FIXSTARS Amplify

量子時代のプログラミングセミナー

～ Fixstars Amplifyで実装するシフト最適化（多目的最適化）～

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
  - シフト最適化（多目的最適化）
- 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 でお願いします

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

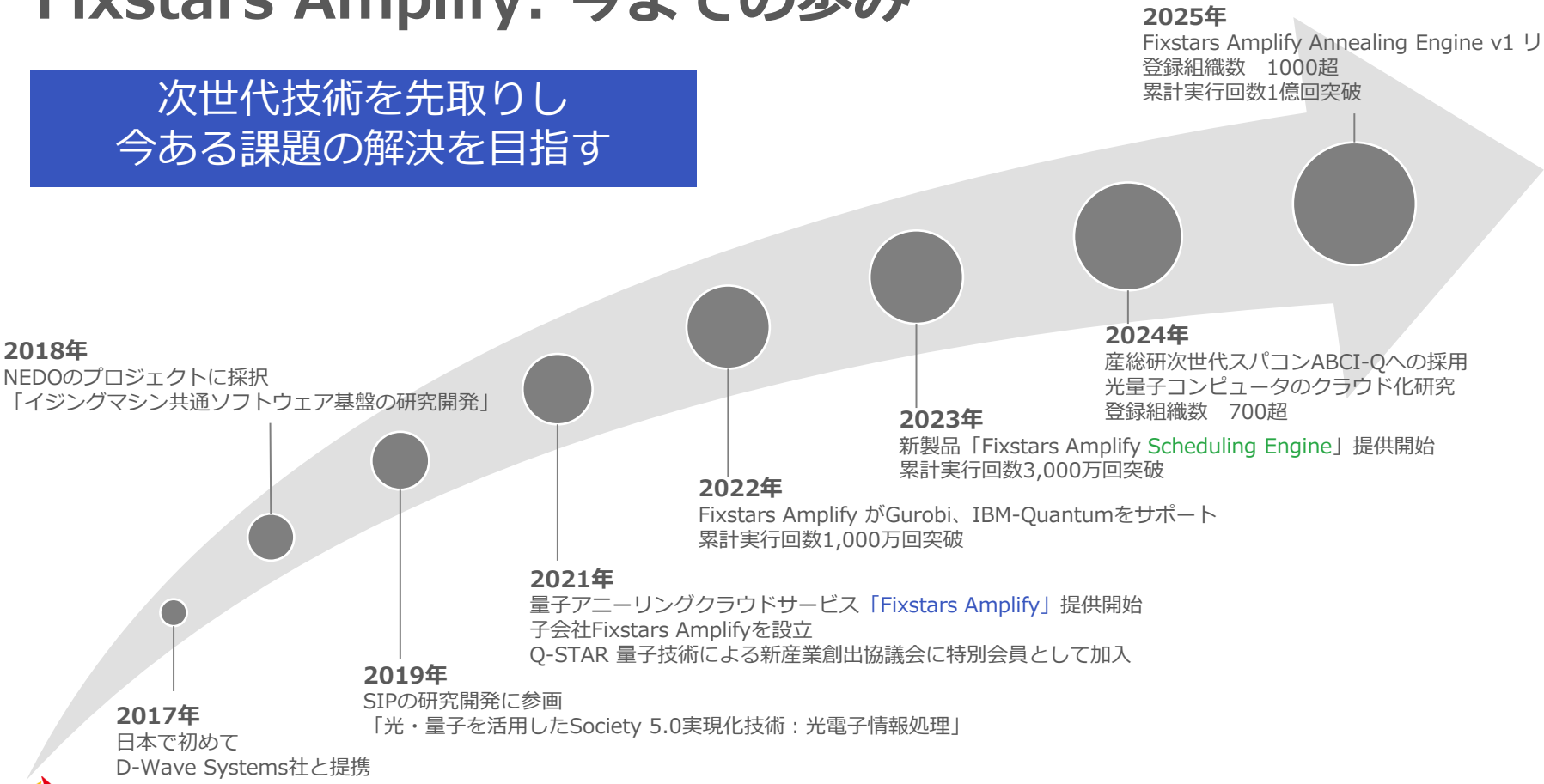
- 2021年に設立（株式会社フィックスターズからスピンアウト）
  - 代表取締役社長CEO：松田 佳希（博士）

- 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

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



Fixstars Amplify: 製品ポートフォリオ

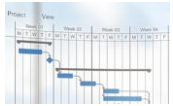
Fixstars Amplify
(量子コンピューティングクラウド, 2021/2～)

- **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

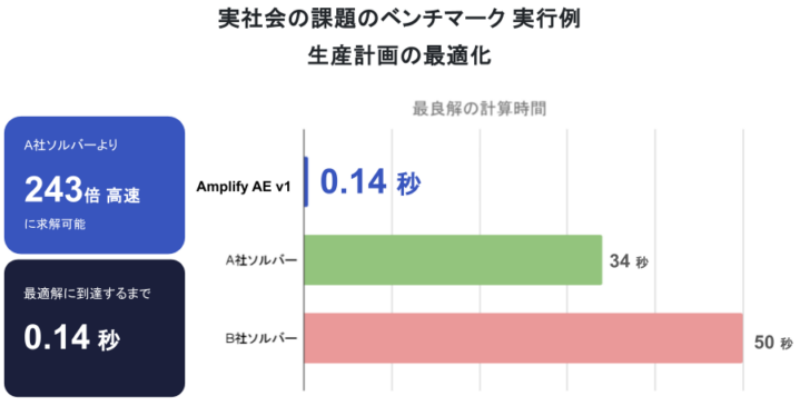


Fixstars Amplify AE v1（2025/10/17リリース）

https://amplify.fixstars.com/ja/news/795/

Amplify AE v1の特徴

- 圧倒的な求解性能
- パラメーターの自動調整機能による使いやすさの実現
- 不等式制約や高次多項式への対応による表現力の強化



AE v0 から AE v1 への移行手順
クライアントを切り替えるだけ！

```python
from amplify import FixstarsClient

client = FixstarsClient()

# API トークンを設定
client.token = "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"

# 実行時間を 1 秒に設定
client.parameters.timeout = 1000
```

```python
from amplify import AmplifyAEClient

client = AmplifyAEClient()

# API トークンを設定
client.token = "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"

# 実行時間を 1 秒に設定
client.parameters.time_limit_ms = 1000
```

本日のセミナーは Amplify AE v1 を使います！

様々な領域での利用拡大中（実稼働含む）

1100 を超える企業、研究所、大学

1.2億 を超える実行回数 (Amplify AE)

https://amplify.fixstars.com/ja/customers

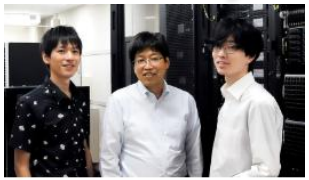
量子コンピューティングクラウドサービス「Fixstars Amplify」のご紹介

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）、エネルギーマネー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

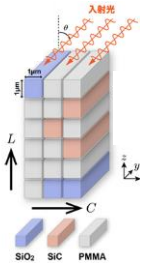
生産計画最適化 / 材料開発（MI） / 経路最適化 / 人員シフト最適化 / 機械学習の特徴量抽出 / 実験のパラメータ最適化 / 設計最適化 / エネルギーマネジメント最適化 / 電力最適化



