アニーリング計算をストリーム処理で実行するフレームワーク「Sawatabi」とデモアプリケーションの開発

メンバー：寺田晃太朗、古山慎悟、臼井純哉、小野和輝

担当PM：棚橋耕太郎

2020年度未踏ターゲット事業 成果報告会

Feb. 11th, 2021

発表アウトライン

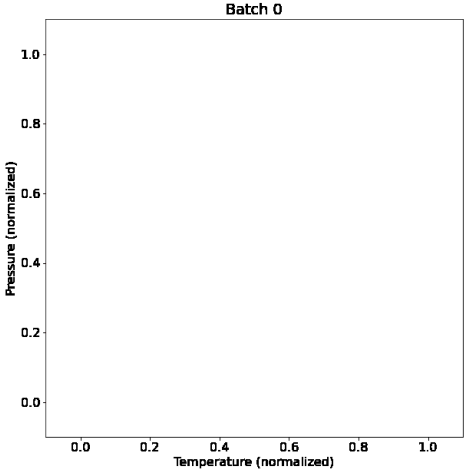
1. プロジェクトの概要
2. 開発フレームワーク — Sawatabi
3. 解決する課題・将来性
4. フレームワークを用いたアプリケーション開発
   - Traveling Salesperson Problem (ルート最適化)
   - Arbitrage Opportunities Finding (最適裁定取引機会の検出)

プロジェクトでやったこと

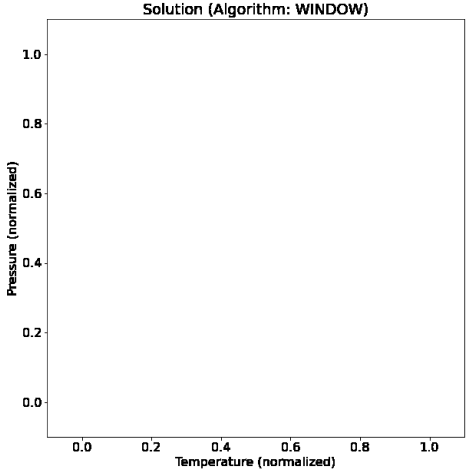
課題

連続してやってくるストリームデータに対してアニーリング計算したい
(問題例: 異常検知)

