大規模ネットワークに対するリンク予測の並列実装

小形 英史[1,a)]　鈴村 豊太郎[2,3,4,b)]

概要：近年、Twitter や FaceBook，LINE などのソーシャルアプリケーションの利用が盛んになっており、このようなアプリケーションのユーザー同士の繋がりを解析することにより、繋がりの構造的特徴およびユーザーの興味の傾向などを明らかにする試みが注目されている。ネットワーク解析手法の中には、クラスタリングなどネットワーク全体の情報を必要とするものもあるが、ソーシャルアプリケーションユーザーの数は日々増加し続けており、クローリングによってユーザーネットワーク全体を取得することは困難になりつつある。この問題に対する１つの解決策として、リンク予測と呼ばれるネットワーク解析手法が存在する。これは、あるネットワークの部分情報が与えられたときに、そこから元のネットワークを予測する手法である。これを利用すると、リンク予測によって Twitter の部分ネットワークから全体を予測し、そうして得られたネットワークを解析することで、真のネットワークに対する解析に近い結果を得ることができるのである。
本研究では、Twitter ネットワークに対するリンク予測アルゴリズムを並列プログラミング言語 X10 で実装し、実行速度及び予測精度の評価を行った。

1. 研究背景

ソーシャルネットワーク解析の分野では Twitter のユーザーネットワークの解析が盛んに行われている。Haewoon ら [1] は 2009 年に Twitter のユーザーネットワーク全体を様々な側面から解析し、Twitter ネットワークが従来のソーシャルネットワークと異なる性質を持つことを明らかにした。また、Bongwon ら [2] は 2010 年に 7400 万ものツイートデータを用いて、「リツイート」が情報の拡散に与える影響を調査した。このような解析を行うことで、Twitter が持つ従来の SNS(ソーシャルネットワーキングサービス) とは異なる特徴や性質を明らかにするだけでなく、特定のユーザーに類似するユーザーの抽出や、ツイート情報に基づく商品のリコメンドなど、Twitter を利用したネットワークサービスの充実にも繋がると期待されている。

Twitter のみならず、ソーシャルアプリケーションユーザーの数は日々増加し続けており、Haewoon らの研究があった 2009 年では 4170 万人であった Twitter ユーザー数が、2012 年 7 月 1 日には 5 億を超えるまでに成長している。

このようなソーシャルアプリケーションのユーザーネットワークを解析することは、一般的には容易ではない。その要因は２つあり、１つは大規模なネットワークデータを丸ごとローカルストレージに確保し、その上で解析を行わなければならないという点であり、もう１つの要因は、ソーシャルネットワーク全体のデータを保存するために莫大な記憶容量を必要とする点である。

このような問題を解決するために、我々は**リンク予測**と呼ばれるネットワーク解析手法の利用を試みた。これは、あるネットワークの部分情報が与えられたときに、そこから元のネットワークを予測する手法である。これを利用すると、ソーシャルネットワークの部分的なデータから全体を予測でき、そうして得られたネットワークを解析することで、真のネットワークに対する解析に近い結果を得ることができると考えられる。リンク予測によって部分ネットワークから元のネットワークを十分な精度で復元することが出来れば、ネットワーク解析のためにネットワーク全体のデータを長い時間を掛けて取得しなくても良くなると考えられる。

そこで本研究では、大規模なグラフデータに対応できるリンク予測アルゴリズムを調査し、Raymond ら [5] が提案した Link Propagation というアルゴリズムを並列プログラミング言語 X10 で実装した。その際、アルゴリズム中の不明瞭な箇所を適切に補完することで、大規模なソーシャルネットワークに適用できるようなものとした。性能評価のためのデータとして Twitter ユーザーネットワークを使

[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 Research
[3] University College Dublin
[4] JST CREST
[a)] ogata.h.ac@m.titech.ac.jp
[b)] suzumura@cs.titech.ac.jp

用し、それに対する Link Propagation アルゴリズムの予測精度を評価することにより、リンク予測によるネットワーク構造の復元がどの程度有効であるかを明らかにできると考えられる。また、アルゴリズムの中でどのようなヒューリスティックを用いるとリンク予測精度の向上に寄与するかを調べることで、Twitter ネットワーク特有の構造的特徴などを発見することも期待できる。

2 節でリンク予測および Link Propagation アルゴリズムについて説明し、4.2 節で Link Propagation の X10 による実装方法を述べる。5 節で Link Propagation の実行速度とリンク予測精度の評価を行う。6 節で関連研究を紹介し、最後に 7 節で結論を述べる。

2. リンク予測問題

リンク予測問題とは、あるネットワークが与えられたときに、その情報を元に将来そのネットワークに追加されるであろうリンクを予測する問題である。初めに既知のリンクに関する情報が与えられ、そこから未知のリンクを予測する問題は、半教師付き学習問題の一つと見なすことができる。リンク予測はソーシャルネットワークの分析やタンパク質の相互作用の分析などで重要な役割を持つ。例えば SNS のあるユーザーに対しておすすめのユーザーを提示する機能は、リンク予測を使って実現可能である。また、既知のタンパク質間の相互作用を元に未知のタンパク質との相互作用を予測することもできる [3]。これらは、現時点でのネットワーク構造から未来のネットワーク構造を予測するためにリンク予測を用いた例である。

