Vol.2022-ICS-206 No.13
2022/3/10

大規模グラフサンプリングと並行したノードID再配置の分割実行によるグラフ演算の高速化

土田 康平[1,a)]　金子 晋丈[2,3,b)]

概要：近年，大規模グラフの高速解析を目的として，サンプリングで得た部分グラフを対象に演算することが一般的である．このとき，サンプリングにより部分グラフのノード ID の連続性が損なわれるため，キャッシュミスによる演算速度の低下が招かれる．部分グラフのノード ID の再配置は効果的だが，既存手法を適用するにはサンプリングの完了を待つ必要があり，大きな時間コストが発生する．そこで，サンプリングと並行して ID 再配置を分割実行する手法として *WalkOrder* を提案する．具体的には，Metropolis-Hastings Random Walk によるサンプリングの過程で新規ノードへの訪問数が閾値を超えるたびに，それらのノード集合から構成される部分グラフで Gorder を踏襲した再配置を行う．実世界のグラフデータセットを用いた実験により，サンプリング後に再配置を行わない場合およびサンプリング後に Gorder を適用した場合の両者と比較し，サンプリング開始時点からグラフ演算終了時点までの総所要時間が減少したことを示した．

Speed Up Graph Processing by Combining Graph Reordering and Sampling from Large Graphs

Kohei Tsuchida[1,a)]　Kunitake Kaneko[2,3,b)]

1. はじめに

近年，実世界の複雑ネットワークをグラフとして解析することへの注目が高まっている．タンパク質の構造特定 [1] やソーシャルネットワーク解析 [2] など多様な分野でグラフ解析が利用されており，これらのグラフは急速に大規模化している [3].

これらの大規模グラフを解析するために様々なグラフ処理系 [4–7] が提案されてきた．DRAM の大容量化などを背景に，単一マシンで大規模グラフの解析が可能となって以降は，単一マシンによるマルチコアを駆使した大規模グラフ解析の高速化について盛んに研究されている [8]．特に近年は，サンプリングによって規模を縮小した部分グラフに対して解析を行う手法が注目されている [9–12].

部分グラフのサンプリングでは，次数分布の Power-Law 性 [13] を筆頭に，全体グラフの特徴を維持することが求められる．一般に，サンプリング手法はランダムサンプリング [9,10] とクローリングベースのサンプリング [11,12] に分類されるが，サンプリングのコストを抑えつつ，Power-Law を維持することが可能な点からクローリングベースの手法である Metropolish-Hastings Random Walks（MHRW）[11] が広く利用されている [14].

サンプリングした部分グラフは全体グラフよりも小規模となる一方で，ノード ID の連続性が損なわれ演算速度の低下を引き起こす．ノード ID が非連続な場合，全ノードの値を格納するためのメモリ領域が不必要に拡大してしまう．その結果，キャッシュの使用効率が低下し，演算途中でのキャッシュミスの増加が引き起こされる．

[1] 慶應義塾大学大学院理工学研究科
Graduate School of Science and Technology, Keio University, Yokohama, Kanagawa, 223–8522, Japan
[2] 慶應義塾大学理工学部
Faculty of Science and Technology, Keio University
[3] 慶應義塾大学デジタルメディア・コンテンツ統合研究センター
Research Institute for Digital Media Content, Keio University
[a)] nora@inl.ics.keio.ac.jp
[b)] kaneko@inl.ics.keio.ac.jp

ID の連続性を保ち，キャッシュの使用効率を向上させることを目的として様々なノード ID 再配置手法が提案されてきた [15–17]．ノード ID の再配置は任意のグラフ演算においてキャッシュの使用効率を向上させるため，グラフ解析の前処理として広く用いられるようになった．ID の再配置によりグラフ演算の高速化が実現された一方で，以下で述べる 3 つの課題が存在する．まず 1 点目に，既存の再配置手法はグラフの全体構造を把握していることを前提としているため，サンプリングで得た部分グラフを対象とする場合，サンプリングの完了を待つための時間コストが発生する．2 点目に，Gorder [15] のようにキャッシュの使用効率が大きく向上する手法は再配置処理に要する計算コストが極めて大きい．3 点目に，既存の再配置手法はマルチコアを使用したグラフ解析に適していない．既存手法により ID を再配置すると，ID をの大小に応じた次数の偏りが発生する．この次数の偏りがコア間の処理量の不均一化を引き起こし，演算速度低下を招く [18, 19]．グラフパーティショニング [20, 21] によるコア間の負荷調整も検討はされてきたが，パーティショニングに要する計算コストが未だに大きな課題となっている [22]．

本稿では，MHRW によるサンプリングと並行して Gorder を踏襲した ID 再配置を分割実行する手法として $WalkOrder$ を提案する．具体的には，MHRW の過程で新規ノードのサンプリング数が閾値を超えるたびに，それらのノード集合から構成される部分グラフで Gorder を踏襲した再配置を行う．

$WalkOrder$ はサンプリング途中での再配置計算およびサンプリング後のマルチコアを使用したグラフ演算の両者で高速化を実現する．サンプリングの途中で構成される部分グラフはサンプリング完了時点での部分グラフに比べて，ノード数および各ノードの次数が小さい．そのため，再配置に要する計算コストが削減され，前者の高速化を実現する．一方で後者は，MHRW による次数分布の維持を利用し，高速化を実現する．MHRW によってサンプリングしたノード集合は全体グラフの次数分布を維持している．そのため，サンプリングの途中で ID を再配置することで，マルチコア演算におけるコア間の処理量を均等化することが可能となる．

