続グラフ・アナリティクス・ベンチマークの冒険

この記事は、インテル® デベロッパー・ゾーンに公開されている「More Adventures in Graph Analytics Benchmarking」の日本語参考訳です。

Louvain アルゴリズムのベンチマーク

以前の 2 つの記事、グラフ・アナリティクス・パフォーマンスの測定 (英語) とグラフ・アナリティクス・ベンチマークの冒険をお読みになった方は、私が最近グラフ・アナリティクス・ベンチマークの話題を多く取り上げていることをご存じでしょう。簡単に実行でき、複数のグラフ・アルゴリズムとトポロジーをテストして、グラフ・アナリティクス全体を網羅していること、また包括的、客観的、かつ再現可能な結果が得られること (最も重要です) から、カリフォルニア大学バークレー校の GAP Benchmark Suite (英語) を使用していることもご承知かもしれません。しかし、GAP はソーシャル・ネットワークでのコミュニティー検出に対応していません。

大規模ネットワークでコミュニティーを検出する Louvain アルゴリズム[1] は、このギャップを埋める可能性のある候補です。第 1 に、これは広く研究されているアプローチです。Google Scholar* によると、このアルゴリズムに関する記事の引用数は、2019 年 11 月時点で 1 万を超えています。第 2 に、このアルゴリズムは大規模なグラフ向けに設計されています。オリジナルの記事では、1.18 億個の頂点と 10 億個以上のエッジを持つウェブグラフを解析しています。この記事では、Louvain アルゴリズムのベンチマークについて考察します。

最近、私は RAPIDS* cuGraph と NetworkX の Louvain パフォーマンスの比較を目にする機会がありました (**図 1**)。NetworkX は、ネットワーク解析向けの包括的なライブラリーで、ネットワーク研究の中でも最も難しい分野、例えば Triadic Census などもカバーしています。非常に小さなグラフを扱う場合に適したライブラリーです。NetworkX は純粋な Python* ライブラリーであるため、パフォーマンスはあまりよくありません。スピードアップを誇張することが目的でなければ、cuGraph のような最適化されたライブラリーと、NetworkX のような最適化されていないライブラリーを比較することは無意味です。しかし、このグラフにはさらに大きな問題があります。**図 1** の出典元の記事 (英語) は、cuGraph と NetworkX の結果が同じであったかどうかを明らかにしていません。そこで、私は実際に自分で比較してみることにしました。

