Vol.2017-HPC-160 No.30
2017/7/28

疎行列における非零要素の値を考慮したリオーダリング手法の検討

櫻井隆雄[†1]　中島研吾[†2]

概要：疎行列向けの反復解法では，収束性を向上させるために不完全 LU 分解前処理が良く用いられる．不完全 LU 分解前処理において，対象とする疎行列に対角から離れた位置に値の大きな非零要素が存在すると，値の大きな fill-in が多く発生することがある．このような場合に，収束性を向上させる前処理行列とするためには，その非零要素数を多くする必要があり，メモリ量と計算量が増大する問題がある．そこで，本研究では，値の大きい非零要素を対角に近づけるリオーダリング手法を提案する．提案方式と RCM による不完全 LU 分解前処理の収束性改善効果を評価した結果，全 60 ケースの内，RCM は 46 ケースで収束したのに対し，提案方式は 50 ケースで収束しており，有効性が確認できた．

1. はじめに

大規模な係数行列に対する連立一次方程式の求解は，分子構造や電磁場の解析といった科学技術計算やデータ同化，機械学習において，主要な演算であり，最も多くの計算時間を必要とする．そのため，この求解を高速に演算する方法は常に必要とされている．大規模な疎行列を係数とする連立一次方程式を解く際に使われる反復解法において，収束を加速させる重要な前処理方式の一つとして不完全 LU 分解前処理(以降，ILU 前処理)[1]が知られている．ILU 前処理では，係数行列の行の順序により，その収束性の改善効果が大きく異なる．一方で，ILU 前処理の収束性の改善効果を高めること，IL 前処理を並列処理することを目的として係数行列の行を適切な順番に並べ替えるリオーダリング方式が検討されてきた[2]。

リオーダリング方式として，疎行列の非零要素の構造をグラフ解析し，ILU 前処理において fill-in 数の低減や並列処理を可能とする方式が一般的である．fill-in 数を低減する方式としては Reverse Cuthill-McKee(RCM)[3]や Minimum Degree(MD)[4]，Nested Dissection(ND)[5]などが知られ，並列処理を可能とする方式としては Red-Black，Multicolor 法[6]などが知られている．

一方，ILU 前処理による収束性の改善効果は分解後の下三角成分と上三角成分の積が元の疎行列とどの程度離れているかと密接に関係することが知られている．

そこで，筆者らはリオーダリングにおいて従来検討されていた非零要素のグラフ構造だけではなく，非零要素の値の大きさも検討する必要があると考えた．本報告では非零要素の値の大きさを考慮したリオーダリング方式の提案と，従来のリオーダリング方式と提案方式の効果を ILU 系統の前処理の収束性の評価に寄り比較した結果を述べる．

†1 東京大学大学院情報理工学系研究科
Graduate School of Information Science and Technology, The University of Tokyo
†2 東京大学情報基盤センター
Information Technology Center, The University of Tokyo

2. 疎行列反復解法を加速させる方式

2.1 不完全 LU 分解前処理

本論文で対象とする連立一次方程式は以下の式であらわされる．

$$Ax = b \quad (2.1)$$

ここで，(2.1)式の連立一次方程式を反復解法で解く場合，解法の収束性を高めるために前処理が良く用いられる．前処理は，(2.1)式を以下のように変換する．

$$(M_1^{-1}AM_2^{-1})(M_2x) = M_1^{-1}b \quad (2.2)$$

この変換により，係数行列 A が$(M_1^{-1}AM_2^{-1})$となる．このとき，$(M_1^{-1}AM_2^{-1})$が収束しやすくなるようにM_1^{-1}，M_2^{-1}を選択する．ここで，$M = M_1M_2$は前処理行列と呼ばれ，一般的に$M^{-1} = M_1^{-1}M_2^{-1}$が$A^{-1}$に近ければ反復解法の収束性は良くなると言われている．

