

量子コンピュータ時代のプログラミングセミナー

～ Fixstars Amplify で実装する経路最適化 ～



お知らせ

- 本日のスライドは後日配布いたします
- アンケートへのご協力をよろしくお願いいたします

本日のAgenda

- 本セミナーのゴール
- 会社および量子コンピューティングクラウドサービス「Fixstars Amplify」のご紹介
- Fixstars Amplify を用いた最適化ワークショップ
 - 経路最適化
- 事例・価格・今後の進め方等のご紹介

質問は随時 Zoom のチャットか Q&A でお願ひします

本セミナーのゴール

- 身の回りには組合せ最適化問題がたくさんあることを知る
- 組合せ最適化問題を解くための専用マシン（量子アニーリング・イジングマシン）があることを知り、解くための流れを理解する（決定変数、目的関数、制約条件など）
- ワークショップを通して、実際にイジングマシンを動かしてみることで、実問題への適用の足掛かりを得る

会社紹介

(株) Fixstars Amplify の紹介

- **組合せ最適化**のための量子コンピューティングクラウドプラットフォーム「Fixstars Amplify」の提供
- 2021年に設立（株式会社フィックスターズからスピンアウト）
 - 代表取締役社長CEO：松田 佳希（博士）
- 親会社 (株) フィックスターズ
 - 東証プライム市場上場
 - 約300名の従業員のうち、約9割がソフトウェアエンジニア
 - ソフトウェア高速化プロフェッショナル集団



Fixstars Amplify: 今までの歩み

次世代技術を先取りし
今ある課題の解決を目指す

2018年

NEDOのプロジェクトに採択
「イジングマシン共通ソフトウェア基盤の研究開発」

2017年

日本で初めて
D-Wave Systems社と提携

2019年

SIPの研究開発に参画
「光・量子を活用したSociety 5.0実現化技術：光電子情報処理」

2021年

量子アニーリングクラウドサービス「Fixstars Amplify」提供開始
子会社Fixstars Amplifyを設立
Q-STAR 量子技術による新産業創出協議会に特別会員として加入

2022年

Fixstars Amplify がGurobi、IBM-Quantumをサポート
累計実行回数1,000万回突破

2024年

産総研次世代スパコンABCI-Qへの採用
光量子コンピュータのクラウド化研究
登録組織数 700超

2025年

Fixstars Amplify Annealing Engine v1 リリース
登録組織数 1,000超
累計実行回数1億回突破

Fixstars Amplify: 製品ポートフォリオ

Fixstars Amplify (量子コンピューティングクラウド, 2021/2~)

- **汎用的な問題**に対応する SDK と実行環境 (AE)
- 非線形な最適化問題が得意
- シフト最適化、経路最適化、設計変数最適化向けに自分で**自由に定式化**した目的関数や制約条件の中で一番良い組み合わせを高速・高精度に探索
- 量子アニーリング・イジングマシンを活用してブラックボックス最適化問題を解くことも可能



本日のセミナーはこちら

Fixstars Amplify Scheduling Engine (スケジューリング最適化クラウド, 2023/9~)

- **スケジューリング問題に特化**した SDK と実行環境 (Scheduling Engine)
- 最適化の目的を Makespan の最小化に絞り、**定式化不要**で、複雑な制約条件を通常のプログラミングの延長で実装可能
- 生産計画・人員シフト計画・配送計画など幅広いスケジューリングに適用可能



様々な領域での利用拡大中（実稼働含む）

1000 を超える企業、研究所、大学

1億 を超える実行回数 (Amplify AE)



量子コンピューティングクラウド
サービス「Fixstars Amplify」
のご紹介

組合せ最適化問題

- 組合せ最適化問題とは、膨大な選択肢の中から、制約条件を満たし、ある評価値（目的関数値）が最もよくなる（最小 or 最大）解を求めること。量子アニーリング・イジングマシンは組合せ最適化問題を解くための専用マシン
- 組合せ最適化問題は、生産計画最適化や、シフト最適化、構造物の設計最適化、材料開発（MI）、エネルギーマネジメント最適化など、様々な領域に存在し、研究・開発や社会実装が積極的に進められている

Fixstars Amplify の活用事例

生産計画
最適化

材料開発
(MI)

経路最適化

人員
シフト
最適化

機械学習の
特徴量抽出

実験の
パラメータ
最適化

設計最適化

エネルギー
マネジメント
最適化

電力最適化

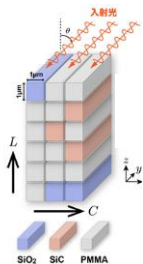


ユーザーインタビュー

2023年8月

東京大学

ブラックボックス最適化手法
(FMQA) の開発に成功



ユーザーインタビュー

2023年6月

住友商事株式会社

現場で愛されて育つ、量子コン
ピュータ技術活用



ユーザーインタビュー

2024年10月

マツダ株式会社

ブラックボックス最適化を活用
した車両設計最適化



組合せ最適化問題、量子技術、Fixstars Amplify の関係

1. 量子コンピュータ

(量子ゲート方式)

- 古典汎用コンピュータの上位互換。量子ゲートを操作。エラー訂正機能の無いNISQ型実機がクラウド利用可能
- QAOAにより**組合せ最適化問題 (QUBO)** を取り扱うことが可能
- 演算規模：～数100ビット

1.
量子
コンピュータ

IBM
Google
Rigetti
IonQ
...

2.
量子
アニーリング

D-Wave
NEC

3.
その他の
イジングマシン

Fixstars Amplify
TOSHIBA, Fujitsu
HITACHI, ...

3. その他のイジングマシン

(半導体技術に基づくイジングマシン)

- 二次の多変数多項式で表される目的関数の**組合せ最適化問題 (QUBO)** 専用マシン
- 統計物理学におけるイジング模型に由来。様々な実装により実現。
- 演算規模：
260,000+ビット (*Amplify AE*)

2. 量子アニーリング (量子焼きなまし法式のイジングマシン)

- イジングマシンの一種。量子イジング模型を物理的に搭載したプロセッサで実現。量子効果を物理的に調整し、自然計算により低エネルギー状態が出力
- **組合せ最適化問題 (QUBO)** を扱う専用マシン
- 演算規模：～数1,000ビット

組合せ最適化問題 (QUBO)

数理最適化問題

- 連続最適化問題
 - ・ 決定変数が連続値 (実数など)
- 決定変数が離散値 (整数など)
 - ・ 整数計画問題 (決定変数が整数)
 - ・ 0-1整数計画問題 (決定変数が二値)

QUBO目的関数 (0-1整数二次計画問題)

$$f(\mathbf{q}) = \sum_{i < j} Q_{ij} q_i q_j + \sum_i Q_{ii} q_i$$

f : 目的関数

q : 決定変数

Q : 係数

量子アニーリング・イジングマシン

Q uadratic	二次形
U nconstrained	制約条件なし
B inary	0-1整数 (二値)
O ptimization	計画 (最適化)



$f(\mathbf{q})$ を最小化するような \mathbf{q} を求める



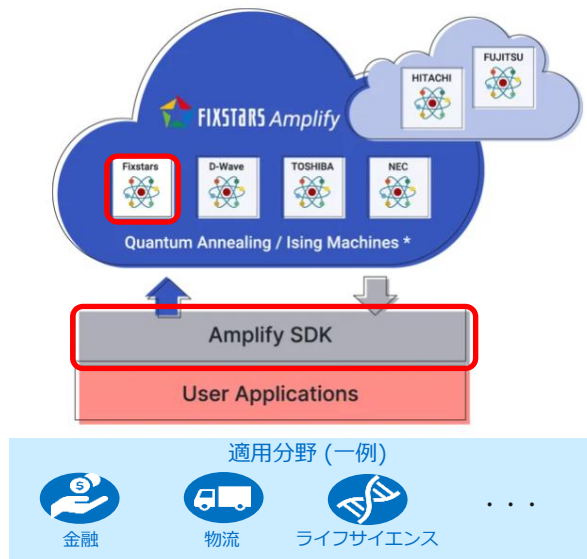
クラウドサービス : **Fixstars Amplify**

量子コンピューティングクラウド: Fixstars Amplify

- 量子コンピューティングを想定したシステム開発・運用のクラウドプラットフォーム
- 量子コンピュータやGPUベースのイジングマシンなど、組合せ最適化問題の専用マシンを効率的に実行できる