一方で、部分的に失われたリンクを現在のネットワーク構造から予測し、復元するためにリンク予測を用いるケースも存在する。本研究ではこの側面に着目し、Twitter ネットワークの部分構造から全体をどの程度正確に予測できるのかを調査した。

2.1 定義

ネットワークを $G = \langle V, E, W \rangle$ とし、V を頂点集合、$E = \{\langle u, v \rangle | u, v \in V\}$ をエッジ集合、$W : E \longrightarrow \mathbb{R}$ をエッジの重み集合と定義する。リンク予測問題は、既知のネットワーク G が与えられたときに未知のネットワーク $G' = \langle V, E', W' \rangle$(但し $E \subset E', W \subset W'$) を求める問題と表される。$u, v \in G$ に対するリンク予測指標を $score : (G, u, v) \longmapsto r \in \mathbb{R}$、閾値を α としたとき、リンク予測結果を求める関数は式 (1) となる。従って、リンク予測とはリンク予測指標 $score$ と閾値 α をいかに上手く定義するか、という問題と考えることができる。

$$pred(u, v) = \begin{cases} 0 : score(G, u, v) < \alpha (\text{リンク無し}) \\ 1 : score(G, u, v) \geq \alpha (\text{リンク有り}) \end{cases} \quad (1)$$

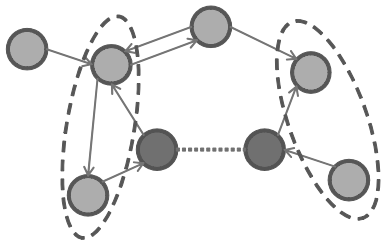

図 1 2 ノード間のリンク予測に隣接ノードの情報を利用する予測手法の例。点線で囲まれた隣接ノードの情報を元に、濃灰色の 2 ノード間のリンクの有無を予測する。

3. ペアワイズ予測モデル

リンク予測の手法の 1 つに、2 ノード間のリンク予測を全てのノードペアに対して独立に行う**ペアワイズ予測モデル**がある。ペアワイズ予測モデルでは、ノードペアのリンク予測にそのノードペア周辺の情報あるいはネットワーク全体の情報を用いる。ノードペア周辺の情報とは、例えば隣接ノードの集合である（図 1)。ペアワイズ予測モデルに基づくリンク予測においては、*common neighbors* や *Jaccard's coefficient* などが既存研究として有名である [4]。頂点 v の隣接ノード集合を $\Gamma(v)$ としたときの、common neighbors および Jaccard's coefficient のリンク予測指標を式 (2), (3) に示す。

$$Common(u, v) = |\Gamma(u) \cap \Gamma(v)| \quad (2)$$

$$Jaccard(u, v) = \frac{|\Gamma(u) \cap \Gamma(v)|}{|\Gamma(u) \cup \Gamma(v)|} \quad (3)$$

ペアワイズ予測モデルは、2 ノード間のリンク予測をそれぞれ独立に行えるという特徴があるが、これはあくまで近似であり、本来ならば全てのリンクに対する同時確率を考えなければならない。そのような予測モデルを関係ネットワーク予測モデルと呼ぶ。しかし、ペアワイズ予測モデルはこの近似のおかげで、アルゴリズムの並列化が容易になるというメリットがあるため、リンク予測の計算速度を重視するならば、関係ネットワーク予測モデルよりもペアワイズ予測モデルを選択すべきであると言える。

ペアワイズ予測モデルにおけるリンク予測は 2 タイプ存在する。1 つは「類似しているノード間にリンクがある」と予測するノード特徴ベース推論で、もう 1 つは「類似しているノードペア間はリンクの有無も類似している」と予測するペアワイズ特徴ベース推論である (図 2)。本研究で実装した Link Propagation アルゴリズムは、後者のペアワイズ特徴ベース推論に基づくリンク予測アルゴリズムである。

4. Link Propagation アルゴリズム

本研究で実装を行った Link Propagation アルゴリズムについて概説する。Link Propagation は、Raymond ら [5]

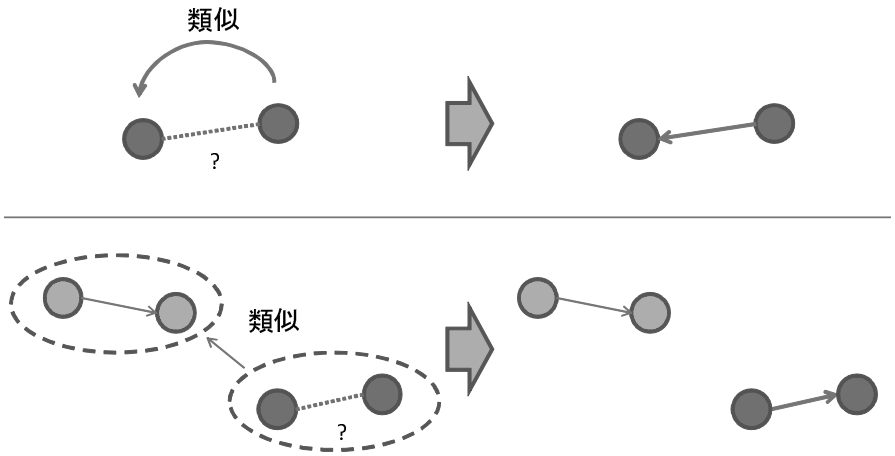

図 2 上図がノード特徴ベース推論、下図がペアワイズ特徴ベース推論のイメージである。ノード特徴ベース推論では類似している頂点間にリンクがあると予測するが、ペアワイズ特徴ベース推論では類似しているノードペア間でリンクの張り方も類似するような予測をする。

