B-014

GPU による看板配置問題の効率的並列計算

A Work-Efficient Parallelized Calculation of K-Sign Location Problems Using GPU

大石 真生 † Mao Oishi　　**渡邉 貴之** † Takayuki Watanabe

1. はじめに

看板配置問題とは, 道路網上に看板を配置する際に少数でより多くの人の目に留まる効果的な配置場所を求める問題である. この問題は, 文献 [1] で提案されている集合媒介中心性を配置の有効性の尺度として用いることで解くことができる. 集合媒介中心性とは, 単独のノードに着目した媒介中心性を, 集合としてのノード群の協調的振る舞いに拡張した中心性である. 先行研究 [2] では, 集合媒介中心性に基づく看板配置問題の近似解を GPU による edge-based の並列処理によって求めるアルゴリズムを提案している. しかし, スケールフリー性を持たず平面的で直径が大きい道路網に対しては, 並列処理による高速化の効果が得られていない. 本研究では, 不要なメモリアクセスや分岐を削減するために文献 [3] において提案されている work-efficient 技法を取り入れてアルゴリズムの効率化を行う. work-efficient 技法は,vertex-based の並列処理に基づく媒介中心性の計算を高速化するための技法である. 本研究では,edge-based, vertex-based 及び work-efficient 技法付き vertex-based による計算時間について比較する.

2. 媒介中心性と集合媒介中心性

まず, ノード集合 V, リンク集合 E からなる無向ネットワーク $G = (V, E)$ が与えられたとき, ノード $v \in V$ の媒介中心性 $BC(v)$ は以下 のように定義される.

$$BC(v) = \sum_{s \in V} \sum_{t \in V} \frac{\sigma_{s,t}(v)}{\sigma_{s,t}} \tag{1}$$

このとき,$\sigma_{s,t}$ はノード s, t 間の最短パス数を表しており,$\sigma_{s,t}(v)$ はノード s, t 間の最短パスの中でノード v を通るパス数である. 式 (1) から, 多くのノード間の橋渡しをしているノードほど, 媒介中心性の値は高くなる. ここで, 媒介中心性を尺度として看板配置問題を解くことを考える. 単純に媒介中心性の上位から K 個のノードを選択した場合, 選択されるノードは他の上位のノードが媒介するパス上に同様に媒介する可能性が高く, 言い換えれば同じ移動者が複数回同じ看板を見る可能性が高くなる.

次に, ノード集合 R に対する集合媒介中心性 $SB(R)$ は以下のように定義される.

$$SB(R) = \sum_{s \in V} \sum_{t \in V} \frac{\sigma_{s,t}(R)}{\sigma_{s,t}} \tag{2}$$

ここで,$\sigma_{s,t}(R)$ は, ノード s, t 間の最短パスの中で集合 R に含まれるノード $r \in R$ を通るパス数である. 従って, 最短パスが集合 R に含まれるノードを何度通過したとしても通過回数は 1 回と算定される. 仮に集合の要素数 K が $|R| = K = 1$ であれば, 集合媒介中心性の値は媒介中心性の値と一致する. しかし,$K > 1$ では上位のノードが媒介するパスが除外されて媒介度が評価されるため, 異なる移動者が 1 度でも看板を見る可能性が高いノードを選定することができる. 結果として, 既存の媒介中心性の上位 K 個のノード群と, 集合媒介中心性を最大化する K 個のノード群は必ずしも一致しないことになる.

†静岡県立大学経営情報イノベーション研究科, Graduate School of Management and Informatics of innovation, University of Shizuoka

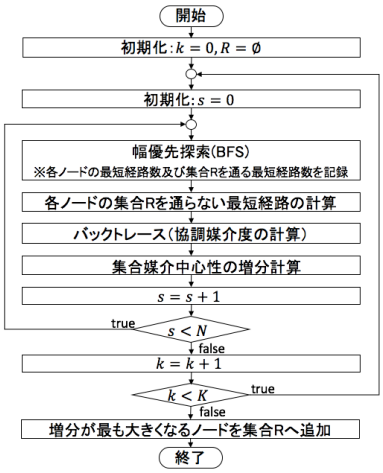


図 1: 貪欲法による解法アルゴリズム

3. 看板配置問題の貪欲法による解法

集合媒介中心性を尺度として看板配置問題の厳密解を求める問題は一般に NP 完全であり, 文献 [1] では貪欲法による近似解法が提案されている. 図 1 に示す手順では, 集合 R にノードを 1 つ追加する際, 集合媒介中心性の増分が最も高くなるノードを選定する処理を K 回繰り返している. 集合媒介中心性の増分計算は以下の式により行う.