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

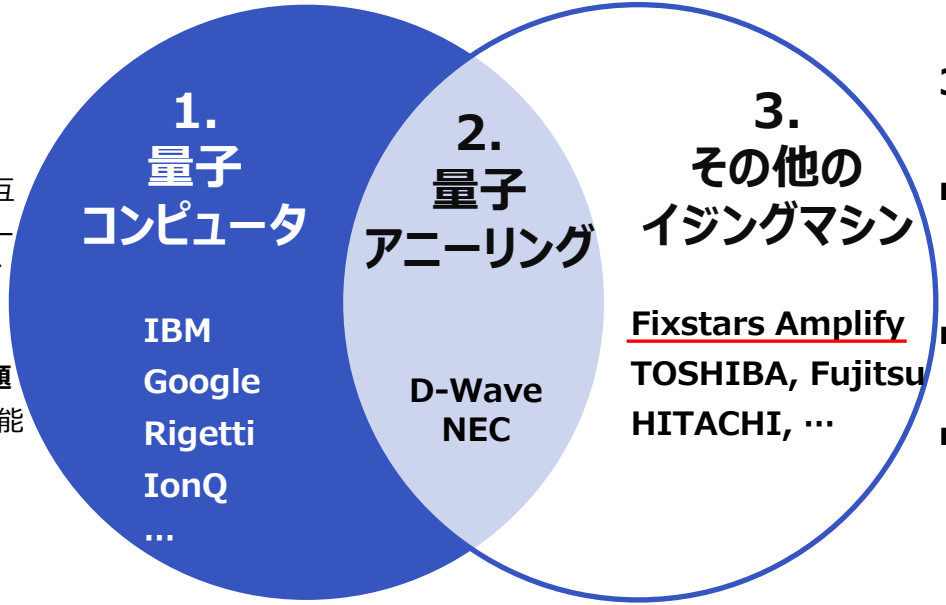
ブラックボックス最適化を活用した車両設計最適化

FIXSTARS Amplify

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**



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)

量子アニーリング・イジングマシン

Quadratic	二次形
Unconstrained	制約条件なし
Binary	0-1整数 (二値)
Optimization	計画（最適化）

QUBO目的関数 (0-1整数二次計画問題)

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

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

f: 目的関数　q: 決定変数　Q: 係数

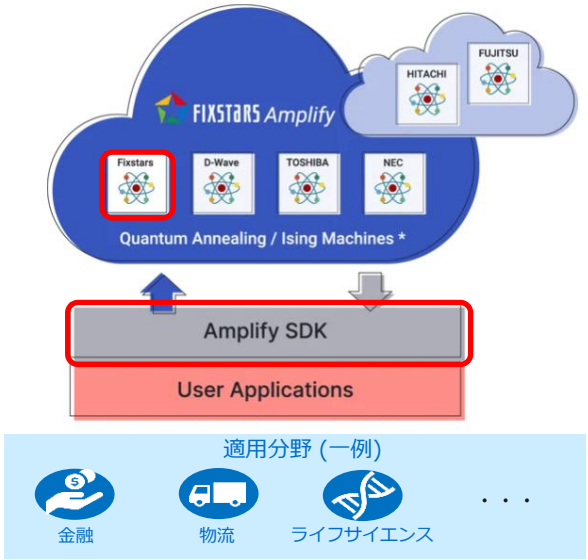
クラウドサービス：**Fixstars Amplify**

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

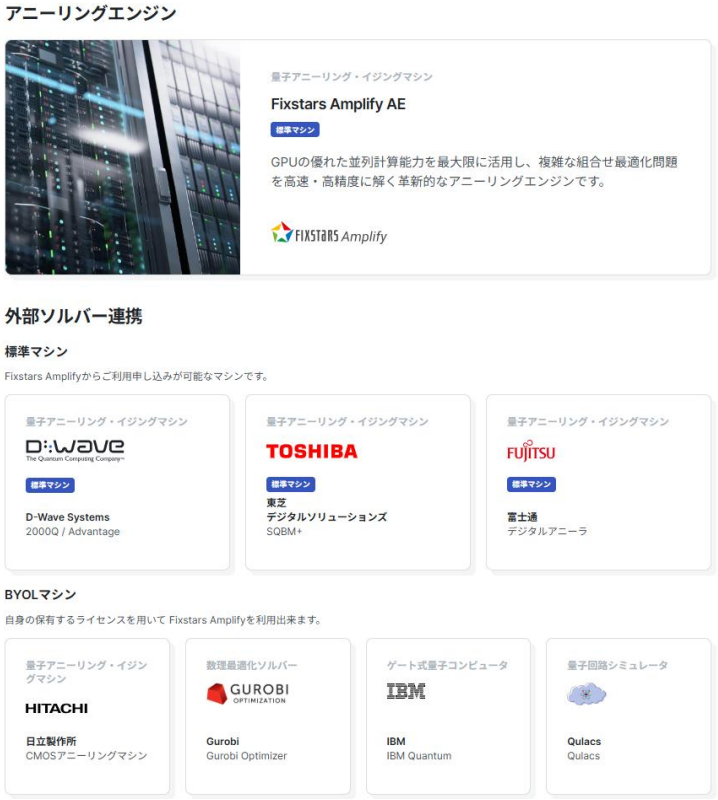
ポータブル

- 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



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

最適エネルギーマネジメント

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、疑似的な高温超電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化学反応に関する物理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

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、翼の揚抗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

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 ★★★★☆

運送業における効率的な配送計画の策定やごみ収集や道路清掃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

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し得る都市における、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、刻一刻と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、その様な信号機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ずれ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、最も早く全ジョブが完了するような割り当て方を求めます。

サンプルコード

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

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
2. **マシンを実行**
   複数マシンの中から選択可能

