Vol.2013-HPC-141 No.11
2013/10/1

大規模ネットワーク解析のための スペクトラルクラスタリング

小形 英史[1,a)]　鈴村 豊太郎[1,2,b)]

概要：大規模グラフ解析は，近年最も活発に研究されている分野の１つである．中でも，クラスタリングはネットワークのコミュニティ抽出や，タンパク質相互作用の解析など，多様な分野で利用される重要なアルゴリズムである．スペクトラルクラスタリングはクラスタリングの一種で，比較的精度が高いアルゴリズムであることが知られているが，固有ベクトル計算のコストが高いため，大規模グラフの解析には適していない．この問題を解決するため，我々は，大規模行列のための固有値計算ライブラリ P_ARPACK を用いた，並列分散環境で実行できるスケーラブルなスペクトラルクラスタリングを提案し，それを並列プログラミング言語 X10 で実装した．

1. はじめに

最近，大規模ソーシャルネットワークの解析が盛んになっている．クラスタリングやコミュニティ抽出はソーシャルネットワークの解析に有用である [1] [2]．中でも，スペクトラルクラスタリングは比較的精度が高いアルゴリズムであるが，計算量が大きいために大規模ネットワークには適用し辛いことが問題となっている．一方で，IBM による並列プログラミング言語 X10[3] が開発され，並列分散環境における高性能な並列プログラムの生産性が高まっている．そこで本研究では，スーパーコンピュータでの超大規模並列計算を視野に入れた，スケーラブルなスペクトラルクラスタリングを X10 で実装した．

2. スペクトラルクラスタリング

頂点集合 V とエッジ集合 $E = \{(u,v)|u,v \in V\}$ およびエッジの重み $W : E \to \mathbb{R}^+$ からなるネットワーク $G = (V, E, W)$ に対する k 分割スペクトラルクラスタリングは，Normalized Cut と呼ばれる以下の式 (1) を最小化するようなクラスタ $C_1, ..., C_k$ を求める操作である．

$$NCut(C_1, ..., C_k) = \sum_{i=1}^{k} \frac{Cut(C_i, V \setminus C_i)}{Assoc(C_i)} \qquad (1)$$

ここで，$C_1 \cup C_2 \cup ... \cup C_k = V$ であり，$Cut(C_1, V \setminus C_i)$ は C_i に属する頂点と $V \setminus C_i$ に属する頂点に張られたエッジの重みの総和を意味し，$Assoc(C_i)$ は C_i に属する頂点同士に張られたエッジの重みの総和を意味する．従って，Normalized Cut を小さくするということはすなわち，各クラスタの内部を密にし，クラスタ間の繋がりをなるべく疎にするということである．上記の目的関数の最小化問題は NP 困難であることが知られているが，この式を適切に近似することによって，クラスタを求めることが可能である．スペクトラルクラスタリングは，目的関数の近似の仕方によって様々なバリエーションがあるが，いずれもある行列の固有方程式を解く形に帰着される．スペクトラルクラスタリングのバリエーションの例を表 1 にまとめた [4] [5] [6]．表中の A, D はそれぞれネットワークの隣接行列と次数行列である．ただし，隣接行列 A の (i,j) 要素は v_i, v_j 間のエッジの重み，すなわち $W(v_i, v_j)$ である．

我々が実装したアルゴリズムは，表 1 の Melia, Shi のアルゴリズム [5] である．このアルゴリズムを選択した理由は，アルゴリズムのスケーラビリティを保つためであるが，詳細は 3.2 節で述べる．実装したスペクトラルクラスタリングのアルゴリズムを Algorithm 1 に示す．

Algorithm 1 スペクトラルクラスタリング

1: $L \leftarrow D^{-1}A$
2: $Lu = \lambda u$ を解き，k 個の固有ベクトル $u_1, ..., u_k$ を求める．
3: $U \leftarrow (u_1, ..., u_k)$
4: U の各行をユークリッド空間の座標と見なし，K-means を適用する．