ストリームデータ



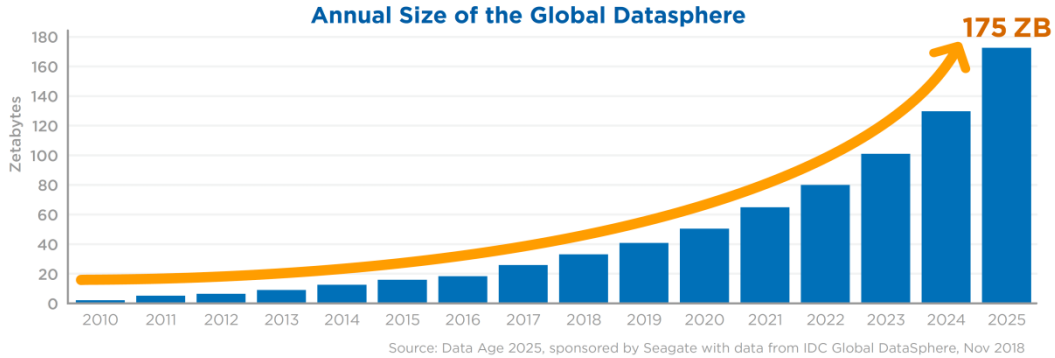
各時間におけるデータの
継続的なアニーリング結果



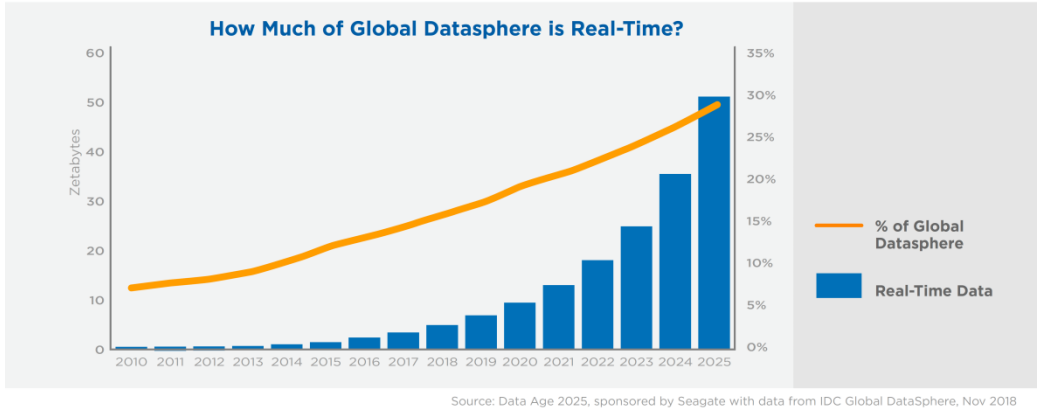
やったこと

このようなイジングアプリケーションを効率的に開発・実行する
フレームワーク「Sawatabi」を開発しました

背景 — ビッグデータ・ストリームデータ



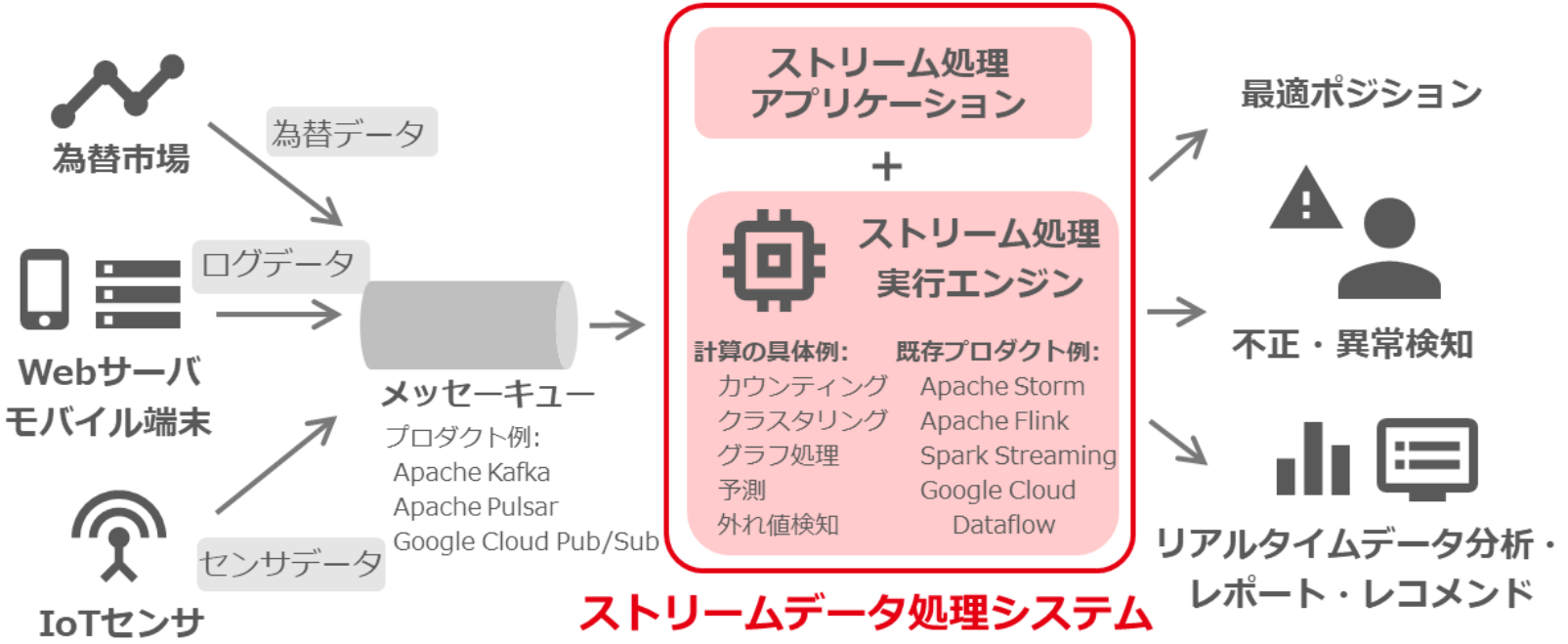
全世界での
データ総量予測



その中でも
リアルタイムデータ
(ストリームデータ)
が占める割合の増加

出典: IDC Whitepaper https://www.seagate.com/files/www-content/our-story/trends/files/idc-seagate-dataage-whitepaper.pdf

従来のストリームデータ処理システム



Continuous Input → Data Processing → Output

ストリーム処理とアニーリング

ストリーム処理 Key Challenges

- ストリーム処理は通常のデータ処理 (バッチ処理) に無い特徴
- これに対してイジングマシンの特徴を生かしてアプローチ

	ストリーム処理
計算の正確性	完全な正確性を必要しないケースあり
速度	リアルタイム性必要
データサイズ	小さい (数B～数KBオーダー)
入力データ	連続で終わりないストリームデータ

アプローチ

	イジングマシン
計算の正確性	確率的
計算速度	問題によっては速い
マシンサイズ	(今は) 小さい (全結合換算で数千スピン)
アプリ+ソルバ	**ストリーム処理アプリ・アルゴリズム実装 + ソルバの工夫**