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。

```python
# 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 = AmplifyAEClient()
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

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:
Wikipdia

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！

FIXSTARS Amplify

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	262,144以上	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)/ 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

※評価・検証用途では無償提供

FIXSTARS Amplify

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

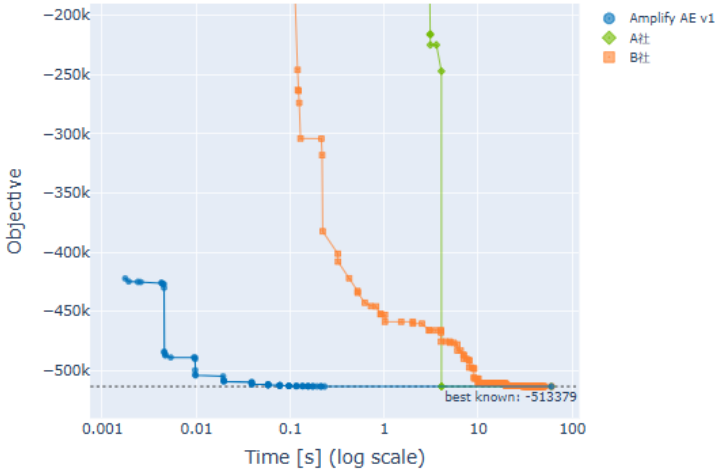
に求解可能*

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



https://amplify.fixstars.com/ja/engine

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

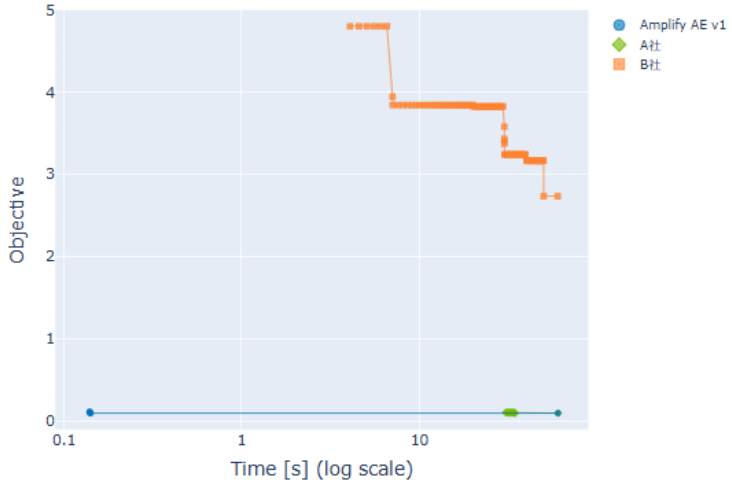
0.14 秒

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



FIXSTARS Amplify

https://amplify.fixstars.com/ja/engine

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/1HrRKai4jA0u-r0XhZHtvh-OKowfXWCGL
Step2	https://colab.research.google.com/drive/1JZ5rut0NMKLx_xTeYCZpHyMZXIxw3efw
Step3	https://colab.research.google.com/drive/1kZM-eXwp6u8yk12gaIWra05aDGTGGFJi
Step4	https://colab.research.google.com/drive/1R_REoXiM5ixZCstLwQemBiW4F7xnkULb
Step5（発展）	https://colab.research.google.com/drive/1JnOYiME1QAwDm4k708nwNJw4QW20aSUX

質問は随時 Zoom のチャットか Q&A でお願いします

ワークショップ
～シフト最適化～

最適シフト作成

業務で求められる役割・役職・スキルと個人の能力や要求を考慮した最適シフトとは

業務要求

要求管理者数

ライン	責任者	主任技師	合計
LineA	1	1	2
LineB	1	1	2
LineC	1	1	2
合計	3	3	6

要求スキル量（各従業員が持つスキル値の合計）

前工程	後工程	組立て	合計
8	4	5	17
6	9	3	18
7	6	5	18
21	19	13	53

要求配置(0=NG, 1=OK, 2=要求)

従業員ID	LineA	LineB	LineC
0	1	0	0
1	1	1	1
2	1	2	0
⋮	⋮	⋮	⋮
23	1	1	1
24	1	1	0

マッチング

各従業員情報

従業員ID	担当可能役職			スキル値(1=初級, 2=中級, 3=上級)		
	責任者	主任技師	技師	前工程	後工程	組立て
0	1	1	1	3	3	3
1	0	1	1	3	3	2
2	0	0	1	2	1	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮
23	0	0	1	0	0	2
24	0	0	1	1	0	1

FIXSTARS Amplify

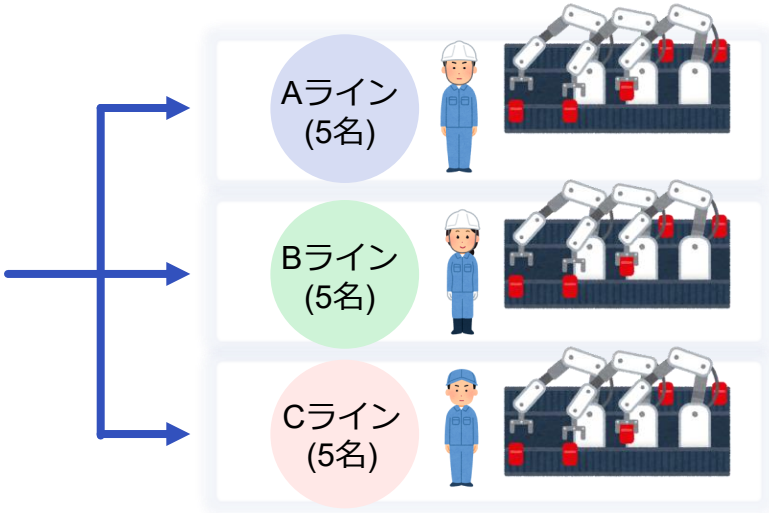
ワークショップ: 問題設定

15名の従業員を5名ずつ3つの生産ラインへ振り分けます。各従業員は、各ラインに対するスキル値を持ちます。
各ラインに配置された従業員のスキル値（ラインスキル値）の**合計がなるべく高く（目的1）**、また、
各ラインのラインスキル値の**ばらつきが少ない（目的2）**、という2つの目的のバランスの取れたシフトの作成を目指します。

従業員の各ラインのスキル値

worker_id	worker_skill		
	line_A	line_B	line_C
0	130	70	90
1	120	70	80
2	110	60	80
3	100	60	80
4	90	55	70
5	90	130	70
6	80	120	70
7	80	110	60
8	80	100	60
9	70	90	55
10	70	90	130
11	70	80	120
12	60	80	110
13	60	80	100
14	55	70	90

組み合わせの数は約1,400万(3^15) 通り!



ワークショップ: 試しに人の手でやってみましょう

上から順に A → B → C と割り振る場合

worker_id	line_A	line_B	line_C	配属ライン
0	130	60	70	→ line_A
1	120	55	60	→ line_B
2	110	130	60	→ line_C
3	100	120	55	→ line_A
4	90	110	130	→ line_B
5	90	100	120	→ line_C
6	80	90	110	→ line_A
7	80	90	100	→ line_B
8	80	80	90	→ line_C
9	70	80	90	→ line_A
10	70	80	80	→ line_B
11	70	70	80	→ line_C
12	60	70	80	→ line_A
13	60	70	70	→ line_B
14	55	60	70	→ line_C

ラインスキル値

worker_id	line_A
0	130
3	100
6	80
9	70
12	60
合計	440

worker_id	line_B
1	55
4	110
7	90
10	80
13	70
合計	405

worker_id	line_C
2	60
5	120
8	90
11	80
14	70
合計	420

全ライン合計 1,265

1か所だけ微調整

worker_id	line_A	line_B	line_C	配属ライン
0	130	60	70	→ line_A
1	120	55	60	→ line_C
2	110	130	60	→ line_B
3	100	120	55	→ line_A
4	90	110	130	→ line_B
5	90	100	120	→ line_C
6	80	90	110	→ line_A
7	80	90	100	→ line_B
8	80	80	90	→ line_C
9	70	80	90	→ line_A
10	70	80	80	→ line_B
11	70	70	80	→ line_C
12	60	70	80	→ line_A
13	60	70	70	→ line_B
14	55	60	70	→ line_C

ラインスキル値

worker_id	line_A
0	130
3	100
6	80
9	70
12	60
合計	440

worker_id	line_B
2	130
4	110
7	90
10	80
13	70
合計	480

worker_id	line_C