[1] 東京工業大学 情報理工学研究科 計算工学専攻
Tokyo Institute of Technology
[2] IBM 東京基礎研究所
IBM Research - Tokyo
[a)] ogata.h.ac@m.titech.ac.jp
[b)] suzumura@cs.titech.ac.jp

考案者	Affinity Matrix L	固有方程式	求める固有ベクトル
Ng, Jordan, Weiss	$D^{-0.5}AD^{-0.5}$	$Lv = \lambda v$	最大から k 個
Melia, Shi	$D^{-1}A$	$Lv = \lambda v$	最大から k 個
Shi, Malik	$D - A$	$Lv = \lambda Dv$	最小から k 個

表 1 種々のスペクトラルクラスタリング

3. アルゴリズム

3.1 固有ベクトル計算

スケーラブルなスペクトラルクラスタリングを実装するうえで我々が着目した点は，行列 L の固有ベクトル計算を如何に高速化するかということである．我々の実測では，スペクトラルクラスタリングの実行時間の中で固有ベクトル計算の占める割合は約 90%であり，実行時間のほとんどが固有ベクトル計算に費やされていると言って差し支えない．大規模行列向けの並列固有値計算に関しては多くの既存のアルゴリズムが存在し，山本 [7] による対象密行列のための高速な固有値計算や，片桐ら [8] による並列固有値ソルバーなどが提案されているが、我々が対象とするのは超大規模疎行列であるため，疎行列向けの高速な固有値計算アルゴリズムが必要である。そこで，我々は P_ARPACK[9] という大規模疎行列向け固有値計算ライブラリを用いて，スペクトラルクラスタリングにおける固有ベクトル計算を実装した．

3.2 固有値計算ライブラリ P_ARPACK

P_ARPACK は，大規模疎行列に対する少数の固有値計算に特化したライブラリ ARPACK[10] の，拡張パッチである．ARPACK は計算機 1 ノードのシングルプロセスで実行可能なライブラリであるが，これにパッチを適用した P_ARPACK は，複数ノードに並列分散されたライブラリとなっている．ARPACK の固有値計算アルゴリズムは Arnoldi 法に基づいており，Arnoldi Iteration と呼ばれる行列計算の反復実行によって固有値，固有ベクトルの近似解を得る．Arnoldi Iteration は対象の行列とあるベクトルとの積，および対象の行列に依らない幾つかの計算から構成されている．Arnoldi 法の計算で行列に依存する部分が行列ベクトル積だけであることから，ARPACK では Reverse Communication というインターフェースを採用し，行列ベクトル積をユーザーが実装，それ以外の計算を ARPACK ライブラリが提供することによって，固有値計算を実現している．

この Reverse Communication の大きな特徴として，計算対象の行列のデータ形式を ARPACK 側が制限しないという利点がある．LAPACK 等の行列計算ライブラリでは，ライブラリ関数の引数に与える行列は密行列かつ Column Major でなければならないという制限がある．しかし ARPACK ではそのような制限はなく，どのような形式の行列でも，その行列に対する行列ベクトル積さえ実装すれば必ず計算可能である．これは，ユーザーが ARPACK に渡す引数が，行列そのものではなく行列ベクトル積の計算結果であることによる．そのため，ARPACK は疎行列向けのライブラリではあるものの，実際には密行列や帯行列等，あらゆる行列に対する固有値を計算することが出来る．

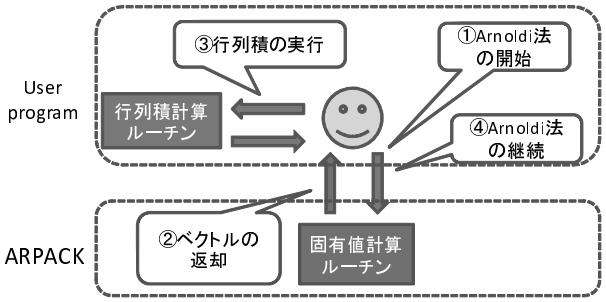

図 1 ARPACK の計算過程

ただし，ARPACK には 1 つ欠点があり，行列の最大固有値の計算は高速であるが，最小固有値の計算は収束が遅く，極端に時間がかかるのである．これは Arnoldi 法の特性からくる問題であるため，Arnoldi 法をベースとしている以上避けられない問題であるが，アプリケーションによっては A の最小固有値の代わりに A^{-1} の最大固有値を求めることで問題を回避できることもある．スペクトラルクラスタリングの場合，前述の通り固有方程式にはいくつかのバリエーションがあるため，最小固有値ではなく最大固有値を必要とするアルゴリズムを選択することで問題を解決できる．これが，我々が Melia, Shi のアルゴリズムを選択した理由である．ARPACK による固有値計算過程を図 1 に示す．