<https://amplify.fixstars.com/ja/>

サービス概要



簡単

- SDKをインストールするだけですぐに使える (pip install amplify)
- ハードウェアの専門知識不要でアプリケーションが開発できる

ポータブル

- すべての量子アニーリング/イジングマシンに対応
- Fixstars の26万ビット級のアニーリングマシン実行環境も利用可能

始めやすい

- 評価・検証用途には開発環境と実行環境が**無償で利用可能**
- 多くのチュートリアル、サンプルコードを整備・拡充

Fixstars Amplify の対応マシンの一例

アニーリングエンジン



量子アニーリング・イジングマシン

Fixstars Amplify AE

標準マシン

GPUの優れた並列計算能力を最大限に活用し、複雑な組合せ最適化問題を高速・高精度に解く革新的なアニーリングエンジンです。



外部ソルバー連携

標準マシン

Fixstars Amplifyからご利用申し込み可能なマシンです。

量子アニーリング・イジングマシン

D-Wave Systems
2000Q / Advantage

標準マシン

量子アニーリング・イジングマシン

東芝
デジタルソリューションズ
SQBM+

標準マシン

量子アニーリング・イジングマシン

富士通
デジタルアニーラ

標準マシン

BYOLマシン

自身の保有するライセンスを用いて Fixstars Amplify を利用出来ます。

量子アニーリング・イジングマシン

日立製作所
CMOSアニーリングマシン

数値最適化ソルバー

Gurobi
Gurobi Optimizer

ゲート式量子コンピュータ

IBM
IBM Quantum

量子回路シミュレータ

Qulacs
Qulacs

標準マシン は、

- ベンダ各社と個別マシン利用契約なし、
 - 評価・検証用ベーシックプランなら無料、
- で利用可能！ ←「いつでも」、「誰でも」

今後も幅広い対応マシンの追加が続々と行われる予定です！ ←「あらゆる」

オンラインデモ & チュートリアル

Amplify デモ

検索

<https://amplify.fixstars.com/ja/demo>



チュートリアル応用編

最適エネルギー管理 ト

プログラミング難易度 ★★★★★

本サンプルプログラムでは、ホーム・エネルギー・マネジメント・システム (HEMS) を想定し、種々のエネルギーコストや供給量に基づき、2日分の最適エネルギーミックスの探索を行います。

サンプルコード



チュートリアル応用編

ブラックボックス最適化 (1)

プログラミング難易度 ★★★★★

複雑で未知な目的関数にも適用可能な、機械学習と組み合わせ最適化を組み合わせたブラックボックス最適化手法を紹介し、Amplifyを用いて実装します。

サンプルコード



チュートリアル応用編

ブラックボックス最適化 (2)

プログラミング難易度 ★★★★★

機械学習と量子アニーリング・イジングマシンを活用するブラックボックス最適化の適用例として、疑似的な高温超伝導を実現する材料探索を取り扱います。

サンプルコード



チュートリアル応用編

ブラックボックス最適化 (3)

プログラミング難易度 ★★★★★

化学プラントにおける生産量を最大化するための運転条件最適化を行います。最適化には、機械学習モデルに基づくブラックボックス最適化と流体シミュレーションを用い、真の最適化を最大化するように異型の探索を行います。

サンプルコード



チュートリアル応用編

ブラックボックス最適化 (4)

プログラミング難易度 ★★★★★

流体機器設計に不可欠な異型の最適化問題を取り上げます。最適化には、組み合わせ最適化や機械学習に基づくブラックボックス最適化と流体シミュレーションを用い、真の最適化を最大化するように異型の探索を行います。

サンプルコード



デモアプリケーション

容量制約付き運搬経路問題 (CVRP)

プログラミング難易度 ★★★★★

運送業における効率的な配達計画の策定やごみ収集や道路清掃における巡回順序の最適化等での応用が期待される容量制約付き運搬経路問題 (CVRP) を取り扱います。

デモアプリ

サンプルコード



チュートリアル応用編

ブラックボックス最適化 (5)

プログラミング難易度 ★★★★★

ブラックボックス最適化により、高層ビル群による交通渋滞が発生し得る都市における、交通渋滞を低減するような信号制御の最適化を実施します。最適化の実施及び実証には、マルチエージェント・シミュレーションによる交通シミュレーションを用います。

サンプルコード



チュートリアル応用編

定式化による交通信号機の最適化

プログラミング難易度 ★★★★★

都市における渋滞を最小化するために、第一級に変化する交通状況に応じ、組合せ最適化を用いてリアルタイムに信号機の最適制御を実施します。また、その様な信号制御を実施した際の都市の交通量をシミュレーションします。

サンプルコード



チュートリアル応用編

10. 整数長ジョブスケジューリング問題

プログラミング難易度 ★★★★★

あらかじめ決まった数のジョブとマシンがあり、それぞれのジョブにかかる時間が分かっているとします。それぞれのジョブをいくつかのマシンに割り当てます。ジョブスケジューリング問題では、最も早く全てのジョブが完了するよう割り当て方を求めます。

サンプルコード



チュートリアル応用編

画像のノイズ除去

プログラミング難易度 ★★★★★

画像のノイズ除去を行うアプリケーションを開発します。

サンプルコード



チュートリアル応用編

会議室割当問題

プログラミング難易度 ★★★★★

制約条件を用いて定式化するアプリケーションの例として会議室割当問題のアプリケーションを開発します。

サンプルコード



チュートリアル応用編

タクシーマッチング問題

プログラミング難易度 ★★★★★

目的関数と制約条件を用いて定式化するアプリケーションの例としてタクシーマッチング問題のアプリケーションを開発します。

サンプルコード



デモアプリケーション

グラフ彩色問題

プログラミング難易度 ★★★★★

Fixstars Amplifyによる、グラフ彩色問題の定式化を体験します。

デモアプリ

サンプルコード



デモアプリケーション

巡回セールスマン問題

プログラミング難易度 ★★★★★

Fixstars Amplifyによる、巡回セールスマン問題の定式化を体験します。

デモアプリ

サンプルコード



デモアプリケーション

数独

プログラミング難易度 ★★★★★

Fixstars Amplifyによる、数独の定式化を体験します。

デモアプリ

サンプルコード



デモアプリケーション

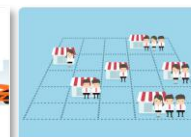
ライドシェア

プログラミング難易度 ★★★★★

集合型ライドシェアの最適化アプリケーションを体験します。

デモアプリ

サンプルコード



デモアプリケーション

タスク割当問題

プログラミング難易度 ★★★★★

店舗とタスクに従業員を割り当てる組合せ最適化問題のアプリケーションを体験します。

デモアプリ

サンプルコード



デモアプリケーション

ポートフォリオ最適化

プログラミング難易度 ★★★★★

リスクとリターンを考慮した株式ポートフォリオの最適化アプリケーションを体験します。

デモアプリ

サンプルコード

Fixstars Amplify の内容と特徴

- 開発環境 : Amplify SDK
- 実行環境 : Amplify Annealing Engine (AE)

開発環境 : Fixstars Amplify SDK

Fixstars Amplify SDK を使うと組合せ最適化問題の実装が圧倒的に直感的で簡単です

通常のプログラミング

1. 課題を定式化

マシンのSDKやAPI仕様に合わせて物理モデルをデータ化

2. 論理モデルへ変換

目的関数をマシンの動作モデルで再定義

3. 物理モデルへ変換

マシン仕様や制約を考慮した物理モデルに再変換

4. マシンにデータを入力

マシンのSDKやAPI仕様に合わせて物理モデルをデータ化

5. マシンを実行

特定マシンのみで実行可能

Fixstars Amplifyを用いたプログラミング

1. 課題を定式化

定式化された数式をプログラムコードで表現

SDKが提供するAPIが、自動で各マシンに合った形式へ多段変換をして入力。実行結果は逆変換をして、ユーザーにとって結果の解釈が容易な形式で返却されます。

2. マシンを実行

複数マシンの中から選択可能

```
# 決定変数の発行
q = VariableGenerator().array("Binary", 2)

# 目的関数と制約条件の定義
objective = q[0] - 2 * q[0] * q[1]
constraint = one_hot(q)

# モデルの構築
model = Model(objective, constraint)

# ソルバーの指定
client = AmplifyAECClient()
client.token = "Amplify AE のアクセストークン"
client.parameters.time_limit_ms = timedelta(seconds=1)

# 最適化の実行
result = solve(model, client)

# 結果の表示
print(q.evaluate(result.best.values))
print(objective.evaluate(result.best.values))
```

Amplify AE v1
(2025/10/17 リリース)

