

1. 担当 PM

プロジェクトマネージャー：藤井 彰人
(KDDI 株式会社 ソリューション事業本部 ソリューション事業企画本部長
兼 クラウドサービス企画部長)

2. 契約者氏名

クリエータ：伊藤 竜一 (大阪大学 大学院)

3. 委託金支払額

2,021,200 円

4. テーマ名

様々なデータソースに対応する高速なグラフ処理エンジンの開発

5. 関連 Web サイト

- GitHub : <https://github.com/hoshizora-project>
- DockerHub : <https://hub.docker.com/u/hoshizora/>
- PyPI (amanogawa) : <https://pypi.python.org/pypi/amanogawa>
- PyPI (hoshizora) : <https://pypi.python.org/pypi/hoshizora>

6. テーマ概要

本プロジェクトでは、データ前処理基盤と統合されたグラフ構築・分析基盤を開発した。それにより、ユーザはパラメータの選択のみで様々なデータソースを利用したグラフ構築・分析が可能となり、Trial-and-Error しながらのグラフ処理を容易に行うことが可能となった。

7. 採択理由

大規模データから有益な情報を得て活用するニーズが高まる中、現実にはそのデータソースはフォーマット含め様々であり「データ分析作業の 9 割以上が前処理」など揶揄されるように、実際には有効に活用することには多くの作業が必要である。本提案は、様々なデータソースに対応できるだけでなく、高速なグ

ラフ処理も可能にするエンジンを開発するものであった。未踏期間中にデータアナリストが喜ぶ、簡単に利用できかつ高速なツール群を開発してくれることを期待した。

8. 開発目標

近年、ネットワークやストレージなどの進歩によって様々な大規模データが収集可能になった。そして、それらのデータから有益な情報を得るデータマイニングが活発に行われている。データマイニングには多くのアプローチがあるが、中でも物事の関係性を顕著に捉えることができるグラフ構造に注目したグラフ処理は強力である。例えばソーシャルネットワークのユーザの繋がりを表すグラフ構造であれば、コミュニティの抽出や影響力のあるユーザの特定といった分析を効率的に行うことが可能である。

実際のグラフ処理には 2 つのフェーズが必要である。まずデータ間の何らかの類似度に基づいてグラフを構築し、その後にグラフ分析アルゴリズムを適用して分析を行う。これらは 1 回きりのものではなく、良い結果が得られるまでグラフ構築・分析を繰り返し行うこととなる。このグラフ構築とグラフ分析にはそれぞれ以下のような問題点が存在している。

- グラフ構築

構築方法・利用する特徴量がデータに強く依存するため、処理の一般化が難しく、安定したライブラリが存在しない。そのため、構築方法や特徴量に応じて煩雑なグラフ構築処理を自前で準備する必要がある。また、簡易な実装では時間計算量・空間計算量共に $O(n^2)$ となってしまうため、極小規模（～5000 程度）のデータしか扱えない。

- グラフ分析

グラフ分析アルゴリズムは、繰り返しが多くコストが高いものが多い。大規模データも増えてきており、処理時間が問題となる。特に Trial-and-Error しながらの分析では致命的なボトルネックとなる。

本プロジェクトでは、これらの問題点を解決したグラフ処理基盤の開発を目的とした。グラフ構築とグラフ分析を可能な限り簡単に、かつ高速に実行可能なシステムとすることで、ラップトップ PC のような手軽な環境で Trial-and-Error しながらのグラフ処理を実現する。加えて、広く利用してもらうため、既存のデータ分析技術とシームレスに組合せて利用できるシステムとする。

9. 進捗概要

本プロジェクトでは効率的なグラフ処理を実現するために「amanogawa」と「hoshizora」と呼ばれる 2 つのシステムを作成した (図 1)。いずれも Python