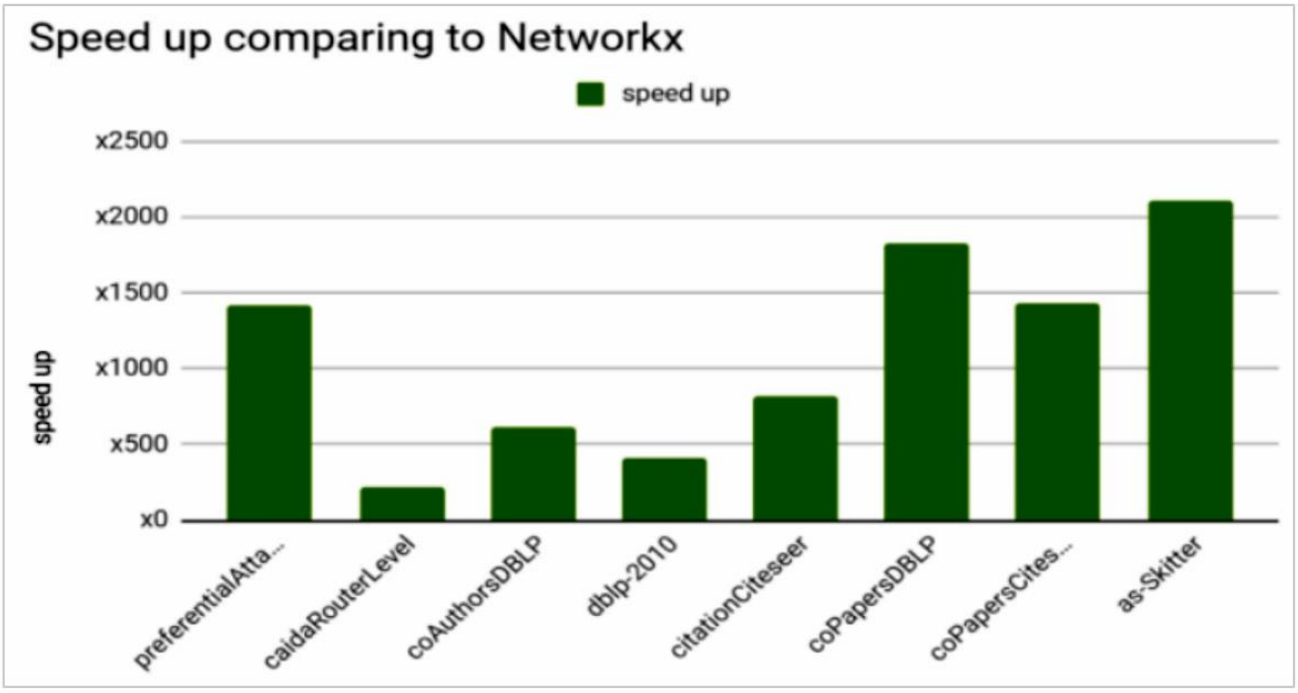

図 1. いくつかの小さなグラフでの cuGraph と NetworkX の Louvain クラスタリング・パフォーマンスの比較 (出典: https://medium.com/rapids-ai/rapids-cugraph-1ab2d9a39ec6 (英語))

検証には、Python* よりも優れたパフォーマンスをもたらすと考えられる C++ で記述された Louvain アルゴリズムのリファレンス実装 (英語) を使用しました (2019 年 11 月にダウンロード)[2]。以下のコマンドを使用して、このパッケージをコンパイルしました。

```
$ tar -xzf louvain-generic.tar.gz
$ cd gen-louvain
$ make
```

これにより、convert (未処理のエッジリストを効率良いバイナリー形式に変換します)、louvain (モジュール方式の最適化を使用してグラフをコミュニティーに分割します[3])、および hierarchy (コミュニティーの構造を表示します) の 3 つの実行ファイルが生成されました。以前の GAP の実験と同様に、GNU* C++ コンパイラーの標準ビルドを使用しました。ソースコードは変更せずにそのまま使用し、オリジナルの Makefile のコンパイラー・オプションを使用しました。

(louvain-generic.tar.gz に含まれる) README.txt ファイルの手順に従って、入力データを準備し、Louvain リファレンス実装を実行しました。

```
$ convert -i graph.txt -o graph.bin
$ louvain graph.bin -v -l -1 -e 0.001 > graph.tree
$ hierarchy graph.tree -n
```

図 1 のグラフは、SuiteSparse Matrix Collection (英語) からダウンロードしてエッジリストに変換したものです。これらは小さなグラフです (**表 1**)。最も大きいものでも 170 万個の頂点しかありません。

Graph	*Vertices*	*Edges*
preferentialAttachment	100,000	999,970
caidaRouterLevel	192,244	1,218,132
coAuthorsDBLP	299,067	1,955,352
dblp-2010	326,186	1,615,400
citationCiteseer	268,495	2,313,294
coPapersDBLP	540,486	30,491,458
coPapersCiteseer	434,102	32,073,440
as-Skitter	1,696,415	22,190,596