本PJで開発

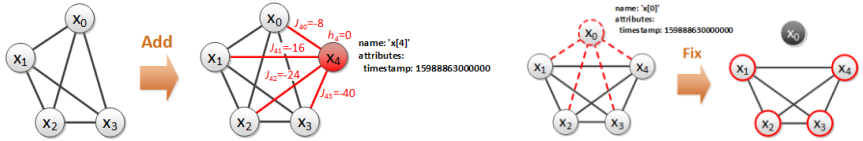
開発したフレームワーク — Sawatabi

「アニーリング計算するストリーム処理アプリ」を Sawatabi を利用することで開発可能に

ライブラリ (Python パッケージ)

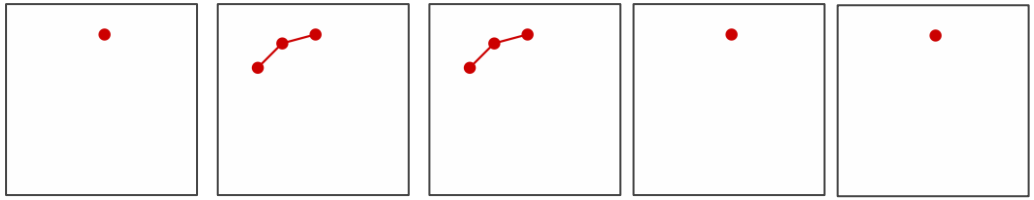
Model

ストリームデータに対応したイジングモデルの差分更新
（追加・削除・固定）と検索機能のインターフェイスを提供



Algorithms

ストリームデータに対するイジングモデル更新をパターン化



イジング
マシン
実行

Demo App & Visualization

アプリケーション動作結果を
アニメーションで可視化

Solver (Sawatabi Solver)

「ストリーム処理」を効率的に
実行するための工夫を検証する
ソフトウェア実装のソルバ

工夫点

- 初期ステート利用リバースアニーリング
- 冷却スケジューリングの部分的な変更

D-Wave The Quantum Computing Company™ **Advantage**

FIXSTARS Speed up your Business **Optigan**

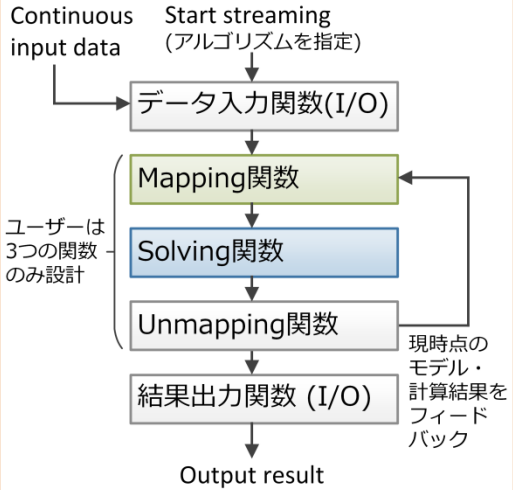
(拡張可能)

Sawatabi Architecture

Algorithm

ストリーム処理部分のパイプライン設計

(1) 関数を利用したパイプライン設計



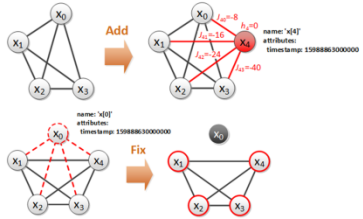
(2) Beam を利用したパイプライン設計

パイプラインのバックエンドは Apache Beam で実装されているので、I/O・計算の関数を自由に組み合わせて設計

Model

Logical Model with Metadata

ストリームデータに対応したイジングモデルの差分更新（追加・削除・固定）



SQLライクなスピン・相互作用検索機能

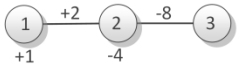
Ex) attributes.currency == 'JPY'

アプリケーションレベル制約クラス

- N-hot constraint
- Equality constraint
- Zero-or-one-hot constraint

Physical Model

Logical Modelの係数や制約を値に変換したイジングモデル



Solver

Solver Connectors

- Local Solver
- D-Wave Solver
- Optigan Solver
- Sawatabi Solver

各マシンと共通のインターフェイスで接続し、物理イジングモデルの計算をすることができる
(拡張可能)

依存ライブラリ

- Apache Beam (ストリームパイプライン)
- Pandas (属性データ管理)
- PyQUBO(モデル管理)
- dwave-neal (ローカルソルバ)
- dwave-ocean-sdk (D-Waveソルバ)

Sawatabi on Google Cloud Dataflow

Sawatabi アプリケーションの実行環境

ローカル (PC・サーバー)、ストリーム処理エンジン

Input Pub/Sub (message queue)