1	60
5	120
8	90
11	80
14	70
合計	420

全ライン合計 1,340

組合せ最適化問題を解くイメージ

問題を設定

15名の従業員を5名ずつ3つの生産ラインへ振り分けます。各従業員は、各ラインに対するスキル値を持ちます。各ラインに配置された従業員のスキル値（ラインスキル値）の**合計がなるべく高く（目的1）**、また、各ラインのラインスキル値の**ばらつきが少ない（目的2）**、という2つの目的のバランスの取れたシフトの作成を目指します。

入力情報を準備

従業員の各ラインのスキル値

worker_id	worker_skill		
	line_A	line_B	line_C
0	130	70	90
1	120	70	80
2	110	60	80
3	100	60	80
4	90	55	70
5	90	130	70
6	80	120	70
7	80	110	60
8	80	100	60
9	70	90	55
10	70	90	130
11	70	80	120
12	60	80	110
13	60	80	100
14	55	70	90

決定変数を用意

1: 配置、0: 非配置

worker_id (変数: i)	line_A	line_B (変数: l)	line_C
0	q_0,0 (0 or 1)	q_0,1 (0 or 1)	q_0,2 (0 or 1)
1	q_1,0 (0 or 1)	q_1,1 (0 or 1)	q_1,2 (0 or 1)
2	q_2,0 (0 or 1)	q_2,1 (0 or 1)	q_2,2 (0 or 1)
3	q_3,0 (0 or 1)	q_3,1 (0 or 1)	q_3,2 (0 or 1)
4	q_4,0 (0 or 1)	q_4,1 (0 or 1)	q_4,2 (0 or 1)
5	q_5,0 (0 or 1)	q_5,1 (0 or 1)	q_5,2 (0 or 1)
6	q_6,0 (0 or 1)	q_6,1 (0 or 1)	q_6,2 (0 or 1)
7	q_7,0 (0 or 1)	q_7,1 (0 or 1)	q_7,2 (0 or 1)
8	q_8,0 (0 or 1)	q_8,1 (0 or 1)	q_8,2 (0 or 1)
9	q_9,0 (0 or 1)	q_9,1 (0 or 1)	q_9,2 (0 or 1)
10	q_10,0 (0 or 1)	q_10,1 (0 or 1)	q_10,2 (0 or 1)
11	q_11,0 (0 or 1)	q_11,1 (0 or 1)	q_11,2 (0 or 1)
12	q_12,0 (0 or 1)	q_12,1 (0 or 1)	q_12,2 (0 or 1)
13	q_13,0 (0 or 1)	q_13,1 (0 or 1)	q_13,2 (0 or 1)
14	q_14,0 (0 or 1)	q_14,1 (0 or 1)	q_14,2 (0 or 1)

定式化

目的関数1: 全ラインの合計スキル値の最大化

$$skill\_score = \sum_{l}^{3}\sum_{i}^{15} q_{i,l} \cdot s_{i,l}$$

2次

目的関数2: ラインスキル値のばらつき (分散) を最小化

$$variance = \frac{\sum_{l}^{3}\left(\sum_{i}^{15} q_{i,l} \cdot s_{i,l}\right)^2}{3} - \left(\frac{\sum_{l}^{3}\sum_{i}^{15} q_{i,l} \cdot s_{i,l}}{3}\right)^2$$

制約条件1: 従業員は同時に1つの製造ラインのみに配置可能

$$\sum_{l}^{3} q_{i,l} = 1$$

⋮

Pythonで記述してマシンで求解

worker_id (変数: i)	line_A	line_B (変数: l)	line_C
0	1	0	0
1	0	1	0
2	0	1	0
3	0	0	1
4	1	0	0
5	0	0	1
6	0	1	0
7	1	0	0
8	1	0	0
9	0	1	0
10	0	0	1
11	0	1	0
12	0	0	1
13	1	0	0
14	0	0	1

ワークショップ: 4 Step

15名の従業員を5名ずつ3つの生産ラインへ振り分けます。各従業員は、各ラインに対するスキル値を持ちます。各ラインに配置された従業員のスキル値（ラインスキル値）の**合計がなるべく高く（目的1）**、また、各ラインのラインスキル値の**ばらつきが少ない（目的2）**、という2つの目的のバランスの取れたシフトの作成を目指します。

全てを一度にやるのは難しいので4つのステップに分けてプログラムの完成を目指します

Step1

まず、2つの制約だけを考慮して配置シフトを求めます
制約1: 従業員は同時に1つの製造ラインのみに配置が可能
制約2: 各ラインの配置人数が5名ずつになること

解の候補多数あり

Step2

Step1に「ラインスキル値の合計 (= 全ラインの合計スキル値) を最大化」という目的1を追加し、複数の解の候補から目的を実現するシフトを求めます

Step3

Step2に「ラインスキル値のばらつきを最小化」という目的2を追加して、2つの目的を同時に実現するシフトを求めます

Step4

Step3に目的1と目的2の重みを調整する「パラメーター」を追加し、最適なバランスのシフトを作成します

Step1　まず、2つの制約だけを考慮して配置シフトを求めます

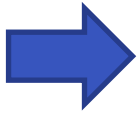
決定変数の準備

Binary の PolyArray
15×3 = 45 [qubit]

1: 配置
0: 非配置

worker_id (変数: i)	line_A	line_B (変数: l)	line_C
0	q_0,0 (0 or 1)	q_0,1 (0 or 1)	q_0,2 (0 or 1)
1	q_1,0 (0 or 1)	q_1,1 (0 or 1)	q_1,2 (0 or 1)
2	q_2,0 (0 or 1)	q_2,1 (0 or 1)	q_2,2 (0 or 1)
3	q_3,0 (0 or 1)	q_3,1 (0 or 1)	q_3,2 (0 or 1)
4	q_4,0 (0 or 1)	q_4,1 (0 or 1)	q_4,2 (0 or 1)
5	q_5,0 (0 or 1)	q_5,1 (0 or 1)	q_5,2 (0 or 1)
6	q_6,0 (0 or 1)	q_6,1 (0 or 1)	q_6,2 (0 or 1)
7	q_7,0 (0 or 1)	q_7,1 (0 or 1)	q_7,2 (0 or 1)
8	q_8,0 (0 or 1)	q_8,1 (0 or 1)	q_8,2 (0 or 1)
9	q_9,0 (0 or 1)	q_9,1 (0 or 1)	q_9,2 (0 or 1)
10	q_10,0 (0 or 1)	q_10,1 (0 or 1)	q_10,2 (0 or 1)
11	q_11,0 (0 or 1)	q_11,1 (0 or 1)	q_11,2 (0 or 1)
12	q_12,0 (0 or 1)	q_12,1 (0 or 1)	q_12,2 (0 or 1)
13	q_13,0 (0 or 1)	q_13,1 (0 or 1)	q_13,2 (0 or 1)
14	q_14,0 (0 or 1)	q_14,1 (0 or 1)	q_14,2 (0 or 1)

イジングマシンで最適な(0,1)の組み合わせを探す



得られる解の例

worker_id (変数: i)	line_A	line_B (変数: l)	line_C
0	1	0	0
1	0	1	0
2	0	1	0
3	0	0	1
4	1	0	0
5	0	0	1
6	0	1	0
7	1	0	0
8	1	0	0
9	0	1	0
10	0	0	1
11	0	1	0
12	0	0	1
13	1	0	0
14	0	0	1

従業員3はline_Cに配置することを意味する

実装

決定変数