ライブラリ又はコマンドラインツールとして利用可能である。

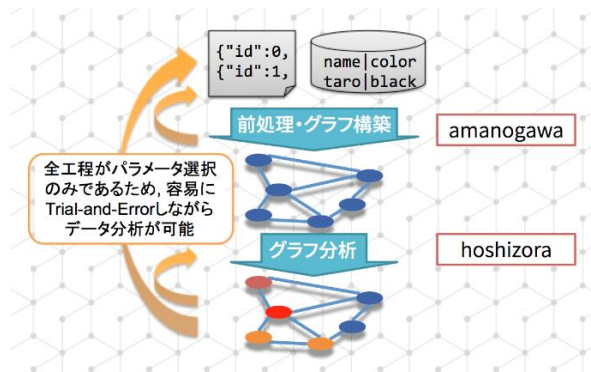


図 1. システム概要

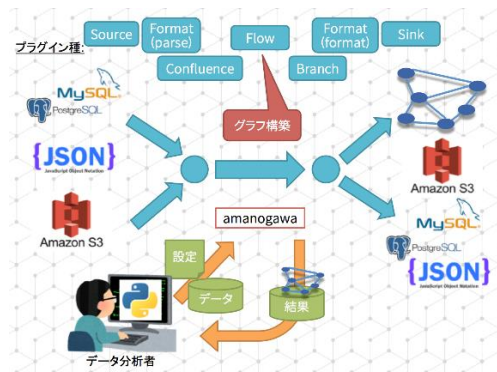


図 2. amanogawa 概要

amanogawa はグラフ処理のうち、グラフ構築を担当するシステムである (図 2)。データ前処理と統合することで、グラフ構築を一般化し、ユーザはパラメータを選択するだけで処理可能である。データ中に含まれるグラフを意味する要素の単純な抽出だけでなく、数値特徴量を組合せた類似度に基づく k 近傍グラフの構築や、自然言語処理による文字列特徴量のベクトル化を通したグラフの構築が可能である。また、amanogawa の主目的はグラフ構築であるが、その前処理にも注力している。「データ分析では前処理が 8 割」と揶揄されることがあるほど前処理は手間であることが多い。そこで、グラフ構築のためのデータ一般化を拡張し、様々な前処理に応用可能なものとした。ローカルファイルやリレーショナルデータベースといったデータソースに対応するためのデータ入力用プラグイン (Source)、CSV や JSON といったフォーマットに対応するためのデータ整形用プラグイン (Format)、グラフ構築やフィルタリングといったデータ加工パターンに対応するためのデータ加工プラグイン (Flow)、指定属性の一致等から複数データを Join するためのデータ結合用プラグイン (Confluence)、条件に基づいて行や列を分割するためのデータ分割用プラグイン (Branch)、ローカルファイルやリレーショナルデータベースといったデータの出力先のためのデータ出力用プラグイン (Sink)、計 6 種のプラグインの組合せで動作する。これらのプラグインはそれぞれ入力・出力のいずれか又は両方を持ち、決められたインターフェースでデータのやり取りを行う。その組合せでデータフローグラフを構築する (※ここでの「グラフ」はグラフ構築や分析対象となる「グラフ」とは異なる)。頂点がプラグインを、エッジがプラグイン間のデータのやり取りを意味する。グラフ構築はこのうち FlowPlugin として提供している。利用方法は主に 2 つあり、ユーザは自由に選択することができる。1 つ目は TOML 形式の設定ファイルを用いる方法である。図 3 のように、Source、Flow、Sink 等を記述する。それぞれの頂点に ID を振り、どの頂点 (=プラグイン) からデータが渡されるかを from という属性として指定することでデータフローグラフを表現する。今回の例のうち Flow の部分に注目してみると、`type = to_graph` と