Fixstars Amplify SDK によるシンプルプログラミング

数独を解くサンプルアプリ

SDKなし
最適化しても
200行以上

SDKあり
30行程度

5	3		7					
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

出典:
Wikipedia

富士通・デジタルアニーラの設定用コード

SDKなし
59行

SDKあり
1行

日立CMOSアニーリングマシンの設定用コード

SDKなし
183行

SDKあり
1行

SDKが、各マシンに対して最適な形式に実装式を多段変換！

実行環境 : Fixstars Amplify Annealing Engine (AE)

NVIDIA GPU V100/A100/H100 で動作

- 独自の並列化シミュレーテッド
アニーリングアルゴリズム

WEB経由で計算機能を提供

- 社会実装・PoC・検証が加速
- Amplify SDK の実装を直ぐに実行可能

商用マシンでは最大規模・最高速レベル

- 120,000 ビット (全結合)
- 260,000 ビット超 (疎結合)

	標準マシン Fixstars Amplify AE	標準マシン D-Wave 2000Q/Advantage	標準マシン 東芝 SQBM+	日立 CMOS Annealing	富士通 Digital Annealer
装置型式	GPU	量子回路	GPU	デジタル回路	デジタル回路
最大ビット数	<u>262,144以上</u>	2,048 (16x16x8)/ 5,760 (16x15x24)	100,000 (SQBM+)/ 10,000 (SBM PoC 版)	61,952 (352x176)	8,192 (DA2)/ 100,000 (DA3)
係数パラメータ	デジタル (32/64bit)	アナログ (5bit程度)	デジタル (32bit)	デジタル (3bit)	デジタル (16/64 bit)
結合グラフ	全結合	キメラグラフ/ ペガサスグラフ	全結合	キンググラフ	全結合
全結合換算ビット 数	131,072	64/124	31,000程度 (SQBM+) ^(*) / 1,000 (SBM PoC 版)	176	8,192 (DA2)/ 100,000 (DA3)
APIエンドポイン ト	Fixstars Amplify	D-Wave Leap	Fixstars Amplify / AWS	Annealing Cloud Web	DA Cloud

Fixstars Amplify AE パフォーマンス

Amplify AE v1
(2025/10/17 リリース)

Fixstars Amplify AEは、**最速・最高精度**の求解を達成しています

A社ソルバーより

18 倍 高速

に求解可能*

B社ソルバーより

217 倍 高速

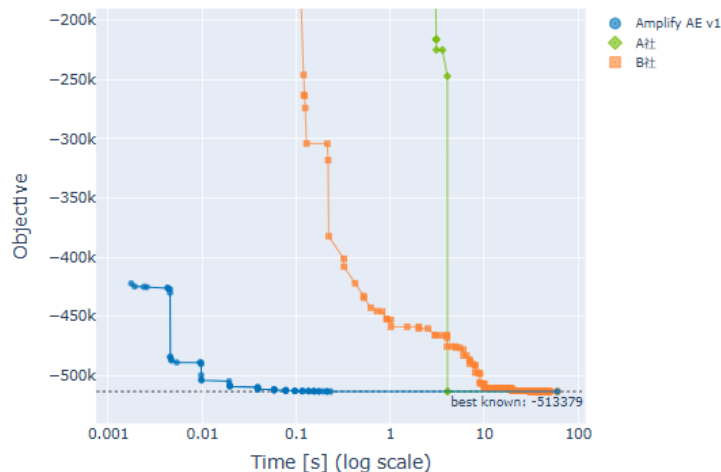
に求解可能*

通常のナップサック問題は、荷物の価値と重みが与えられて、重みが一定の値に収まるような荷物の選び方のうち価値の総和が最大になる選び方を求める。これに対しQKPでは荷物のペアで追加の価値が生じることを考慮する、より難しい問題。

ベンチマークデータは、それぞれ11回実行した中央値をプロット。

* 各社ソルバーが最適解に到達するまでの時間で比較。

2次ナップサック問題



最適解への到達時間



Fixstars Amplify AE パフォーマンス

Amplify AE v1
(2025/10/17 リリース)

Fixstars Amplify AEは、**最速・最高精度**の求解を達成しています

A社ソルバーより

243 倍 高速

に求解可能*

最適解に到達するまでの時間

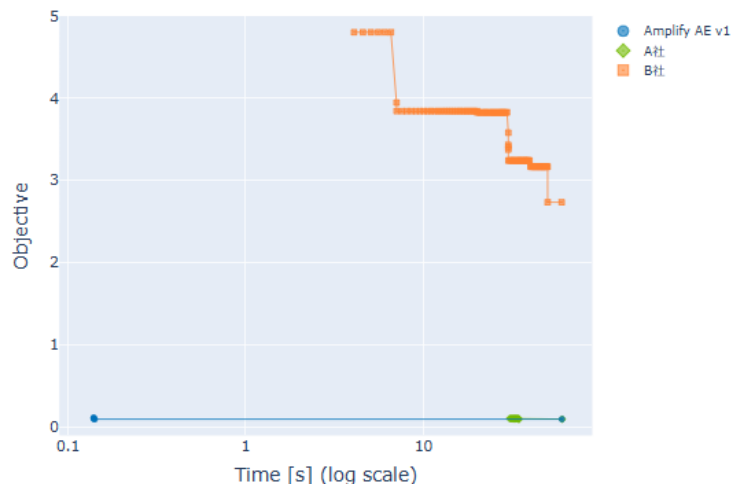
0.14 秒

工場の生産スケジュールにおける生産コストを最小化する業務課題。2次の目的関数と933個の線形不等式制約を含む。

ベンチマークデータは、それぞれ11回実行した中央値をプロット。

* 最良解を得るのに必要とした時間で比較。

実際の生産計画最適化



最良解の計算時間



ワークショップ

～事前準備～

ワークショップの事前準備 (1)

- ご自身の PC のブラウザ上で Python のプログラミングを行います。Google Colaboratory を使うので、事前に Google Colaboratory にログインできることをご確認ください (Google アカウントが必要です)。

Google Colab 検索

<https://colab.research.google.com/>

- Fixstars Amplify の無料トークンの取得有無をご確認ください。まだの方は、[こちら](#) からユーザー登録をして無料トークンを取得してください (1分で完了します)。

Fixstars Amplify 検索

<https://amplify.fixstars.com/>



ワークショップの事前準備 (2)

取得された Fixstars Amplify AE の無料トークンを用いてトークンチェック用のサンプルコードが動くか、以下のステップでご確認をお願いします。

1. 以下の URL にアクセスしてください。サンプルコードは閲覧のみ可能な状態なので、「ファイル」→「ドライブにコピーを保存」して、ご自身の Google ドライブにコピーを作成してください。
<https://colab.research.google.com/drive/1xdrIVKEK0ndQYMMIhjGXFb3yIw4Ci05M>
2. コピーしたファイルの1番目のセルにご自身の無料トークンを入力してください（***印の部分を書き換えてください）。ご自身の無料トークンは、「アクセストークン」ページの「Fixstars Amplify AE」のセクションでご確認いただけます。トークンを入力後、再生ボタンまたは Shift +Enter で1番目のセルを実行して下さい。

```
token = """*****""" # ご自身のトークンを入力
```

3. 1番目のセルの実行が完了したら、2番目のセルも再生ボタンまたは Shift + Enter で実行してください。実行後、以下の結果が出力されればOKです。

```
result: [q_0, q_1] = [1. 1.] (f = 0.0)
```



ワークショップの事前準備 (3)

- ワークショップで使うサンプルコードを以下のURLより取得して下さい
- それぞれのサンプルコードにご自身のトークンを入力いただく必要があります。それぞれのサンプルコードを「ドライブにコピー」の上、トークンを入力し実行して下さい

▶ サンプルコード

Step1	https://colab.research.google.com/drive/198AxYGkbqEEK-Ol4sWBc92zsRkpWI3Uu
Step2	https://colab.research.google.com/drive/1-7Goe4wrY6sQNzXVzuXc4cr5sUCOlplL4
Step3	https://colab.research.google.com/drive/1MJr_Uws9HlyfUYoE0AEGMz6LkzK_hL50