本稿の貢献を以下にまとめる．

- サンプリングした部分グラフの非連続な ID に起因する演算速度低下が起こることを明らかにした．
- マルチコアを搭載した単一マシンにおいて，サンプリングと並行して ID 再配置を行うことで再配置処理およびサンプリング後のグラフ演算をともに高速化する手法を提案した．
- 実世界のグラフデータセットを使った実験により，ID 再配置を行わない場合と比べてサンプリング開始時点からグラフ演算終了時点までの処理速度が平均 1.52 倍，最大 1.95 倍減少したことを示した．

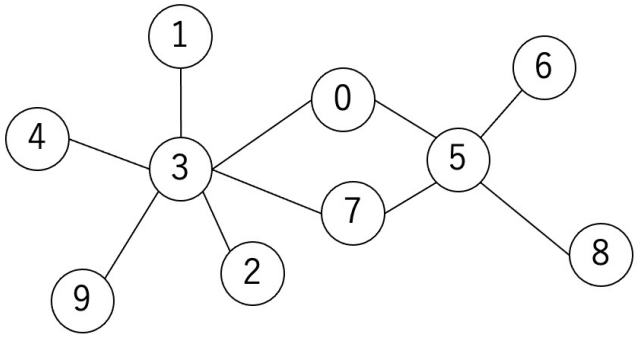


図 1: 重みなしの連結無向グラフ

最後に本稿の構成について述べる．2 章では，本稿と関連研究との相対的な位置づけについて述べ，3 章では，我々の提案手法である $WalkOrder$ について述べる．4 章では，実世界のグラフデータセットを用いた実験により $WalkOrder$ の有用性を示す．最後に 5 章では，本稿の結論および今後の課題を述べる．

なお，本稿で使用する記法の数学的な定義は以下の通りである．

実世界の複雑ネットワークは無向グラフ $G = (V, E)$ として表される．なお，本稿では G をセルフループを含まない重み無しの連結グラフと定義する．図 1 にグラフの例を示す．$V = \{v_0, v_1, ..., v_{n-1}\}$ はノード集合を指し，ノード数 $|V| = n$ である．また，$E = \{(v_i, v_j)|v_i, v_j \in V, i \neq j\}$ はエッジ集合を指し，エッジ数 $|E| = m$ である．さらに，ノード v_i と隣接するノード集合を $N(v_i) = \{v_j \in V|(v_i, v_j) \in E\}$，ノード v_i の次数を $d_i = |N(v_i)|$ と定義する．なお，本稿ではグラフ構造の表現に CSR 形式 [23] を採用している．

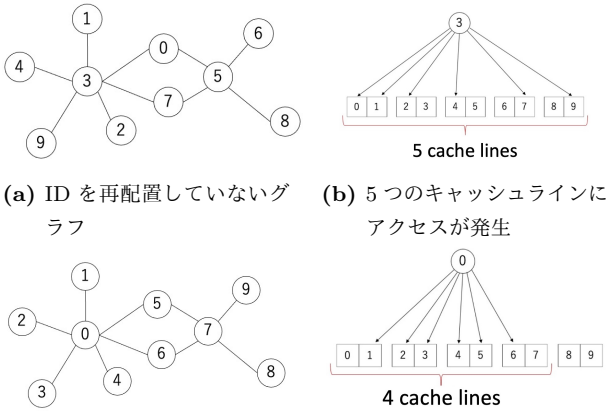
2. 関連研究

2.1 グラフサンプリング

グラフサンプリングはグラフの可視化や特徴量推定など多様な目的のために使用される．特に近年は，全体グラフの特徴を維持した小規模な部分グラフに対して高速な解析を行う目的で使用されている．グラフサンプリング手法はランダムサンプリング [9, 10] とクローリングベースのサンプリング [11, 12] に分類される．部分グラフのサンプリングでは，全体グラフの特徴量の維持がどの程度達成されているかが重要となる．

ランダムサンプリンでは，各手法毎に予め定めた確率に基づいてノード [9,24,25] やエッジ [10] を選択し，それらから部分グラフを構成する．これらの手法は部分グラフの非連結化を招いたり，予めグラフの構造を詳細に把握している必要があるなど適用範囲の狭さが課題視されている [14]．

クローリングベースのサンプリングには，BFS に基づく手法 [26] やランダムウォーク（RW）に基づく手法 [11,12,27] などが存在する．BFS に基づく手法ではサンプリング結果



(a) ID を再配置していないグラフ
(b) 5 つのキャッシュラインにアクセスが発生
(c) ID を再配置したグラフ
(d) アクセス対象のキャッシュライン数が減少

図 2: ID 再配置によるメモリアクセスの局所性が向上

が探索開始ノードの位置に大きく依存するため，安定した特徴量の維持が困難である．一方で，RW に基づくサンプリングでは他のサンプリング手法と比較して，少数のサンプリング数で次数分布を含む多くの特徴量を維持できることが知られている [28, 29]．特に，サンプリングされる確率が全ノードで一様となる Metropolis-Hastings Random Walk（MHRW）[11] は，RW に基づくサンプリング手法の中でも特に次数分布を維持することが可能である．

WalkOrder により再配置後のグラフ演算を高速化させるためには，サンプリンの途中で構成する部分グラフが常に同様の次数分布を持つことが求められる．MHRW によるサンプリングでは，サンプリングした順番に従って部分グラフを構成するだけで各部分グラフの次数分布を揃えることができる．一方で MHRW と比較した場合，他のサンプリング手法は，サンプリングで得た部分グラフが全体グラフの次数分布を維持していたとしても，その部分グラフから同様の次数分布を持つ複数の部分グラフに分割することは困難である．