```
# 従業員(i)をライン(l)に配置することを表現する決定変数
from amplify import VariableGenerator

gen = VariableGenerator()  # 変数のジェネレータを宣言
q = gen.array("Binary", shape=(num_workers, num_locations))
display(q)
```

Step1　まず、2つの制約だけを考慮して配置シフトを求めます

定式化

制約1: 従業員は同時に1つの製造ラインのみに配置可能
→ one_hot 制約

$$\sum_{l}^{3} q_{i,l} = 1$$

制約2: 各ラインの配置人数が要求人数（5名）と一致する
→ equal_to 制約 (等式制約)

$$\sum_{i}^{15} q_{i,l} = 5$$

worker_id (変数: i)	line_A	line_B (変数: l)	line_C	
0	q_0,0 (0 or 1)	q_0,1 (0 or 1)	q_0,2 (0 or 1)	→ one_hot
1	q_1,0 (0 or 1)	q_1,1 (0 or 1)	q_1,2 (0 or 1)	→ one_hot
2	q_2,0 (0 or 1)	q_2,1 (0 or 1)	q_2,2 (0 or 1)	→ one_hot
3	q_3,0 (0 or 1)	q_3,1 (0 or 1)	q_3,2 (0 or 1)	→ one_hot
4	q_4,0 (0 or 1)	q_4,1 (0 or 1)	q_4,2 (0 or 1)	→ one_hot
5	q_5,0 (0 or 1)	q_5,1 (0 or 1)	q_5,2 (0 or 1)	→ one_hot
6	q_6,0 (0 or 1)	q_6,1 (0 or 1)	q_6,2 (0 or 1)	→ one_hot
7	q_7,0 (0 or 1)	q_7,1 (0 or 1)	q_7,2 (0 or 1)	→ one_hot
8	q_8,0 (0 or 1)	q_8,1 (0 or 1)	q_8,2 (0 or 1)	→ one_hot
9	q_9,0 (0 or 1)	q_9,1 (0 or 1)	q_9,2 (0 or 1)	→ one_hot
10	q_10,0 (0 or 1)	q_10,1 (0 or 1)	q_10,2 (0 or 1)	→ one_hot
11	q_11,0 (0 or 1)	q_11,1 (0 or 1)	q_11,2 (0 or 1)	→ one_hot
12	q_12,0 (0 or 1)	q_12,1 (0 or 1)	q_12,2 (0 or 1)	→ one_hot
13	q_13,0 (0 or 1)	q_13,1 (0 or 1)	q_13,2 (0 or 1)	→ one_hot
14	q_14,0 (0 or 1)	q_14,1 (0 or 1)	q_14,2 (0 or 1)	→ one_hot
	↓	↓	↓	
	equal_to	equal_to	equal_to	

Step1　まず、2つの制約だけを考慮して配置シフトを求めます

worker_id (変数: i)	line_A	line_B (変数: l)	line_C	
0	q_0,0 (0 or 1)	q_0,1 (0 or 1)	q_0,2 (0 or 1)	→ one_hot
1	q_1,0 (0 or 1)	q_1,1 (0 or 1)	q_1,2 (0 or 1)	→ one_hot
2	q_2,0 (0 or 1)	q_2,1 (0 or 1)	q_2,2 (0 or 1)	→ one_hot
3	q_3,0 (0 or 1)	q_3,1 (0 or 1)	q_3,2 (0 or 1)	→ one_hot
4	q_4,0 (0 or 1)	q_4,1 (0 or 1)	q_4,2 (0 or 1)	→ one_hot
5	q_5,0 (0 or 1)	q_5,1 (0 or 1)	q_5,2 (0 or 1)	→ one_hot
6	q_6,0 (0 or 1)	q_6,1 (0 or 1)	q_6,2 (0 or 1)	→ one_hot
7	q_7,0 (0 or 1)	q_7,1 (0 or 1)	q_7,2 (0 or 1)	→ one_hot
8	q_8,0 (0 or 1)	q_8,1 (0 or 1)	q_8,2 (0 or 1)	→ one_hot
9	q_9,0 (0 or 1)	q_9,1 (0 or 1)	q_9,2 (0 or 1)	→ one_hot
10	q_10,0 (0 or 1)	q_10,1 (0 or 1)	q_10,2 (0 or 1)	→ one_hot
11	q_11,0 (0 or 1)	q_11,1 (0 or 1)	q_11,2 (0 or 1)	→ one_hot
12	q_12,0 (0 or 1)	q_12,1 (0 or 1)	q_12,2 (0 or 1)	→ one_hot
13	q_13,0 (0 or 1)	q_13,1 (0 or 1)	q_13,2 (0 or 1)	→ one_hot
14	q_14,0 (0 or 1)	q_14,1 (0 or 1)	q_14,2 (0 or 1)	→ one_hot
	↓	↓	↓	
	equal_to	equal_to	equal_to	

実装

```
####################
# 制約条件の定式化
####################

from amplify import one_hot, equal_to, sum as amplify_sum

# 制約1: 従業員は同時に1つの製造ラインのみに配置が可能
loc_constraints = amplify_sum(one_hot(q[i]) for i in range(num_workers))
display(loc_constraints)

# 制約2: 各ラインの配置人数が要求人数(5名)と一致すること
req_constraints = amplify_sum(
    equal_to(q[:, l], df_req["worker"][l])
    for l in range(num_locations)
)
display(req_constraints)

# 制約
constraints = loc_constraints + req_constraints
```

FIXSTARS Amplify

Step1　まず、2つの制約だけを考慮して配置シフトを求めます

求解

```
####################
# マシンで求解
####################

from amplify import AmplifyAEClient, solve
from datetime import timedelta

# 実行マシンクライアントの設定
client = AmplifyAEClient()
client.token = token
client.parameters.time_limit_ms = timedelta(milliseconds=1000)  # タイムアウトは 1 秒