が 2010 年に提案した、大規模グラフに適用可能なリンク予測アルゴリズムである。Link Propagation はペアワイズ特徴ベース推論では、既にリンクが張られているノードの情報を元に、まだリンクが張られていないノード間でのリンク予測を行う。Link Propagation ではノード間の情報として**類似度**を利用するため、このアルゴリズムが必要とする情報として、ネットワークの隣接行列の他に、既知のリンクに対応する 2 ノード間の類似度を表した類似度行列を事前に生成しておかなければならない。

Raymond らは論文の中で、厳密解を求める Link Propagation と近似解を求める Link Propagation の 2 つを提案している。厳密解のバージョンでは与えられた類似度行列をそのまま行列計算に使うのに対し、近似解のバージョンでは類似度行列を低階数近似によって小行列の積に分解し、その小行列を行列計算に使うことでメモリ使用量を削減する工夫がなされている。これにより予測精度は多少悪くなるものの、大規模グラフであっても任意のサイズの小行列に落とし込んでリンク予測を適用させることが可能となっている。本研究では大規模な Twitter ネットワークをリンク予測の対象とするため、近似解を求める Link Propagation を採用した。

4.1 アルゴリズム

本節では近似解を求める Link Propagation のアルゴリズムを説明する。初めにアルゴリズムの大筋を示し、その後各ステップについて詳しく述べる。

Link Propagation アルゴリズムは以下の手順で実行される。

(1) ネットワークの隣接行列 W から類似度行列 S を求める。
(2) 類似度行列 S を低階数近似し、小行列の積 $S = GG^T$ に分割する。
(3) 小行列 G に対する行列計算を施し、Core Matrix と呼ばれる行列を求める。
(4) Core Matrix に対する行列計算により、任意のノードペアに対するリンク予測指標を得る。

4.1.1 類似度行列

類似度行列は隣接行列から求まる行列であるが、Raymond らは具体的な類似度の定義を示していない。そこで本研究ではいくつか類似度の定義を決め、それぞれについてリンク予測精度を算出し比較することで、適切な類似度の定義を定めることとした。ただし後々の処理の関係上、隣接行列は対称行列でなければならないことに注意する。

類似度の定義としては次のようなものが考えられる。

$$S_1(u,v) = \begin{cases} |\Gamma(u) \cap \Gamma(v)| & : (u,v) \in E \\ 0 & : (u,v) \notin E \end{cases} \tag{4}$$

$$S_2(u,v) = \begin{cases} \dfrac{|\Gamma(u) \cap \Gamma(v)|}{|\Gamma(u) \cup \Gamma(v)|} & : (u,v) \in E \\ 0 & : (u,v) \notin E \end{cases} \tag{5}$$

ここで、類似度行列の疎性について少し述べておきたい。一般的に、実世界の大規模なネットワークの構造は疎であることが知られている。そのようなネットワークから得られる隣接行列はもちろん疎行列となり、前述の定義から得られる類似度行列も疎行列となる。我々が扱う Twitter ネットワークも疎なネットワークの 1 つなので、これ以降、類似度行列は疎行列であることを前提として議論を進めてゆく。

疎行列は密行列に比べて値が 0 である要素を多く持つので、非ゼロの要素のみを格納するようなデータ形式（Compressed Row Storage など）を用いることで扱うデータ量

を大きく減らすことができる。大規模ネットワークが疎行列であることは非常に重要である。

4.1.2 低階数近似

アルゴリズムのステップ 2 において、類似度行列 S を低階数近似によって小行列の積に近似する。ここで用いる低階数近似は、$S = GG^T$ という形に分解できるならば何でもよい。本研究では、コレスキー分解を独自に変形したアルゴリズムを利用する。

コレスキー分解とは、正定値エルミート行列を下三角行列とその共役転置との積に分解することである。実行列の範囲では、正定値エルミート行列は正定値対称行列と等しい。

計算機上でコレスキー分解を求めるアルゴリズムを Algorithm 1 に示す。このアルゴリズムは、外側の for ループの j 回目で L の j 列目の値を決定している。従って、この for ループを $j = 1 \ldots M(M < N)$ としたならば、L の 1 列目から M 列目までが計算できることになる。低階数近似としてコレスキー分解を使う場合、このように M 列目で計算を打ち切って、$N \times M$ 行列となった L で $A \approx LL^T$ と近似する方法が良く用いられる。これにより行列の保持に必要なメモリ量を M/N に削減することができる。