$$\begin{aligned} &SB(R \cup \{v\}) - SB(R) \\ &= \sum_{s \in V} \sum_{t \in V} \frac{\sigma_{s,t}(v; R)}{\sigma_{s,t}} = \sum_{s \in V} \delta_s(v; R) \end{aligned} \tag{3}$$

ここで, $\sigma_{s,t}(v; R)$ は, s から t までのパスの中で, v を通過し, R 内のどのノードも通らないパス数である. また, $\delta_s(v; R)$ は, v の s を起点とした集合 R に対する協調媒介度であり,

$$\begin{aligned} &\delta_s(v; R) \\ &= \sum_{w \in C_s(v;R)} \left\{ \frac{\sigma_{s,v}(v; R)}{\sigma_{s,w}} + \frac{\sigma_{s,v}(v; R)}{\sigma_{s,w}(w; R)} \delta_s(w; R) \right\} \end{aligned} \tag{4}$$

となる. ただし, $C_s(v; R)$ は, s を起点とし R 内のどのノードも通らない非巡回有向グラフにおける v の子ノード集合である. 式 (4) から, v の協調媒介度 $\delta_s(v; R)$ と, その子ノード w の協調媒介度 $\delta_s(w; R)$ の間には, 再帰的な関係があることが分かる. 従って, 起点 s から幅優先探索によって各ノードへの最短経路数と集合 R 内のノードを通らない経路数を計算し, その後, 探索した経路を逆にたどる（バックトレースする）ことで再帰的に各ノードの協調媒介度を計算することができる.

4.GPU による看板配置問題の並列計算

本章では, 第 3 章で紹介した看板配置問題の貪欲解法を対象に, 先行研究 [2] における GPU（NVIDIA 社の GPU で採用されている CUDA 環境）を用いた並列化とその改良の詳細について述べる.

4.1. 先行研究における GPU を用いた並列化

先行研究では, 図 1 の開始から終了までの各処理を GPU 側で実行される複数の kernel 関数として実装している. 例えば, 幅優先探索の kernel 関数 (bfs_kernel) の実装例を図 2(a) に示す. この実装では 1 つの thread に 1 つのリンクの処理を割り当てており,edge-based の並列処理と言える. 従って,kernel 関数が実行される総 thread 数はネットワーク内の総リンク数 $M = |E|$ となる.

幅優先探索は Level-Synchronized に則って, 起点 s から深さを 1 つずつ増加させながら探索を遠方へ前進させていく. CUDA では, 同一 block 内での thread 間の同期は可能だが, 異なる block 間での同期は一旦 kernel 関数を終了する必要がある. そのため, 図 2(a) の実装例では, 深さが増すたびに kernel 関数を終了し同期を行っており, 直径の大きいネットワークでは kernel 関数呼び出しが増加し, オーバーヘッドは無視できないものとなる. また, 幅優先探索以外の各処理も各 block 間での同期を取るために, 個別の kernel 関数として実装されており, 並列化の効果が得られ難い実装となっている.

さらに, スケールフリー性を持たず平面的な道路網では, 同一の深さで処理の対象となるリンクは数百程度と GPU 内のコア数よりもかなり少ない. しかし, 図 3 の edge-based に示すように処理の対象となっていない（探索済み, またはより遠方の）リンクについても thread に割り当てられているため, 無駄の多い実装といえる. 図 2(a) において, 1 つの thread に 1 つのノードを割り当てる vertex-based の並列化も可能であるが, 同様の問題が生じることは明らかである.

4.2.work-efficient 技法を用いた改良

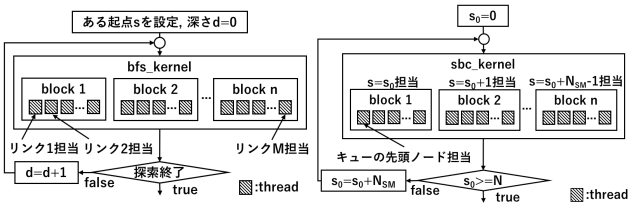
本研究におけるアルゴリズムの効率化について述べる. まず, 先行研究では同一ノードを起点とする幅優先探索等の処理を複数の block に分割して行っていたため,block 間での同期が必要となり kernel 関数の粒度も細い. そのため本研究では, 図 2(b) に示すように,1 つの起点ノードに対する協調媒介度の計算処理の全て（幅優先探索から集合媒介中心性の増分計算まで）を 1 つの block に割り当てる. これによって block 間の同期が不要となり, 全ての処理を単一の kernel 関数 (sbc_kernel) に集約できる.