ストリームデータ

```
{“number”: 64}
{“number”: 62}
{“number”: 30}
{“number”: 6}
{“number”: 27}
{“number”: 12}
{“number”: 28}
{“number”: 67}
```

時系列イベントデータ

時間

Google Cloud Dataflow

(永続的なストリーム処理でのアニーリング計算を実行)

Output Pub/Sub (message queue)

```
{“solution”: [
 [62, 6, 27], [64, 30]]}
{“solution”: [
 [62, 6], [30, 27, 12]]}
{“solution”: [
 [30, 28], [6, 27, 12]]}
{“solution”: [
 [6, 67], [27, 12, 28]]}
```

各時刻のアニーリング計算結果

Sawatabi on GitHub

開発フレームワークはOSSとして公開中

GitHub

https://github.com/kotarot/sawatabi

Installation

```
$ pip install sawatabi
```

Sawatabi が解決する課題・将来性

解決する課題

イジング計算をするストリーム処理
アプリの開発効率向上

イジングモデル差分更新を利用した
計算効率の向上

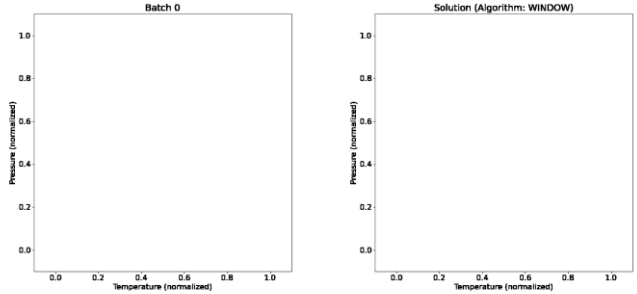
将来性

イジング計算が活用できる
アプリ領域の拡大

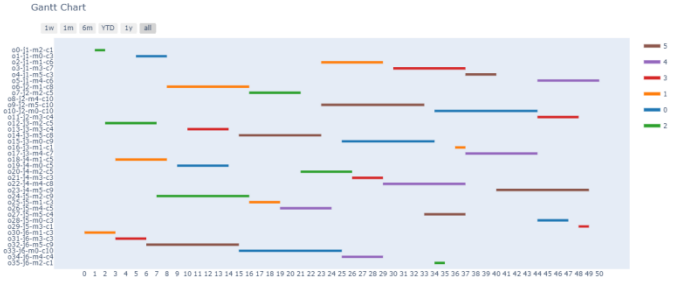
計算性能向上への
さらなる可能性

取り組んだアプリケーション開発

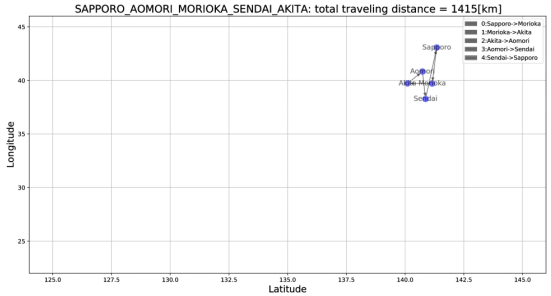
(1) 異常検知 (Unit Clustering)