2.2 ノード ID の再配置

グラフ演算において，各ノードの値はノード ID をアドレスとしてメモリ上へ格納される．そのため．ノード ID を適切に再配置することでメモリアクセスの局所性を向上させることが可能となる．特に，グラフ演算では各ノードが隣接ノードの値を参照することが多いため，ノード間の近接性を考慮した ID の再配置が求められる．図 2a，図 2c にそれぞれ ID 再配置前後のグラフを示す． 1 つのキャッシュラインで 2 ノードの値を格納するとき，図 2b で示すように，ノード 3 が隣接ノードの値を参照するときは 5 つのキャッシュライン全てにアクセスする必要がある．一方で，ID を再配置した場合は，図 2d で示すように，同位置のノード 0 は 4 つのキャッシュラインへアクセスするだけで良く，アクセスの局所性が向上している

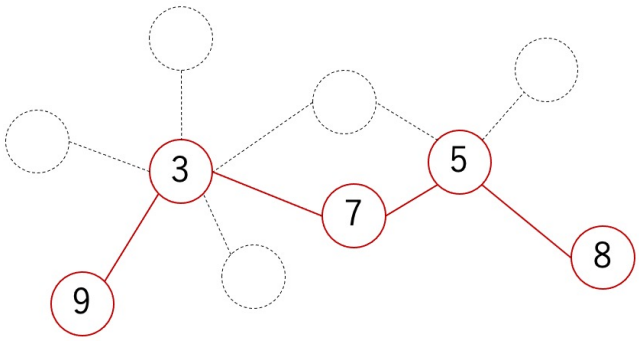


図 3: サンプリングしたノード集合から構成される部分グラフ

ID 再配置によるグラフ演算の高速化という観点では，Gorder [15] が最良の手法である．しかし，Gorder の処理に要する時間計算量は $O(\sum_{u\in V}(d_u)^2)$ と極めて大きいため，より軽量な次数に基づくソートをベースとした再配置手法が提案されている [17, 30, 31]．これらの手法は再配置に要する計算量が減少しているが，Gorder と比べ再配置による演算速度向上は不十分である．

本稿では，MHRW によるサンプリングの途中で部分グラフを構成するが，この部分グラフはサンプリング完了時点のグラフと比較してノード数，エッジ数がともに小さい．そのため，Gorder の時間計算量 $O(\sum_{u\in V}(d_u)^2)$ を減少させることが可能である．

また，通常の ID 再配置処理では ID 再配置に要する時間が全て前処理のオーバーヘッドとみなされる．しかし，サンプリングを伴う状況では，次の再配置対象のノード集合をサンプリングするまでに現在のノード集合に対する再配置処理が完了すればオーバヘッドは発生しない．これらの理由から，*WalkOrder* では [17, 30, 31] で指摘される Gorder の計算コストの償却を実現している．

3. *WalkOrder*

3.1 概要

WalkOrder は MHRW によるサンプリングと並行して Gorder を踏襲した ID 再配置を行う．サンプリングと並行した ID 再配置を行うために，*WalkOrder* は MHRW により L 個の異なる ID を持つノードをサンプリングするとき，L/M 個のサンプリング毎に再配置を行う．ここで，M はサンプリングと並行して再配置を行うスレッド数を指している．MHRW を実行するスレッドは L/M 個のノードをサンプリングした時点で，再配置を行うスレッドにノード集合を送信する．ノード集合を受け取ったスレッドは，それらのノード集合から構成される部分グラフに対して Gorder を踏襲した再配置を行う．図 1 で示す全体グラフにおいて，ノード集合 $V_{sample} = \{3, 5, 7, 8, 9\}$ をサンプリングした場合に構成される部分グラフの例を図 3 に示す．

Algorithm 1 に *WalkOrder* による処理の流れを示す．*WalkOrder* では，L/M 個の全ノード集合が同様の次数分

Algorithm 1 *WalkOrder*

Input: Initial Node: v_{init}, Sampling Number: L, Thread Number: M
Output: Reorder Permutation: $P = \{v_0, v_1, ..., v_L\}$
$P \leftarrow \{\}$
$m \leftarrow 1$
while $m < M$ **do**
  $W \leftarrow$ MHRW-Sampling($v_{init}, L/M$)
  Send W to Reordering Thread m
  $v_{init} \leftarrow$ Last visted node in previous MHRW
  $m \leftarrow m + 1$
end while
for $m \leftarrow 1$ to M **do**
  $P_m \leftarrow$ Reorder Permutation from Thread m
  for $i \leftarrow 0$ to L/M **do**
    Append P_m[i] to P
  end for
end for

布を示すという MHRW によるサンプリングの特徴を利用し，サンプリング後のマルチコアを使用したグラフ演算の高速化を実現している．また，任意のグラフ G における Gorder の計算量は $O(\sum_{u \in V}(d_u)^2)$ である [15]．図 3 で示すように，部分グラフを構成するノード数が少ないほど，各ノードの次数も小さくなる．そのため，再配置のスレッド数 M を増加させることで, 再配置対象のノード数 L/M および再配置に要する計算量を減少させることが可能となる．各スレッドにおいて再配置に要する計算量が大幅に減少することで，[17, 30, 31] で指摘されている Gorder の再配置処理に伴うオーバーヘッドが MHRW によるサンプリング時間内で償却することが可能となる．MHRW によるサンプリングの特徴と再配置処理との関係については 3.2 で詳しく述べる．