質問は随時 Zoom のチャットか Q&A でお願ひします

ワークショップ

～ 経路最適化問題 ～

搬送経路最適化

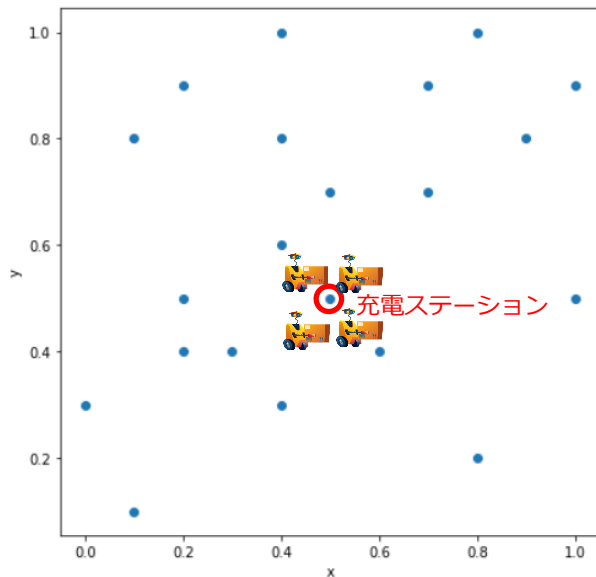
一言に「経路最適化」と言っても、搬送ロボット (AGV) が運用されている環境や要件などは様々で、それぞれに対する問題設定や定式化は異なります

状況や要件の例

- AGVが通過できる経路が定められている
- AGVごとに稼働時間に制限がある
- AGVごとに積載量に制約がある
- 配達地ごとに要求される到着時間がある
- AVGの衝突回避の制御が必要
- 運ぶ材料や部品によって各配達地で作業時間が異なる

搬送経路最適化

【問題】工場内の4台のAGVを用いて、AGVの充電ステーションから20か所の搬送先に部品を届けるときの各搬送ロボットの走行距離の合計が最短になるような経路を求めます。尚、搬送ロボットをできるだけ平均的に利用したいため各AGVの搬送先は5か所とします



搬送経路最適化

【問題】工場内の4台のAGVを用いて、AGVの充電ステーションから20か所の搬送先に部品を届けるときの各搬送ロボットの走行距離の合計が最短になるような経路を求めます。尚、搬送ロボットをできるだけ平均的に利用したいため各AGVの搬送先は5か所とします

全てを一度にやるのは難しいので、3つのステップに分けてプログラムを作成します

Step1

AGVを1台とし、制約を守るだけのプログラムを作ります

制約①: 各順番で回れる訪問先 (充電ステーション+各搬送先) は1か所
制約②: 各訪問先を回るのは1回のみ

解の候補多数あり

Step2

Step1に「総距離の最小化」という目的を追加し、複数の解の候補から目的を実現する解を求めるプログラムを作ります

Step3

AGVを4台に増やします。具体的には、Step2に対して以下の2つの変更を加えます

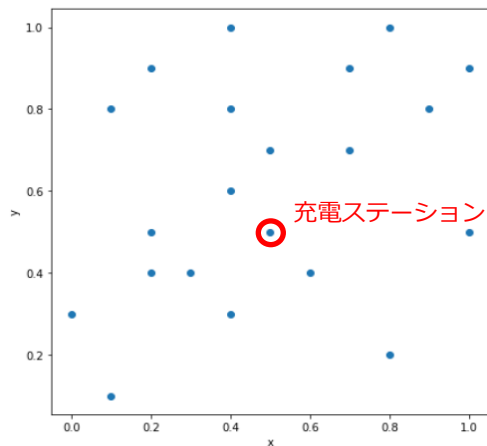
変更①: 決定変数の一部を0/1で固定
変更②: 制約条件の対象範囲を微調整

距離行列の作成

各訪問先の座標

```
[0.5 0.5]
[0.  0.3]
[0.3 0.4]
[0.2 0.5]
[0.4 0.8]
[0.5 0.7]
[0.2 0.4]
[0.2 0.9]
[0.7 0.7]
[0.6 0.4]
[0.8 1. ]
[0.1 0.1]
[0.1 0.8]
[0.4 0.6]
[0.8 0.2]
[0.7 0.9]
[0.4 0.3]
[0.4 1. ]
[0.9 0.8]
[1.  0.5]
[1.  0.9]
```

充電ステーション



実装

```
# 距離行列の作成
all_diffs = np.expand_dims(locations, axis=1) - np.expand_dims(locations, axis=0)
distances = np.sqrt(np.sum(all_diffs**2, axis=-1))
print(distances)
```

各訪問先間の距離行列

訪問先 (0) は充電ステーション

(km)	訪問先 (j)	0	1	2	...	20
訪問先	0	0	0.54	0.22	...	0.64
(i)	1	0.54	0	0.32	...	1.17
	2	0.22	0.32	0	...	0.86
	⋮	⋮	⋮		...	⋮
	⋮	⋮	⋮		...	⋮
	20	0.64	1.17	0.86	...	0

Step1

AGVを1台とし、制約を守るだけのプログラムを作ります
制約①: 各順番で回れる訪問先 (充電ステーション+各搬送先) は1か所
制約②: 各訪問先を回るのは1回のみ

決定変数を準備

順番(n)	訪問先 (i)				
	0	1	2	...	20
0	1	$q_{0,1}(0 \text{ or } 1)$	$q_{0,2}(0 \text{ or } 1)$...	$q_{0,20}(0 \text{ or } 1)$
1	$q_{1,0}(0 \text{ or } 1)$	$q_{1,1}(0 \text{ or } 1)$	$q_{1,2}(0 \text{ or } 1)$...	$q_{1,20}(0 \text{ or } 1)$
2	$q_{2,0}(0 \text{ or } 1)$	$q_{2,1}(0 \text{ or } 1)$	$q_{2,2}(0 \text{ or } 1)$...	$q_{2,20}(0 \text{ or } 1)$
...
...
20	$q_{20,0}(0 \text{ or } 1)$	$q_{20,1}(0 \text{ or } 1)$	$q_{20,2}(0 \text{ or } 1)$...	$q_{20,20}(0 \text{ or } 1)$

イジングマシン
で最適な(0,1)の
組合せを探す



Binary の PolyArray
 $21 \times 21 = 441$ [qubit]
1: 訪問する
0: 訪問しない

実装

```
from amplify import VariableGenerator

gen = VariableGenerator() # 変数のジェネレータを宣言
q = gen.array("Binary", shape=(n_visits, n_locations)) # 決定変数を作成
q[0, 0] = 1 # 出発点 (充電ステーション) を表す決定変数に1を代入
print(q)
```

得られる解のイメージ

順番(n)	訪問先 (i)				
	0	1	2	...	20
0	1	0	0	...	0
1	0	0	0	...	1
2	0	1	0	...	0
...
...
20	0	0	1	...	0

充電ステーション
から出発を示す

充電ステーションの次
に行くのは訪問先 (20)
であることを示す

Step1

AGVを1台とし、制約を守るだけのプログラムを作ります

制約①: 各順番で回れる訪問先 (充電ステーション+各搬送先) は1か所

制約②: 各訪問先を回るのは1回のみ

制約の定式化

制約1: 各順番で回れる訪問先は1か所

→ one hot 制約

$$\sum_{i=0}^{20} q_{n,i} = 1 \quad n \in \{0,1,\dots,20\}$$

制約2: 各訪問先を回るのは1回のみ

→ one hot 制約

$$\sum_{n=0}^{20} q_{n,i} = 1 \quad i \in \{0,1,\dots,20\}$$

順番(n)	訪問先 (i)					
	0	1	2	...	20	
0	1	0	0	...	0	→ one_hot
1	0	0	0	...	1	→ one_hot
2	0	1	0	...	0	→ one_hot
⋮	⋮	⋮	⋮	⋮	⋮	
20	0	0	1	...	0	→ one_hot

↓ one_hot ↓ one_hot ↓ one_hot ↓ one_hot

Step1

AGVを1台とし、制約を守るだけのプログラムを作ります

制約①: 各順番で回れる訪問先 (充電ステーション+各搬送先) は1か所

制約②: 各訪問先を回るのは1回のみ

制約の定式化

制約1: 各順番で回れる訪問先は1か所

→ one hot 制約

$$\sum_{i=0}^{20} q_{n,i} = 1 \quad n \in \{0,1,\dots,20\}$$

制約2: 各訪問先を回るのは1回のみ

→ one hot 制約

$$\sum_{n=0}^{20} q_{n,i} = 1 \quad i \in \{0,1,\dots,20\}$$

実装

```
from amplify import one_hot, ConstraintList

# 制約条件1 各順番で回れる訪問先は1つ
row_constraints = ConstraintList()
for i in range(n_visits):
    row_constraints += one_hot(q[i])

# row_constraints = one_hot(q, axis=1)

# 制約条件2 各訪問先を回るのは1回のみ
col_constraints = ConstraintList()
for i in range(n_locations):
    col_constraints += one_hot(q[:, i])

# col_constraints = one_hot(q, axis=0)

# 制約条件
constraints = row_constraints + col_constraints
```