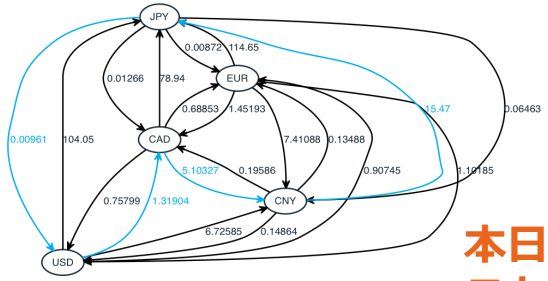
(2) Job-shop Scheduling



(3) Traveling Salesperson Problem



(4) Arbitrage Opportunities Finding



本日は時間の都合で
こちらを説明します

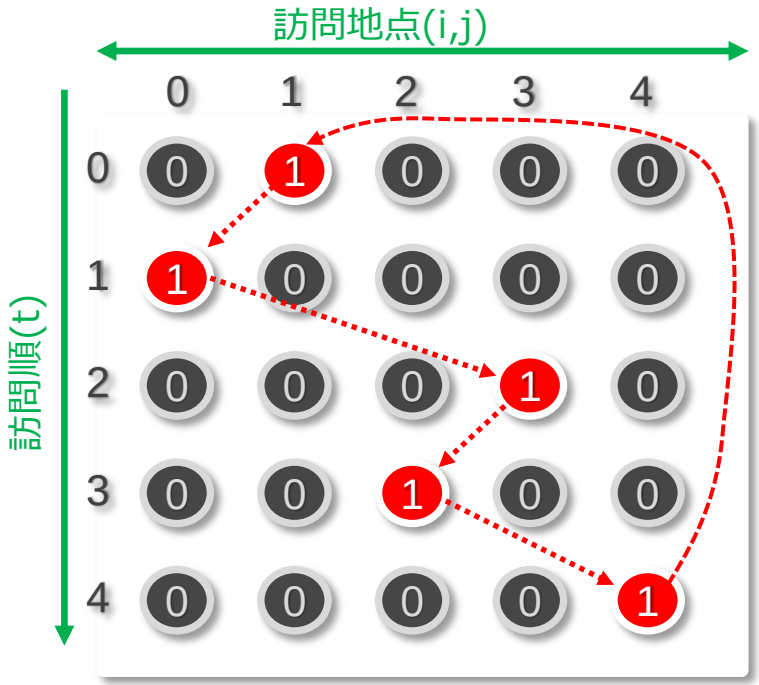
Sample Applications: https://github.com/kotarot/sawatabi/tree/main/sample/algorithm

Sawatabiフレームワークを用いたアプリケーション開発 (1)

Traveling Salesperson Problem

Traveling Salesperson Problem (TSP)

TSP := 各地点を必ず1回通る制約下で距離が最小になる経路を求める最適化問題



TSP + ストリームデータを試した理由

リアルタイムで要求が来るライドシェアや、工場等でのロボットのリアルタイムルーティングへの応用が期待できるため

エネルギー関数 (QUBO)

$$E = \sum_{t,i,j} d_{i,j} x_{t,i} x_{(t+1)j}$$

$where\ \alpha, \beta : const$
$d : distance$
$i, j : point_A, point_B$
$t : visit\ order$

$$+\alpha \sum_{t} (\sum_{i} x_{ti} - 1)^2$$

複数地点に同時に滞在できない

$$+\beta \sum_{i} (\sum_{t} x_{ti} - 1)^2$$

各地点を1回訪問

ストリームデータを TSP に適用

ストリームデータを作成し，window サイズ5都市で 1都市ずつ追加/削除しながら TSP を解いていく

入力データ(json)
{“都市名”: [経度, 緯度]}

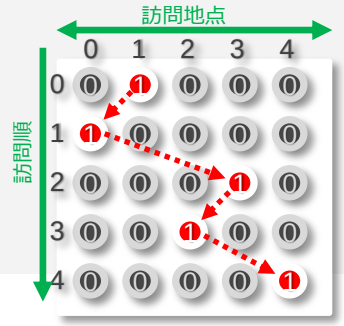
時間

{'Hiroshima': [132.45, 34.39]},
{'Sapporo': [141.34, 43.06]},
{'Yokohama': [139.64, 35.44]},
{'Osaka': [135.52, 34.68]},
{'Sendai': [140.87, 38.26]},
{'Fukuoka': [130.41, 33.60]},
{'Naha': [127.68, 26.21]}
{'Tokyo': [139.69, 35.68]},
{'Kyoto': [135.75, 35.02]}

Window

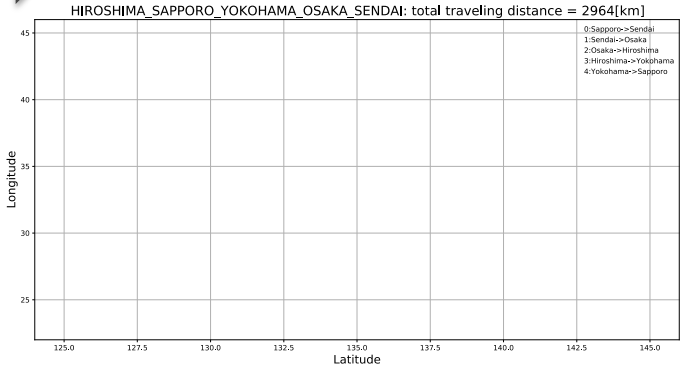
Sawatabi

{'Hiroshima': [132.45, 34.39]},
{'Sapporo': [141.34, 43.06]},
{'Yokohama': [139.64, 35.44]},
{'Osaka': [135.52, 34.68]},
{'Sendai': [140.87, 38.26]},



訪問順を出力
都市A->都市B-> …

Sapporo -> Sendai -> Osaka ->
Hiroshima -> Yokohama



ストリームデータを TSP に適用

ストリームデータを作成し，window サイズ5都市で 1都市ずつ追加/削除しながら TSP を解いていく

入力データ(json)
{“都市名”: [経度, 緯度]}

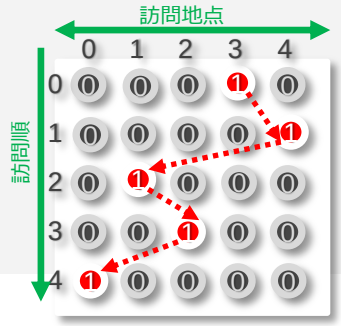
時間

~~{'Hiroshima': [132.45, 34.39]},~~ ← 削除
{'Sapporo': [141.34, 43.06]},
{'Yokohama': [139.64, 35.44]},
{'Osaka': [135.52, 34.68]},
{'Sendai': [140.87, 38.26]},
{'Fukuoka': [130.41, 33.60]}, ← 追加
{'Naha': [127.68, 26.21]}
{'Tokyo': [139.69, 35.68]},
{'Kyoto': [135.75, 35.02]}