また，Gorder は G が有向グラフであることを前提としているため，エッジの向きを区別した処理が必要となる．しかし，*WalkOrder* では G が無向グラフであることを前提としているため，エッジの向きを考慮する必要がなく，処理を簡易化することができる．節 3.3 では，無向グラフを前提とした最適化について述べる．

3.2 MHRW によるサンプリング

MHRW によるサンプリングは始点ノード v_{init} とサンプルするノード数 L を引数にとる．Algorithm 2 に MHRW によるサンプリングの流れを示す．

MHRW によるサンプリングでは，任意のノード v がサンプリングされる確率 π_v が $\pi_v = \frac{1}{|V|}$ となり，定常分布が一様分布に収束する．そのため，サンプリングした L 個のノード集合 $V_{sample} = \{v_0, v_1, ..., v_{L-1}\}$ は全体グラフの次数分布を維持している．*WalkOrder* では，$V_{sample} = \{v_0, v_1, ..., v_{L-1}\}$ を M 個のノード集合に分割するが，定常分布が一様分布に収束する性質は M 個の各ノード集合にも当てはまる．つまり，*WalkOrder* による再配置は，サンプリングしたグラフを同様の次数分布を維持した M 個の部分グラフに分割し，それぞれの部分グラフで ID 再配置を実行することに等しい．

Algorithm 2 MHRW-Sampling

Input: Initial Node: v_{init}, Sampling Number: L
Output: Sampled Nodes: $W = \{v_0, v_1, ..., v_{L-1}\}$
$W \leftarrow \{v_{init}\}$
$v_c \leftarrow v_{init}$
while $|W| < L$ **do**
  $v_n \leftarrow$ select a node uniformly at random from $N(v_c)$
  $p \leftarrow$ generate a random number that meets $0 \leq p \leq 1$
  if $p < \frac{d_c}{d_n}$ and visits v_n for the first time **then**
    Append v_n to W
    $v_c \leftarrow v_n$
  else
    $v_c \leftarrow v_c$
  end if
end while

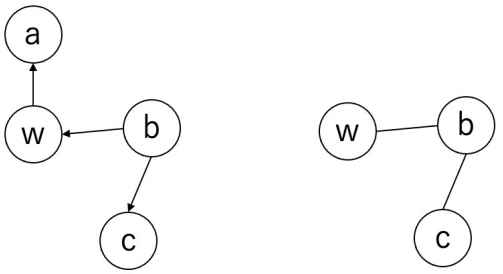
次数分布を維持した部分グラフに分割し，マルチコアを使用したグラフ演算を高速化させることは検討されてきた [19–21]．しかし，分割に要する計算コストをがボトルネックであった．*WalkOrder* では，MHRW によるサンプリングの特徴を利用することで，これら既存手法のボトルネックを解消している．

3.3 サンプリングしたノード集合に対する ID 再配置

WalkOrder では，まず MHRW を実行しているスレッドからノード集合 W を受け取る．そして，受け取ったノード集合 W から部分グラフ G_{sub} を構成する．G_{sub} のエッジ集合 E_{sub} は [9, 24] などと同様に $E_{sub} = \{(v_i, v_j) | v_i, v_j \in W, (v_i, v_j) \in E, i \neq j\}$ を満たすように構成する．

Gorder [15] では，再配置が完了していないノード集合 V_{wait} に含まれる全ノード v に対して，再配置が完了したノード集合 V_{finish} とのグラフ構造に基づく近接度合いのスコアリングを行う．特に，有向グラフを前提とした Gorder では，図 4a で示すように，ノード $w \in V_{finish}$ に対して，a，b，c の 3 種類のノードがスコアリング対象となる．そして，V_{wait} から最大のスコアを持つノードを V_{finish} に加え，再びスコアリングを行うことで全体の処理を進める．CSR 形式によるグラフ表現では w からエッジを受けているノード集合と w にエッジを張っているノード集合を区別している．そのため，a, b, c のそれぞれについて隣接リストの走査が必要となる．

WalkOrder でもこの処理を踏襲するが，G_{sub} が無向グラフであるという特徴を利用した最適化を施す．無向グラフはエッジの向きに区別が無いため，図 4b で示すように $w \in V_{finish}$ に対して b, c の 2 種類のノードのみがスコアリング対象となる．そのため，図 4a で示すような有向グラフにおけるスコアリングと比べて隣接リストの走査が短縮される．



(a) 有向グラフにおけるスコアリング対象のノードの種類　**(b)** 無向グラフにおけるスコアリング対象のノードの種類

図 4: 無向グラフではスコアリング対象のノードの走査が短縮される

Algorithm 3 SubGraph Reordering

Input: Reorder Nodes: W
Output: SubGraph Permutation: $P_{sub} = \{v_0, v_1, ..., v_{L/M}\}$
1: $G_{sub} \leftarrow$ Connecting edges between $v \in W$
2: $P_{sub} \leftarrow OptimizedGorder(W)$

サンプリングしたノード集合に対して再配置を実行するスレッドにおける処理の流れを Algorithm 3 に示す．なお，2 行目で示されている *OptimizedGorder* は無向グラフにおける最適化を施した再配置処理である．

4. 実験・評価

本章では，サンプリング開始時点からグラフ演算終了時点までの総所要時間という観点から $WalkOrder$ の効果を評価した．本実験では，代表的なグラフ演算である PageRank [32] を Ligra [6] で実行した場合の実行時間を測定した．比較対象として，サンプリングしたまま ID の再配置を行わない *Original*，サンプリング完了後に Gorder を適用する $Naive$ を採用した．$Naive$ はサンプリングで得た部分グラフのノード ID を一度でまとめて再配置するため，$Naive$ の結果は $WalkOrder$ を M=1 で実行した場合の結果に等しい．なお，グラフ上で最高次数のノードを MHRW の開始点とし，全ノード数の 20 % をサンプリングするという設定で全ての実験を行った．