Algorithm 1 コレスキー分解

Require: 正定値対称行列 $A = \{a_{ij}\} \in \mathbb{R}^{N \times N}$
Ensure: 下三角行列 $L = \{l_{ij}\} \in \mathbb{R}^{N \times N}, A = LL^T$
for $j = 1$ to N **do**
$$l_{jj} \leftarrow \sqrt{a_{jj} - \sum_{k=1}^{j-1} l_{jk}^2}$$
for $i = j + 1$ to N **do**
$$l_{ij} \leftarrow \frac{a_{ij} - \sum_{k=1}^{j-1} l_{ik} l_{jk}}{l_{jj}}$$
end for
end for

ところで、類似度行列 S は対称行列であるが正定値であるとは限らないため、単純にコレスキー分解を適用することはできない。正定値でない対称行列にコレスキー分解を無理矢理適用すると、Algorithm 1 の $l_{jj} \leftarrow \sqrt{a_{jj} - \sum_{k=1}^{j-1} l_{jk}^2}$ でルートの中身が負になり、L が実行列でなくなってしまう。また、S が疎行列であっても L が疎行列になるとは限らないため、メモリ使用量を削減するために行うコレスキー分解がかえってメモリ量を増大させることにもなりかねない。この問題を解決するために、本研究では正定値でない対称行列にも適用可能で、なおかつ分解して得られる下三角行列が元の行列の疎性を保つような近似コレスキー分解を提案する。

基本的なアイデアは、ルートの中身が負になったらそれをある特定の正数に置き換えることと、$a_{ij} = 0$ となるような全ての $i, j(i \neq j)$ に対して $l_{ij} = 0$ としてしまうことである。近似コレスキー分解のアルゴリズムを Algorithm 2 に示す。

Algorithm 2 近似コレスキー分解

Require: 対称行列 $A = \{a_{ij}\} \in \mathbb{R}^{N \times N}, M < N, \epsilon > 0$
Ensure: 下三角行列 $L = \{l_{ij}\} \in \mathbb{R}^{N \times M}, A \approx LL^T$
for $j = 1$ to M **do**
$$l_{jj} \leftarrow a_{jj} - \sum_{k=1}^{j-1} l_{jk}^2$$
if $l_{jj} \geq 0$ **then**
$l_{jj} \leftarrow \sqrt{l_{jj}}$
else
$l_{jj} \leftarrow \epsilon$
end if
for $i = j + 1$ to N **do**
if $a_{ij} \neq 0$ **then**
$$l_{ij} \leftarrow \frac{a_{ij} - \sum_{k=1}^{j-1} l_{ik} l_{jk}}{l_{jj}}$$
else
$l_{ij} \leftarrow 0$
end if
end for
end for

この近似コレスキー分解で得られた下三角行列 L を、アルゴリズム概要で述べた小行列 G と見なし、次のステップでは G に対する行列計算で Core Matrix を計算する。

4.1.3 Core Matrix の計算

Core Matrix とは、Link Propagation においてリンク予測指標を算出する一歩手前の段階の小さな行列であり、一度 Core Matrix を計算しておけば任意のノード間のリンク予測指標を Core Matrix から求めることができる。G に対する行列計算により Core Matrix を計算する手順を以下に示す。

(1) $d \equiv GG^T\mathbf{1}$
(2) $\tilde{G} \equiv \mathrm{diag}(d)^{-\frac{1}{2}} G$
(3) $\tilde{G}^T\tilde{G}$ を固有値分解し、
$\tilde{G}^T\tilde{G} = U\mathrm{diag}(\lambda^{(1)}, \lambda^{(2)}, \ldots, \lambda^{(M)})U^T$ を得る
(4) $V \equiv \tilde{G}U\mathrm{diag}(\lambda^{(1)}, \lambda^{(2)}, \ldots, \lambda^{(M)})^{-\frac{1}{2}}$
(5) $[\Lambda]_{i,j} \equiv \lambda^{(i)}\lambda^{(j)}$
(6) $[D]_{i,j} \equiv \dfrac{\sigma(1+\sigma)[\Lambda]_{i,j}}{1 + \sigma - \sigma[\Lambda]_{i,j}}$
(7) Core Matrix: $C \equiv D * (V^T W V)$

上の手順 6 で現れる σ はノード間類似度の重み付けに用いられているパラメータである。Raymond らの論文の中では σ を 0.001 に固定して実験を行っているため、本研究でもそれに従うこととする。

また、演算子 $*$ は行列の要素同士の積を表す。すなわち、$C = A * B$ のとき $c_{ij} = a_{ij}b_{ij}$ である。

4.2 リンク予測指標の算出

Core Matrix からリンク予測指標を算出するには式 (7)

を用いる。ただし、$v_{(i)}$ は行列 V の i 行目を要素に持つベクトルである。すなわち $V^T = (v_{(1)}, v_{(2)}, \ldots, v(N))$ が成り立つ。

$$P = \frac{1}{1+\sigma}W + \frac{1}{(1+\sigma)^2}VCV^T \quad (6)$$

$$pred(i,j) = [P]_{i,j} = \frac{1}{1+\sigma}[W]_{i,j} + \frac{1}{(1+\sigma)^2}v_{(i)}^T C v_{(j)} \quad (7)$$