従って，P_ARPACK を使って大規模疎行列の固有値計算を実装するためには，大規模疎行列向けの並列分散された行列ベクトル積を実装すれば良いことになる．疎行列ベクトル積に関しても多くのアルゴリズムが提案されているが，我々は 2011 年に Yoo ら [11] が発表した MatVec アルゴリズムを実装した．このアルゴリズムは，計算機間の通信をなるべく少なくするような行列の二次元分割を構築することで，従来手法よりもスケーラビリティを向上させている．さらに，このアルゴリズムのデータ分散方法は，固有値計算に用いられる行列ベクトル積として適用しやすいように考案されているため，P_ARPACK との相性も良いと考えられる．

3.3 K-means アルゴリズム

固有ベクトル行列を求めた後は，その各行を１つの座標と見なし，$|V|$ 個の点を k-means によってクラスタリングする．k 行目の点のクラスタ割り当てをネットワークの k 番目の頂点のクラスタ割り当てに対応させることで，それをスペクトラルクラスタリングの結果とする．K-means アルゴリズムは，最も一般的な Lloyd アルゴリズムを並列化して実装した．収束条件は，全てのクラスタ中心の変化量が閾値以下になったかどうかで判定するようにした．並列化した Lloyd アルゴリズムを Algorithm 2 に示す．

Algorithm 2 並列化 Lloyd アルゴリズム

```
 1: for p in P do
 2:    C_1^p, ..., C_k^p をランダムに初期化
 3:    Centroid_i ← 0
 4:    loop
 5:       C_i ← C_i^1 + ... + C_i^{|P|} (i = 1, ..., k)
 6:       NextCentroid_i = (Σ_{x∈C_i} ||x||) / |C_i|
 7:       if ∀i, ||NextCentroid_i − Centroid_i|| < t then
 8:          break
 9:       end if
10:       for i in {1, ..., k} do
11:          NextC_i^p ← ∅
12:       end for
13:       for v in V^p do
14:          î ← argmin_{i∈{1,...,k}} ||NextCentroid_i − v||
15:          NextC_î^p ← NextC_î^p ∪ {v}
16:       end for
17:       for i in {1, ..., k} do
18:          C_i^p ← NextC_i^p
19:          Centroid_i ← NextCentroid_i
20:       end for
21:    end loop
22: end for
```

ただし，Algorithm 2 において，P は K-means を実行するプレースの集合，V^p は各プレースに分散された頂点の集合である．すなわち，$V^1 \cup ... \cup V^{|P|} = V$ である．また，t は K-means の収束判定の閾値である．1 行目の for ... do から 22 行目の end for まで，全てのプレースが並列に処理を行うが，5 行目（二重下線の行）でのみ全プレースが同期を取って集団通信を行う必要がある．ここでは各プレースが持っている C_i^p の総和を求めるので，MPI_Allreduce による通信が行われる．それ以外の処理は全てプレースごとに独立した処理なので，並列に実行することが出来る．

4. X10 によるスペクトラルクラスタリングの実装

4.1 並列プログラミング言語 X10

X10 とは，IBM が開発した，並列分散プログラムの記述に特化したプログラミング言語である [3]．X10 は，プログラム実行時に生成される全てのタスクに親子関係を持たせている．プログラム起動時には必ず１つのタスクが生成され，C 言語で言うところの main 関数の実行を開始する．新たなタスクは，async コマンドによって既に生成されているタスクの子タスクとしてのみ生成される．そして，finish コマンドを使うと，その finish 文の中で生成された全てのタスクの終了を待つことが出来る．これらの仕様は，プログラマが不注意によってデッドロックを起こすようなプログラムを書いてしまうことを防止している．

