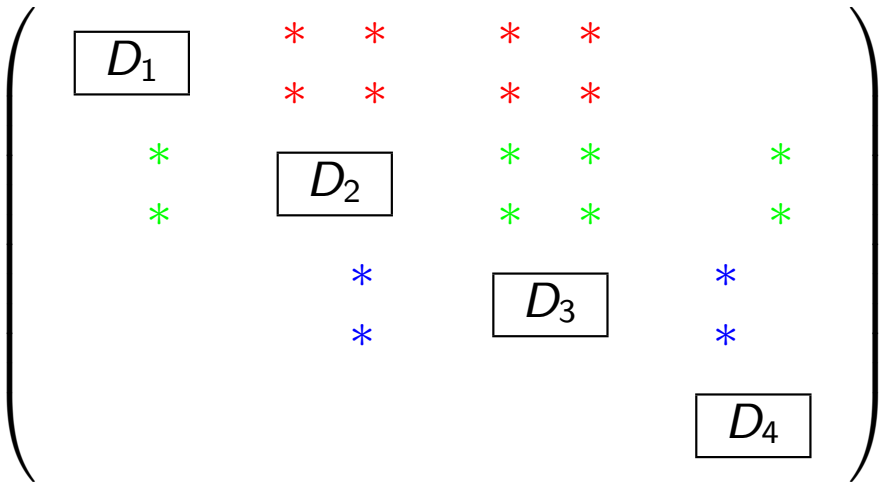
none row neighbours

university of groningen

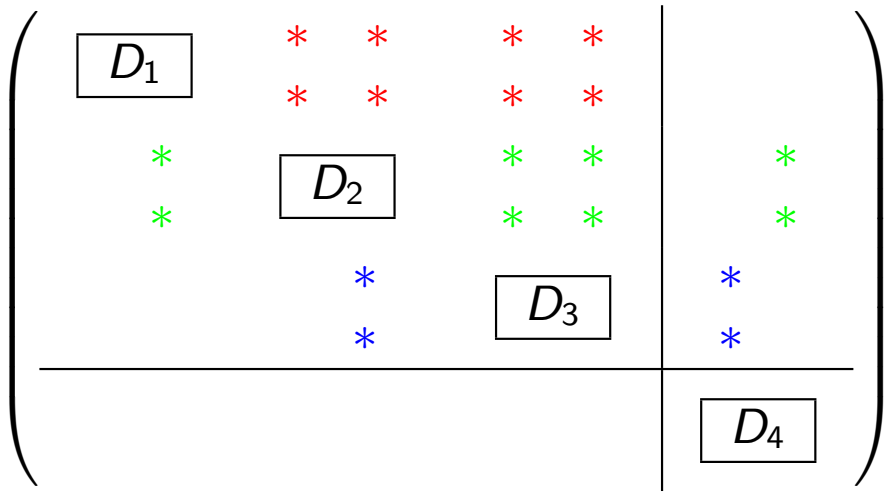
Gathering blocks to decrease the size of the inner-system

If we choose a group and gather it with its row neighbours, the size of the *Inn* will decrease at least by 1.
We call this gathering as Neighbourhood gathering.

The size of *Inn* is 3.
(red, green and blue)



Gathering group 1 with its row neighbours: group 2 and 3.



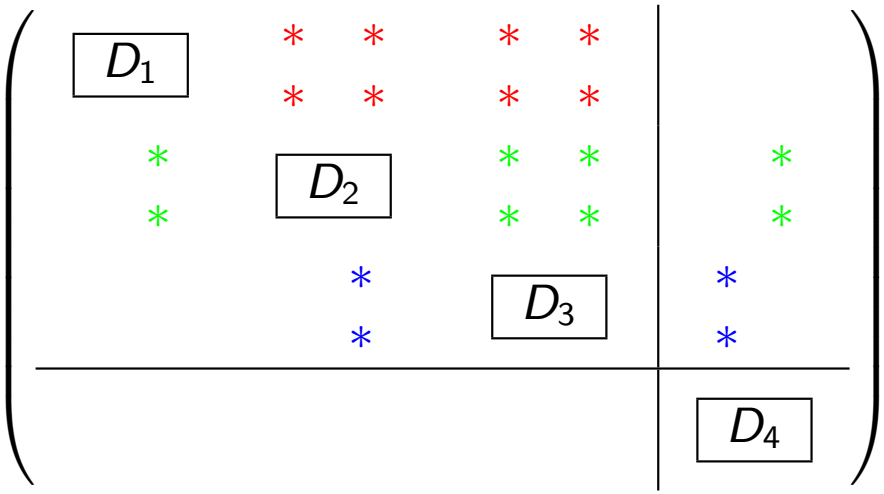
The size of the *Inn* becomes 2.
only green and blue.