ILU 前処理は，この前処理行列 M を LU 分解中に全ての要素を保持しない不完全な LU 分解によって得られた下三角行列 L，上三角行列 U，対角行列 D の積とする方式である．このように全ての要素を保持しないのは，LU 分解中に，係数行列 A では零要素であった位置が分解により非零要素となる fill-in という現象が起こり，これにより LU 分解の処理時間および前処理行列を係数行列および近似解，右辺項に適用する時間が増大するためである．前処理行列保持する要素を選ぶ戦略は様々なものが存在しており，代表的なものとして，係数行列 A にもともと存在する非零要素と同じ位置の要素のみを保持する ILU(0)前処理や，L および U の 1 行あたりの要素数を絶対値の大きいものから p 個を選び，後は保持しない ILUT(p)前処理，ILUT(p)前処理を行う際にさらに絶対値がτ以下の要素を無視して分解および要素の保持を行う ILUT(τ,p)前処理が存在する．ここで，図 1 に ILUT(τ,p)前処理のアルゴリズムを示す．

このように，ILU 前処理では，演算時間や必要メモリ賞の削減のために，保持する要素数を制限しているため，M は A と異なっている．このとき，保持されずに棄却された fill-in により構成される行列$R = A - M$の大きさは前処理による反復解法の収束性改善効果に密接に関係する[7].

1)**do** $i = 1, \cdots, n$
2) $w_j = a_{ij}$ $(j = 1, \cdots, n)$
3) **do** $k = 1, \cdots, i$-1 **and if** $w_k \neq 0$,
4) $w_k = w_k / a_{kk}$
5) **if** $|w_k| < \tau$ **then** $w_k = 0$
6) **if** $w_k \neq 0$ **then**
7) $w = w - w_k u_{kj}$ $(j = k+1, \cdots, n)$
8) **end if**
9) **end do**
10) **if** $|w_j| < \tau$ **then** $w_j = 0$ $(j = i+1, \cdots, n)$
11) Extract the maximum p elements from $w_{1:i-1}$ as l_i
12) Extract the maximum p elements from $w_{i+1:n}$ as u_i
13) enddo

図 1　ILUT(τ,p)前処理のアルゴリズム

2.2 リオーダリング方式

ILU 前処理において，fill-in が多く発生すると，ILU 分解の処理時間の増大，棄却の増加による収束性改善効果の低下などの問題が発生する．それを防ぐために，疎行列の行の順序を ILU 前処理に適したものに入れ替えるリオーダリングが行われることがある．

ここで，A の i 行において，対角要素から最も遠い非零要素との距離をβ_i，全ての行のβ_iの合計値をプロフィル，最大値をバンド幅と定義したとき，このバンド幅とプロフィルは反復解法の収束性そのものに影響する上，fill-in の発生数にも影響する．

これまでに提案されているリオーダリングでは，疎行列の各行をノード，非零要素a_{ij}の存在をノード i と j のエッジの存在としたグラフと見立て，そのグラフの構造からバンド幅とプロフィルを小さくするような順序を決定している．

リオーダリングの中でも代表的なものの一つとして Reverse Cuthill-Mckee(RCM)がある．RCM の基本的なアルゴリズムは以下となる．

① 全ノードの中で最もエッジ数の少ないノードを 1 つ選び，そのノードをレベル 1 とする．
② レベル k-1 と接続しており，まだレベル付けがされていないノードをレベル k とする．全ての点にレベル付けがされるまで k を 1 つずつ増やして繰り返す．
③ 全ての点がレベル付けされたら，レベルの順番に 1 から再番号付けする．このとき，各レベル内ではエッジ数の少ない順で番号付けする．
④ 番号の逆順，すなわち番号 n のノードが 1 行目，番号 1 のノードが n 行目となるように疎行列の行の順序を並べ替える．

ここで，④のように逆順にふり直すのは，こちらの方が fill-in が少なくなるためである．

RCM の他にも，疎行列を表すグラフから接続エッジ数が最小のノードを順番に取り除き，取り除いた順に並べ替える Minimum Degree(MD)や，グラフを 2 分できる最小数のノードを再帰的に選択する Nested Dissection(ND)などのリオーダリング方式が存在する．