また，X10 は言語レベルで並列計算をサポートする，いわゆる並列プログラミング言語であり，ある計算機上で生成したタスクを at コマンドで別の計算機上で実行させることが出来る．X10 では並列計算における実行単位は計算機ノードやプロセッサではなくプレースと呼ぶため，at コマンドはあるプレースから別のプレースへタスクを渡すコマンドであると言い換えられる．プレース間ではメモリを共有しないため，データを受け渡す際には計算機ノード間の通信が発生する．ただし，この通信をプログラマが明示的に記述してやる必要はなく，X10 の処理系が内部でソケットや MPI 等を使って通信を行ってくれる．これによりプログラマが受け渡したいデータのサイズやアドレスを意識することなく，自然に並列プログラムを記述することが出来るため，X10 は生産性の高い並列プログラミング言語と言われている．

4.2 スペクトラルクラスタリングの実装

4.2.1 データの分散

アルゴリズムの実装に当たって，まずはデータをどのように複数の計算機に分散させるかを決定しなければならない．我々のアルゴリズムの中では Yoo らの MatVec アルゴリズムを用いているので，必然的に MatVec が要求する 2 次元分割方式を採用することとなる．行列の 2 次元分割は，まずプロセッサの 2 次元配置が決められた後，それに基づいて決定されるものである．MatVec の 2 次元分割の場合，例えば図 2 のようなプロセッサ配置が与えられたならば，行列は図 3 のように分割される．ベクトルの場合は，プロセッサの数で単純に 1 次元分割される．

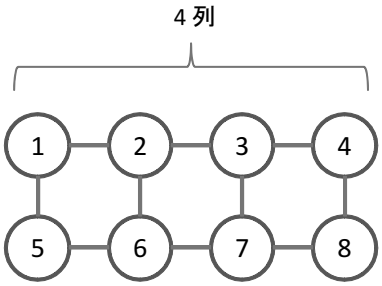

図 2 8 つのプロセッサの 2 次元配置例

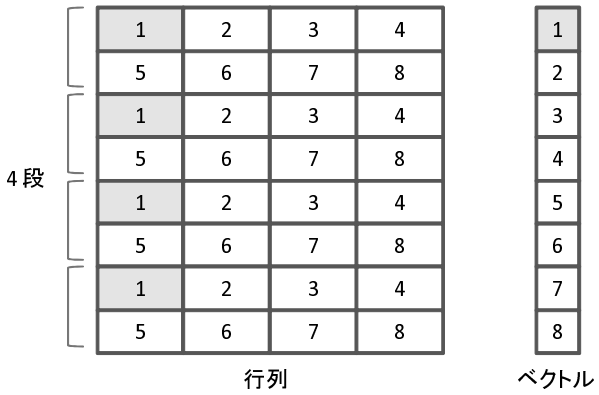

図 3 行列とベクトルの 2 次元分割例．編み掛けの部分がプロセッサ 1 に割り当てられるデータである．

4.2.2 Affinity Matrix の計算

Affinity Matrix $L = D^{-1}A$ を求めるためには，まず隣接行列 A から次数行列 D を求める必要がある．次数行列は，A の各行の要素の総和を対角線状に並べた行列なので，実際にはその対角要素だけを計算できれば良い．これは，A と要素が全て 1 のベクトル $\mathbf{1}$ の積で簡単に求めることが出来る．

$$D = \mathrm{diag}(A\mathbf{1}) \tag{2}$$

$D^{-1}A$ は，A の各行に D^{-1} の各対角要素をそれぞれ掛けていることと等しいので，これは用意に実装でき，並列化も簡単である．

4.2.3 P_ARPACK による固有ベクトル計算

P_ARPACK は元々 Fortran のためのライブラリとして開発されており，これを X10 から利用するには，X10 の @Native アノテーションを使う必要がある．@Native を使うと C++や Java で書かれたコードを X10 プログラムから利用することが出来るようになる．P_ARPACK には C 言語のインターフェースが存在するので，@Native で P_ARPACK の関数を呼び出すことが可能である．