university of groningen

Gathering blocks to decrease the size of the inner-system

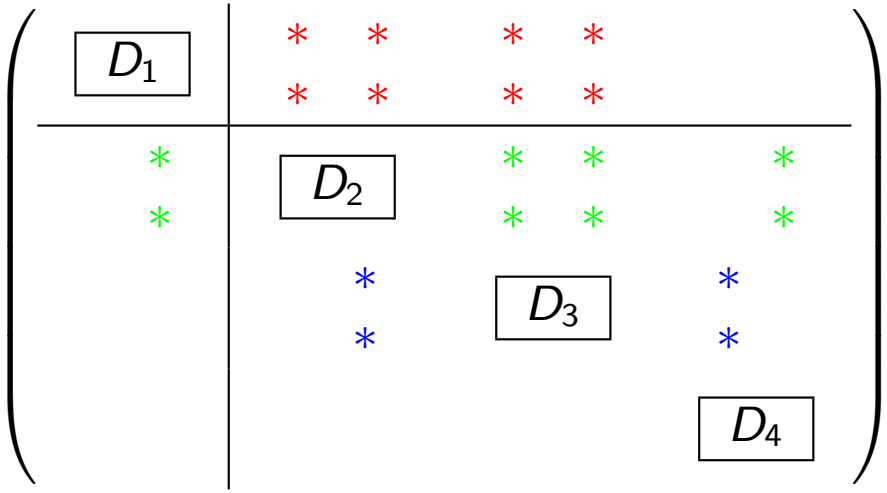
Each step of neighbourhood gathering decreases the size of *Inn*.
However, it also put more elements in the diagonal blocks which can't be compressed immediately. $\longrightarrow$ *tcf* $\downarrow$
We define a index named diagonal fill-in to record this of each group.

neighbourhood gathering on group 1.



its diagonal fill-in is 16.

neighbourhood gathering on group 3.



its diagonal fill-in is 10.

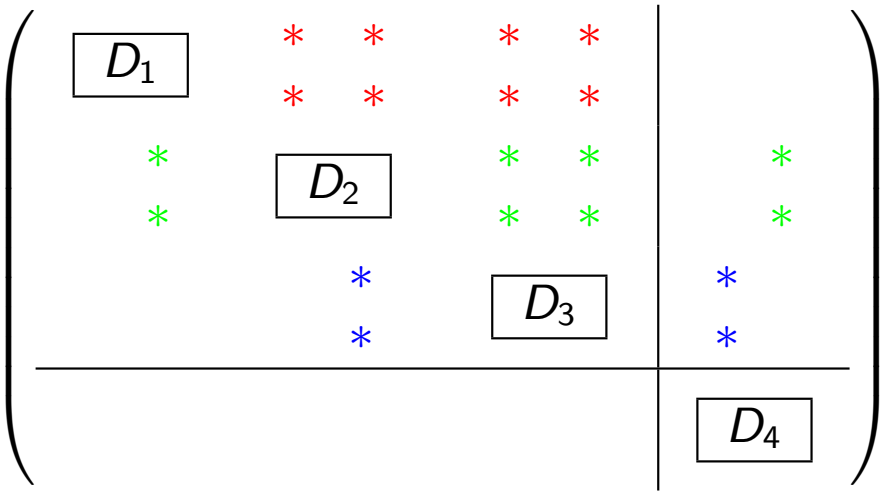
Both decrease the size of *Inn* by 1.