こう書くこともできる

Step1

AGVを1台とし、制約を守るだけのプログラムを作ります

制約①: 各順番で回れる訪問先 (充電ステーション+各搬送先) は1か所

制約②: 各訪問先を回るのは1回のみ

求解

- 先ほど作った constraints (制約) を model に格納して、マシンに投げます
- 制約条件だけを与えた場合、制約条件を満たす解を探してきてくれます

```
#####  
# 求解  
#####  
  
from amplify import AmplifyAECient, Model, solve  
  
# 実行マシクライアントの設定  
client = AmplifyAECient()  
client.token = token  
client.parameters.time_limit_ms = 1 * 1000 # タイムアウト1秒 (1000m秒)  
  
# モデル化  
model = Model(constraints)  
  
# アニーリングマシンの実行  
result = solve(model, client)
```

Amplify AE

無料版は1ジョブ10秒まで設定可
有料版では1分～15分まで設定可能

Step1

AGVを1台とし、制約を守るだけのプログラムを作ります

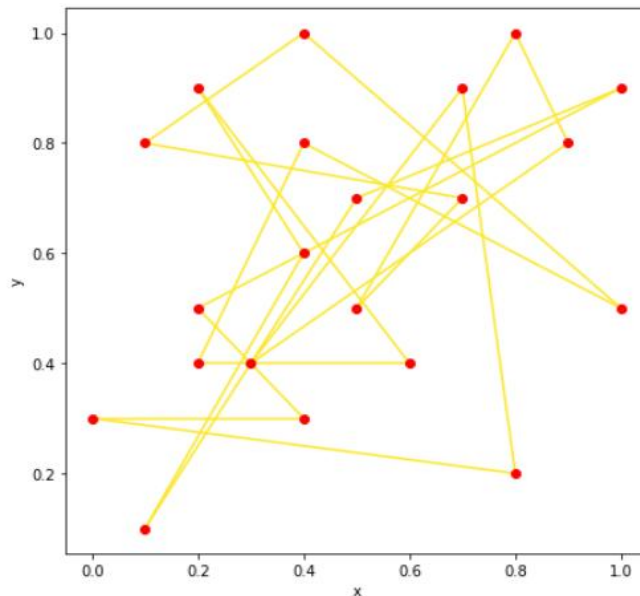
制約①: 各順番で回れる訪問先 (充電ステーション+各搬送先) は1か所

制約②: 各訪問先を回るのは1回のみ

結果の取得

```
#####  
# 結果の取得  
#####  
  
q_solutions = q.evaluate(result.best.values)  
print(q_solutions)
```

可視化



最短経路長: 11.7604

各順番で回れる訪問先は1つ、各訪問先を回るのは1回のみ、という二つの制約を満たす経路を求めることができました。解の候補がたくさんある状況ですので、次のStepでは、目的を追加して解を絞ります

Step2

Step1に「総距離の最小化」という目的を追加し、複数の解の候補から目的を実現する解を求めるプログラムを作ります

目的関数の定式化

総距離の最小化

$$distance = \sum_{n=0}^{20} \sum_{i=0}^{20} \sum_{j=0}^{20} d_{i,j} \cdot q_{n,i} \cdot q_{n+1,j}$$

決定変数 × 決定変数 (2次)

計算のイメージ

n=1のときi=20が1、n=2のときj=1が1なので、 $d_{20,1}$ である1.17がdistanceに加算される

$distance =$

各訪問先間の距離行列 ($d_{i,j}$)

(km)	訪問先 (j)					
	0	1	2	...	20	
訪問先 (i)	0	0	0.54	0.22	...	0.64
1	0.54	0	0.32	...	1.17	
2	0.22	0.32	0	...	0.86	
...
20	0.64	1.17	0.86	...	0	



決定変数 ($q_{n,i}, q_{n+1,j}$)

		訪問先 (i)					
		0	1	2	...	20	
順番(n)	0	1	0	0	...	0	0
	1	0	0	0	...	1	1
	2	0	1	0	...	0	0

	20	0	0	1	...	0	0

Step2

Step1に「総距離の最小化」という目的を追加し、複数の解の候補から目的を実現する解を求めるプログラムを作ります

実装

```
#####  
# 目的関数  
#####  
  
from amplify import Poly  
  
distance = Poly()  
for n in range(n_visits):  
    for i in range(n_locations):  
        for j in range(n_locations):  
            distance += distances[i, j] * q[n, i] * q[(n + 1) % n_visits, j]
```

剰余を使って、最後の訪問先の次は0番目に戻るようにしています

イジングマシンは、制約を満たす中で、このdistance (目的) を最小にする組合せを探します

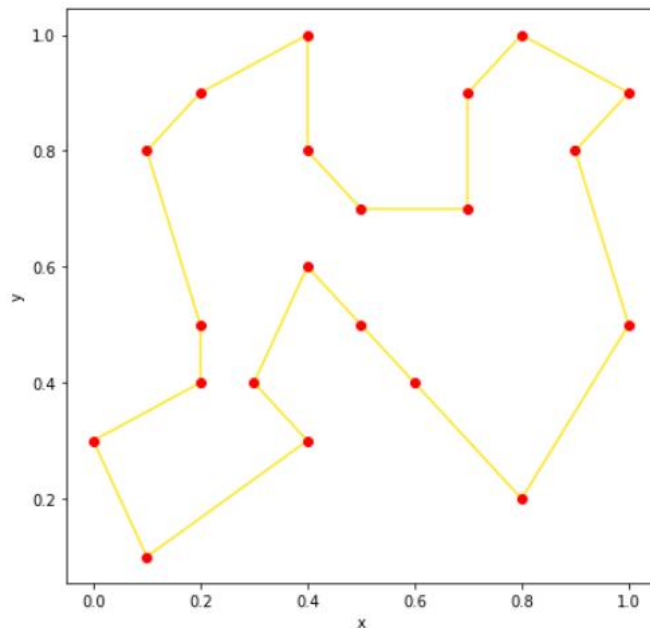
Step1 と同様

```
#####  
# 制約条件  
#####  
  
from amplify import one_hot  
  
# 制約条件1 各順番で回れる訪問先は1つ  
row_constraints = one_hot(q, axis=1)  
  
# 制約条件2 各訪問先を回るのは1回のみ  
col_constraints = one_hot(q, axis=0)  
  
# 制約条件  
constraints = row_constraints + col_constraints
```

```
# モデル化  
model = Model(distance, constraints)  
  
# アニーリングマシンの実行  
result = solve(model, client)
```

Step2

Step1に「総距離の最小化」という目的を追加し、複数の解の候補から目的を実現する解を求めるプログラムを作ります



最短経路長: 4.4444

2つの制約を満たした上で、総距離を最小化する経路が求められるようになりました。次のStepでは、AGVを4台に増やします。

Step3

AGVを4台に増やします ($n_vehicle = 4 \rightarrow n_visits = 24$)

変更①: 決定変数の一部を0/1で固定します。これにより、各AGVの搬送先を5か所に固定しています


決定変数

順番(n)	訪問先 (i)	0	1	2	...	20
0		1	0	0	...	0
1		0	q_1,1	q_1,2	...	q_1,20
2		0	q_2,1	q_2,2	...	q_2,20
3		0	q_3,1	q_3,2	...	q_3,20
4		0	q_4,1	q_4,2	...	q_4,20
5		0	q_5,1	q_5,2	...	q_5,20
6		1	0	0	0	0
7		0	q_7,1	q_7,2	...	q_7,20
8		0	q_8,1	q_8,2	...	q_8,20
9		0	q_9,1	q_9,2	...	q_9,20
10		0	q_10,1	q_10,2	...	q_10,20
11		0	q_11,1	q_11,2	...	q_11,20
12		1	0	0	0	0
13		0	q_13,1	q_13,2	...	q_13,20
	
	
	
23		0	q_23,1	q_23,2	...	q_23,20

充電ステーションには、0番目、6番目、12番目、18番目に訪問します (計4回訪問)