このようにして定義されたリンク予測指標 $pred(i,j)$ を元にペアワイズ予測を行うのが、Link Propagation アルゴリズムである。 sectionX10 による Link Propagation の実装

4.3 並列プログラミング言語 X10

Link Propagation アルゴリズムの実装に用いたプログラミング言語である X10 について説明する。

最近では、複数の実行コアを持つマルチコアプロセッサの利用が一般的になっているが、このようなマルチコアプロセッサで構成された並列分散システムの性能を最大限に発揮させるプログラムを設計することは簡単なことではない。openMP を使えば、シングルスレッド用に作られた C のコードにディレクティブを埋め込むことで、容易にプログラムを並列化することができるが、細かなチューニングはコンパイラ任せになってしまうため、プログラムによっては思うような性能向上が見られないこともある。一方で MPI プログラミングは、チューニングによってプログラマがより良い性能を引き出すことが可能であるが、プロセッサ間通信の手続きを明示的に記述しなければならないため、プログラムが複雑になりやすく openMP に比べて生産性が低いという問題がある。そこで、並列プログラミングの生産性を保ちつつ、並列化による十分な性能向上を達成するために開発されたプログラミング言語が *X10* である。

X10 はいわゆる *PGAS* **モデル**をベースとした言語の 1 つである。PGAS とは Partitioned Global Address Space の略で、並列分散環境を抽象化し、プログラマからはあたかも全てのプロセッサが単一のアドレス空間上に存在しているように見えるような並列プログラミングモデルである。抽象化によりプロセッサごとに独立したアドレス空間を意識する必要がなくなるため、MPI などによる並列化に比べて生産性が高いことが特長である。PGAS モデルのイメージを図 3 に示す。

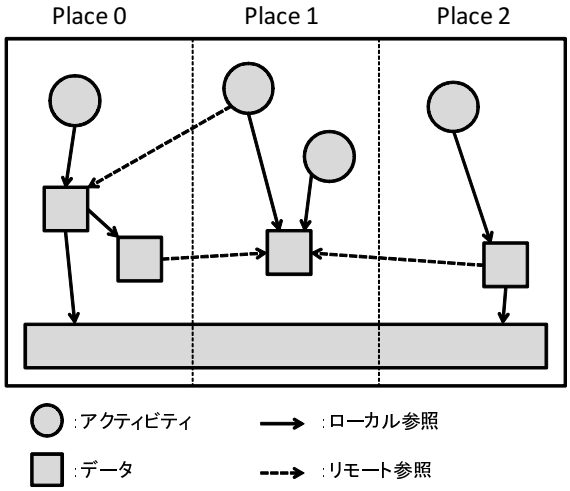

図 3 PGAS モデルの図解

4.4 実装方法

4.4.1 分散疎行列形式によるグラフデータ保持

X10 で Link Propagation を実装するにあたって重要となるのは、ネットワークの隣接行列などのデータをどのようにしてメモリ上に保持するかである。我々が扱いたいのは数億ものノードを持つネットワークであり、全ノードペアに対する情報を全てメモリ上に格納することは不可能である。従ってこの場合には、疎行列形式で隣接行列を保持することが望ましい。大規模ネットワークの隣接行列は疎であるため、疎行列形式にすることでメモリ使用量を大きく削減できる。しかしそれでも、全てのデータを 1 台の計算機で保持できるほどにはならないので、1 つの行列データを複数の計算機に分散して配置する必要がある。

行列の分散方法にはいくつかの方法が考えられる。最も単純なのは行か列を計算機の台数で均等に分割する、いわゆる 1 次元分割である（図 4 左図）。1 次元分割の一般化として、行と列の両方について分割する方法もあり、これを 2 次元分割という（図 4 右図）。分散メモリ環境向けの数値計算ライブラリ ScaLAPACK [6] では、”Block Cyclic Data Distribution”というタイプの 2 次元分割を用いており、これにより 1 次元分割よりも高速な行列計算を実現している。また、Yoo ら [7] が 2011 年に発表した論文では、特殊な 2 次元分割によって分散された行列に対する高速な行列ベクトル積のアルゴリズムが提案されている。行列を 2 次元分割することの利点は、各プレースで必要な行列データを局所化し、プレース間通信のコストを小さくできることである。従って、大規模な行列であるほど通信コストの削減による効果は大きい。しかし小さな行列に対しては、1 次元分割と比べても大した速度向上は見られない。また、分割の仕方が複雑になればその分アルゴリズムも複雑になるので、実装に掛かるコストは 1 次元分割よりも大きい。

本研究における Link Propagation の実装では、行列の分散方法として行方向の 1 次元分割を採用している。なぜなら、Link Propagation アルゴリズムの大部分はコレスキー分解によって得られた小行列に対する行列計算であり、1 次元分割でも十分速く計算できると考えられるからである。さらに、比較的大きな行列の計算として $\tilde{G}^T\tilde{G}$ という行列

