# 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
## ― Hoshizora Project：手軽で高速なグラフ構築・分析基盤 ―

### 1. 背景

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．

### 2. 目的

本プロジェクトの目的は，上記の 2 つの問題点を解決したグラフ処理基盤となるシステムの作成である．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，ノートパソコン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．

### 3. 開発の内容

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．

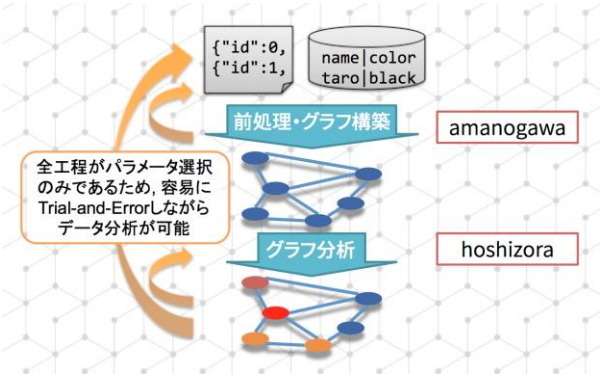


図 1 本プロジェクト概要

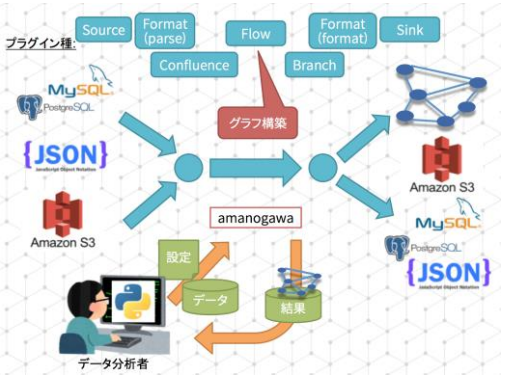


図 2 amanogawa 概要

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ー般化を拡張し，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」とは異なるので注意されたい）．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．

```
[source.r]
type = "file"
path = "data.jsonl"
[source.r.format]
type = "json"
columns = [
  { name = "vpos", type = "int" },
  { name = "date", type = "int" },
  { name = "user_id", type = "string" },
  { name = "content", type = "string" }
]

[flow.f]
type = "to_graph"
from = "r"
mode = "bow"
column = "content"
knn = { k = 2, p = 1.5 }
to = [
  { name = "topo" },
  { name = "meta", columns = [ "vpos", "date", "user_id" ] }
]

[sink.s1]
type = "file"
from = "topo"
path = "graph.tsv"
[sink.s1.format]
type = "csv"
delimiter = "\t"


[sink.s2]
type = "file"
from = "meta"
path = "meta.csv"
[sink.s2.format]
type = "csv"
```

図 3 グラフ構築用 TOML 形式設定ファイル例

```
[source]
type = "file"
path = "kinmosa.csv"
[source.format]
type = "csv"
skip_header = true
columns = [
  { name = "id", type = "int" },
  { name = "name", type = "string" },
  { name = "blood_type", type = "int" }
]

[sink]
type = "file"
path = "kinmosa.tsv"
[sink.format]
type = "csv"
delimiter = "\t"
```

図 4 CSV を TSV に変換する TOML 形式設定ファイル例

```
import amanogawa as am
builder = am.ConfigBuilder()
config = builder \
    .source('file', 'r').set('path', 'data.jsonl').format('json') \
        .set('columns', [{'name': 'vpos', 'type': 'int'},
                         {'name': 'date', 'type': 'int'},
                         {'name': 'user_id', 'type': 'string'},
                         {'name': 'content', 'type': 'string'}]) \
    .flow('to_graph', 'f', 'r').set('mode', 'bow').set('column', 'content') \
        .set('mode', 'bow').set('column', 'content').set('knn', {'k': 2, 'p': 1.5}) \
        .set('to', [{'name': 'topo'},
                    {'name': 'meta', 'columns': ['vpos', 'date', 'user_id']}]) \
    .sink('file', 's1', 'topo').set('path', 'graph') \
        .format('csv').set('delimiter', '\t') \
    .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．

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

図 6 グラフ分析の実行例

4. 従来の技術（または機能）との相違

- グラフ構築

背景でも述べたように，従来，グラフ構築は対象とするデータごと，アルゴリズムごと

に自前で行う必要があった. 加えて, 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という問題があった. 本プロジェクトの amanogawa では, データによる差異・グラフ構築アルゴリズムによる差異のどちらも吸収するため, ユーザはパラメータ選択だけで容易にグラフの構築が可能である. また, オプション 1 つの設定を行うだけでグラフ構築を適切な近似アルゴリズムに切り替えることができるため, 大規模データに対してもスムーズなグラフ構築が可能である.

- グラフ分析

  効率的な並列グラフ計算モデル・グラフ圧縮を中心とした最適化により, ノートパソコンでも Billion-Scale のグラフを分析可能にした. ユーザはグラフ分析を記述する際に hoshizora が提供するインターフェースに則ることで, 一切並列化や細かな最適化を考慮することなく適切に最適化されたグラフ分析を実行することができる. 最良の場合で, 既存のグラフ分析ライブラリであるNetworkXと比較して約50倍の高速化が確認された. また, 100 コアを超えるようなサーバマシンでも処理が適切にスケールすることが確認された. これにより, 超大規模データのサブセットを手元のノートパソコンで Trial-and-Error しながら分析し, 最終的に同じスクリプトを利用して全データをサーバマシンで分析するといった分析フローも可能となった. また, グラフ圧縮の効果により, メモリ使用量のピーク値を 30%程度削減することに成功した.

## 5. 期待される効果

本プロジェクトでは, グラフ構築とグラフ分析の両方を簡単かつ高速に実行可能にすることで,「データ分析は行われているものの, グラフ構築のハードルの高さからグラフ処理が利用されてない」・「試行に時間がかかるためパラメータチューニングが十分に行われずにグラフ処理を活用しきれていない」といった現場でグラフ処理が適切に利用され, グラフ処理の物事の関係性を顕著に捉えるという特徴を元に有益な情報の発見に繋がることが期待される. また, amanogawa はグラフ構築に限らず, ボイラープレート的な前処理をプラグイン化することで, これまでコストの高かったデータ前処理を容易にすることが期待される.

## 6. 普及(または活用)の見通し

本プロジェクトのソースコードは GitHub で公開され, Python ライブラリは PyPI から pip でインストールが可能である. また, チュートリアルとして, 本プロジェクトの全てのシステムをインストールし, 実行環境として Jupyter Notebook がセットアップ済み, 更に使用例として基本的操作集ノートブックとデータセットを含む Docker イメージを DockerHub にて公開しており, Docker 環境さえあればすぐに試すことが可能である. しかしながら, チュートリアル以上の操作をするためのドキュメントが不足しており, より多くのユーザに利用してもらうためにはその拡充が必須である. また, 本プロジェクトの特徴の一つであるプラグイン機構をより活用するためには様々なプラグインを開発・追加していく必要がある. 本クリエータも継続的に開発・追加していく予定だが, 全てに対応させることは困難であるため, OSS としての特徴を活かして開発者を募り開発を加速させていくことが望ましいだろう.

7. クリエータ名（所属）

伊藤 竜一（大阪大学大学院）

（参考）関連 URL

- 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