P_ARPACK は様々なタイプの固有方程式に対応しており，ユーザーがタイプを細かく指定できるように，多くのパラメータが用意されている．我々の実装における P_ARPACK のパラメータを表 2 にまとめた．このようにパラメータを指定することで，我々が求めたい $Lu = \lambda u$ の最大から k 個の固有値に対応する固有ベクトルが計算される．

4.2.4 疎行列ベクトル積

4.2.5 K-means

K-means のアルゴリズムは，Algorithm 2 で示した擬似コードの通りである．この擬似コードを X10 のプログラムにするのは比較的簡単な手順で出来る．例えば，1 行目の for p in P do ... は X10 で `finish for(p in Place.places()) at(p) async { }` と記述でき，5 行目の集団通信は x10.util.Team オブジェクトを利用して，`team.allreduce(role, src, src_off, ...);` のようにシンプルに書くことが出来る．

パラメータ	値	備考
nev	k	求める固有ベクトル数 = クラスタ数
$bmat$	'I'	標準固有方程式の指定
$which$	"LA"	最大固有値計算の指定
tol	0.01	収束判定の閾値
$maxitr$	1000	最大反復回数
$rvec$	1	固有ベクトルを計算するかどうか

表 2 P_ARPACK のパラメータ指定

5. 性能評価

我々の実装したスペクトラルクラスタリングを，速度と精度の 2 つの面から評価する．実行環境は，東京工業大学のスーパーコンピュータ TSUBAME2.0[12] を使用し，計算機を最大 64 ノードまで稼働させて実験を行った．1 ノードにつき Intel Xeon 2.93GHz 6 コア CPU を 2 つ搭載，メモリは最大 54GB 使用可能である．1 ノード当たり 2CPU なので，プログラムの実行は 1 ノード当たり 2 プレースで行うこととした．すなわち，16 プレースでの実行とは，本論文では 8 ノードでの実行であることを意味する．

速度評価においては，特にスケーラビリティを有しているか，どの程度大規模に実行可能か，ということに着目して実験を行った．一方，精度評価においては，スペクトラルクラスタリングによってどの程度良いクラスタ分割を求めることが出来るのか，ということについて，Rand Index という指標を用いて評価を行った．

5.1 速度評価

速度評価に用いるデータセットとして，あらかじめ R-MAT グラフを生成しておいた．スケールは 26 から 29 までの 4 種類とし，どれも頂点数とエッジ数の比が等しくなるようにパラメータを設定した．R-MAT グラフのサイズとパラメータを表 4 にまとめた．

R-MAT パラメータ	$a = 0.45, b = c = 0.15, d = 0.25$	
Scale	$\lvert V\rvert$	$\lvert E\rvert$
26	67,108,863	268,435,456
27	134,217,728	536,870,912
28	268,435,456	1,073,741,824
29	536,870,873	2,147,483,648

図 4 データセットと R-MAT パラメータ

また，スペクトラルクラスタリングのパラメータとしては，クラスタ数や収束判定の閾値などがあるが，今回は精度については考慮せず速度のみに焦点を当てて測定を行った．具体的なパラメータは表 3 にまとめた．

パラメータ	値	備考
k	2	クラスタ数
tol	0.01	ARPACK の収束判定閾値
$maxitr$	1000	K-means の最大反復回数
t	0.00001	K-means の収束判定閾値

表 3 スペクトラルクラスタリングのパラメータ

5.1.1 ストロングスケール

R-MAT グラフに対するストロングスケールの測定結果を図 5 に示す．データは Scale-29 に固定，プレース数を変化させ，実行時間がどのように変化するかを調べた．8 プレース以下での実行も試みたが，メモリ不足のため実行できなかった．256 プレース以上で実験を行うことは可能であるが，諸事情により今回は行っていない．