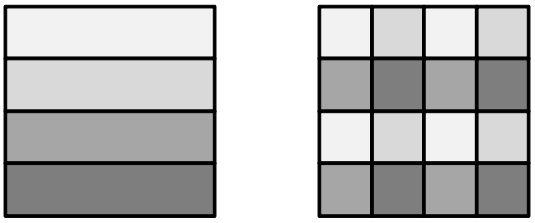

図 4 行列データを 4 台の計算機に分散する例。1 つの計算機が保持する部分を同じ色で表現している。右図の 2 次元分割は ScaLAPACK で用いられているタイプのものである。

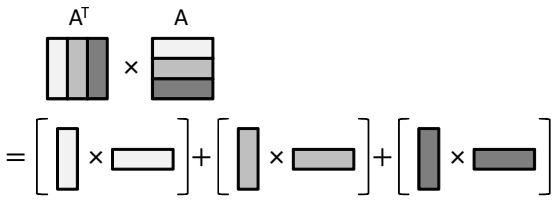

図 5 A^TA の計算。各プレースで自分が持つ行列の積を計算し、最後に全プレースの結果を足し合わせる。

積があるが、この計算はむしろ 1 次元分割を使った方が 2 次元分割よりも効率が良い。図 5 を見れば分かる通り、行方向の 1 次元分割を用いると、A^TA という形式の行列積は各プレースで独立に行列積を計算した後、全てのプレースの結果を合計するだけで良いことが分かる。2 次元分割の場合はこのように上手くいかず、各プレースで必要なデータが他のプレースにまたがることになり、どうしても細かな通信が発生してしまうため効率が悪い。

先ほど例に挙げた $\tilde{G}^T\tilde{G}$ の計算を行う X10 のコードを図 6 に示す。第 1 行で各プレースに DenseMatrix という行列オブジェクトを生成し、2 行目以降で各プレースが並列に行列積の計算を行う。その後、team.allreduce メソッドによって、MPI の AllReduce と同様に全プレースの結果の合計を計算して result オブジェクトに格納する。

4.4.2 近似コレスキー分解の実装

近似コレスキー分解のアルゴリズムは Algorithm 2 で示した通りであるが、このアルゴリズムをそのまま 1 次元分割された行列に適用するのは計算効率が悪いので、アルゴリズムを並列化しやすい形に変形する。アルゴリズム中の $a_{jj} - \sum_{k=1}^{j-1} l_{jk}^2$ と $a_{ij} - \sum_{k=1}^{j-1} l_{ik}l_{jk}$ は、$i = j$ のとき同じ式になるので、それを考慮すると Algorithm 3 のように変形できる。

こうしてできた変形近似コレスキー分解の並列化は容易である。並列化した変形近似コレスキー分解の計算過程を図 7 に示す。このアルゴリズムで行うプロセス間通信は、行列 L の j 行目で値が確定した要素を全プロセスにブロードキャストするだけで、それ以外の計算は全て各プロセスで独立に計算できることが分かる。

```
val result = PlaceLocalHandle.make[DenseMatrix]
               (Dist.makeUnique(),
               () => new DenseMatrix(nCol, nCol));
finish for(p in Place.places()) async at(p) {
  val localResult = result();
  for(var i:Int = 0; i < m.nRow; i++) {
    for(var jj:Int = m.offset(i);
        jj < m.offset(i + 1); jj++) {
      val j = m.columns(jj);
      for(var kk:Int = m.offset(i);
          kk < m.offset(i + 1); kk++) {
        val k = m.columns(kk);
        localResult(j, k) = localResult(j, k) +
                            m.vertices(jj) *
                            m.vertices(kk);
      }
    }
  }
  team.allreduce(here.id, result().data, 0,
                 result().data, 0,
                 result().data.size, Team.ADD);
}
```

図 6 X10 で実装した $\tilde{G}^T\tilde{G}$ の計算。

Algorithm 3 変形近似コレスキー分解

Require: 対称行列 $A = \{a_{ij}\} \in \mathbb{R}^{N\times N}, M < N, \epsilon > 0$
Ensure: 下三角行列 $L = \{l_{ij}\} \in \mathbb{R}^{N\times M}, A \approx LL^T$

```
for j = 1 to M do
  for i = j to N do
    if a_ij ≠ 0 then
      l_ij ← a_ij − Σ_{k=1}^{j−1} l_ik l_jk
    else
      l_ij ← 0
    end if
  end for
  if l_jj ≥ 0 then
    l_jj ← √l_jj
  else
    l_jj ← ε
  end if
  for i = j + 1 to N do
    l_ij ← l_ij / l_jj
  end for
end for
```

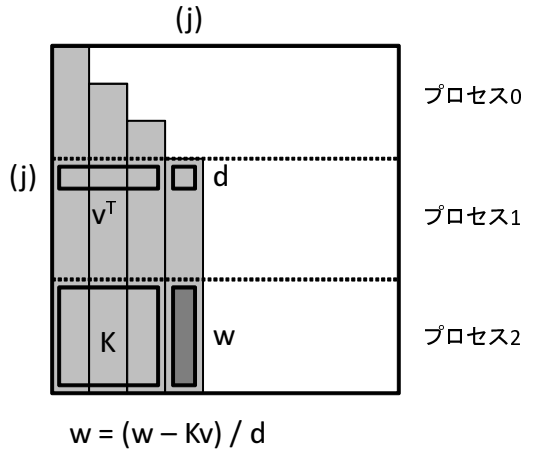

図 7 変形近似コレスキー分解の計算過程。プロセス 2 において w を求めるには、自身が持つ要素 K の他に、第 j 行目を持つプロセスの v と d にあたる要素が必要になる。