表 1. 図 1 で使用されている小さなグラフのサイズ

一方、GAP Benchmark Suite は、最も小さなグラフでも 239 万個の頂点があり、最も大きなグラフでは 1342 万個の頂点があります。オリジナルの記事では、1.18 億個の頂点を持つグラフで Louvain アルゴリズムがテストされています。**図 1** のグラフは、おそらく 2 つの理由で選択されたと考えられます。1 つ目は、cuGraph の Louvain 実装は複数の GPU のメモリーを使用できないため、単一の NVIDIA* Tesla* V100 のメモリーに収まる小さなグラフでなければなりません。2 つ目は、NetworkX と比較しているため、大きなグラフでは純粋な Python* ライブラリーの計算時間が非常に長くなります。

パフォーマンスについて詳しく見る前に、正確さについて考えてみましょう。出典元の記事 (英語) は、NetworkX と cuGraph の結果が同じであったかどうかには触れず、**図 1** のパフォーマンス・データのみを示しています。それぞれの Louvain 実装は、同じコミュニティーを検出したのでしょうか? それぞれの Louvain 実装によって検出されたコミュニティーの数は大きく異なります (**表 2**)。実装ごとに内部ヒューリスティックが異なる可能性があるため、多少のばらつきは想定されます。しかし、表 2 ではパフォーマンスの比較が疑わしくなるほど大きな差も見られます。例えば、1 つの実装はコミュニティーのセットに収束するのが早すぎるように見受けられます。これらのグラフでは何が正しい Louvain 計算結果なのでしょうか? GAP やその他の標準ベンチマークの主な利点の 1 つは、テストの正しい結果が定義されることです。そうでなければ、同一条件でのパフォーマンスの比較は困難、あるいは不可能です。

Graph	**Number of Communities Detected**		
	NetworkX	**Louvain Reference Implementation**	**cuGraph**
preferentialAttachment	22	28	3,004
caidaRouterLevel	631	537	556
coAuthorsDBLP	218	191	583
dblp-2010	23,194	48,727	59,670
citationCiteseer	130	133	47
coPapersDBLP	145	148	455
coPapersCiteseer	271	272	498
as-Skitter	1,150	1,146	1,237

表 2. 各 Louvain 実装で検出されたコミュニティーの数の相違。cuGraph[4] のテストは、AWS* EC2* p3.16xlarge インスタンスを使用して[5]、単一の V100 で実行されました。NetworkX と Louvain リファレンス実装は、単一のインテル® Xeon® プロセッサーで実行されました[6]。

cuGraph と Louvain アルゴリズムのシーケンシャルな C++ リファレンス実装を比較すると、パフォーマンスの差は大幅に縮まります (**図 2**)。いくつかの簡単な変更を加えることで、この差をさらに縮めることができると考えられます。例えば、インテル® コンパイラーを使用して積極的な最適化とベクトル化を有効にしたり、OpenMP* を使用してリファレンス実装を並列化したりできます。しかし、正しい結果が判明するまで、私はその作業に取り掛かる気はありません。現時点では、cuGraph のスピードアップが**図 1** で報告された 200 倍～ 2,100 倍とはかけ離れた 1.4 倍～ 10 倍であることが明らかになり、これらのパフォーマンス比較に問題があることが分かったことに満足しています。ほとんどの「その場しのぎの」未定義のベンチマークと同様に、結果をうのみにすることはできません。

