7M-04

GPU を用いた 5 ノードサブグラフ数え上げの高速化

菅波 柊也 † 天笠 俊之 †† 北川 博之 ††

† 筑波大学 情報学群 情報科学類 †† 筑波大学 計算科学研究センター

1 序論

グラフの部分構造であるサブグラフを数え上げることはネットワーク分析の基本的な手法であり，バイオインフォマティクスなどの分野で用いられている．しかしサブグラフの数え上げはその実用性にも関わらず，既存の多くのアルゴリズムは実行に多くの時間を要するという問題点が存在する．この問題は特に 5 ノード以上のサブグラフの数え上げに対して顕著に現れる．5 ノードまでのサブグラフの数え上げを行う手法の中で高速なものとして Pinar らによる ESCAPE[1] がある．しかし ESCAPE はグラフによっては探索に数時間から数十時間を要するため，より高速に 5 ノードまでサブグラフの数え上げを行う事が必要である．

そこで本稿では，GPU を用いた 5 ノードまでのサブグラフの数え上げを高速化手法を提案する．提案手法では ESCAPE のアイデアに基づいた数え上げのアルゴリズムを GPU を用いて並列化することにより，5 ノードまでのサブグラフの数え上げを高速化する．本稿では実世界のグラフデータセットを用いて既存手法と比較実験を行なった．その結果我々の手法は 5 ノードまでのサブグラフの数え上げを行う state-of-the-art な手法に比べ，約 3 倍から 5 倍の高速化を達成した．

2 関連研究

5 ノードまでのサブグラフの数え上げを行う手法として pinar らによる ESCAPE[1] がある．ESCAPE は 5 ノードまでのサブグラフの数え上げる state-of-the-art な手法であり，4 ノードのサブグラフの数え上げにおいて，PGD よりも高速に数え上げを行うことができる．

GPU を用いてサブグラフの数え上げをアルゴリズムとして Rossi らによる手法 [2] がある．その手法ではグラフ全体として各 4 ノードのサブグラフが何回出現するかだけでなく，各エッジごとに各サブグラフが何回出現するかまでを求めることができる．

Accelerating 5-vertex subgraph counting on GPUs
Shuya Syganami†
Toshiyuki Amagasa†† and Hiroyuki Kitagawa††
†College of Information Science, University of Tsukuba
††Center for Computational Sciences, University of Tsukuba

3 前提知識

入力はノード数 n，エッジ数 m の無向グラフで与えられる．本稿の目的はグラフ中の 5 ノードまでのサブグラフの数え上げを行うことである．5 ノードのサブグラフは図 1 に示すように，連結なものが 21 個，非連結なものは 13 個存在する．

5 ノードまでのサブグラフの数え上げを行う手法である ESCAPE について述べる．ESCAPE は多くの数え上げのアルゴリズムで起こる組合せ爆発を避けることにより，5 ノードまでのサブグラフの数え上げを行う．ESCAPE は組合せ爆発を回避するために以下の 2 つのアイデアを利用する．

アイデア 1: パターンをより小さなパターンに分割.
クリークを除く全てのパターンにおいて，いくつかのノードを取り除くとそのパターンをいくつかの連結なグラフに分割できるようなノード集合が存在する．

アイデア 2: エッジに方向付け.
入力される無向グラフ G のエッジについて $\prec$ に基づいて方向付けを行い $G^{\rightarrow}$ を作成し G' について探索を行う．方向付けにより同じ部分を重複して探索することを防ぎ，探索範囲の削減を行う．

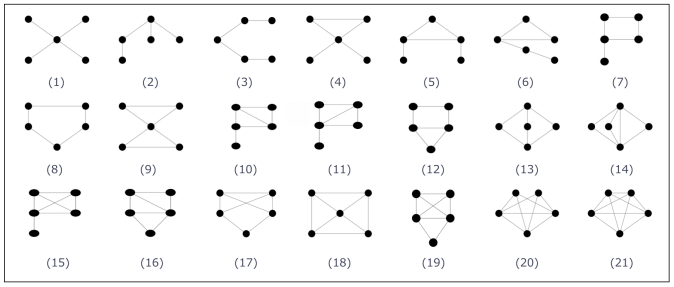
ESCAPE ではこれらを組み合わせパターンをより小さいパターンに分割し，その分割したパターンについて $G^{\rightarrow}$ を探索することでカウンティングを行う．