全てのアルゴリズム，データ構造は C++を用いて実装し，これらの実装を G++ 9.3.0 により O3 最適化を施した上でコンパイルした．また，並行処理に関わる処理は OpenMPI を用いて実現した．全ての実験は Intel Xeon Gold 6348 @ 2.60GHz を 2 個，メインメモリ 1TB 搭載した Linux サーバ上で行っており，OS のバージョンは Ubuntu 20.04.3 であった．使用した CPU は L1 キャッシュ 4.4 MiB，L2 キャッシュ 70 MiB，L3 キャッシュ 84 MiB を搭載しており，最大で 56 スレッドの並行処理が可能である．なお，再配置後の PageRank 演算は 56 スレッドを使用した並列処理を行っている．

本実験では，SNAP ライブラリ [33] で提供されている 3 種類の実世界グラフデータを使用した．表 1 に使用したデータセットを示す．なお，全ての実験を 50 回を行い，その平均値を実験結果として採用している．

表 1: 使用したデータセット

Name	$\|V\|$	$\|E\|$
facebook [34]	4,039	88,234
roadNet [35]	1,088,092	1,541,898
dblp [36]	317,080	1,049,866

4.1 総所要時間の内訳

図 5 に M を変化させたときの総所要時間とその内訳を示す．なお，総所要時間を (1) PageRank 演算に要する時間，(2) MHRW によるサンプリングに要する時間，(3) サンプリング完了時点から全スレッドの再配置処理が終了するまでに要する時間の 3 項目に分け内訳として示している．

サンプリングに要する時間に対して，サンプリング完了時点から全スレッドの再配置処理が終了するまでに要する時間の比率が大きいほど再配置処理のオーバヘッドが大きいことを意味する．全てのデータセットにおいて $Naive$ のときにこの比率が最大となり facebook で 6.02，roadNet で 0.48，dblp で 1.54 の値を示した．$WalkOrder$ では M の増加によりこの比率を減少させており，facebook で最大 4.35，roadNet で最大 0.054，dblp で最大 0.069 まで減少した．

また，図 6 に PageRank 演算のみに要する時間を示す．roadNet，dblp では M の増加により再配置処理のオーバヘッドが減少することに加えて，マルチコアを使用した PageRank 演算の演算時間も減少した．特にこれらのデータセットでは，再配置を実行するスレッドと PageRank 演算を実行するスレッド数が一致する M=56 のとき PageRank 演算に要する時間が最小となった．これは [19] と同様に，$WalkOrder$ がマルチコアを使用したグラフ演算においてコア間の処理量が均等化するように ID を再配置していることを示している．一方で，facebook では M の値と PageRank 演算に要する時間に明確な相関関係は現れなかった．これは [17] で指摘されているように，グラフが比較的小規模な場合，ID 再配置の効果が弱まることを示している．

4.2 *Original*, $Naive$ と比較した場合の速度向上

図 7，図 8 にそれぞれ *Original*，$Naive$ と比較し，再配置を行うスレッド数 M を変化させた場合の $WalkOrder$ による速度向上の結果を示す．

まず，*Original* では，facebook で平均 1.88 倍/最高 1.95 倍，roadNet で平均 1.41 倍/最高 1.49 倍，dblp で平均 1.28 倍/最高 1.34 倍速度が向上した．全てのグラフにおいて