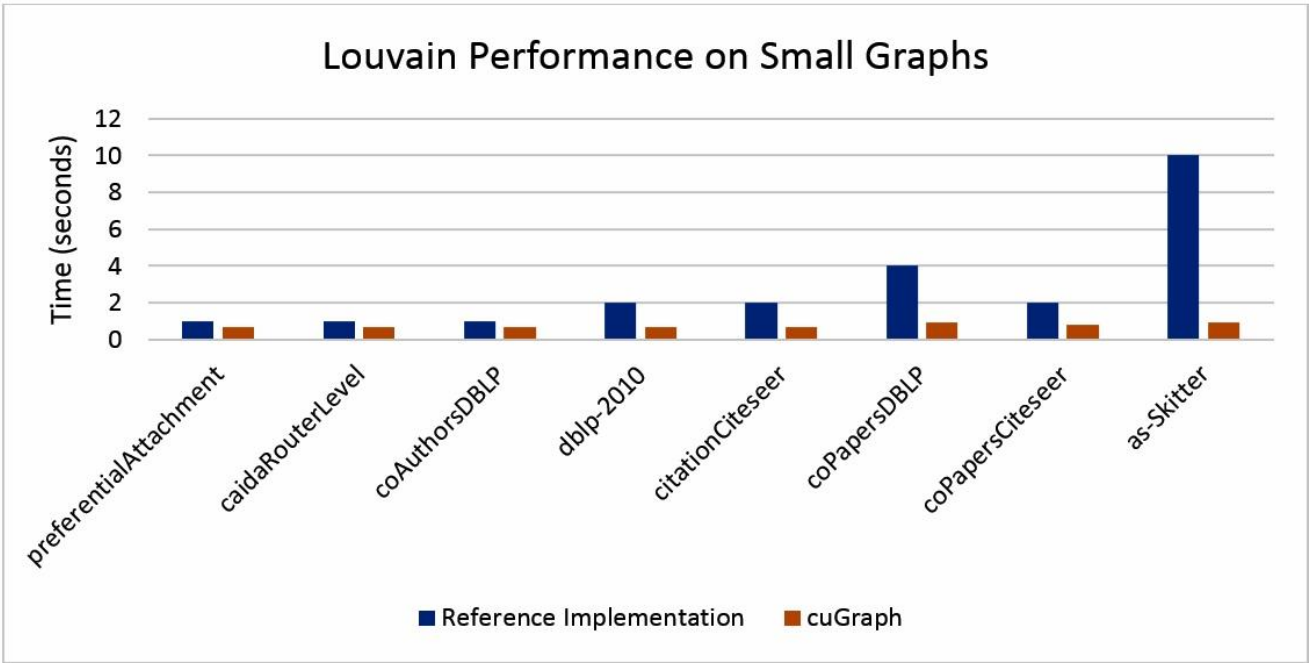

図 2. いくつかの小さなグラフでの Louvain リファレンス実装と cuGraph のパフォーマンスの比較[7]。計算時間 (秒単位、値が低いほうが良い) のみが測定され、グラフの読み込みと設定にかかった時間は無視されています。cuGraph テストは、AWS* EC2* p3.16xlarge インスタンスを使用して単一の V100 で実行されました。Louvain リファレンス実装は、単一のインテル® Xeon® プロセッサーで実行されました。

正しい結果を定義する

正しい結果を定義するため、良く知られている Zachary [8] Karate Club (英語) ソーシャル・ネットワークで各 Louvain 実装を実行しました。このグラフは、すべての実装が同じコミュニティーを生成するかどうかを手動で確認できるぐらい小さいです (34 個の頂点と 78 個のエッジしかありません)。NetworkX、cuGraph、Louvain 実装はそれぞれわずかに異なる結果を生成しました (相違点はハイライト表示しています)。

NetworkX

```
[0, 1, 2, 3, 7, 9, 11, 12, 13, 17, 19, 21]
[4, 5, 6, 10, 16]
[8, 14, 15, 18, 20, 22, 26, 29, 30, 32, 33]
[23, 24, 25, 27, 28, 31]
```

Louvain リファレンス実装

```
[0, 1, 2, 3, 7, 11, 12, 13, 17, 19, 21]
[4, 5, 6, 10, 16]
[8, 9, 14, 15, 18, 20, 22, 26, 29, 30, 32, 33]
[23, 24, 25, 27, 28, 31]
```

cuGraph

```
[0, 1, 2, 3, 7, 9, 11, 12, 13, 17, 19, 21]
[4, 5, 6, 10, 16]
[8, 14, 15, 18, 20, 22, 26, 28, 29, 30, 31, 32, 33]
[23, 24, 25, 27]
```