4 提案手法

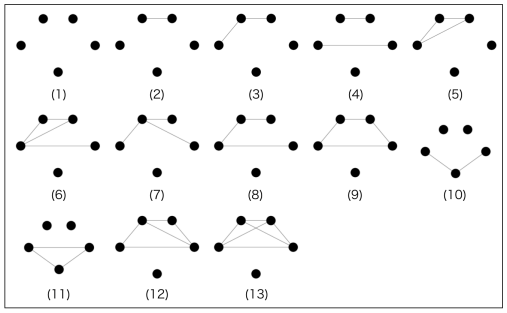
本節では GPU を用いた 5 ノードまでのサブグラフの数え上げについて述べる．本稿では前節で説明した ESCAPE の cutting フレームワークに基づいた数え上げアルゴリズムを GPU 用いて並列に実行することにより，5 ノードのサブグラフの数え上げを高速化する．各パターンの探索において，探索は全ノードについてや全エッジについて行う．この探索は各ノードやエッジ毎に独立であるため，ノード単位やエッジ単位でのスレッド並列化を行うことにより並列に探索することができる．

4.1 データ構造

GPU を用いて並列に数え上げを行う際にボトルネックとなるのは数え上げに wedge を利用する場合である．数え上げに wedge に利用するためには，全てのノード



(a) 連結な 5 ノードのグラフ



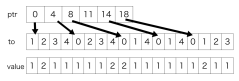
(b) 非連結な 5 ノードのグラフ

図 1: 5 ノードのグラフ

について，そのノードがどのノードと wedge を形成するかを記録しておく必要がある．ここで，あるノードと wedge を形成するノードとは図 2b において，ノード i に対してノード j となる位置のノードのことである．ただし i, j 間のエッジの接続関係は問わない．この情報を記録するためには $O(V^2)$ のメモリが必要となる．しかし我々の検証では，実世界のグラフにおいて各ノードと wedge となるノードの配列は比較的疎となる．そのため本稿では wedge の配列を CSR (compressed sparse row) 形式に基づいた表現法により表す．CSR はグラフを表現するために一般的に使用されているデータレイアウトである．wedge を CSR に基づいた方式で表現する方法を説明する．ptr 配列，to 配列と value 配列を用いることで，あるノードがどのノードと何個 wedge を形成するかを表現する．例として図 2a のグラフは wedge は図 2c のように表される．図 2a において，ノード 0 はノード 2 と $(0, 1, 2)$ と $(0, 4, 2)$ の 2 つの wedge を形成するため value は 2 となる．

(a) グラフの例

(b) wedge



(c) wedge の格納方式の例

図 2: データの格納方式の例

5 実験

本節では提案手法の性能を評価するため，5 ノードまでのサブグラフの数え上げの state-of-the-art な手法である ESCAPE と実行速度の比較評価を行う．本実験には CPU として Intel(R) Xeon(R) CPU E5-2660 v4 @ 2.00GHz m，GPU として NVIDIA Tesla V100，32GB を搭載しているマシンを使用した．ESCAPE と提案手法の実行時間の比較を表 1 に示す．5 ノードのサブグラフの数え上げにおいて，約 3 倍から 5 倍の高速化を達成している．

表 1: サブグラフの数え上げの実行時間 (単位は全て秒)

データセット	$\|V\|$	$\|E\|$	$\|T\|$	ESC-5	proposal-5
soc-brightkite	56.7K	426K	494K	7.16	1.786
tech-as-skitter	1.69M	28.8M	28.8M	1.27K	479.6
web-wiki-ch-internal	1.93M	8.5M	18.19M	1.73K	316.98
web-hudong	1.98M	14.43M	21.61M	2.55K	788.56
web-baidu-baike	2.14M	17.01M	25.2M	3.61K	1.31K

6 まとめ

本稿では GPU を用いた 5 ノードまでのサブグラフの数え上げの手法を提案した．提案手法は 5 ノードのサブグラフの数え上げを行う手法 state-of-the-art な手法に比べ，約 3 倍から 5 倍の高速化を達成した．

今後の課題としては，複数の GPU を用いた数え上げの並列化の検討を行う予定である．

参考文献

[1] Ali Pinar, C Seshadhri, and Vaidyanathan Vishal. Escape: Efficiently counting all 5-vertex subgraphs. In *Proceedings of the 26th International Conference on World Wide Web*, pages 1431–1440. International World Wide Web Conferences Steering Committee, 2017.

[2] Ryan A Rossi and Rong Zhou. Leveraging multiple gpus and cpus for graphlet counting in large networks. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, pages 1783–1792. ACM, 2016.