これらの中で
合計が最短と
なる経路を探
します

実装

 : 追加 or 変更部分

```
#####  
# 決定変数の作成  
#####  
  
from amplify import VariableGenerator  
  
gen = VariableGenerator() # 変数のジェネレータを宣言  
q = gen.array("Binary", shape=(n_visits, n_locations)) # 決定変数を作成  
  
if len(locations) // n_vehicle <= 1:  
    raise RuntimeError(f"n_vehicle is too large: {n_vehicle}")  
  
def split_number(number, divisor):  
    quotient, remainder = divmod(number, divisor)  
    return [quotient + 1] * remainder + [quotient] * (divisor - remainder)  
  
i = 0  
q[:, 0] = 0  
for j in split_number(n_visits, n_vehicle):  
    q[i] = 0  
    q[i, 0] = 1  
    i += j
```

Step3

AGVを4台に増やします ($n_vehicle = 4 \rightarrow n_visits = 24$)


変更②: 制約条件の対象範囲を微調整します。訪問先 (0) である充電ステーションを、one_hot制約の対象から外します

決定変数

one_hot制約対象外

順番 (n)	訪問先 (i)	0	1	2	...	20
0	0	1	0	0	...	0
1	0	q_1,1	q_1,2	q_1,20
2	0	q_2,1	q_2,2	q_2,20
3	0	q_3,1	q_3,2	q_3,20
4	0	q_4,1	q_4,2	q_4,20
5	0	q_5,1	q_5,2	q_5,20
6	1	0	0	0	...	0
7	0	q_7,1	q_7,2	q_7,20
8	0	q_8,1	q_8,2	q_8,20
9	0	q_9,1	q_9,2	q_9,20
10	0	q_10,1	q_10,2	q_10,20
11	0	q_11,1	q_11,2	q_11,20
12	1	0	0	0	...	0
13	0	q_13,1	q_13,2	q_13,20
	
	
	
23	0	q_23,1	q_23,2	q_23,20

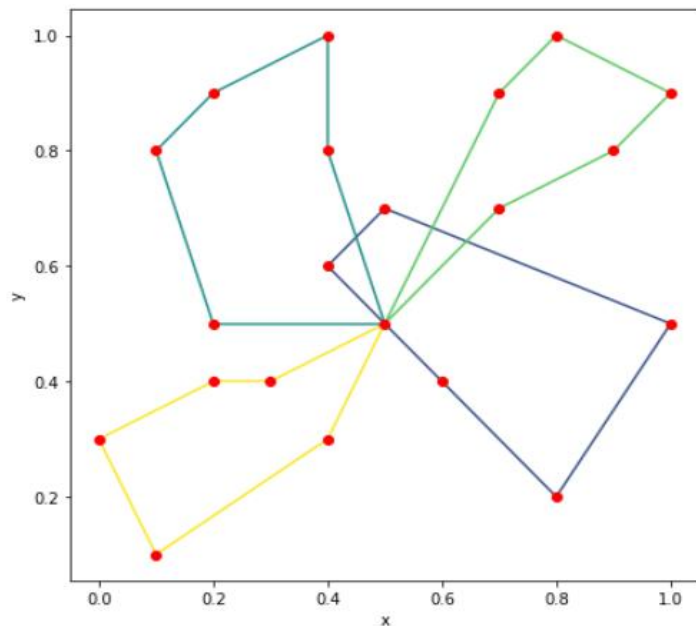
実装

 : 追加 or 変更部分

```
#####  
# 制約条件  
#####  
  
from amplify import one_hot  
  
# 制約条件1 各順番で回れる訪問先は1つ  
row_constraints = one_hot(q, axis=1)  
  
# 制約条件2 各訪問先を回るのは1回のみ  
col_constraints = one_hot(q[:, 1:], axis=0)  
  
# 制約条件  
constraints = row_constraints + col_constraints
```

Step3

AGVを4台に増やします ($n_vehicle = 4 \rightarrow n_visits = 24$)

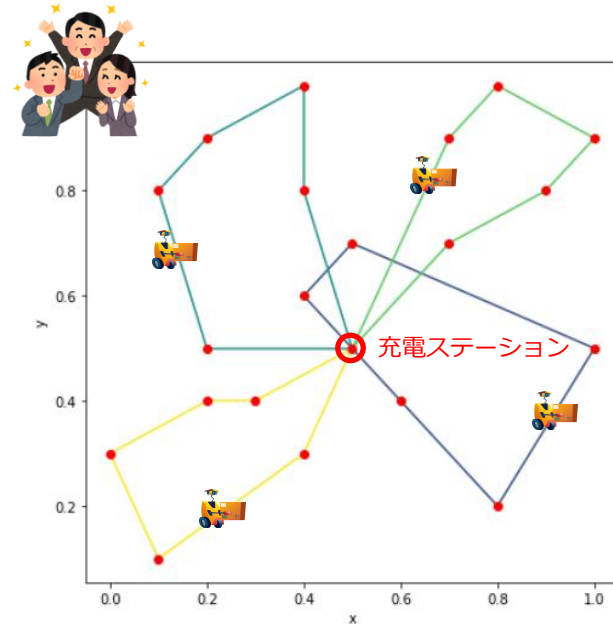
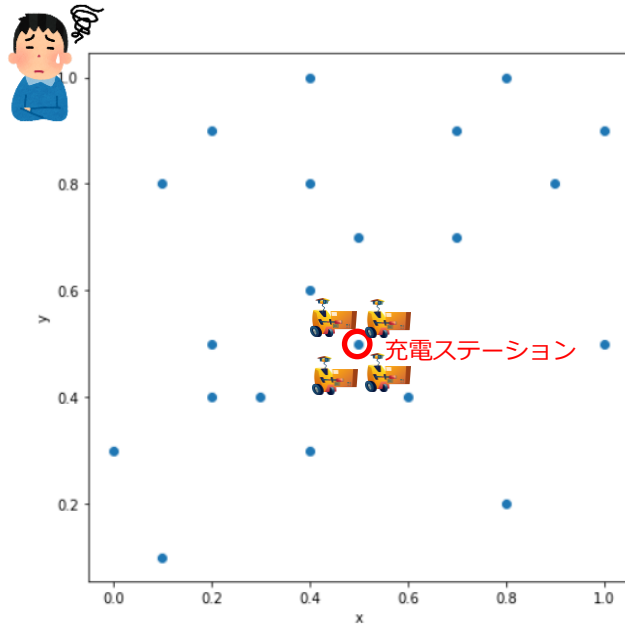


最短経路長: 5.9188

4台のAGVそれぞれが5か所ずつ搬送し、総走行距離が最短になるような経路を求める事ができました！

ワークショップ : おさらい

AGV1台の制約のみから始めて、目的を追加し、最終的には、4台のAGVそれぞれが5か所ずつ搬送し総走行距離が最短になるような経路を求めることができました！



事例・価格・今後の進め方のご紹介

シフト最適化の事例：物流倉庫の最適配置

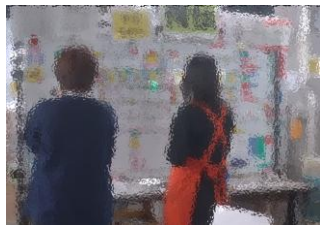
<https://www.fixstars.com/ja/services/cases/amplify-bellemaison>

業務内容：

梱包業務担当者（1チーム3名）を
コンベア前のブースに割り当て

従来の方法：

前日夕方に、翌日の予測出荷目標数と
出勤予定に基づいて、3人程度のリー
ダーが相談し数時間をかけて決定



課題：

公平になるよう、様々な配慮を行う必
要があり、割り当て担当者に心理的負
担がかかっていた



成果： → 2022年10月より実稼働開始！

アニーリング技術を活用して自動化・デジタル化

- 作業時間 → 15分程度に
- 心理負担 → ほぼゼロに



手動配置

一部事前配置
自動配置結果の微調整

自動配置 (アニーリング)

各種条件を満たす形で
未配置のメンバーを一
括割り当て

割当業務の
時間削減

担当者の
心理的負担低減

配置情報の
デジタル化

Smileboard
Connectと連携

国家プロジェクトSIP「光・量子を活用したSociety 5.0実現化技術」の一環として、住友商事、SCSK、ベルメゾンロジスコと、2019年より共同研究



車体構造の設計最適化 (マツダ株式会社)

<https://amplify.fixstars.com/ja/customers/interview/vehicle-co-optimization>

- 複数車種の車体内部の各部位に異なる厚さの鋼板を割り当てる問題（設計変数が計222個の大規模な問題）。衝突性能などの品質特性の制約条件を満たした上で、**部品の軽量化**と3車種間の**共通部品数の最大化**の実現する**多目的最適化問題**
- 2017年にマツダがベンチマーク問題として一般公開し*1、進化学会でのコンペ*2や英国オックスフォード大学や米国Meta社の研究者によるベイズ最適化に基づく解法*3など様々な手法が試されてきており、**1~3万回程度の試行**を繰り返せばある程度よい解が見つけれられることは確認されていた
- 2023年より量子アニーリング・イジングマシを活用したブラックボックス最適化（FMQA）に着目し、検証を開始