実験結果を見る限り，128 プレースまでは良くスケールしているようである．実行時間の内訳を見ると，固有ベクトル計算が占める割合が 16 プレースのときは約 87%であるのに対し，128 プレースのときは約 92%となっている．これは，固有ベクトル計算よりもそれ以外の計算（Affinity Matrix の計算や K-means）のスケール性が良いためだと考えられる．実際，Affinity Matrix の計算は単純な 2 回の行列演算で済むため，効率よく並列化することが出来ているし，K-means は Allreduce 以外全ての処理を並列化できている上，反復回数も P_ARPACK に比べると遥かに少ない．それに対し，固有ベクトル計算は疎行列ベクトル積を 200 回程反復しなければならないため，疎行列ベクトル積の通信のオーバーヘッドが必然的に顕著になりやすい．

5.1.2 ウィークスケール

次に，ウィークスケールの測定結果を図 6 に示す．1 プレース当たりのデータサイズを固定し，データサイズを変化させ，実行時間がどのように変化するかを調べた．データのスケールが 1 上がるとデータサイズは 2 倍になるので，それと同時にプレース数を 2 倍にすることで，1 プレース当たりのデータサイズを固定した．理想的に並列化されたプログラムの場合，1 プレース当たりのタスクが同じ量ならば，常に一定の実行時間となるはずであるが，実験結果はそのようになっておらず，むしろ右肩上がりのグラフになってしまっている．この結果から，並列化されていない処理，特にノード間の集団通信に掛かるコストの大きさを見ることが出来る．

実行時間の増加の仕方は，ほぼ線形であるように見える．Scale-26 と Scale-29 で実行速度が 3 倍程違うため，ウィークスケーリングの観点からは，あまり良くスケールしていないと思われる．本来ならば，実行時間のうち通信に掛かった時間とそれ以外の処理にかかった時間の割合を調査すべきであったが，時間の都合により未調査であり，本論文には掲載できなかった．もし実行時間の増加分が全て通信によるものであったならば，疎行列ベクトル積における集団通信の高速化を図ることが重要であると考える．

5.2 精度評価

スペクトラルクラスタリングの精度の評価には比較的小規模なネットワークを用い，Rand Index と呼ばれるクラスタリング指標でその精度を測った．データは Internet データセットと Flickr データセットの 2 種類を用意した．Internet データセットは頂点数 65,536，エッジ数 96,872 のネットワークであり，Flickr データセットは頂点数 105,938，エッジ数 2,316,948 のネットワークである．実行時のパラメータは，クラスタ数以外は表 3 の通りとし，クラスタ数を変化させることで精度にどのような変化があるかを調べた．

5.2.1 Rand Index

Rand Index は，クラスタリングの精度を測る指標として一般的に用いられている．定義を式 (3) に示す．

$$RI = \frac{TP + TN}{TP + TN + FP + FN} \tag{3}$$

ただし，

- TP : 同じクラスタに属する，接続されている頂点ペア数
- TN : 別のクラスタに属する，接続されていない頂点ペア数
- FP : 同じクラスタに属する，接続されていない頂点ペア数
- FN : 別のクラスタに属する，接続されている頂点ペア数

であるとする．このように定義すると，任意の頂点ペアは TP, TN, FP, FN のどれかに必ず属すので，頂点数を n とすると，$TP + TN + FP + FN = {}_nC_2 = n(n-1)/2$ である．エッジで接続されている全ての頂点を同じクラスタに，接続されていない頂点を異なるクラスタに割り当てることが出来た場合，RI(Rand Index) の値は 1 になる．

5.2.2 実験結果

Flickr, Internet データセットにおける精度評価を表 4 に示す．Internet については，クラスタ数 2 の分割で既に RI が 0.9 を超えており，ネットワーク構造が元々クラスタリングしやすい形状をしていた可能性が考えられる．また，クラスタ数を増やすと RI 値が増加するのは，Internet データセットの最適な分割におけるクラスタ数が 64 よりも大きいからであると推察するが，Internet のような非常に疎なネットワークのクラスタリング指標が 1 に近い値を取るとは考えにくいため，実験方法に何らかのミスがあるかもしれない．現在，これに関して調査を行っている．