# モデル化
model = constraints

# アニーリングマシンの実行
result = solve(model, client)  # 問題を入力してマシンを実行
```

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

- 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。

Step1　まず、2つの制約だけを考慮して配置シフトを求めます

結果の取得

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

可視化

line_Aのラインスキル値: 490

loc	worker_id	worker_skill
line_A	0	130
line_A	1	120
line_A	5	90
line_A	6	80
line_A	9	70

line_Bのラインスキル値: 440

loc	worker_id	worker_skill
line_B	3	120
line_B	4	110
line_B	8	80
line_B	11	70
line_B	14	60

line_Cのラインスキル値: 390

loc	worker_id	worker_skill
line_C	2	60
line_C	7	100
line_C	10	80
line_C	12	80
line_C	13	70

全ラインの合計スキル値: 1320

確認

worker_id	worker_skill		
	line_A	line_B	line_C
0	130	60	70
1	120	55	60
2	110	130	60
3	100	120	55
4	90	110	130
5	90	100	120
6	80	90	110
7	80	90	100
8	80	80	90
9	70	80	90
10	70	80	80
11	70	70	80
12	60	70	80
13	60	70	70
14	55	60	70

同じ人が複数のラインに配置される事無く、各ラインに5人ずつ配置するという、２つの制約を満たすシフトを作ることができました。ただし、解の候補はたくさん有り、最適化の余地も大きそうです。

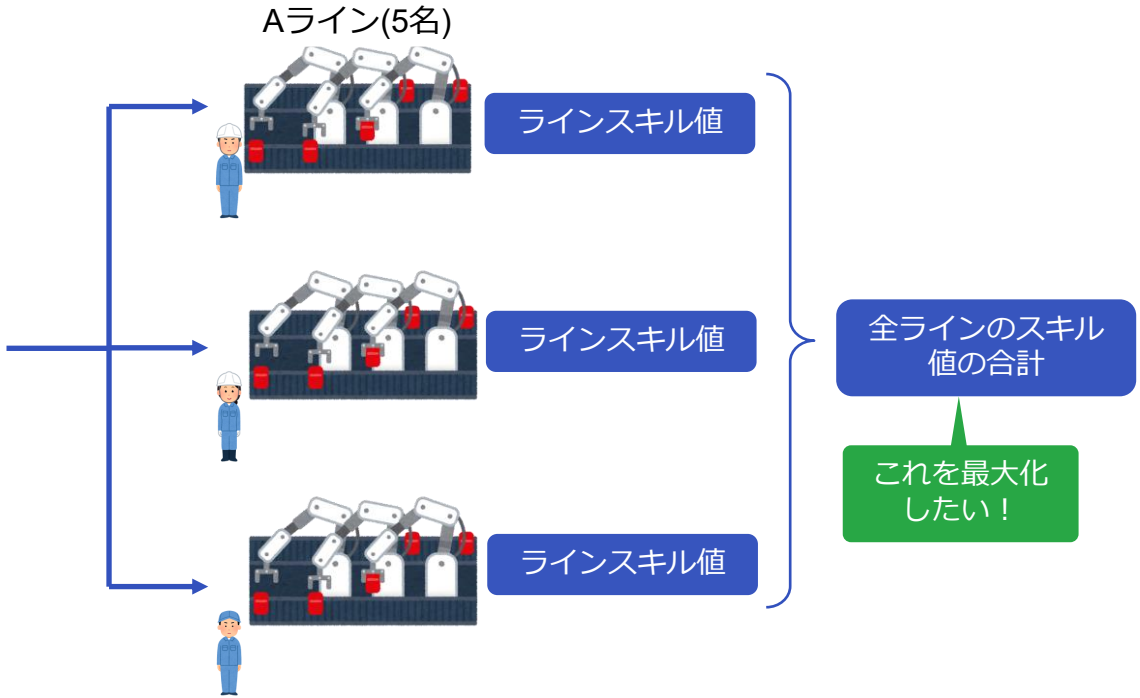
Step2

Step1に「ラインスキル値の合計 (= 全ラインの合計スキル値) を最大化」という目的1を追加し、複数の解の候補から目的を実現するシフトを求めます

ここでは、工場全体のアウトプットの最大化を目指し、各従業員をできるだけ高いスキル値を持つラインへ配置することを目指します

従業員の各ラインのスキル値

worker_id	worker_skill		
	line_A	line_B	line_C
0	130	70	90
1	120	70	80
2	110	60	80
3	100	60	80
4	90	55	70
5	90	130	70
6	80	120	70
7	80	110	60
8	80	100	60
9	70	90	55
10	70	90	130
11	70	80	120
12	60	80	110
13	60	80	100
14	55	70	90



Step2

Step1に「ラインスキル値の合計 (= 全ラインの合計スキル値) を最大化」という目的1を追加し、複数の解の候補から目的を実現するシフトを求めます

定式化

目的1: 全ラインの合計スキル値の最大化

$$skill\_score = \sum_{l}^{3} \sum_{i}^{15} q_{i,l} \cdot s_{i,l}$$

決定変数

決定変数

worker_id (変数: i)	line_A	line_B (変数: l)	line_C
0	1	0	0
1	0	1	0
2	0	1	0
3	0	0	1
4	1	0	0
5	0	0	1
6	0	1	0
7	1	0	0
8	1	0	0
9	0	1	0
10	0	0	1
11	0	1	0
12	0	0	1
13	1	0	0
14	0	0	1

各従業員のスキル値

worker_id (変数: i)	worker_skill (変数 s) line_A	line_B	line_C
0	130	60	70
1	120	55	60
2	110	130	60
3	100	120	55
4	90	110	130
5	90	100	120
6	80	90	110
7	80	90	100
8	80	80	90
9	70	80	90
10	70	80	80
11	70	70	80
12	60	70	80
13	60	70	70
14	55	60	70

各列の要素同士の掛け算の和 ⇒ 各ラインのラインスキル値

Step2

Step1に「ラインスキル値の合計 (= 全ラインの合計スキル値) を最大化」という目的1を追加し、複数の解の候補から目的を実現するシフトを求めます

実装

```
####################
# 目的関数の定式化
####################

from amplify import sum as amplify_sum

# 各ラインのスキル値
a = amplify_sum(q[:, 0] * df_worker_skill.iloc[:, 0 + 1].values)
b = amplify_sum(q[:, 1] * df_worker_skill.iloc[:, 1 + 1].values)
c = amplify_sum(q[:, 2] * df_worker_skill.iloc[:, 2 + 1].values)

# 目的1: 全ラインのスキル値の合計（最大化したいもの）
skill_score = a + b + c
display(skill_score)

# 目的関数（最小化問題とするためにスキル値の項にはマイナスを追加）
objective = -skill_score
display(objective)
```

Step1 と同様

```
####################
# 制約条件の定式化
####################

from amplify import one_hot, equal_to

# 制約1 従業員は同時に1つの製造ラインのみに配置が可能
loc_constraints = amplify_sum(one_hot(q[i]) for i in range(num_workers))
# display(loc_constraints)

# 制約2 各ラインの配置人数が要求人数(5名)と一致すること
req_constraints = amplify_sum(
    equal_to(q[:, l], df_req["worker"][l]) for l in range(num_locations)
)
# display(req_constraints)

# 制約
constraints = loc_constraints + req_constraints
```

イジングマシンは、この objective の値が最小になる組合せを探します (全ラインの合計スキル値は大きいものを選びたいので、skill_score にマイナスをつけたものを objective としています)

```
# モデル化
model = objective + constraints
```

Step2

Step1に「ラインスキル値の合計 (= 全ラインの合計スキル値) を最大化」という目的1を追加し、複数の解の候補から目的を実現するシフトを求めます

結果の取得

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

可視化

ばらつき大

line_Aのラインスキル値: 455

loc	worker_id	worker_skill
line_A	0	130
line_A	1	120
line_A	8	80
line_A	11	70
line_A	14	55

line_Bのラインスキル値: 470

loc	worker_id	worker_skill
line_B	2	130
line_B	3	120
line_B	10	80
line_B	12	70
line_B	13	70

line_Cのラインスキル値: 550

loc	worker_id	worker_skill
line_C	4	130
line_C	5	120
line_C	6	110
line_C	7	100
line_C	9	90

確認

worker_id	worker_skill		
	line_A	line_B	line_C
0	130	60	70
1	120	55	60
2	110	130	60
3	100	120	55
4	90	110	130
5	90	100	120
6	80	90	110
7	80	90	100
8	80	80	90
9	70	80	90
10	70	80	80
11	70	70	80
12	60	70	80
13	60	70	70
14	55	60	70

全ラインの合計スキル値: 1475

二つの制約を満しながら、全ラインの合計スキル値が最大化されたシフトを作ることができました (Step1の全ラインの合計スキル値は1,320) 。但し、ライン間のスキル値のばらつきが大きいので、更なる最適化をかけたい状況です

Step3

Step2に「ラインスキル値のばらつきを最小化」という目的2を追加して、2つの目的を同時に実現するシフトを求めます

定式化

目的2: ラインスキル値のばらつき(分散)を最小化

2次

$$variance = \frac{\sum_l^3 \left(\sum_i^{15} q_{i,l} \cdot s_{i,l}\right)^2}{3} - \left(\frac{\sum_l^3 \sum_i^{15} q_{i,l} \cdot s_{i,l}}{3}\right)^2$$

(分散 = (2乗の平均) − (平均の2乗))

実装

：追加コード

```
####################
# 目的関数の定式化
####################

from amplify import sum as amplify_sum

# 各ラインのスキル値
a = amplify_sum(q[:, 0] * df_worker_skill.iloc[:, 0 + 1].values)
b = amplify_sum(q[:, 1] * df_worker_skill.iloc[:, 1 + 1].values)
c = amplify_sum(q[:, 2] * df_worker_skill.iloc[:, 2 + 1].values)