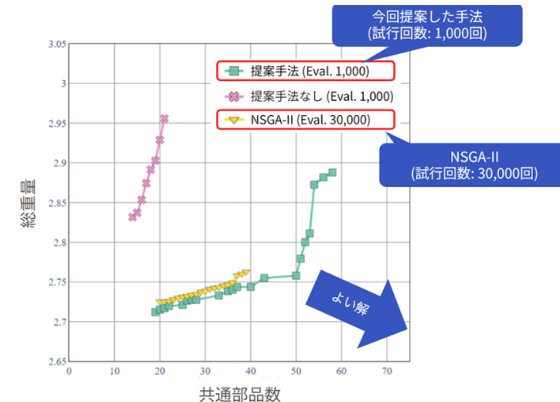
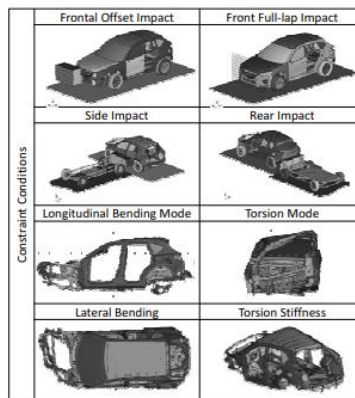


- 量子アニーリング・イジングマシを活用した FMQA により、従来の代表的な手法の**1/30の1,000回**の試行で同等以上の解を見つけることに成功！
- 今後は、QUBO 式近似の計算コストを削減しつつ、最適化性能も向上させるような手法の検討を予定

*1 応答曲面法を用いた複数車種の同時最適化ベンチマーク問題の提案

*2 進化計算コンペティション2017開催報告

*3 [Multi-objective Bayesian optimization over high-dimensional search spaces](#)



Fixstars Amplify ユーザーインタビュー

Amplify インタビュー 検索

amplify.fixstars.com/ja/customers/interview

・ 業務・研究開発利用



ユーザーインタビュー 2024年10月

マツダ株式会社

ブラックボックス最適化を活用した車両設計最適化



開発事例 2024年5月

日本テレビ放送網

数理最適化技術で地上波テレビの広告取引を最適化



ユーザーインタビュー 2023年6月

住友商事株式会社

現場で愛されて育つ、量子コンピュータ技術活用



ユーザーインタビュー 2023年11月

株式会社アパルデータ

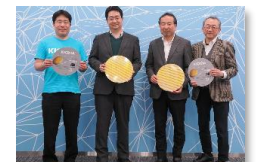
生産計画の属人化解消と設備投資計画への活用を目指して



パートナー事例 2022年10月

通販向け物流倉庫の人員最適配置自動作成サービス

住友商事株式会社と、物流倉庫での実運用を開始



ユーザーインタビュー 2024年6月

会津大学・東京工業大学・キオクシア株式会社

半導体製造におけるマスク最適化に量子アニーリング・イジングマシンを活用



ユーザーインタビュー 2023年8月

東京大学

ブラックボックス最適化手法 (FMQA) の開発に成功



有識者インタビュー 2023年2月

慶應義塾大学

量子コンピュータの活用を促進する「量子バイリンガル」というキャリアデザイン



ユーザーインタビュー 2022年11月

慶應義塾大学

Fixstars Amplifyを使って、次世代技術を活用した高速な解析手法の開発に成功



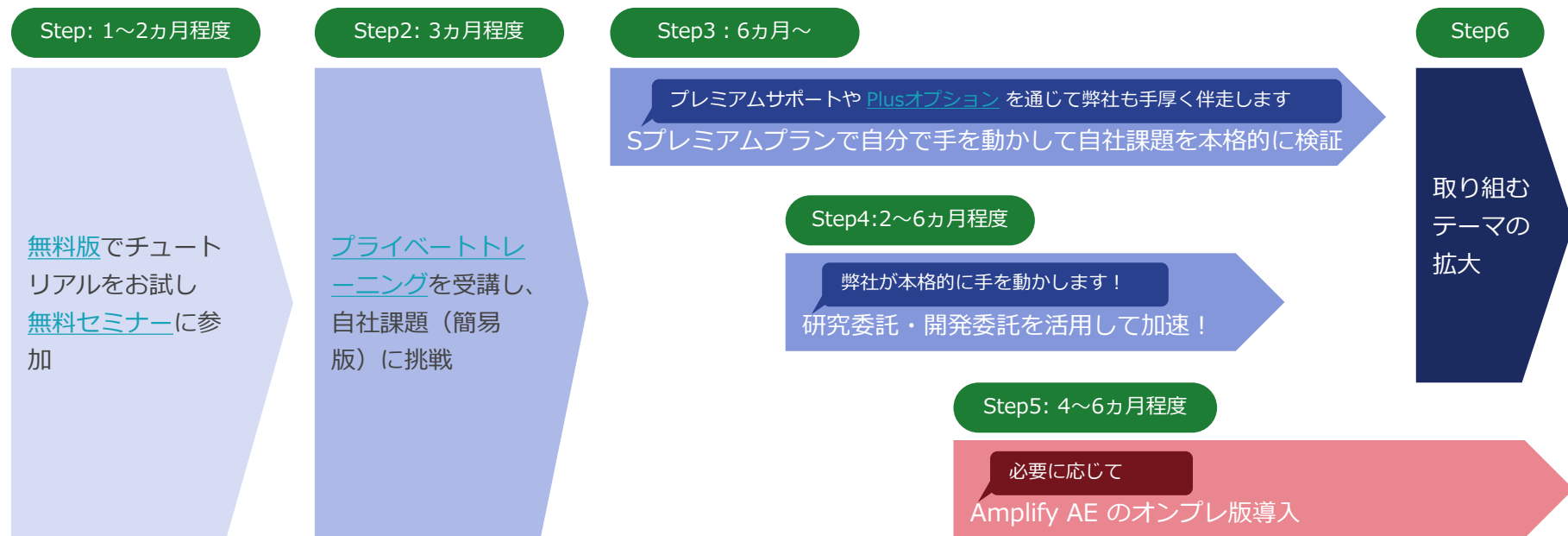
ユーザーインタビュー 2023年4月

早稲田大学

情報工学領域で進む量子コンピュータ・イジングマシンの活用

研究・開発者向けおすすめの実進め方

二次・非線形を上手に使いこなせるように、**弊社と一緒に**取り組みを進めていきましょう！



プライベートトレーニング

<https://amplify.fixstars.com/ja/seminar/private-training>



全4回のレクチャーとお客様に実施いただく「課題」を含む約2カ月のコースです。コースの前半では、量子アニーリング・イジングマシン専用の開発／実行環境である Fixstars Amplify を用いて Python 言語による組合せ最適化アプリケーション開発方法を学びます。コースの後半では、お客様が解きたい課題を持ち込んでいただき、弊社のエンジニアと一緒に解きます。量子アニーリング・イジングマシンを使って実課題の解決に取り組みたい方に最適なコースです。

第1回
3時間



同日実施
または1週間

第2回
3時間

課題
2〜4週間

第3回
1.5時間



2〜4週間

第4回
1.5時間

第1回・第2回クラス (各3時間)

前半は、以下を段階的に学び、持ち込み課題や実課題を解くための準備を行います

- 量子アニーリング・イジングマシンの概要
 - 組合せ最適化問題の定式化
 - 学習環境 Fixstars Amplify の使い方
- プログラミングハンズオン
 - 目的関数の定式化：数の分割問題、画像のノイズ除去
 - 制約条件の定式化：会議室割り当て問題、数独
 - 高度な組合せ最適化問題の定式化：巡回セールスマン問題
 - 応用事例を用いたハンズオン：生産計画（定式化ベース）、材料探索や設計最適化（ブラックボックス最適化ベース）等
- 第3回・第4回のワークショップに向けて、持ち込み課題の内容や進め方に関するディスカッション

課題

第3回、第4回の持ち込み課題のワークショップに向けて、事前準備を進めていただきます

第3回・第4回クラス (各1.5時間)

後半は、お客様の持ち込み課題を使ったワークショップを行います

- 持ち込み課題の内容に関するディスカッション
- 利用するデータの確認
- 定式化や実装の方向性の確認や、実際の実装で苦労している点などに関してディスカッション