3. 非零要素の値の大きさを考慮したリオーダリング方式

これまで述べたリオーダリング方式は，疎行列 A の非零要素の構造にのみ注目していた．しかし，前処理行列 M が A の特徴をどの程度保持できているかは棄却する fill-in の数だけではなく，値の大きさも強く影響を受けると考えられる．そこで，本論文では，グラフ構造のみではなく，非零要素の値の大きさも考慮したリオーダリング方式を提案する．

一般的に，値の大きな fill-in は ILU 分解前の行列における値の大きな要素から発生しやすく，対角に近い非零要素から発生する fill-in の数は少ない傾向にある．そこで，提案法では値の大きな非零要素を対角要素の近くに集中させるように行の順序を入れ替える．従来のリオーダリング方式と同様に疎行列の各行をノード，非零要素a_{ij}の存在をノード i と j のエッジとしたグラフと見立てた場合，基本方針として，既に番号付けしたノードから見て，最も強く(絶対値が大きく)エッジで接続しているノードを選択していく．しかし，これだけで選択していくと，非常に値の小さいエッジが無視されるため，プロフィルやバンド幅が増大する可能性があるため，あるエッジの片方のノードが番号付けされてから無関係のノードが選択されるたびにエッジの強さを増大させるよう重み付けを行う．

この重み付けの方法はいくつか考えられるが，本論文では n 個前に選ばれたノードからつながっているエッジは n 倍の強さとして考える方法と，2^n倍の強さと考える方法の 2 つとした．これらのアルゴリズムは以下となる．

提案方式のアルゴリズム

① 全ノードの中で最もエッジ数の少ないノードを 1 つ選び，そのノードを 1 番目とする．
② 1~k-1 番目のノードと接続しており，番号付けがされていないノードの中で，接続されているエッジの強さの合計が最も大きなノードを k 番目とする．このとき，各エッジの強さe_{ij}は i 行目が p 番目に選ばれたとすると，以下の式(3.1)もしくは式(3.2)で計算される．

$$e_{ij} = (k-p)\left|a_{ij}/a_{jj}\right| \quad (3.1)$$

$$e_{ij} = 2^{(k-p)}\left|a_{ij}/a_{jj}\right| \quad (3.2)$$

③ 各行に付けられた番号に従い，疎行列の行の順序を並べ替える．

ここで，③において，行の順序を入れ替える際に，番号

順に従い昇順で並べ替えるか，RCM と同様に降順(逆順)で並べ替えるかという点でも 2 種類考えられる．そこで，本論文は今後，エッジの強さを決める際に(3.1)と(3.2)のどちらの式を用いたかと，③で昇順と降順のどちらで並び変えたかの 4 種類を表 1 のように呼称する．

表 1　提案方式の表記

方式名	エッジの強さ	並び変え
Proposal1-s	(3.1)	昇順
Proposal1-r	(3.1)	降順
Proposal2-s	(3.2)	昇順
Proposal2-r	(3.2)	降順

4. 数値実験

4.1 実験環境

提案法により，ILU 前処理の収束性改善効果が向上するか数値実験を行った．評価に用いた計算機環境を表 2 に示した．また，表 3 に評価に用いた反復解法と ILU 前処理，およびそのパラメータを示した．

表 2　計算機環境

OS	CentOS 6.6
CPU	Intel(R) Xeon(R)　E5-2697 v2
Memory	256GByte
Compiler	Intel Fortran Compiler 15.0.1

表 3　反復解法，前処理とパラメータ

反復解法	BiCGStab
前処理法	ILUT(τ,p)
収束条件	10^{-8}
最大反復回数	1000
T	10^{-8}
P	10, 20, 30, 40, 50, 60, 70, 80, 90, 100