表 2 のコミュニティーの数のばらつきから、各パッケージが異なるコミュニティーを生成しているのは当然のことです。各パッケージで検出されたコミュニティーは、Girvan および Newman (2002 年、図 4b を参照) によって報告されたものとは完全に一致しませんが、「主観的に」妥当であると言えます。

```
[0, 1, 3, 7, 11, 12, 13, 17, 19, 21]
[4, 5, 6, 16]
[10]
[8, 9, 14, 15, 18, 20, 22, 23, 25, 26, 29, 30, 31, 32, 33]
[2, 24, 27, 28]
```

まとめ

この実験の最初の目的は、cuGraph のような最適化されたライブラリーを NetworkX のような純粋な Python* ライブラリーと比較することの愚かさを示すことでした。C++ の Louvain リファレンス実装は、パフォーマンスの差を大幅に縮めましたが、コミュニティーの数に大きなばらつきがあることは、パフォーマンス以外の問題、例えば、各入力グラフの正しいコミュニティーは何か? 1 つの Louvain 実装が正しくて、それ以外は正しくないのか? があることを示唆しています。

広く研究されている Zachary ソーシャル・ネットワークでさえ、正しい結果を定義するのが困難なことから、「客観的な」明確に定義されたグラフ・アナリティクス・ベンチマークの重要性が浮き彫りになりました。Louvain アルゴリズムは、GAP Benchmark Suite にはないコミュニティー検出機能で GAP を拡張できますが、「その場しのぎの」パフォーマンス測定では十分ではありません。グラフ・アナリティクス・コミュニティーによって、合理的で再現性のあるテストが定義される必要があります。それまでは、意思決定にコミュニティー検出のパフォーマンスが重要な場合は、LDBC Graphalytics (英語) ベンチマークを使用することを推奨します。

---

[1] Blondel et al. (2008). "Fast unfolding of Communities in Large Networks," J Stat Mech, P10008.

[2] このパッケージの詳細は、「Louvain method: Finding Communities in Large Networks」(英語) を参照してください。リファレンス実装は並列化されていません。

[3] Girvan and Newman (2002). "Community Structure in Social and Biological Networks," Proc Natl Acad Sci, 99(12), 7821-7826.

[4] cuGraph コードは、https://github.com/rapidsai/notebooks/blob/branch-0.11/cugraph/community/Louvain.ipynb (英語) から引用したものです。

[5] GPU: NVIDIA Tesla V100 with 16 GB HBM; System memory: 480 GB; Operating system: Ubuntu Linux release 4.15.0-1054-aws, kernel 56.x86_64; Software: RAPIDS v0.1.0 (downloaded and run November 2019).

[6] CPU: Intel® Xeon® Gold 6252 (2.1 GHz, 24 cores), HyperThreading enabled (48 virtual cores per socket); Memory: 384 GB Micron DDR4-2666; Operating system: Ubuntu Linux release 4.15.0-29, kernel 31.x86_64; Software: GNU g++ v7.4.0, NetworkX v2.3, community API v0.13, Louvain reference implementation (downloaded and run November 2019).

[7] For more complete information about performance and benchmark results, visit www.intel.com/benchmarks.

[8] Zachary (1977). "An Information Flow Model for Conflict and Fission in Small Groups," J Anthropol Res, 33(4), 452-473.

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more complete information visit www.intel.com/benchmarks.

Performance results are based on testing as of November 2019 by Intel Corporation and may not reflect all publicly available security updates. See configuration disclosure for details. No product or component can be absolutely secure. For more complete information about performance and benchmark results, visit www.intel.com/benchmarks.

Intel does not control or audit third-party data. You should consult other sources to evaluate accuracy.

Cost reduction scenarios described are intended as examples of how a given Intel-based product, in the specified circumstances and configurations, may affect future costs and provide cost savings. Circumstances will vary. Intel does not guarantee any costs or cost reduction.