# 目的1: 全ラインのスキル値の合計（最大化したいもの）
skill_score = a + b + c
# display(skill_score)

# 目的2: 各ラインのスキル値の分散（最小化したいもの）
variance = (a * a + b * b + c * c) / num_locations - ((a + b + c) / num_locations) ** 2
display(variance)

# 目的関数（1つの最小化問題とするためにスキル値の項にはマイナスを追加）
objective = -skill_score + variance
# display(skill_score)
```

ばらつきは小さい方が選ばれるようにしたいのでプラスで足します

Step3

Step2に「ラインスキル値のばらつきを最小化」という目的2を追加して、2つの目的を同時に実現するシフトを求めます

結果の取得

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

可視化

ばらつきなし！

line_Aのラインスキル値: 480

loc	worker_id	worker_skill
line_A	0	130
line_A	1	120
line_A	4	90
line_A	8	80
line_A	13	60

line_Bのラインスキル値: 480

loc	worker_id	worker_skill
line_B	2	130
line_B	3	120
line_B	7	90
line_B	10	80
line_B	14	60

line_Cのラインスキル値: 480

loc	worker_id	worker_skill
line_C	5	120
line_C	6	110
line_C	9	90
line_C	11	80
line_C	12	80

全ラインの合計スキル値: 1440

確認

worker_id	worker_skill		
	line_A	line_B	line_C
0	130	60	70
1	120	55	60
2	110	130	60
3	100	120	55
4	90	110	130
5	90	100	120
6	80	90	110
7	80	90	100
8	80	80	90
9	70	80	90
10	70	80	80
11	70	70	80
12	60	70	80
13	60	70	70
14	55	60	70

二つの制約を満しながら、全ラインの合計スキルが高く、各ライン間のばらつきが全くないシフトを作ることができました（Step1の全ラインの合計スキル値は1,320で、Step2は1,475）。最後に、目的1と目的2のバランスをチューニングして最適なシフトの作成を目指します。

Step4

Step3に目的1と目的2の重みを調整する「パラメーター」を追加し、最適なバランスのシフトを作成します

実装

：追加コード

```
####################
# 目的関数の定式化
####################

from amplify import sum as amplify_sum

# 各ラインのスキル値
a = amplify_sum(q[:, 0] * df_worker_skill.iloc[:, 0 + 1].values)
b = amplify_sum(q[:, 1] * df_worker_skill.iloc[:, 1 + 1].values)
c = amplify_sum(q[:, 2] * df_worker_skill.iloc[:, 2 + 1].values)

# 目的1: 全ラインのスキル値の合計（最大化したいもの）
skill_score = a + b + c
# display(skill_score)

# 目的2: 各ラインのスキル値の分散（最小化したいもの）
variance = (a * a + b * b + c * c) / num_locations - ((a + b + c) / num_locations) ** 2
# display(variance)

# それぞれの目的の重みを調整するためのパラメータ
skill_priority = 2
var_priority = 1

# 目的関数（最小化問題とするためにスキル値の項にはマイナスを追加）
objective = -(skill_priority * skill_score) + var_priority * variance
```

Step4

Step3に目的①と目的②の重みを調整する「パラメーター」を追加し、最適なバランスのシフトを作成します

結果の取得

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

可視化

ばらつき極小！

line_Aのラインスキル値: 490

loc	worker_id	worker_skill
line_A	0	130
line_A	1	120
line_A	5	90
line_A	8	80
line_A	11	70

line_Bのラインスキル値: 480

loc	worker_id	worker_skill
line_B	2	130
line_B	3	120
line_B	9	80
line_B	10	80
line_B	13	70

line_Cのラインスキル値: 490

loc	worker_id	worker_skill
line_C	4	130
line_C	6	110
line_C	7	100
line_C	12	80
line_C	14	70

全ラインの合計スキル値: 1460

確認

worker_id	worker_skill		
	line_A	line_B	line_C
0	130	60	70
1	120	55	60
2	110	130	60
3	100	120	55
4	90	110	130
5	90	100	120
6	80	90	110
7	80	90	100
8	80	80	90
9	70	80	90
10	70	80	80
11	70	70	80
12	60	70	80
13	60	70	70
14	55	60	70

最適なバランスのシフトが完成しました！

ワークショップ：おさらい

制約のみからスタートして、複数の目的を加え、重みを調整する事で、最適なバランスのシフトを作りました。

Step1

スキル値合計：**1,320**

制約

worker_id	worker_skill		
	line_A	line_B	line_C
0	130	60	70
1	120	55	60
2	110	130	60
3	100	120	55
4	90	110	130
5	90	100	120
6	80	90	110
7	80	90	100
8	80	80	90
9	70	80	90
10	70	80	80
11	70	70	80
12	60	70	80
13	60	70	70
14	55	60	70

Step2

スキル値合計：**1,475**

制約

＋

スキル値最大化

worker_id	worker_skill		
	line_A	line_B	line_C
0	130	60	70
1	120	55	60
2	110	130	60
3	100	120	55
4	90	110	130
5	90	100	120
6	80	90	110
7	80	90	100
8	80	80	90
9	70	80	90
10	70	80	80
11	70	70	80
12	60	70	80
13	60	70	70
14	55	60	70

Step3

スキル値合計：**1,440**

制約

＋

スキル値最大化

＋

スキル値ばらつき調整

worker_id	worker_skill		
	line_A	line_B	line_C
0	130	60	70
1	120	55	60
2	110	130	60
3	100	120	55
4	90	110	130
5	90	100	120
6	80	90	110
7	80	90	100
8	80	80	90
9	70	80	90
10	70	80	80
11	70	70	80
12	60	70	80
13	60	70	70
14	55	60	70

Step4

スキル値合計：**1,460**

制約

＋

スキル値最大化

＋

スキル値ばらつき調整

＋

重みを調整

worker_id	worker_skill		
	line_A	line_B	line_C
0	130	60	70
1	120	55	60
2	110	130	60
3	100	120	55
4	90	110	130
5	90	100	120
6	80	90	110
7	80	90	100
8	80	80	90
9	70	80	90
10	70	80	80
11	70	70	80
12	60	70	80
13	60	70	70
14	55	60	70

FIXSTARS Amplify

Step5
（発展）

重みの自動調整を導入したサンプルコードです

2025年2月に実施した技術解説セミナー（定式化ベース）の資料にある「目的関数のスケーリング係数」を実装したもの

https://amplify.fixstars.com/ja/seminar/formulation20250219

p19/25 多目的最適化：発展的なスケーリング係数決定

- スケーリング係数の自動決定 (AE)
  1. 制約問題として求解
     - ➔ 短いタイムアウトで多くの解を取得
     - ➔ 制約を満たすランダム解とみなせる
  2. 解を個々の目的関数に代入、代入結果を目的関数のスケーリング係数とする
  3. 目的関数を除算しスケーリングを実施
     - ➔ 目的関数の取りうる値が 1 程度になることが期待される
  4. スケーリング後の目的関数と制約条件を考慮し、最適化問題として求解

FIXSTARS Amplify

```
obj_1, obj_2 = ...
const_1, const_2 = one_hot(...), less_equal (...)