続いて，表 4 に今回の評価で用いる疎行列を示した．1~4 は The SuiteSparse Matrix Collection[8]から取得した．5,6 は立方体を z 方向に 1.1 倍の長さになるように力を加えた場合に物体がどのように変形するかを有限要素法で求める場合に解く疎行列で，立方体のヤング率は 0.3，ポアソン比が項番 5 は 0.4999，項番 6 が 0.49999999 としている．要素数は x,y,z の各方向が 20 要素となるようにしている．また，これらの行列では 3 行で接点のそれぞれ x 軸,y 軸,z 軸方向の処理をしているため，リオーダリングを行う場合は，この連続する 3 行を一組であると処理し，この 3 行が離れないようにした．

数値実験は表 4 で示した各行列に対し，RCM および表 1 の 4 種の提案方式を適用し，これとリオーダリングをかけなかった 6 種類の行列に対し，p を 10 から 100 まで 10 刻みで動かした ILUT(τ,p)前処理をかけ，BiCGStab 法で説いたときの反復回数と演算時間を評価した．また，p が 100 の場合の ILUT(τ,p)前処理における fill-in も測定し，比較した．

表 4　評価対象の行列

#	Matrix	N	NZ	Field
1	Baumann	112,211	748,331	chemical master eq.
2	nmos3	18,588	237,130	semiconductor
3	sme3Dc	42,930	3,148,656	structural
4	xenon1	48,600	1,181,120	materials
5	fem1	27,783	2,042,829	finite element method
6	fem2	27,783	2,042,829	finite element method

4.2 実験結果

反復回数と演算時間の評価の結果を行列ごとに表 5〜10 に示した．これらの表では，一番左に ILUT(τ,p)前処理における p の値，続いて左からリオーダリングをかけないオリジナルの行列(Original)，RCM をかけた行列(RCM)，4 つの提案方式をそれぞれかけたものとなっており，それぞれ収束までに要した反復回数(Iter)と秒単位の演算時間(Time(s))が示されている．最大反復回数と設定した 1000 回で収束しなかった場合や破綻した場合は Iter の項に”not conv.”と記載している．

表 5 によると，行列 Baumann では，リオーダリングをかけなかった場合，p=10 では解けず，20 以上で解けるようになる．しかし，20 では反復回数が 30 以上に比べ非常に多い．これが，RCM をかけることにより，p=10 でも解けるようになり，反復回数，演算時間ともに小さい．一方，提案方式をかけた場合，Proposal1-s では p=30 まで解けなくなり，反復回数，演算時間ともにオリジナルより増加している．Proposal1-r では p=10 で解けるようになっているものの，反復回数は RCM よりも多く，演算時間は 30 以上でオリジナルより悪化している．Proposal2-s では，p=10 で解けないものの，50 以上で反復回数，演算時間が RCM より少なくなる．Proposal2-r では反復回数，演算時間ともに RCM と同等から 1 割悪化の範囲となっている．以上から，Baumann では RCM をかけたものが最も解きやすく，続いて Proposal2-r, Proposal2-s となり，オリジナルと Proposal1-r がほぼ同等，Proposal1-s が最も解きづらいと考えられる．

続いて表 6 によると，nmos3 はオリジナル，Proposal1-s, Proposal2-s では全く解けなかった．一方，RCM, Proposal1-r, Proposal2-r では全ての p で解けている．反復回数や演算時間を見ると，Proposal1-r が最も少なく，続いて RCM，Proposal2-r の順となる．