CPU	Intel Xeon 2.93GHz (6 cores) x 2
メモリ	54GB
MPI	MVAPICH2-1.9a2
X10	version 2.3.1
GotoBLAS2	version 1.13

表 1 Thin ノードと使用したソフトウェアのバージョン

	ノード数
データセット A	594,455
データセット B	2,650,983

表 2 各データセットに含まれるノード数

4.4.3 行列計算ライブラリ GotoBLAS2 による固有値分解

Link Propagation の中で行列を固有値分解するステップが存在するが、対象となる行列は 1 プロセスで扱える程度の小さな行列なので、既存の行列計算ライブラリである GotoBLAS2 [8] で固有値分解を行った。

5. 性能評価

5.1 実行環境およびデータセット

Link Propagation の実行は全て TSUBAME2.0 の Thin ノードで行う。Thin ノードの実行環境を表 1 に示す。

実行速度の測定および予測精度の評価に用いるデータセットは、2012 年に我々がクローリングによって取得した Twitter ユーザーネットワークの部分ネットワークである。2 種類のデータセット A, B を用いて実験を行った。それぞれが持つノード数は表 2 の通り。

5.2 実験方法

リンク予測はネットワークの欠けたエッジを予測する手法であるから、Link Propagation に渡すネットワークは Twitter ネットワーク全体ではなく、そこから一部のエッジを取り除いたネットワークとする。エッジの除去はランダムサンプリングにより行う。すなわち、全てのエッジに対して一定の確率で取り除くかどうかを決定する。このようにして作られた不完全なネットワークに Link Propagation を適用し、除去されたエッジをどの程度復元できるかを測定する。リンク予測精度の判定には、ROC 曲線や AUC[12] といった評価基準が一般的に用いられている。本研究でもそれに倣い、Link Propagation の精度を AUC で評価する。

実験の流れを以下に示す。

(1) Twitter ユーザーネットワークデータを X10 で並列に読み込み、データをサンプリングする。
(2) サンプリングされたデータから隣接行列を生成する。
(3) 隣接行列に Link Propagation を適用し、Core Matrix を得る。
(4) Core Matrix を元に、エッジの無い全てのノードペアに対するリンク予測を行う。
(5) リンク予測結果から AUC を算出する。

5.3 スケーラビリティ

Link Propagation のスケーラビリティに関する評価結果を図 8, 図 9 に示す。計算機 4 ノードの環境では、データセット B を現実的な時間で処理できなかったため、図 9 の 4 ノードの評価は行っていない。A, B どちらにおいても、16 ノードまでスケールすることを確認できた。Link Propagation の計算過程はほとんど行列計算で構成されているため、並列化による効果が大きかったと考えられる。

5.4 予測精度

予測精度の評価を評価を図 10 に示す。我々の期待とは異なり、コレスキー分解の近似の度合いを変化させても精度に違いが現れなかった。この原因としては、パラメータ $M = 1000$ でも近似が粗すぎるのか、あるいは実装上のミスなのか色々考えられるが、現在はまだ不明である。

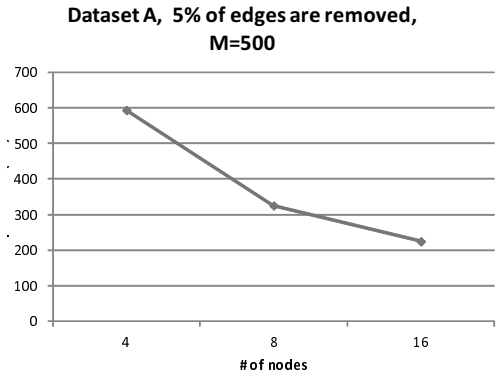

図 8 データセット A におけるスケーラビリティ評価

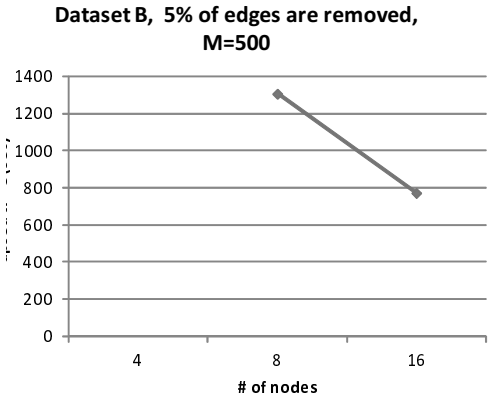

図 9 データセット B におけるスケーラビリティ評価

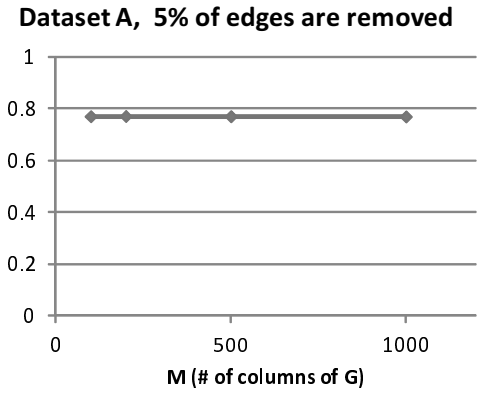

図 10 データセット A における精度評価

6. 関連研究

6.1 リンク予測に関する研究

Song ら [9] は、proximity sketch と proximity embedding という 2 つの手法を元にリンク予測を行うアルゴリズムを提案している。このアルゴリズムは動的に変化するネットワークに対してリアルタイムにリンク予測を行うことができるなどの特徴を持つ。しかし、並列分散環境を意識したアルゴリズムではないため、スケーラビリティの面では Link Propagation に劣る。