Window

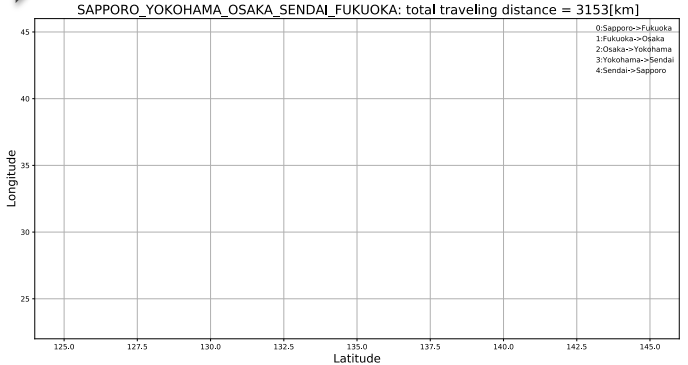
Sawatabi

{'Sapporo': [141.34, 43.06]},
{'Yokohama': [139.64, 35.44]},
{'Osaka': [135.52, 34.68]},
{'Sendai': [140.87, 38.26]},
{'Fukuoka': [130.41, 33.60]},



訪問順を出力
都市A->都市B-> …

Sapporo -> Fukuoka -> Osaka -> Yokohama -> Sendai



ストリームデータを TSP に適用

ストリームデータを作成し，window サイズ5都市で 1都市ずつ追加/削除しながら TSP を解いていく

入力データ(json)
{“都市名”: [経度, 緯度]}

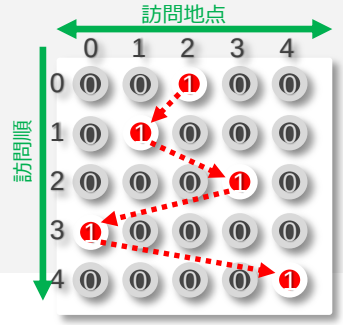
時間

~~{'Hiroshima': [132.45, 34.39]},~~
~~{'Sapporo': [141.34, 43.06]},~~ ← 削除
{'Yokohama': [139.64, 35.44]},
{'Osaka': [135.52, 34.68]},
{'Sendai': [140.87, 38.26]},
{'Fukuoka': [130.41, 33.60]},
{'Naha': [127.68, 26.21]}, ← 追加
{'Tokyo': [139.69, 35.68]},
{'Kyoto': [135.75, 35.02]}

Window

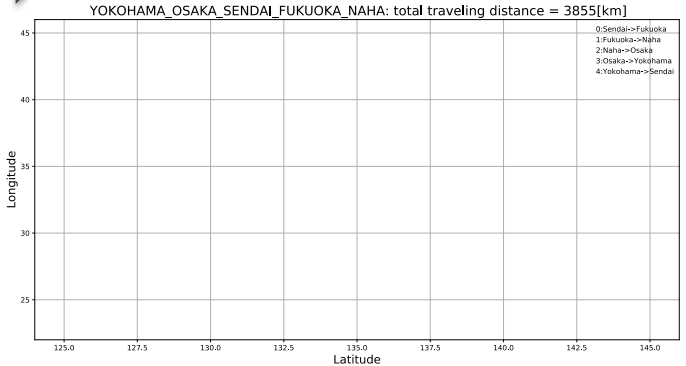
Sawatabi

{'Yokohama': [139.64, 35.44]},
{'Osaka': [135.52, 34.68]},
{'Sendai': [140.87, 38.26]},
{'Fukuoka': [130.41, 33.60]},
{'Naha': [127.68, 26.21]},