表 5　Baumann の反復回数と演算時間

	Original		RCM		Proposal1-s		Proposal1-r		Proposal2-s		Proposal2-r	
p	Iter	Time(s)	Iter	Time(s)	Iter	Time(s)	Iter	Time(s)	Iter	Time(s)	Iter	Time(s)
10	not conv.	-	27	0.62	not conv.	-	424	6.14	not conv.	-	30	0.68
20	249	6.88	13	0.95	not conv.	-	46	3.86	110	2.93	14	1.01
30	32	4.38	11	1.29	not conv.	-	25	5.61	18	1.55	11	1.32
40	19	5.40	9	1.54	231	20.69	14	7.46	9	1.61	10	1.60
50	20	6.75	8	1.76	428	33.73	10	9.11	5	1.63	9	1.99
60	13	7.62	7	1.95	73	20.07	9	10.72	4	1.73	8	2.02
70	12	8.67	7	2.16	19	19.04	8	12.34	3	1.81	7	2.32
80	12	9.71	6	2.36	16	20.63	8	13.99	3	1.91	7	2.37
90	11	10.61	6	2.46	29	23.33	7	15.50	3	2.00	7	2.54
100	7	11.14	6	2.60	10	23.34	7	17.06	2	2.03	6	2.64

表 6　nmos3 の反復回数と演算時間

	Original		RCM		Proposal1-s		Proposal1-r		Proposal2-s		Proposal2-r	
p	Iter	Time(s)	Iter	Time(s)	Iter	Time(s)	Iter	Time(s)	Iter	Time(s)	Iter	Time(s)
10	not conv.	-	708	1.20	not conv.	-	477	0.85	not conv.	-	711	1.31
20	not conv.	-	774	1.94	not conv.	-	281	0.74	not conv.	-	366	0.97
30	not conv.	-	463	1.75	not conv.	-	244	0.82	not conv.	-	289	1.04
40	not conv.	-	485	1.98	not conv.	-	181	0.85	not conv.	-	244	1.12
50	not conv.	-	408	2.01	not conv.	-	117	0.65	not conv.	-	224	1.47
60	not conv.	-	361	2.08	not conv.	-	103	0.67	not conv.	-	170	1.18
70	not conv.	-	312	2.21	not conv.	-	78	0.63	not conv.	-	171	1.42
80	not conv.	-	262	2.20	not conv.	-	91	0.75	not conv.	-	270	2.49
90	not conv.	-	206	2.18	not conv.	-	99	1.07	not conv.	-	145	1.65
100	not conv.	-	134	1.60	not conv.	-	78	0.80	not conv.	-	160	2.00

表 7　sme3Dc の反復回数と演算時間

	Original		RCM		Proposal1-s		Proposal1-r		Proposal2-s		Proposal2-r	
p	Iter	Time(s)	Iter	Time(s)	Iter	Time(s)	Iter	Time(s)	Iter	Time(s)	Iter	Time(s)
10	not conv.	-	not conv.	-	not conv.	-	not conv.	-	not conv.	-	not conv.	-
20	not conv.	-	not conv.	-	not conv.	-	not conv.	-	not conv.	-	not conv.	-
30	not conv.	-	not conv.	-	not conv.	-	533	10.44	not conv.	-	not conv.	-
40	not conv.	-	469	16.35	not conv.	-	358	10.42	not conv.	-	610	55.44
50	not conv.	-	356	17.26	not conv.	-	276	10.76	not conv.	-	544	67.33
60	not conv.	-	493	23.62	823	33.72	254	12.33	not conv.	-	473	78.36
70	not conv.	-	316	21.95	308	23.21	221	13.72	not conv.	-	387	88.05
80	not conv.	-	263	23.16	461	31.47	160	13.35	not conv.	-	753	113.54
90	876	320.47	227	24.32	218	25.65	147	14.57	577	369.89	928	134.15
100	604	308.09	231	26.83	188	26.98	121	15.03	404	354.87	386	123.63

