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 ～

14:00 開始予定

マイク、カメラをOFFにしてしばらくお待ちください

Zoomの表示名は、セミナー申し込み時の
お名前としていただけますようご協力お願いいたします

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 ～

本日のAgenda

第一部 14:00-14:45

- はじめに
- 会社紹介
- 組合せ最適化事例のご紹介
- Fixstars Amplifyのご紹介
- ワークショップ事前準備のご連絡

第二部 14:50-16:00

- Fixstars Amplifyを用いた最適化ワークショップ
  - シフト最適化

- Wrap Up 及び Q&A

はじめに

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（イジングマシン）があることを知り、イジングマシンを使って問題を解くための統一的なフレームワークや、問題設定の考え方、目的関数や制約条件の定式化のポイントを理解する
- ワークショップを通して、実際に量子アニーリング・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

質問は随時Zoomのチャットか Q&A でお願いします

会社紹介

フィックスターズの基本情報

会社名	株式会社フィックスターズ
本社所在地	東京都港区芝浦3-1-1 msb Tamachi 田町ステーションタワーN 28階
設立	2002年8月
上場区分	東証プライム（証券コード：3687）
代表取締役社長	三木　聡

資本金	5億5,446万円
社員数（連結）	263名（2022年9月現在）
主なお客様	キオクシア株式会社 ルネサスエレクトロニクス株式会社 トヨタグループ（トヨタ自動車株式会社・ 豊田通商株式会社・株式会社デンソー） みずほ証券株式会社 キヤノン株式会社