訪問順を出力
都市A->都市B-> …

Sendai -> Fukuoka -> Naha ->
Osaka -> Yokohama



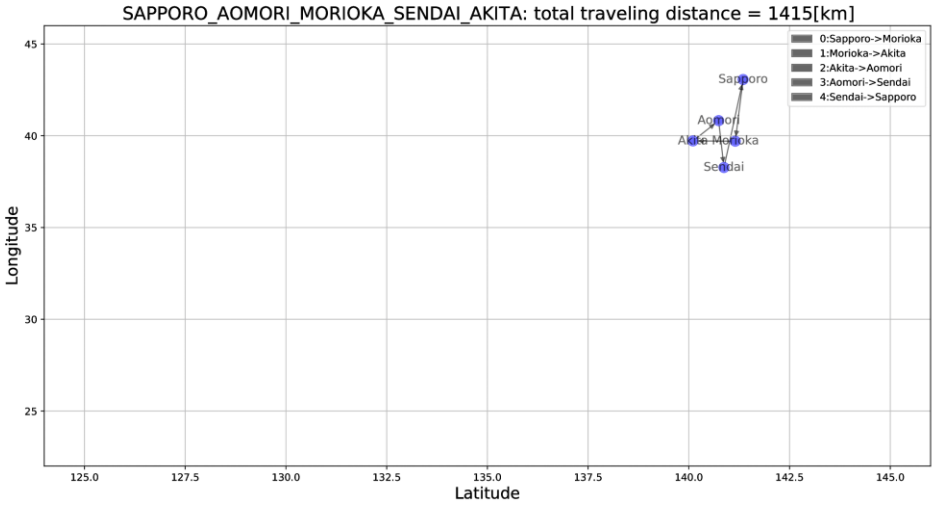
ストリームデータを TSP に適用 — 実験

【実験】ストリームデータに対して1都市ずつ追加/削除しながら5都市の TSP

実験環境：Laptop PC ローカル
入力データ：47都道府県の県庁所在地データ

{“都市名”: [緯度, 経度]}
{"Sapporo": [141.34694, 43.064170000000004]}
{"Aomori": [140.74, 40.82444]}
{"Morioka": [141.1525, 39.70361]}
{"Sendai": [140.87194, 38.26889]}
{"Akita": [140.1025, 39.71861]}
{"Yamagata": [140.36333, 38.240559999999995]}
{"Fukushima": [140.46778, 37.75]}
{"Mito": [140.44666999999998,
36.341390000000004]}
{"Utsunomiya": [139.88361, 36.56583]}
{"Maebashi": [139.06083, 36.39111]}
…
{"Naha": [127.68111, 26.2125]}

各 window の入力に対して最適経路を計算する



Sawatabi を用いたアプリケーション開発

TSP実装を通した、これまでのやり方と比較した Sawatabi の Pros/Cons

Pros

- 入力データ (windowなど) 処理のパイプラインの自前設計が**不要**
- スピン・相互作用のメタデータを利用した**自由度高い**モデル設計
  (例: 問題経路の途中変化への対応)
- 各ソルバとの**共通インタフェイス**
- Google Cloud Dataflow などのストリーム処理PFへ**デプロイ容易** + **Pub/Sub** の利用が可能

Cons

- 既存のアプリケーションがすでに存在する場合、sawatabi記述での再実装が必要

Sawatabiフレームワークを用いたアプリケーション開発 (2)

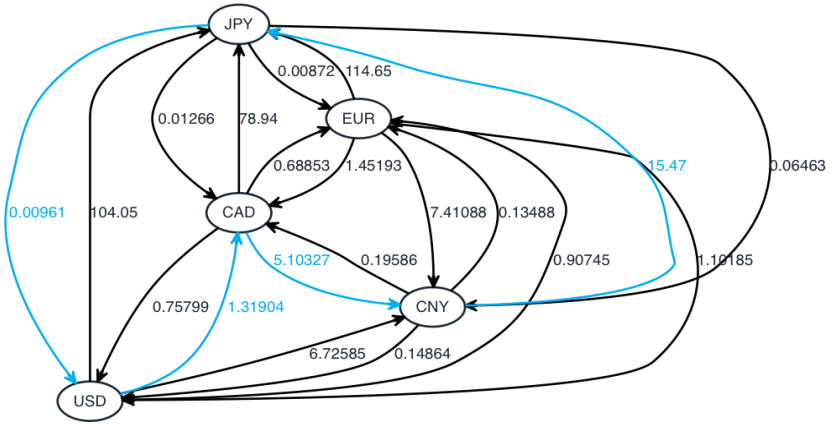
Arbitrage Opportunities Finding

Arbitrage Finding (1/3) – アプリケーションの概要

- 為替レートの価格の歪みを検出する問題（グラフアルゴリズムのサンプルとしても使われている）
- 為替レートは逐次更新されるのが自然なので、データが更新される例として取り上げた

①:データと解のサンプル

以下は[1]のFigure 1から抜粋したもの。青線の、JPY/USD->USD/CAD->CAD/CNY->CNY/JPYのパスで取引することによって0.074%の利益を得ることができる例になっている