表 8　xenon1 の反復回数と演算時間

	Original		RCM		Proposal1-s		Proposal1-r		Proposal2-s		Proposal2-r	
p	Iter	Time(s)	Iter	Time(s)	Iter	Time(s)	Iter	Time(s)	Iter	Time(s)	Iter	Time(s)
10	not conv.	-	not conv.	-	not conv.	-	not conv.	-	not conv.	-	not conv.	-
20	not conv.	-	46	2.57	not conv.	-	48	3.52	630	11.23	49	2.98
30	36	8.19	27	4.39	41	7.83	29	6.39	38	9.83	27	5.27
40	23	10.53	20	6.30	26	11.86	25	9.46	23	13.13	19	7.52
50	19	12.91	16	8.13	21	16.06	18	12.33	20	16.33	16	9.62
60	16	15.20	13	10.02	18	20.14	16	15.14	17	19.27	15	11.66
70	15	17.51	14	12.03	18	24.18	14	17.79	14	21.88	14	13.35
80	17	19.93	12	13.82	16	27.95	12	20.31	14	24.45	13	15.05
90	14	22.26	11	15.61	14	31.50	12	22.83	13	26.90	12	16.66
100	14	24.66	10	17.36	13	34.90	12	25.29	12	29.14	11	18.20

表 9　fem1 の反復回数と演算時間

	Original		RCM		Proposal1-s		Proposal1-r		Proposal2-s		Proposal2-r	
p	Iter	Time(s)	Iter	Time(s)	Iter	Time(s)	Iter	Time(s)	Iter	Time(s)	Iter	Time(s)
10	not conv.	-	not conv.	-	not conv.	-	not conv.	-	not conv.	-	not conv.	-
20	not conv.	-	not conv.	-	not conv.	-	not conv.	-	not conv.	-	not conv.	-
30	not conv.	-	617	8.99	not conv.	-	828	15.32	not conv.	-	694	10.40
40	747	13.31	378	10.33	not conv.	-	902	21.22	578	12.84	578	13.89
50	719	16.09	497	15.39	728	30.32	682	22.99	499	16.47	515	17.11
60	487	16.23	297	15.64	724	35.02	650	26.51	448	20.44	523	21.25
70	406	17.49	338	19.54	872	42.84	481	27.84	549	26.13	330	22.20
80	531	22.49	394	23.58	not conv.	-	696	35.12	458	28.80	242	23.12
90	804	30.98	398	26.57	675	47.55	454	33.80	318	29.62	249	26.12
100	596	28.80	314	27.52	961	58.55	460	36.94	302	32.55	253	28.81

表 10　fem2 の反復回数と演算時間

	Original		RCM		Proposal1-s		Proposal1-r		Proposal2-s		Proposal2-r	
p	Iter	Time(s)	Iter	Time(s)	Iter	Time(s)	Iter	Time(s)	Iter	Time(s)	Iter	Time(s)
10	not conv.	-	not conv.	-	not conv.	-	not conv.	-	not conv.	-	not conv.	-
20	not conv.	-	not conv.	-	not conv.	-	not conv.	-	not conv.	-	not conv.	-
30	not conv.	-	not conv.	-	not conv.	-	not conv.	-	not conv.	-	not conv.	-
40	not conv.	-	not conv.	-	not conv.	-	not conv.	-	not conv.	-	not conv.	-
50	not conv.	-	not conv.	-	not conv.	-	not conv.	-	not conv.	-	955	22.67
60	not conv.	-	not conv.	-	not conv.	-	599	25.77	not conv.	-	not conv.	-
70	805	24.37	not conv.	-	not conv.	-	568	29.24	not conv.	-	not conv.	-
80	not conv.	-	not conv.	-	not conv.	-	447	30.48	not conv.	-	not conv.	-
90	not conv.	-	602	31.04	not conv.	-	334	31.19	not conv.	-	not conv.	-
100	not conv.	-	585	33.88	not conv.	-	286	32.87	not conv.	-	505	34.53

表 7 の sme3Dc では，オリジナルでは p=80 まで解けず，90 以上でも 300 秒以上の時間がかかる．これに対し，RCM は 40 から解けるようになり，反復回数，演算時間ともに大幅に減少する．Proposal1-r は 30 から解け，反復回数，演算時間も RCM よりさらに小さい．Proposal1-s と Proposal2-r はオリジナルよりは収束性が良いが，Proposal1-r 空は大きく劣っていた．Proposal2-s では反復回数はオリジナルよりやや減るが，演算時間は増加していた．