# 制約問題として求解
model = const_1 + const_2

# 同じ目的関数値の解を複数取得するオプション
# （制約問題では、目的関数は常に0であるため）
client.parameters.outputs.duplicate = True

result = solve(model, client)

# s_1 と s_2 はそれぞれ obj_1 と obj_2 の代表値
s_1 = max([obj_1.evaluate(sol.values) for sol in result])
s_2 = max([obj_2.evaluate(sol.values) for sol in result])

# obj_1 と obj_2 のレンジが大きく異なる場合でも、対応可
model = obj_1 / s_1 + obj_2 / s_2 + const_1 + const_2

# 実際の最適化問題として求解
result = solve(model, client)
```

19/25

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

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

- 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

※Smileboard Connect: 住友商事様開発の物流業務可視化サービス

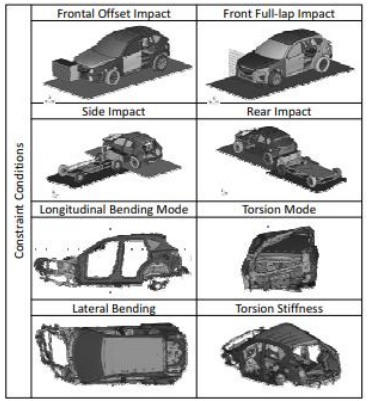
FIXSTARS Amplify

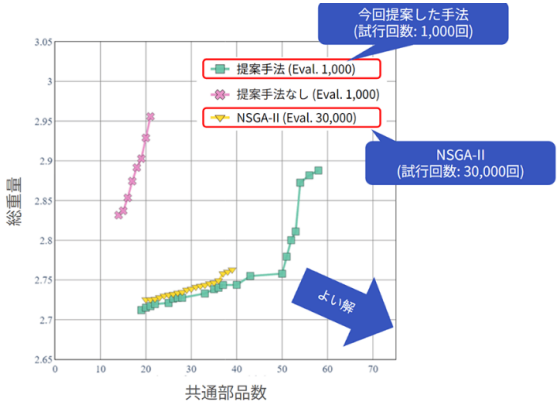
車体構造の設計最適化（マツダ株式会社）

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、で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

株式会社アパールデータ

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

学術利用

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

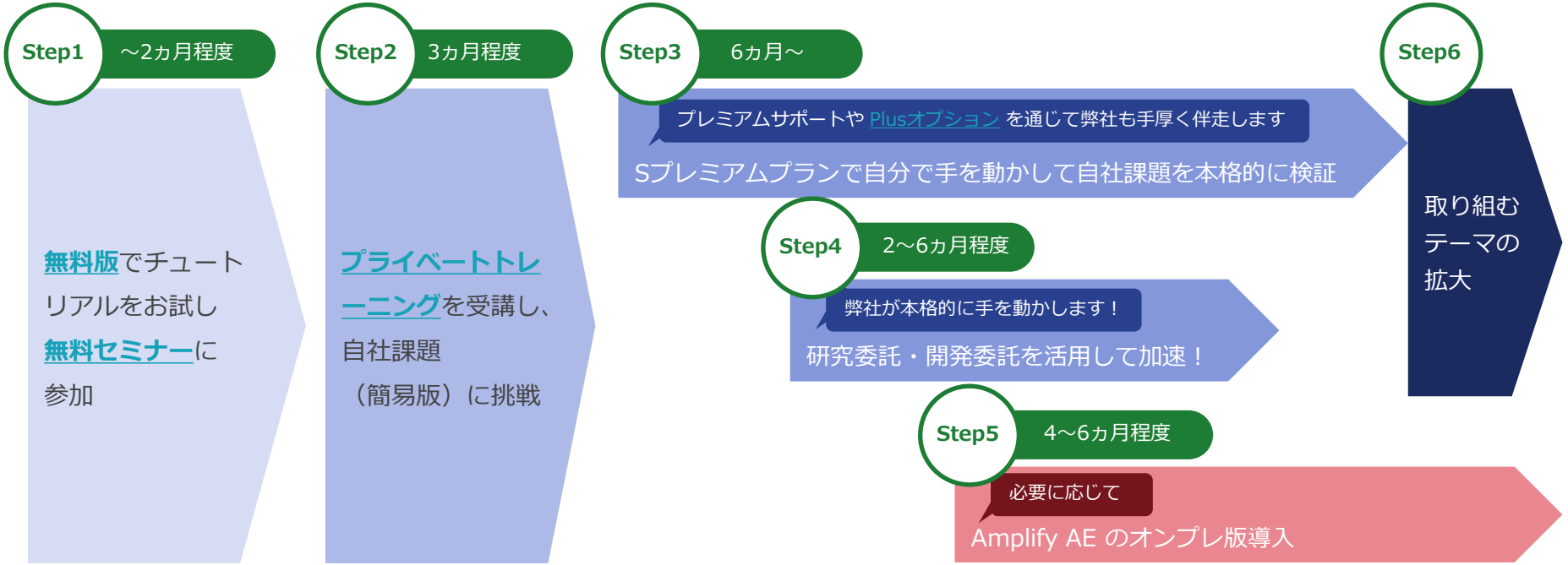
ユーザーインタビュー 2023年4月

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なって解きます。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・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
- 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)	ベーシック	スタンダード	プレミアム	Sプレミアム	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	**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)	**20万円** (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)	**40万円** (1アプリ)	**60万円** (1アプリ)
計算環境	**スモール**	**ミディアム**	**ラージ**	**スーパーラージ**	**ミディアム**	**ラージ**	**スーパーラージ**
利用GPU (マルチGPUオプションあり)	NVIDIA V100	NVIDIA V100	NVIDIA A100	NVIDIA H100	NVIDIA V100	NVIDIA A100	NVIDIA H100
1ジョブの実行時間 (実行時間延長オプションあり)	10秒	1分	10分	15分	1分	10分	15分
月間累計実行回数 (実行回数追加オプションあり)	制限の可能性あり	無制限			- (上限を設定する可能性あり)		
月間累計実行時間	制限の可能性あり	無制限			- (上限を設定する可能性あり)		
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**	**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	-	-	月額50万/人		-		

次ページ

FIXSTARS Amplify

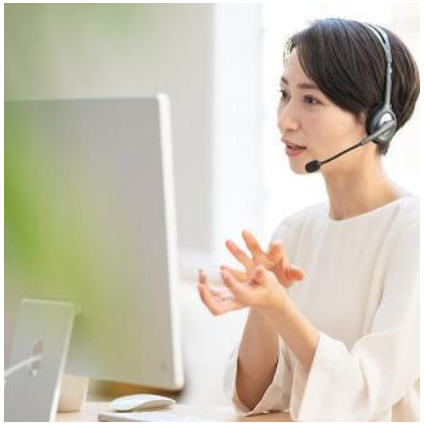
Plusオプション（プレミアムプラン/Sプレミアムプランで追加可能なオプション）

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が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

FIXSTARS Amplify

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

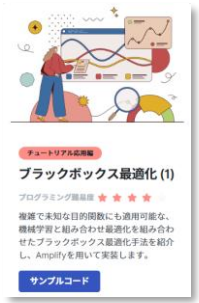
目的関数のみで定式化

制約条件のみで定式化

目的関数 + 制約条件

ブラックボックス最適化問題

概要



材料探索

翼型最適化

信号機制御

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に開催します！

2026/05/14（木）

2枠実施！

「Amplify BBOpt 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 BBOpt 技術解説
- Wrap Up & QA

2025/05/21（木）

「Amplify-BBOpt 技術解説」

~ 機械学習の機械学習の特徴量選択・コスト削減・精度向上 ~

- 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

2025/06/11（木）

2枠実施！

「ビジネスセミナー」

~ AI との違いと意思決定プロジェクトの進め方 ~

- 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
- 最適化のビジネス価値への転換プロセス
- 開発アプローチのご紹介
- 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

FIXSTARS Amplify

ご参加ありがとうございました

アンケートへのご回答お願いします！