グループ会社

2021/10/1 設立

Fixstars Solutions, Inc.
完全子会社
米国での営業及び開発を担当

(株)Fixstars Autonomous Technologies
株式会社ネクスティ エレクトロニクスとのJV
自動運転向けソフトウェアを開発

(株)Fixstars Amplify
完全子会社
量子コンピューティングのクラウド事業を運営

(株)Sider
完全子会社
開発支援SaaS「Sider」を運営

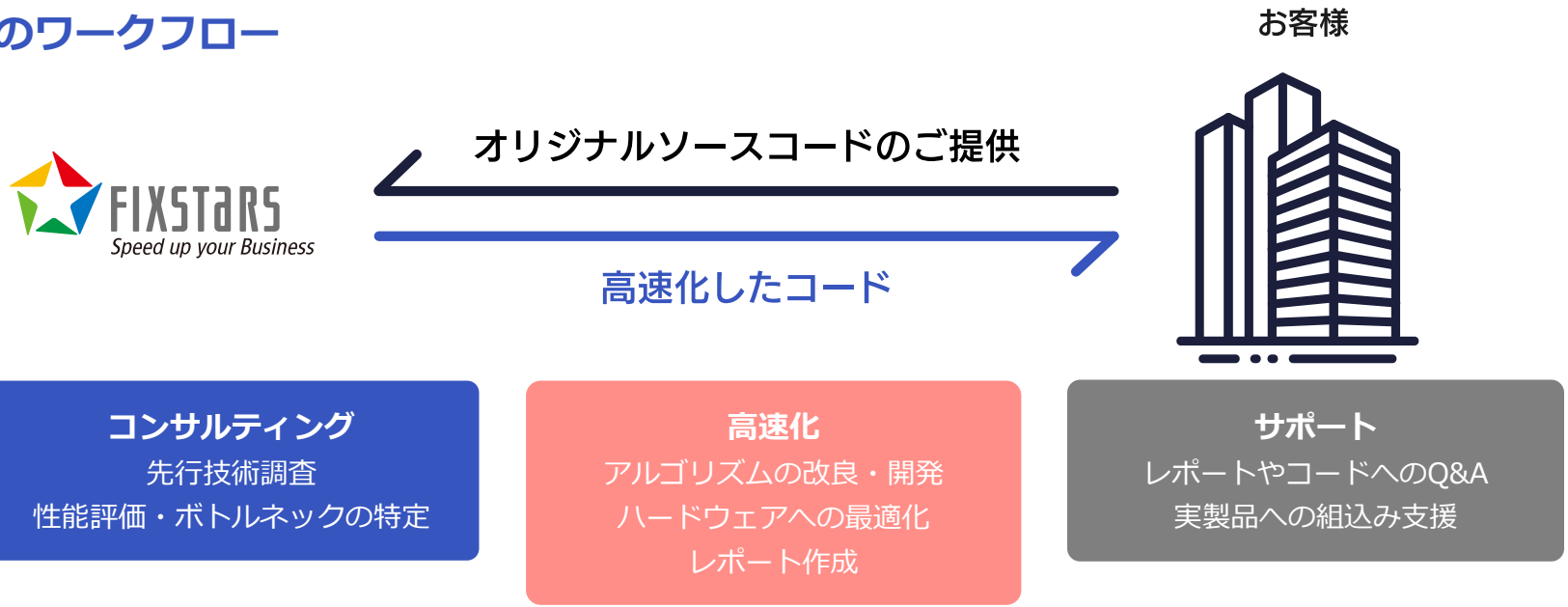
(株)Smart Opinion
連結子会社
乳がんAI画像診断支援事業を運営

オスカーテクノロジー(株)
連結子会社
ソフトウェア自動並列化サービスを提供

フィックスターズのサービス概要

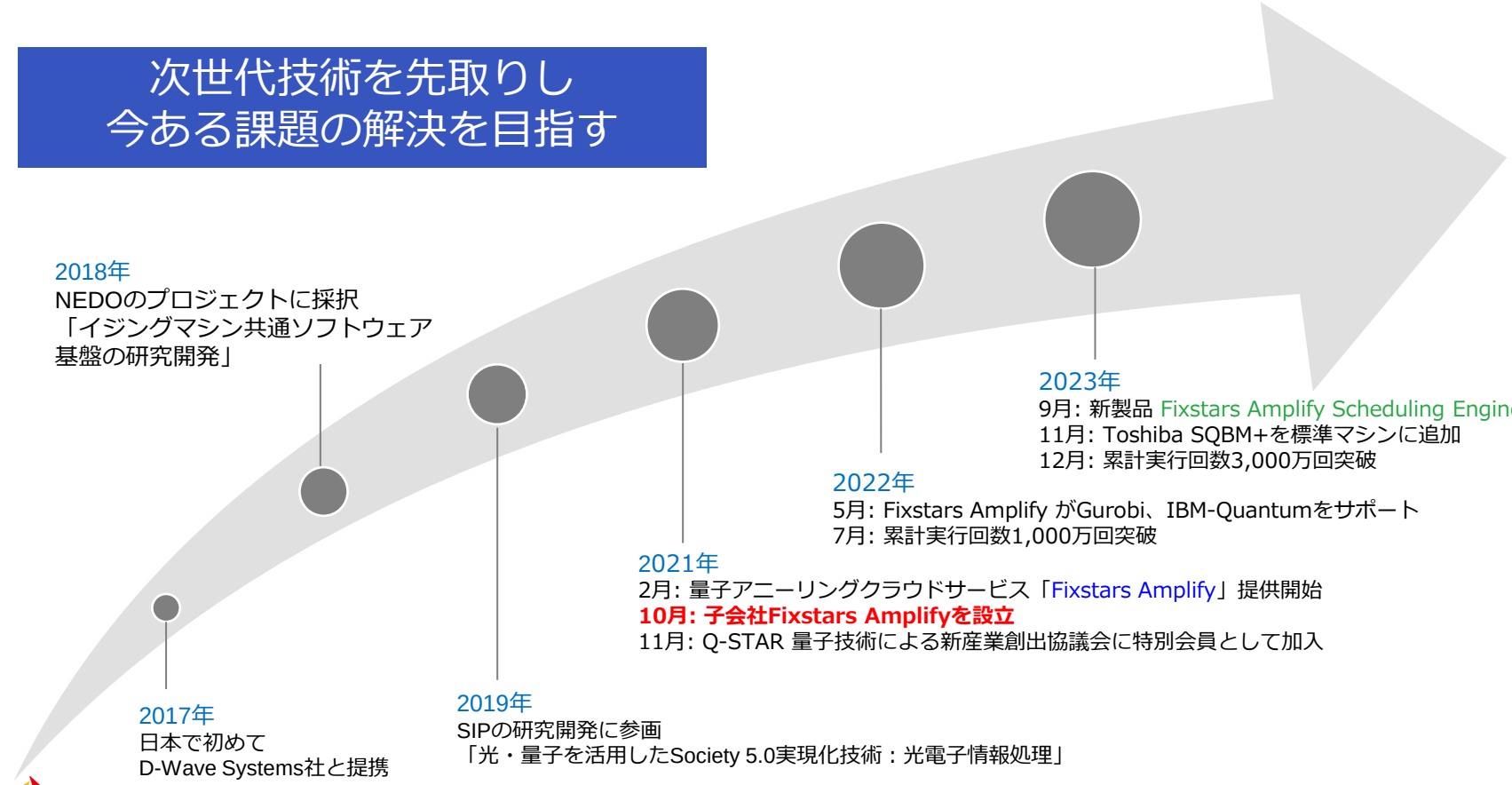
お客様専任のエンジニアが直接ヒアリングを行い、**高速化**を実現するために乗り越えるべき課題や問題を明確にしていきます。