表 8 の xenon1 では，RCM，Proposal1-r，Proposal2-r はオリジナルが p=30 以上で解けるのに対し，p=20 以上で解けるようになり，この 3 者はほぼ同等の反復回数となっていた．しかし，演算時間で比較すると，RCM，Proposal2-r，Proposal1-r の順となる．Proposal1-s と Proposal2-s はオリジナルより反復回数は同等だが，演算時間は増加していた．

表 9 の fem1 では，オリジナルが p=40 から解けるのに対し，表 8 と同じように RCM, Proposal1-r, Proposal2-r が p=30 以上で解けるようになっていた．また，演算時間が RCM, Proposal2-r, Proposal1-r の順となっていたのも同様である．一方，Proposal2-s はオリジナルと比べ，演算時間が若干増加するものの，反復回数が減少しているのに対し，Proposal1-s は解けないパターン，反復回数，演算時間の全てが増加していた．

最後に，表 10 の fem1 では，オリジナルでは p=70 のときのみ解けたが，それ以上の場合も含め他では解けなかった．RCM では p=90 と 100 で解けていた．これに対し，Proposal1-r では p=60 以上で解けるようになっており，Proposal2-r では p=50 と 100 で解けていた．最も少ない p で解けていたのは Proposal2-r であるが，Proposal1-r は 60 以上の全てで解けており，最も ILUT(τ,p)前処理に適した行列に並べ替えているのは Proposal1-r と考えられる．Proposal1-s と Proposal2-s では解けているケースは無かった．

ここまでをまとめると，演算時間で比較した場合，RCM が Baumann，xenon1，fem1，Proposal1-r が nmos3，sme3Dc，fem2 でそれぞれ最も良い結果を示し，この 2 方式が同等であった．一方，表 5〜10 の全 60 ケースで各方式により解けたケース数を比較した結果を図 2 に示した．

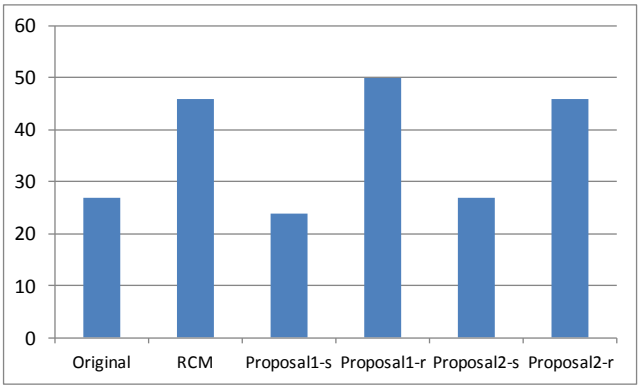

図 2 各リオーダリング方式により解けたケース数

図 2 によると，Original では 27 ケース解けていたのに対し，RCM と Proposal2-r は 46 ケース，Proposal1-r は 50 ケース解けていた．このことから，今回の評価対象の行列に対して，Proposal1-r は RCM 以上に ILUT(τ,p)前処理に適した行列に並べ替えていたと言える．なお，Proposal1-s は 24 ケース，Proposal2-s は 27 ケースであり，有効なリオーダリング方式であるとは言えない．

最後に，各リオーダリング方式により並べ替えられた行列を p=100 とし ILUT(τ,p)前処理をかけた際に発生した fill-in 数を測定し，図 3 に示した．図 3 によると，fill-in 数は RCM では fem1,2 をのぞいて Original より少ないが，Proposal1-r は Baumann，xenon1，fem1,2 の 4 つで増加している．このように fill-in 数が多いのが xenon1 や fem1 で RCM と反復回数がほぼ同数にも関わらず，演算時間が多かった原因と考えられる．また，提案方式において降順で並べ替えたものと昇順で並べ替えたものを比較すると，Baumann においける Proposal2 をのぞいて全て降順(r)の方が fill-in 数が少なくなっており，前述の収束したケース数も考慮すると降順が昇順より優位であると判断できる．