Flickr についてであるが，こちらは Internet と比べると RI があまり高くない傾向にある．クラスタ数 4 から 16 辺りの RI 値は割と妥当な結果であると思われるが，16 以降，クラスタ数を増やせば増やす程 RI 値が際限なく増加するのは，実験方法の誤りによるものだと考えている．

6. 関連研究

我々が以前発表したスペクトラルクラスタリング [13] は、

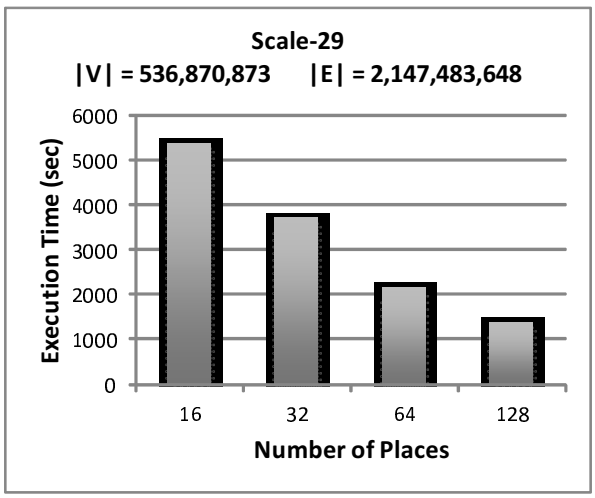

図 5 ストロングスケールによる速度評価

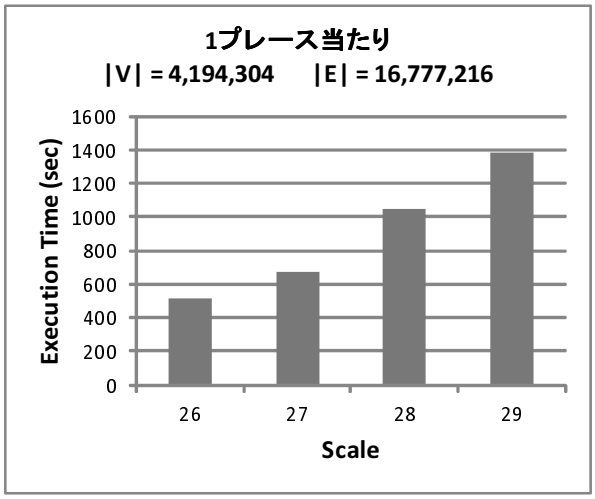

図 6 ウィークスケールによる速度評価

データセット	クラスタ数	RI
Internet	2	0.9471
	4	0.9773
	8	0.9814
	16	0.9857
	32	0.9800
	64	0.9936
Flickr	2	0.1500
	4	0.6187
	8	0.7243
	16	0.8095
	32	0.8998
	64	0.9387

表 4 Rand Index を用いた精度評価結果

P_ARPACK の代わりに ScaLAPACK を使って固有ベクトルを求める手法であったが、ScaLAPACK が疎行列データ形式に対応していなかったこともあり、数十万頂点規模のグラフまでしかスケールしなかった。本研究はそれを踏まえ、疎行列の固有値計算に特化した ARPACK ライブラリの利用を試みた。

2011 年に Chen ら [14] が発表した Parallel Spectral Clustering は，我々が大規模ネットワーク向けのスペクトラルクラスタリングを提案する上で大きな指針となった．彼らが対象としているのはネットワーク構造ではないものの，スペクトラルクラスタリングの並列化手法を複数実装し，結果の比較を行っている．

また、Dhillon ら [15] はスペクトラルクラスタリングの目的関数が Weighted Kernel K-means という別の問題の目的関数の特殊ケースと等価であることを示し、スペクトラルクラスタリングそのものを K-means に帰着させて解く手法を提案した。

7. まとめ

本研究では，クラスタリングアルゴリズムの中でも比較的精度が高いことで知られるスペクトラルクラスタリングを，大規模なネットワークデータに適用することを目的とし，大規模ネットワーク向けのスケーラブルなスペクトラルクラスタリングを提案，そしてそれを並列プログラミング言語 X10 で実装した．その中で，高速な固有ベクトル計算を P_ARPACK ライブラリの利用と Yoo らが提案した大規模疎行列ベクトル積の実装によって実現した．