してグラフ構築用プラグインを呼び出し、*mode = bow*として自然言語処理を用いたグラフ構築を、更に *knn, to* といったオプションを指定している。また、データフローに分岐の発生しない簡単なタスクでは、図 4 のような ID を利用しない簡易記法を用いることもできる。これらの TOML 形式の設定ファイルは Python ライブラリ・コマンドラインツールのどちらからでもロードして利用することができる。2 つ目は Python の Embedded Domain Specific Language (eDSL) を用いる方法である。eDSL は図 5 のように記述することで図 3 と同じ内容のデータフローグラフを記述することができる。

```

1 [source.r]
2 type = "file"
3 path = "data.jsonl"
4 [source.r.format]
5 type = "json"
6 columns = [
7   { name = "vpos", type = "int" },
8   { name = "date", type = "int" },
9   { name = "user_id", type = "string" },
10  { name = "content", type = "string" }
11 ]
12
13 [flow.f]
14 type = "to_graph"
15 from = "r"
16 mode = "bow"
17 column = "content"
18 knn = { k = 2, p = 1.5 }
19 to = [
20   { name = "topo" },
21   { name = "meta", columns = [ "vpos", "date", "user_id" ] }
22 ]
23
24 [sink.s1]
25 type = "file"
26 from = "topo"
27 path = "graph.tsv"
28 [sink.s1.format]
29 type = "csv"
30 delimiter = "\t"
31
32
33 [sink.s2]
34 type = "file"
35 from = "meta"
36 path = "meta.csv"
37 [sink.s2.format]
38 type = "csv"

```

図 3. グラフ構築用 TOML 形式設定ファイル例

```

1 [source]
2 type = "file"
3 path = "kinmosa.csv"
4 [source.format]
5 type = "csv"
6 skip_header = true
7 columns = [
8   { name = "id", type = "int" },
9   { name = "name", type = "string" },
10  { name = "blood_type", type = "int" }
11 ]
12
13 [sink]
14 type = "file"
15 path = "kinmosa.tsv"
16 [sink.format]
17 type = "csv"
18 delimiter = "\t"

```

図 4. CSV を TSV に変換する TOML 形式設定ファイル例

```

1 import amanogawa as am
2 builder = am.ConfigBuilder()
3 config = builder \
4   .source('file', 'r').set('path', 'data.jsonl').format('json') \
5   .set('columns', [{ 'name': 'vpos', 'type': 'int' },
6     { 'name': 'date', 'type': 'int' },
7     { 'name': 'user_id', 'type': 'string' },
8     { 'name': 'content', 'type': 'string' }]) \
9   .flow('to_graph', 'f', 'r').set('mode', 'bow').set('column', 'content') \
10  .set('mode', 'bow').set('column', 'content').set('knn', { 'k': 2, 'p': 1.5 }) \
11  .set('to', [{ 'name': 'topo' },
12    { 'name': 'meta', 'columns': [ 'vpos', 'date', 'user_id' ] }]) \
13  .sink('file', 's1', 'topo').set('path', 'graph') \
14  .format('csv').set('delimiter', '\t') \
15  .sink('file', 's2', 'meta').set('path', 'meta.csv').format('csv').build()

```

図 5. Python eDSL によるグラフ構築設定例

hoshizora はグラフ処理のうちグラフ分析を担当するシステムである。汎用的なグラフ分析に対応しており、グラフ分析用のインターフェースを提供する。このインターフェースでは 4 つの関数に基づいて 1 つのグラフ分析パターンを実現する。実装されたグラフ分析パターンは自動的に hoshizora の効率的な並列グラフ計算モデルに基づいて実行されるため、ユーザは並列化等の最適化を考慮する必要がない。実行する際は、図 6 のように amanogawa で構築されたグラフをパラメータとともに渡すだけである。

```
1 import hoshizora as hz
2 result = hz.pagerank(graph, num_iters=100)
```

図 6. グラフ分析の実行例

10. プロジェクト評価

グラフ分析における前処理の煩雑さを解消し、かつグラフ分析の高速化を実現することで、Trial & Error をしながらデータ分析を行えるツール基盤を構築したことは、高く評価されるべきと考えている。特に従来のグラフ解析ツール対比で 50 倍の高速化を実現した hoshizora と Jupyter Notebook 連携は秀逸である。成果報告会において、多くの人に理解してもらえるユースケースとともにその成果を効果的に説明できなかつたことは残念であり、PM として反省する点であるが、個々の実用的なツールは将来必ずデータ分析分野で活用されるものになるであろうと考えている。

11. 今後の課題

複数データソースに対応するための amanogawa 用 Plugin の拡充は、本システムの更なる発展のために重要である。このためにも amanogawa/hoshizora の先進性とその効果を、多くのデータアナリストに知ってもらい、プラグイン開発の協力者を得ることが大切である。