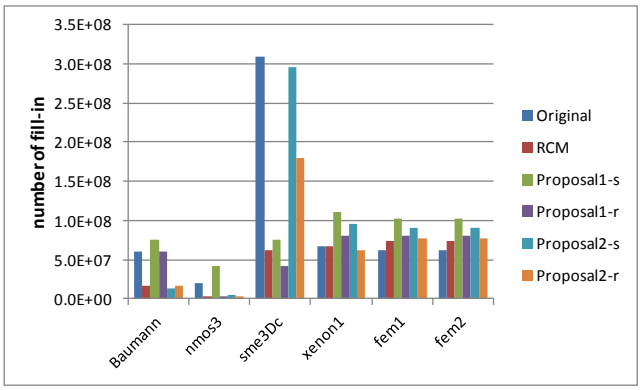

図 3 各リオーダリング方式による fill-in 数の比較

ここまでの評価から，提案した非零要素の値を考慮したリオーダリング方式には，距離の重みを Proposal1 のように n 個前に選ばれたノードからつながっているエッジは n 倍の強さとして考え，降順で並べ替えることにより，ILUT(τ,p)前処理による収束性改善効果を RCM 以上に向上させることがあるとわかった．

5. おわりに

本論文では ILU 系統の前処理の収束性改善効果を向上させるために，従来の疎行列のグラフ構造だけではなく，非零要素の値の大きさまで考慮するリオーダリング方式を提案した．提案方式は非零要素の値の大きさをエッジの強さと見立て，既に番号付けられているノードと最も強くエッジで接続しているノードを選択する．この際，弱いエッジを完全に無視しないように，あるノードが選択されてから

別のノードが選択されるたびにエッジを強くするよう重み付けをするようにした.

6 種の行列を対し，ILUT(τ,p)前処理による収束性改善効果を評価する数値実験を行った結果，エッジの重み付けを n 個前に選ばれたノードからつながっているエッジは n 倍の強さとして考え，番号付けしたノードを降順で並べ替える方式が, 全 60 ケースの内, 50 ケースを解くことができ, RCM が 46 ケースであったことから同等以上の効果があると確認できた.

今後の課題としては以下が考えられる.

一つ目は，提案方式は多くの行列でオリジナルよりも ILUT(τ,p)前処理を行う際に発生する fill-in 数を増加させており，この増加を抑制することである.

また，本論文で提案したリオーダリング方式は，並列化に対応していないため，これを可能とする改良案の考案を行う.

参考文献

[1] Meijerink, J. A., and H. A. van der Vorst. “An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix.” Mathematics of computation 31.137 (1977): 148-162.
[2] Y. Saad. “Iterative methods for sparse linear systems.” Society for Industrial and Applied Mathematics, 2003.
[3] W. Chan, and A. George. “A linear time implementation of the reverse Cuthill-McKee algorithm.” BIT Numerical Mathematics 20.1 (1980): 8-14.
[4] P. R. Amestoy, T. A. Davis, and I. S. Duff. “An approximate minimum degree ordering algorithm.” SIAM Journal on Matrix Analysis and Applications 17.4 (1996): 886-905.
[5] A. George. “Nested dissection of a regular finite element mesh.” SIAM Journal on Numerical Analysis 10.2 (1973): 345-363.
[6] T. Iwashita, and M. Shimasaki. “Algebraic multicolor ordering for parallelized ICCG solver in finite-element analyses.” IEEE transactions on magnetics 38.2 (2002): 429-432.
[7] 岩下武史, 島崎眞昭. “不完全コレスキー分解前処理に関するオーダリングの新評価法.” 情報処理学会研究報告ハイパフォーマンスコンピューティング (HPC) 2005.19 (2004-HPC-101) (2005): 109-114.
[8] “The SuiteSparse Matrix Collection”. https://www.cise.ufl.edu/research/sparse/matrices/, (参照 2017-06-25).