②: 定式化

[1]の定式化にあるQUBO （下式）をそのまま使用

$$x = \mathrm{argmax}_x [\sum_{(i,j)\in E} x_{ij} \log c_{ij} - M_1 \sum_{i\in V} (\sum_{j,(i,j)\in E} x_{ij} - \sum_{j,(i,j)\in E} x_{ji})^2 - M_2 \sum_{i\in V} \sum_{j,(i,j)\in E} x_{ij} (\sum_{j,(i,j)\in E} x_{ij} - 1)]$$

argmax内の第一項は所与の為替レートの組合せにおいて取引戦略に取り入れた結果の対数収益率を表現する目的項、第二項・第三項は売買をリスクニュートラルにするための制約項

[1] G. Rosenberg, "Finding optimal arbitrage opportunities using a quantum annealer," 1QBit White Paper, 2016. https://1qbit.com/whitepaper/arbitrage/

Arbitrage Finding (2/3) – sawatabiによる実装

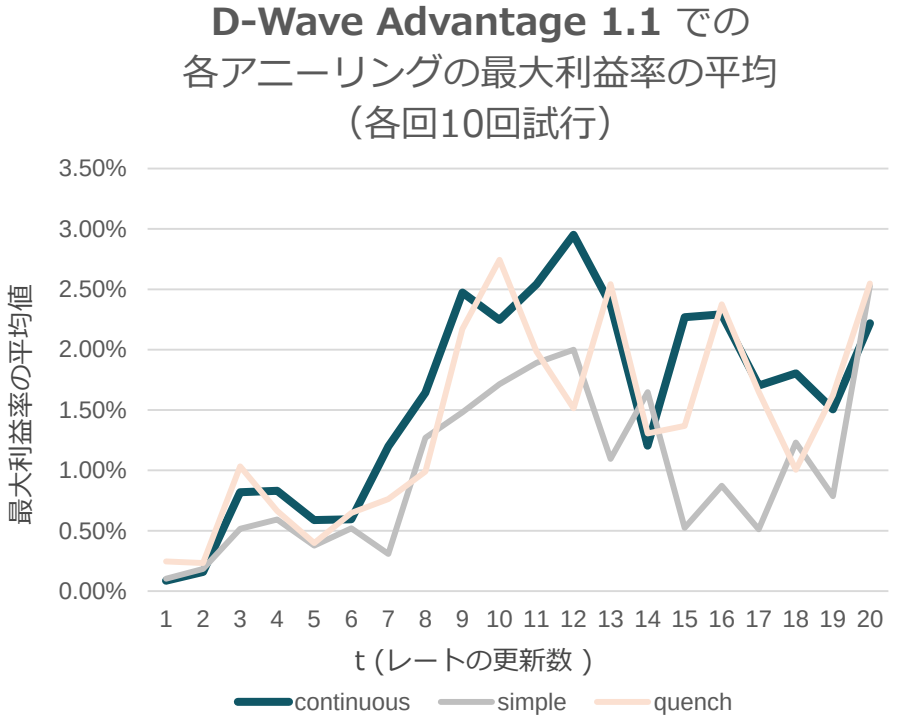
- Algorithm層でApache Beamの機能を使う例として実装
- Beam組み込みの関数やアプリケーション実装として必要な関数を組み合わせてデータのパイプラインを表現する

```
def run(argv=None):
    with beam.Pipeline() as p:
        (p |
         'Load input from memory' >> beam.Create(conversions_1qbit_example) |
         'To key value pair' >> beam.Map(lambda e: (1, e)) |
         'Update logical model and run annealing’ >> beam.ParDo(ArbFindingDoFn())|
         'Print the result of the annealing' >> beam.Map(print))
```

- 今回は為替レートの更新があるたびに論理モデルを更新し、D-Waveでアニーリングを実行するようにしたので、ArbFindingDoFnでモデルの更新とアニーリングの実行をおこなっている

Arbitrage Finding (3/3) – 初期解を与える効果

- Arbitrage findingのアプリケーションは変数の構造を固定して扱うことができるので、複数回のアニーリングのうち前回のアニーリングの結果をリバースアニーリングの初期解として与えることができる
- 前回の解を初期解として与える戦略をcontinuous, 線形のスケジュールをsimple, クエンチするスケジュールをquenchとした際に、平均値（ならびに中央値で比較すると）continuousが最も良い結果となった(※)



※ AR(1)を相関構造とするrepeated measures ANOVAでは、continuous, simple, quenchに差はみられなかった

まとめ

- **アニーリング計算をストリーム処理で実行するための**
  フレームワーク「Sawatabi」を開発しました
- **アプリケーション実装を通して次のことを確認しました**
  1. ストリーム処理エンジン上でストリームデータに対する
     継続的なアニーリング計算が可能であること
  2. ストリームデータの特徴を利用することで、既存の量子アニーリング
     での計算と比較してより質の良い解を得られる可能性があること

Thank you !