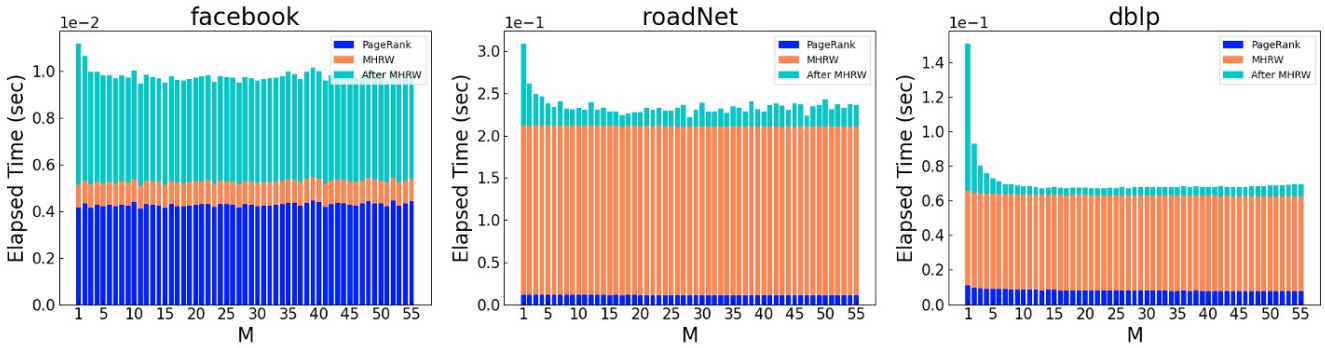


図 5: 総所要時間（sec）

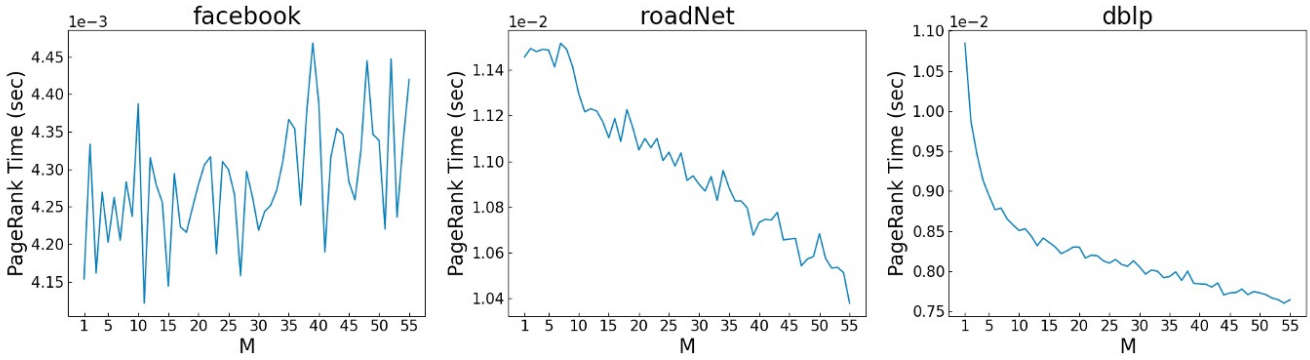


図 6: PageRank 演算に要する時間（sec）

M=1 から M=10 にかけて急速に速度が向上するが，M=10 以降は再配置を実行するスレッド数の増加による速度向上への効果が得られないことが判明した．これは，M の増加による再配置処理におけるコスト減少の効果よりも同期期処理におけるコスト増加の効果が上回ることが原因である．*WalkOrder* では，M の増加により，再配置対象のノード数および各ノードの次数を減少させることで再配置の計算コストを減少させている．M=1 から M=10 における M の増加では，ノード数と各ノードの次数が大きく減少するため，再配置処理に要する時間の減少も大きい．しかし，M=10 以降は，ノード数および各ノードの次数の減少に伴う計算量減少の効果が収束する．一方で，*WalkOrder* は全 M スレッドが再配置処理を完了した段階で同期処理を行うため，M が増加するほど同期処理対象のスレッド数が増加する．特に M が大きい範囲では各スレッドにおける計算量減少の効果に対して，同期処理コストにおけるコスト増加の効果大きく上回っている．

また，facebook，roadNet では *Naive* のとき *Original* より速度が向上しているのに対して，dblp では *Original* よりも速度が低下している．これは，dblp が facebook，roadNet と異なりノード数および平均次数の両者が大きいため，再配置処理に要するコストも大きくなることが原因である．このように，*Naive* は再配置処理に要するコストが全体グラフの構造的な特徴によって決まる．一方で，*WalkOrder* は facebook，roadNet，dblp の全てで速度向上を実現したことから，対象のグラフデータに依存せず速度向上を実現することが可能である．これは，本実験で使用したソーシャルネットワーク，道路網ネットワーク，共著関係のグラフデータに限らず幅広い種類のグラフデータに対して *WalkOrder* が有効であることを示している．

Naive と比較した場合，facebook で平均 1.27 倍/最高 1.31 倍，roadNet で平均 1.32 倍/最高 1.41 倍，dblp で平均 2.27 倍/最高 2.38 倍速度が向上した．dblp のように *Naive* のとき *Original* より速度が低下するグラフデータに対して，*WalkOrder* は特に効果を発揮しているが，これは，dblp と比較してノード数および平均次数がより大きいグラフデータに対して *WalkOrder* がより効果的であることを示している．

5. おわりに

本稿では，MHRW によるサンプリングと並行して Gorder を踏襲したノード ID の再配置を分割実行する手法として *WalkOrder* を提案した．*WalkOrder* は MHRW により次数分布を維持してサンプリングしたノード集合からノード数および各ノードの次数が小さい部分グラフを構成する．この部分グラフに対して ID 再配置を実行することで，再配置処理と再配置後のマルチコアを使用したグラフ演算の両者で高速化を実現した．実世界のグラフデータを用いた実験により，サンプリング完了後に ID 再配置を行わない場合と比べて，サンプリング開始時点からグラフ演算終了