実行速度の面では，R-MAT グラフの Scale-29 において十分なスケール性を示したが，プレース数が多くなると疎行列ベクトル積における集団通信がボトルネックとなり，スケーラビリティの低下を引き起こすことが分かった．疎行列ベクトル積の実行時間が全体時間の 90%近くを占めるため，この部分の更なる高速化が非常にであると考える．

精度の面では，Internet, Flickr 双方でクラスタリング精度が良好であることを示した．しかし，Rand Index の値に若干疑問が残る結果であったため，現在調査を行っているところである．また，今回は他のクラスタリング手法との比較を行わなかったため，それを含めた詳細な精度検証を今後の課題とする．

謝辞 本論文の執筆においては，東京工業大学客員准教授の鈴村豊太郎先生にさまざまなご指導，ご教示を頂いたことを深謝する．

参考文献

[1] 橋本康弘, 陳 Yu, and 大橋弘忠. ”ソーシャルネットワークからのコミュニティ時系列の抽出と可視化分析.” 情報処理学会研究報告. MPS, 数理モデル化と問題解決研究報告 2008.85 (2008): 63-66.

[2] 松尾豊, et al. ”Web 上の情報からの人間関係ネットワークの抽出.” 人工知能学会論文誌 20.1 (2005): 46-56.

[3] Saraswat, Vijay A., Vivek Sarkar, and Christoph von Praun. "X10: concurrent programming for modern architectures." Proceedings of the 12th ACM SIGPLAN symposium on Principles and practice of parallel programming. ACM, 2007.

[4] Ng, Andrew Y., Michael I. Jordan, and Yair Weiss. "On spectral clustering: Analysis and an algorithm." Advances in neural information processing systems 2 (2002): 849-856.

[5] M.Melia and J Shi, "A Random Walks View of Spectral Segmentation," Proc Int 'l conf Artificial Intelligence and Statistics, 2001.

[6] Shi, Jianbo, and Jitendra Malik. "Normalized cuts and image segmentation." Pattern Analysis and Machine Intelligence, IEEE Transactions on 22.8 (2000): 888-905.

[7] 山本有作. "キャッシュマシン向け対称密行列固有値解法の性能・精度評価." 情報処理学会論文誌: コンピューティングシステム 46: 81-91.

[8] 片桐孝洋, and 金田康正. "並列固有値ソルバーの実現とその性能." 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] 97.121 (1997): 49-54.

[9] Maschhoff, Kristyn J., and D. C. Sorensen. "P_ARPACK: An efficient portable large scale eigenvalue package for distributed memory parallel architectures." Applied Parallel Computing Industrial Computation and Optimization. Springer Berlin Heidelberg, 1996. 478-486.

[10] Lehoucq, R. B., et al. "ARPACK: An implementation of the Implicitly Re-started Arnoldi Iteration that computes some of the eigenvalues and eigenvectors of a large sparse matrix, 1995." Available from netlib@ ornl. gov under the directory scalapack.

[11] Andy Yoo, Allison H. Baker, and Roger Pearce. "A scalable eigensolver for large scale-free graphs using 2D graph partitioning." Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis. ACM, 2011.

[12] Matsuoka, Satoshi, et al. "The Total Picture of TSUBAME2. 0." Tsubame e-Science J 1 (2010): 16-18.

[13] Ogata, Hidefumi, Miyuru Dayarathna, and Toyotaro Suzumura. "Towards highly scalable X10 based spectral clustering." High Performance Computing (HiPC), 2012 19th International Conference on. IEEE, 2012.

[14] Chen, Wen-Yen, et al. "Parallel spectral clustering in distributed systems." Pattern Analysis and Machine Intelligence, IEEE Transactions on 33.3 (2011): 568-586.

[15] Dhillon, Inderjit S., Yuqiang Guan, and Brian Kulis. "Kernel k-means: spectral clustering and normalized cuts." Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2004.