一方,block 毎に異なる起点ノードに対する幅優先探索等の処理を実行するためには, ノードの深さや最短経路数などのメモリを block 毎に個別に確保しておく必要がある. 従って,block の総数を起点となるノード数 $N = |V|$ に設定することは, 限られたメモリ容量から鑑みて現実的でない. そのため本研究では,1 回の kernel 関数呼び出しで起動する block 数を GPU の SM(Streaming Multiprocessor) 数 N_{SM} とし,N_{SM} 単位で kernel 関数を実行する方式とした.

次に,kernel 関数内の幅優先探索及びバックトレース部分では work-efficient 技法 [3] に基づく vertex-based の並列処理を導入する. work-efficient 技法は媒介中心性の計算を効率化するために提案された技法であるが, 集合媒介中心性の計算過程にも適用可能である. 図 3 に示すように,work-efficient 技法は探索の最前線以外のノードを処理の対象から除外することで,Level-Synchronized の過程を効率化する技法である. これは, 次に探索する (最前線の) ノードを記憶しておくための明示的なキューを用意することで実装できる.

5. 評価実験と考察

本研究で改良を行った看板配置問題の並列計算処理性能を評価する. 評価実験の実行環境は,CPU: Intel Core-i7-3930K,RAM: 64GB,OS: Cent OS 6.8（64bit）である. また,GPU は NVIDIA 社の GTX1080 8GB RAM を使用し,Compiler は NVCC8.0RC を使用した. 使用したデータセットは, Shizuoka 道路網ネットワーク (ノード数 : 31, 041 , リンク数 : 87, 216) と,Twitter ユーザ関係性ネットワーク (ノード数 : 9, 481, リンク数 : 245, 044) の 2 つである.



(a) edge-based　　(b) work-efficient

図 2: 各手法毎の kernel 関数実装の比較

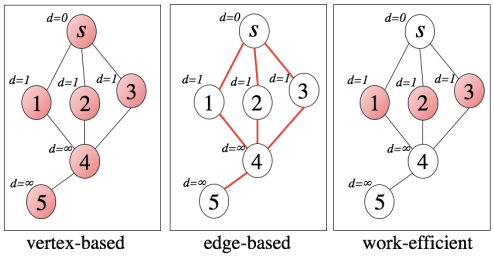


図 3: 各手法毎の thread 割り当ての比較（s は起点ノード, 赤色は深さ 1 で thread に割り当てられるノードまたはリンク）

ノード数 K を増加させた際に要した計算時間について図 3 に示す. まず,Shizuoka ネットワークでは work-efficient 技法による処理が最も高速であり,edge-based に比較して約 7.8 倍高速であった. 次に,Twitter ネットワークでは,work-efficient 技法は vertex-based に比較すると大幅に高速化されているが,edge-based と比較すると約 0.6 倍の計算速度となった. これは,Twitter ネットワークは直径が小さく, 同一の深さを持つノード数が数千ノードと GPU のコア数以上となり,edge-based に有利なネットワーク構造だったことが原因である.

6. まとめ

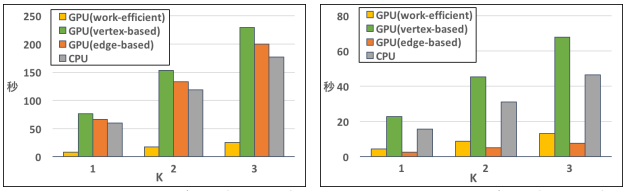
本研究では,GPU による看板配置問題の求解アルゴリズムの work-efficient 技法による効率化を試みた. スケールフリー性を持たない直径の大きい道路網ネットワークにおいて,GPU による並列計算によって高速化の効果が得られることを示した.

参考文献

[1] 伏見, 斉藤, 池田, 武藤, “ノード群の協調的振舞いに着目した集合媒介中心性の提案と応用”, 電子情報通信学会論文誌 D. Vol.J96-D, No.5, pp.1158–1165, 2013 年 5 月.

[2] 赤池, 大久保, 武藤, 斉藤, 渡邉, “GPU による集合媒介中心性に基づく看板配置問題の並列計算”, 第 12 回情報学ワークショップ, 2014 年 11 月.

[3] A.McLaughlin and D.A.Bader, “Scalable and high performance betweenness centrality on the GPU,” in Proc. on *IEEE SC14: International Conference for High Performance Computing, Networking, Storage and Analysis*, pp.572–583, Nov. 2014.



(a) Shizuoka ネットワーク　　(b) Twitter ネットワーク

図 4: K を増加させた際の計算時間の比較