Papadimitriou ら [10] は、ペアワイズのリンク予測指標である Katz_β や RWR に代わる指標として FriendLink というアルゴリズムを提案し、計算速度が Katz_β と比較して 2 倍速くなることを示したが、予測精度は RWR より少し高い 55%程度に留まっている。

6.2 低階数近似に関する研究

我々が実装した Link Propagation では低階数近似にコレスキー分解を用いたが、それ以外にもいくつか有力な手法が存在する。

Drineas ら [11] は、行列 A の行と列をそれぞれサンプリングした行列 C, R を作り、$A \approx CUR$ を満たすように U を決定して A を分解する CUR アルゴリズムを提案した。サンプリングする行数や列数は任意に指定できる上、非常に高速なアルゴリズムであるが、$A = LL^T$ の形で分解することはできないため、Link Propagation の低階数近似には適さなかった。

7. 結論

本研究では大規模ネットワークに適用可能なリンク予測アルゴリズムを並列プログラミング言語 X10 で実装した。そこで我々はコレスキー分解アルゴリズムを修正し、大規模疎行列の疎性を保ったまま下三角行列の積に分解できるよう修にした。これにより計算過程における行列の疎性が維持され、Link Propagation のスケーラビリティが向上した。しかしながら、リンク予測精度は期待通りとは行かず、満足のいく結果が得られなかった。

今後この問題を解決するために、類似度の定義や近似コレスキー分解が予測精度にどの程度の影響を与えるかを調査し、適切な類似度の定義を発見することや近似コレスキー分解の精度向上を行うことが重要であると考える。

謝辞 本論文の執筆に際して、様々なご指導、ご教示を頂きました鈴村豊太郎先生に深く感謝致します。

参考文献

[1] Haewoon Kwak, Changhyun Lee, Hosung Park, Sue Moon, "What is Twitter, a Social Network or a News Media?", *WWW '10 Proceeding of the 19th international conference on World wide web*, 591-600, 2010

[2] Bongwon Suh, Lichan Hong, Peter Pirolli, Ed H. Chi, "Want to be Retweeted? Large Scale Analytics on Factors Impacting Retweet in Twitter Network", *Social Computing, 2010 IEEE Second International Conference*, 177-184, 2010

[3] 鹿島 久嗣："ネットワーク構造予測"，人工知能学会誌, 22, 3, 344, 2007

[4] David Liben-Nowell, Jon Kleinberg, "The Link-Prediction Problem for Social Networks", *Journal of The American Society for Information Science and Technology*, 58, 7,

1019-1031, 2007

[5] Rudy Raymond, Hisashi Kashima, "Fast and Scalable Algorithms for Semi-supervised Link Prediction on Static and Dynamic Graphs", *Machine Learning and Knowledge Discovery in Databases*, 6323, 131-147, 2010

[6] J. Choi, J. Demmel, I. Dhillon, J. Dongarra, S. Ostrouchov, A. Petitet, K. Stanley, D. Walker, R.C. Whaley, "ScaLAPACK: a portable linear algebra library for distributed memory computers – design issues and performance", *Computer Physics Communications*, 97, 1-2, 1-15, 1996

[7] Andy Yoo, Allison H. Baker, Roger Pearce, Van Emden Henson, "A Scalable Eigensolver for Large Scale-Free Graphs Using 2D Graph Partitioning", *High Performance Computing, Networking, Storage and Analysis*, 1-11, 2011

[8] K. Goto, K. Milfeld, Texas Advanced Computing Center - GotoBlas2, http://www.tacc.utexas.edu/tacc-projects/gotoblas2

[9] Han Hee Song, Tae Won Cho, Vacha Dave, Yin Zhang, Lili Qiu, "Scalable Proximity Estimation and Link Prediction in Online Social Networks", *IMC '09* 322-335, 2009

[10] Alexis Papadimitriou, Panagiotis Symeonidis, Yannis Manolopoulos, "Scalable Link Prediction in Social Networks Based on Local Graph Charasteristics", *Information Technology: New Generations*, 738-743, 2012

[11] Petros Drineas, Ravi Kannan, Michael W. Mahoney, "Fast Monte Carlo Algorithms for Matrices I: Approximating Matrix Multiplication", *SIAM Journal on Computing*, 36, 1, 132-157, 2006

[12] 奥村晴彦, "ROC 曲線", http://oku.edu.mie-u.ac.jp/ okumura/stat/ROC.html, 2009