university of groningen

Gathering blocks to decrease the size of the inner-system

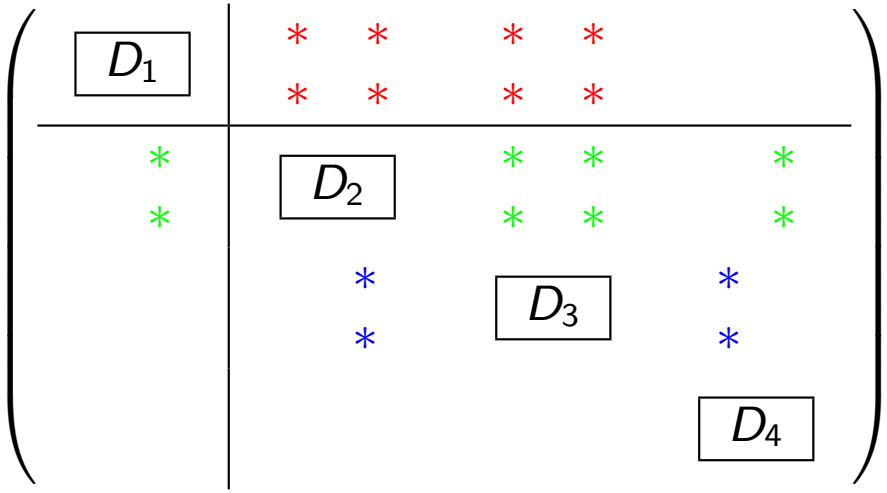
Each step of neighbourhood gathering decreases the size of *Inn*.
However, it also put more elements in the diagonal blocks which can't be compressed immediately. $\longrightarrow$ *tcf* $\downarrow$
We define a index named diagonal fill-in to record this of each group.

neighbourhood gathering on group 1.



its diagonal fill-in is 16.

neighbourhood gathering on group 3.



its diagonal fill-in is 10.

Both decrease the size of *Inn* by 1.
We prefer the latter case which causes fewer diagonal fill-in.

university of groningen

Gathering groups to decrease the size of the inner-system

Basic idea: Each step we choose a contribution group with minimal diagonal fill-in, implement neighbourhood gathering to this group.

Given an upper bound τ for the size(Inn).
Algorithm 2:

1) **while** size($inner$)$\geq \tau$
2) implement neighbourhood gathering to a contribution group which causes minimal diagonal fill-in.
3) update group information

The whole partition method combines Ag 1 and Ag 2.

university of groningen

Gathering groups to decrease the size of the inner-system

Compare the whole partition method with the algorithm 1 alone.
Simply set $\tau = size(A)/20$, compare the performance of this preconditioner on $GMRES(20)$[1].

web crawl of .in with size 7000.

methed	size(Inn)	$\frac{nnz(D)}{nnz(A)}$	it	time	$\frac{nnz(M)}{nnz(A)}$
whole partition	346	48.84%	8	0.095s	1.17
Algorithm 1	3306	28.08%	15	0.135s	1.30

web crawl of .eu with size 7000.

methed	size(Inn)	$\frac{nnz(D)}{nnz(A)}$	it	time	$\frac{nnz(M)}{nnz(A)}$
whole partition	347	34.25%	9	0.096s	1.26
Algorithm 1	4026	13.04%	16	0.146s	1.86

- residual threshold is $1e-8$
- we use the *ilupack* as solver for $D^{-1}v$ and $Inn^{-1}v$
- all the parameters for *ilupack* keep the same

nnz of D increase little compared with the size decreased of Inn

memory saved; increased convergence rate; less time

university of groningen

[1] we choose $\alpha = 0.999 \Rightarrow$ very accurate and difficult PageRank model

Compared with *ilupack*

As the solver for $D^{-1}v$ and $Inn^{-1}v$ is *ilupack* which is a state-of-the-art solver(preconditioner).
Some experiments of our preconditioner against the standalone *ilupack* preconditioner are presented to test the effect of our strategy[2].

Droptol for our preconditioner keeps *1e-1*. Tune the droptol of the ilupack to limit the memory cost($\frac{nnz(M)}{nnz(A)}$) nearly the same.

web crawl of .eu domain with size 7,000.

methed	it	time	memory
`our method`	9	0.096s	1.26
`ilupack`	38	0.195s	1.27
`Gmres(20)`	241	0.695s	#

web crawl of .in domain with size 7,000.

methed	it	time	memory
`our method`	8	0.095s	1.17
`ilupack`	40	0.221s	1.18
`Gmres(20)`	1031	3.140s	#

web crawl of stanford-berkeley with size 70,000.

methed	it	time	memory
`our method`	13	0.371s	1.19
`ilupack`	90	4.120s	1.19
`Gmres(20)`	>5000	>67.5s	#

university of groningen

[2] we choose $\alpha = 0.999 \Rightarrow$ very accurate and difficult PageRank model

Conclusion and following work

Conclusion

- Both our preconditioner and ilupack preconditioners are efficient for PageRank problems
- Our method shows the potential for improving preconditioners' performance by exploiting the special property of PageRank problems

Following work

- Optimize the implementation of the partition method
- Test larger problems

university of groningen