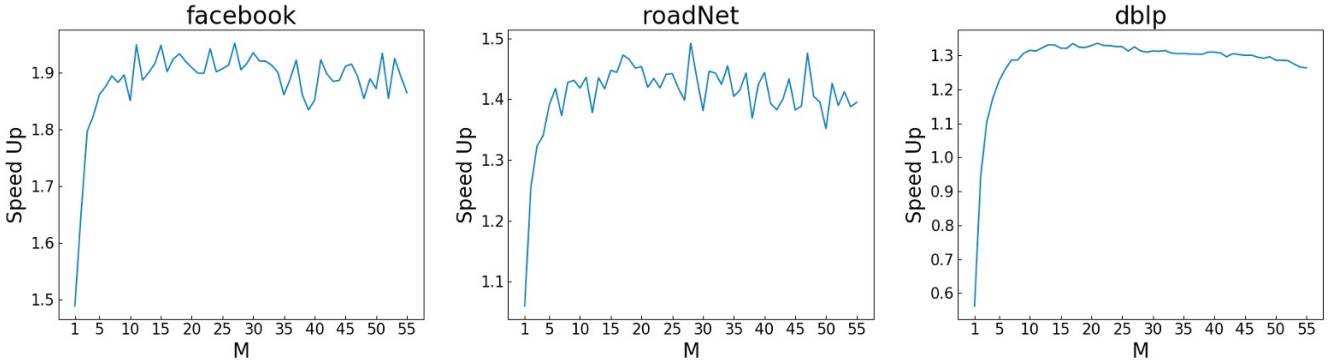


図 7: *Original* と比較した場合の速度向上

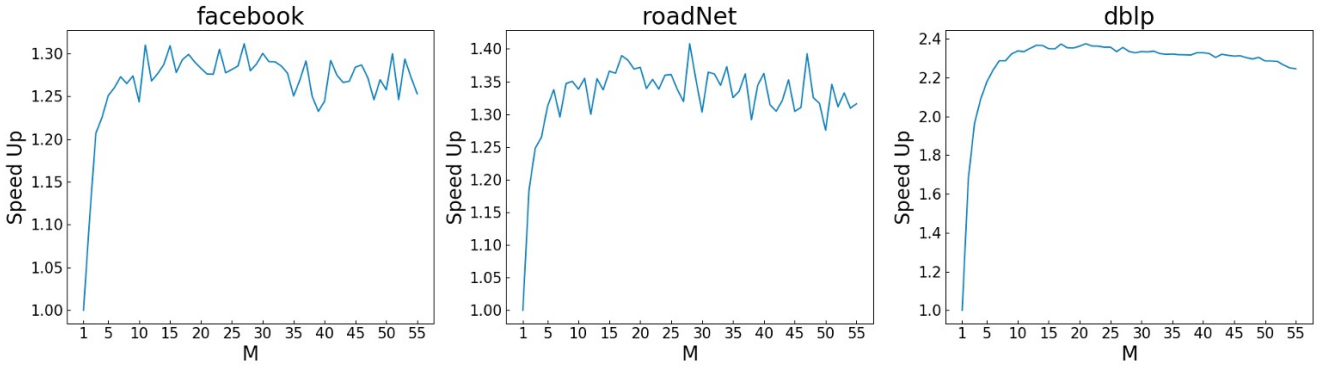


図 8: *Naive* と比較した場合の速度向上

時点までの処理速度が平均 1.52 倍，最大 1.95 倍向上した．

今後の課題について述べる．まず 1 点目は，より大規模かつ多様なカテゴリーのグラフデータに対して *WalkOrder* が効果を発揮するか検証することである．本稿で行った実験では全てのグラフデータにおいて *WalkOrder* を適用したことによる処理速度の向上を確認した．*WalkOrder* の汎用性を示すために，本稿で使用したグラフデータ以外の様々なグラフデータに対して *WalkOrder* を適用し，効果を検証することが必要である．次に 2 点目は，ID 再配置後の演算速度を保ったまま，全スレッドが再配置処理を完了するまでの時間を短縮する方法について検討することである．*WalkOrder* は再配置処理を行う全スレッドが処理を完了するタイミングで同期処理を行う．そのため，サンプリングの前半で再配置を行うスレッドは，サンプリングの後半で再配置を行うスレッドの完了を待つ必要がある．そこで，サンプリングの後半で再配置を行うスレッドを意図的に早く終わらせながらも，再配置後の演算速度を保つ最適化手法を組み込むことで *WalkOrder* のさらなる速度向上を目指す．

参考文献

[1] N Kannan and S Vishveshwara. Identification of side-chain clusters in protein structures by a graph spectral method. *Journal of molecular biology*, Vol. 292, No. 2, pp. 441–464, 1999.

[2] Peter J Carrington, John Scott, and Stanley Wasserman. *Models and methods in social network analysis*, Vol. 28. Cambridge university press, 2005.

[3] Avery Ching, Sergey Edunov, Maja Kabiljo, Dionysios Logothetis, and Sambavi Muthukrishnan. One trillion edges: Graph processing at facebook-scale. *In Proceedings of the VLDB Endowment*, Vol. 8, No. 12, pp. 1804–1815, 2015.

[4] Joseph E Gonzalez, Reynold S Xin, Ankur Dave, Daniel Crankshaw, Michael J Franklin, and Ion Stoica. Graphx: Graph processing in a distributed dataflow framework. In *Proceedings of the 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, pp. 599–613, 2014.

[5] Yucheng Low, Joseph Gonzalez, Aapo Kyrola, Danny Bickson, Carlos Guestrin, and Joseph M Hellerstein. Distributed graphlab: A framework for machine learning and data mining in the cloud. *In Proceedings of the VLDB Endowment*, Vol. 5, No. 8, 2012.

[6] Julian Shun and Guy E Blelloch. Ligra: a lightweight graph processing framework for shared memory. In *Proceedings of the 18th ACM SIGPLAN symposium on Principles and practice of parallel programming*, pp. 135–146, 2013.

[7] Aapo Kyrola, Guy Blelloch, and Carlos Guestrin. Graphchi: Large-scale graph computation on just a {PC}. In *Proceedings of the 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12)*, pp. 31–46, 2012.

[8] Frank McSherry, Michael Isard, and Derek G Murray. Scalability! but at what COST? In *Proceedings of the 15th Workshop on Hot Topics in Operating Systems (HotOS XV)*, 2015.

[9] Elli Voudigari, Nikos Salamanos, Theodore Papageorgiou, and Emmanuel J Yannakoudakis. Rank degree: An efficient algorithm for graph sampling. In *Proceedings of the 2016 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, pp. 120–129. IEEE, 2016.

[10] Ruohan Gao, Huanle Xu, Pili Hu, and Wing Cheong Lau. Accelerating graph mining algorithms via uniform random edge sampling. In *Proceedings of the 2016 IEEE International Conference on Communications (ICC)*, pp. 1–6. IEEE, 2016.