Fixstars Amplify: クラウド利用料

個人単位のプラン ～ 主に研究者・開発者向け～

組織単位のビジネスプラン ～ 社内システムの利用向け～

(金額は税抜)

月額利用料

計算環境

利用GPU

(マルチGPUオプションあり)

1ジョブの実行時間

(実行時間延長オプションあり)

月間累計実行回数

(実行回数追加オプションあり)

月間累計実行時間

東芝 SQBM+ オプション

富士通 DA オプション

D-Wave オプション

サポート

Plus オプション

次ページ

	ベーシック	スタンダード	プレミアム	Sプレミアム
月額利用料	無料	10万円 (1名) 30万円 (最大5名)	20万円 (1名) 60万円 (最大5名)	30万円 (1名) 90万円 (最大5名)
計算環境	スモール	ミディアム	ラージ	スーパーラージ
利用GPU	NVIDIA V100	NVIDIA V100	NVIDIA A100	NVIDIA H100
1ジョブの実行時間	10秒	1分	10分	15分
月間累計実行回数	制限の可能性あり	無制限		
月間累計実行時間	制限の可能性あり	無制限		
東芝 SQBM+ オプション	10秒	30万円 (1名)、90万円 (最大5名)		
富士通 DA オプション	10秒	50万円 (1名)、150万円 (最大5名)		
D-Wave オプション	3分/月	ご相談可		
サポート	ベーシック	スタンダード サポート	プレミアム サポート	プレミアム サポート
Plus オプション	-	-	月額50万/人	

	ビジネス スタンダード	ビジネス プレミアム	ビジネス Sプレミアム
月額利用料	20万円 (1アプリ)	40万円 (1アプリ)	60万円 (1アプリ)
(同一組織内であれば同一アプリのユーザー数は無制限)			
計算環境	ミディアム	ラージ	スーパーラージ
利用GPU	NVIDIA V100	NVIDIA A100	NVIDIA H100
1ジョブの実行時間	1分	10分	15分
月間累計実行回数	- (上限を設定する可能性あり)		
月間累計実行時間	- (上限を設定する可能性あり)		
東芝 SQBM+ オプション	-		
富士通 DA オプション	-		
D-Wave オプション	-		
サポート	スタンダード サポート	スタンダード サポート	スタンダード サポート
Plus オプション	-		

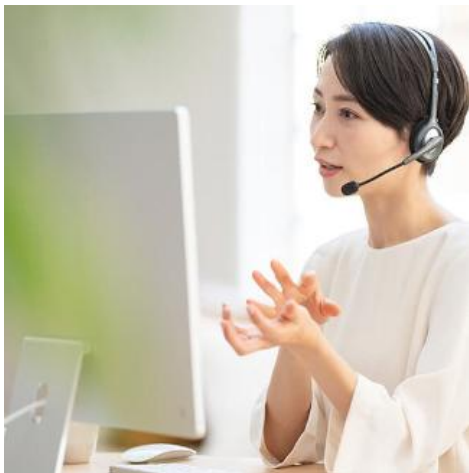
Plusオプション (プレミアムプラン/Sプレミアムプランで追加可能なオプション)

Plusオプション

料金：月額50万円 (税込55万円) /ユーザー

- 問い合わせ回数は無制限
- ご質問には翌営業日までに回答 (目安)
- 定式化・実装等のご相談
- 特別対応窓口や定例会の設置
- 特別技術支援*

*特別技術支援の内容に応じて期間等は個別にご相談



特別技術支援の例

□ 開発支援

- ユーザー様の実装にお困りの部分に関して、弊社エンジニアがサンプルコードを作って提供します
- 開発支援にかかる期間については個別相談となります

□ コードレビュー

- 弊社エンジニアがユーザー様の実装したコードを確認し、よりよい実装などがあればサンプルコードを作って提供します

□ 評価支援

- ユーザー様にご提供いただく問題設定で、弊社のエンジニアが様々な計算環境で実験・評価して結果をレポートします
- 複雑な問題になると限られた計算環境では十分な精度の解が得られない可能性があります。本支援では、異なる GPU (V100/A100/H100) や、GPU 数 (1機~4機)、実行時間 (~1時間) で実験・評価し、最適な計算環境の評価・検討のご支援をします
- 問題設定については、ユーザー様にプログラムやデータを送付してもらう、もしくは、問題の概要をテンプレートで回答いただく形になります
- 評価支援にかかる期間については個別相談となります

今後について

ぜひ、デモ・チュートリアルにあるサンプルコードにも挑戦してください！

一般的な組合せ最適化問題

目的関数のみ
で定式化



チュートリアル基礎編

画像のノイズ除去

プログラミング難易度 ★★★★★
画像のノイズ除去を行うアプリケーションを開発します。

サンプルコード

制約条件のみ
で定式化



チュートリアル応用編

会議室割当問題

プログラミング難易度 ★★★★★
制約条件を用いて定式化するアプリケーションの例として会議室割当問題のアプリケーションを開発します。

サンプルコード

目的関数 + 制約条件



デモアプリケーション

巡回セールスマン問題

プログラミング難易度 ★★★★★
Fixstars Amplifyによる、巡回セールスマン問題の定式化を体験します。

デモアプリ

サンプルコード



デモアプリケーション

容量制約つき運搬経路問題 (CVRP)

プログラミング難易度 ★★★★★
運送における効率的な配送計画の指定やコスト削減や資源消費における訪問順序の最適化等での応用が期待される容量制約つき運搬経路問題 (CVRP) を取り扱います。

デモアプリ

サンプルコード

ブラックボックス最適化問題

概要



チュートリアル応用編

ブラックボックス最適化 (1)

プログラミング難易度 ★★★★★
複雑で未知な目的関数にも適用可能な、機械学習と組み合わせた最適化を組み合わせたブラックボックス最適化手法を紹介し、Amplifyを用いて実装します。

サンプルコード

材料探索



チュートリアル応用編

ブラックボックス最適化 (2)

プログラミング難易度 ★★★★★
機械学習と量子アニーリング・イジングマシンを活用するブラックボックス最適化の適用例として、疑似的な高温超伝導を実現する材料探索を取り扱います。

サンプルコード

翼型最適化



チュートリアル応用編

ブラックボックス最適化 (4)

プログラミング難易度 ★★★★★
流体力学設計に不可欠な翼型の最適化問題を取り上げます。最適化には、組み合わせた最適化及び機械学習に基づくブラックボックス最適化と流体シミュレーションを用い、翼の揚力比を最大化するように翼型の探索を行います。

サンプルコード

信号機制御



チュートリアル応用編

ブラックボックス最適化 (5)

プログラミング難易度 ★★★★★
ブラックボックス最適化により、商業施設による交通集中が発生する都市における、交通渋滞を低減するような信号機群の最適制御を実装します。最適化の実施及び検証には、マルチエージェント・シミュレーションによる交通シミュレーションを用います。

サンプルコード

困った時はドキュメンテーションを！

<https://amplify.fixstars.com/docs/amplify/v1/index.html>



今後のセミナーのご案内

<https://amplify.fixstars.com/ja/news/seminar>

今後も研究・開発者向けの無料セミナーを定期的 to開催します！

2025/1/15 (木)

「Amplify AE 技術解説」

- ・ 会社および量子コンピューティングクラウド「Fixstars Amplify」のご紹介
- ・ Amplify AE 技術解説
- ・ 事例や今後の進め方のご紹介
- ・ Wrap Up & QA

2025/1/29 (木)

「設計パラメータBBO最適化」

- ・ 会社および量子コンピューティングクラウド「Fixstars Amplify」のご紹介
- ・ Fixstars Amplify を用いたワークショップ
 - ・ ブラックボックス最適化
- ・ 事例や今後の進め方のご紹介
- ・ Wrap Up & QA

2025/2/12 (木)

「多目的最適化 定式化解説」

- ・ 会社および量子コンピューティングクラウド「Fixstars Amplify」のご紹介
- ・ Fixstars Amplify を用いたワークショップ
 - ・ 多目的最適化の定式化
- ・ 事例や今後の進め方のご紹介
- ・ Wrap Up & QA

ご質問・ご不明点がありましたら、お問い合わせフォームでご連絡下さい

<https://amplify.fixstars.com/ja/contact>

ご参加ありがとうございました

アンケートへのご回答をお願いします！

A decorative graphic at the bottom of the slide consists of a complex network of white lines connecting various sized white dots, set against a dark blue background. The dots and lines are scattered across the lower half of the image, creating a sense of connectivity and data flow.