[11] Minas Gjoka, Maciej Kurant, Carter T Butts, and Athina Markopoulou. Walking in facebook: A case study of unbiased sampling of osns. In *Proceedings of the 2010 IEEE Infocom*, pp. 1–9. Ieee, 2010.

[12] Bruno Ribeiro and Don Towsley. Estimating and sampling graphs with multidimensional random walks. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, pp. 390–403, 2010.

[13] Michalis Faloutsos, Petros Faloutsos, and Christos Faloutsos. On power-law relationships of the internet topology. *ACM SIGCOMM computer communication review*, Vol. 29, No. 4, pp. 251–262, 1999.

[14] Rong-Hua Li, Jeffrey Xu Yu, Lu Qin, Rui Mao, and Tan Jin. On random walk based graph sampling. In *Proceedings of the 2015 IEEE 31st international conference on data engineering*, pp. 927–938. IEEE, 2015.

[15] Hao Wei, Jeffrey Xu Yu, Can Lu, and Xuemin Lin. Speedup graph processing by graph ordering. In *Proceedings of the 2016 International Conference on Management of Data*, pp. 1813–1828, 2016.

[16] Junya Arai, Hiroaki Shiokawa, Takeshi Yamamuro, Makoto Onizuka, and Sotetsu Iwamura. Rabbit order: Just-in-time parallel reordering for fast graph analysis. In *Proceedings of the 2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pp. 22–31. IEEE, 2016.

[17] Vignesh Balaji and Brandon Lucia. When is graph reordering an optimization? studying the effect of lightweight graph reordering across applications and input graphs. In *Proceedings of the 2018 IEEE International Symposium on Workload Characterization (IISWC)*, pp. 203–214. IEEE, 2018.

[18] Baofu Huang, Zhidan Liu, and Kaishun Wu. Structure preserved graph reordering for fast graph processing without the pain. In *Proceedings of the 2020 IEEE 22nd International Conference on High Performance Computing and Communications (HPCC)*, pp. 44–51. IEEE, 2020.

[19] YuAng Chen and Yeh-Ching Chung. Workload balancing via graph reordering on multicore systems. *IEEE Transactions on Parallel and Distributed Systems*, Vol. 33, No. 5, pp. 1231–1245, 2021.

[20] George Karypis and Vipin Kumar. Metis: A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices. 1997.

[21] Kartik Lakhotia, Rajgopal Kannan, Sourav Pati, and Viktor Prasanna. Gpop: A scalable cache-and memory-efficient framework for graph processing over parts. *ACM Transactions on Parallel Computing (TOPC)*, Vol. 7, No. 1, pp. 1–24, 2020.

[22] Aydın Buluç, Henning Meyerhenke, Ilya Safro, Peter Sanders, and Christian Schulz. Recent advances in graph partitioning. *Algorithm engineering*, pp. 117–158, 2016.

[23] Jeremy Siek, Lie-Quan Lee, Andrew Lumsdaine, et al. *The boost graph library*, Vol. 243. Pearson India, 2002.

[24] Tianyi Wang, Yang Chen, Zengbin Zhang, Tianyin Xu, Long Jin, Pan Hui, Beixing Deng, and Xing Li. Understanding graph sampling algorithms for social network analysis. In *Proceedings of the 2011 31st international conference on distributed computing systems workshops*, pp. 123–128. IEEE, 2011.

[25] Jie Chen, Tengfei Ma, and Cao Xiao. Fastgcn: Fast learning with graph convolutional networks via importance sampling. In *Proceedings of the International Conference on Learning Representations*, 2018.

[26] Maciej Kurant, Athina Markopoulou, and Patrick Thiran. Towards unbiased bfs sampling. *IEEE Journal on Selected Areas in Communications*, Vol. 29, No. 9, pp. 1799–1809, 2011.

[27] Bruno Ribeiro, Pinghui Wang, Fabricio Murai, and Don Towsley. Sampling directed graphs with random walks. In *Proceedings of the 2012 IEEE INFOCOM*, pp. 1692–1700. IEEE, 2012.

[28] Arun S Maiya and Tanya Y Berger-Wolf. Benefits of bias: Towards better characterization of network sampling. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 105–113, 2011.

[29] Pili Hu and Wing Cheong Lau. A survey and taxonomy of graph sampling. *arXiv preprint arXiv:1308.5865*, 2013.

[30] Priyank Faldu, Jeff Diamond, and Boris Grot. A closer look at lightweight graph reordering. In *Proceedings of the 2019 IEEE International Symposium on Workload Characterization (IISWC)*, pp. 1–13. IEEE, 2019.

[31] Yunming Zhang, Vladimir Kiriansky, Charith Mendis, Saman Amarasinghe, and Matei Zaharia. Making caches work for graph analytics. In *Proceedings of the 2017 IEEE International Conference on Big Data (Big Data)*, pp. 293–302. IEEE, 2017.

[32] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab, 1999.

[33] Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection, June 2014.

[34] Julian J McAuley and Jure Leskovec. Learning to discover social circles in ego networks. In *Proceedings of the NIPS*, Vol. 2012, pp. 548–56. Citeseer, 2012.

[35] J Leskovec, KJ Lang, A Dasgupta, and MW Mahoney. Community structure in large networks: natural cluster sizes and the absence of large well-defined clusters. internet math. 6 (1), 29–123 (2009). *Advances in Neural Information Processing Systems*, pp. 539–547, 2012.

[36] Jaewon Yang and Jure Leskovec. Defining and evaluating network communities based on ground-truth. *Knowledge and Information Systems*, Vol. 42, No. 1, pp. 181–213, 2015.