高速化のワークフロー



フィックスターズの量子技術への取り組み

次世代技術を先取りし
今ある課題の解決を目指す



2017年
日本で初めて
D-Wave Systems社と提携

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」

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
2月: 量子アニーリングクラウドサービス「Fixstars Amplify」提供開始
10月: 子会社Fixstars Amplifyを設立
11月: 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
5月: Fixstars Amplify がGurobi、IBM-Quantumをサポート
7月: 累計実行回数1,000万回突破

2023年
9月: 新製品 Fixstars Amplify Scheduling Engine リリース
11月: Toshiba SQBM+を標準マシンに追加
12月: 累計実行回数3,000万回突破

FIXSTARS Amplify

量子コンピューティング事業

多様なハードウェアでのソフトウェア高速化サービスに加え、
量子コンピュータ活用支援とシステム開発を提供しています。

お客様の課題

量子コンピューティングが課題の解決に役立つか確信が持てない

量子コンピューティングの検討をどう進めたら良いかわからない

作りたいアプリケーションがあるが、開発が難しい

ご支援内容

セミナー・トレーニング
量子コンピュータの研究動向や活用事例、実際の利用方法等

クラウド実行環境のご提供
クラウド経由での量子コンピュータ利用サービスを提供

コンサルティング
セットアップ支援、処理の分割や変換等のコンサル

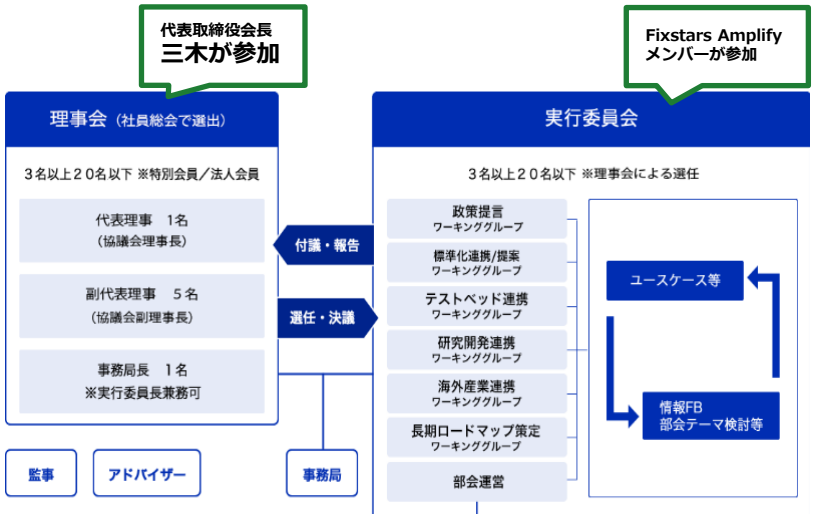
ソフトウェア高速化・開発支援サービス
量子コンピュータを組み合わせてシステムの高速化を実現

FIXSTARS Amplify

産業界での活動

Q-STAR 量子技術による新産業創出協議会

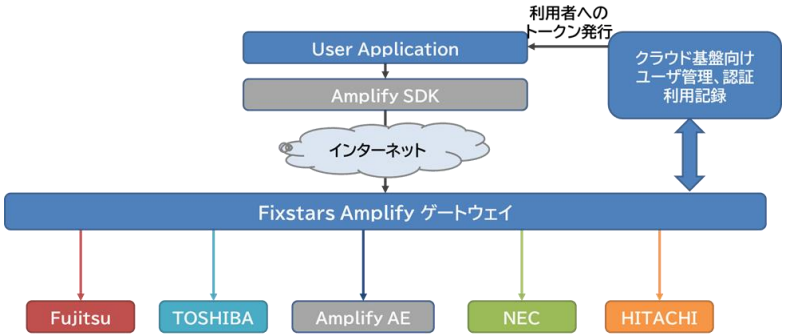
量子関連の産業・ビジネスの創出を目的として設立された協議会



出展：一般社団法人　量子技術による新産業創出協議会　https://qstar.jp/

Q-STAR/NEDO 量子テストベッド*に Amplifyが採用

*国内の量子アニーリングマシンを統一されたインターフェースで利用できるクラウド基盤の開発・検証プロジェクト



※構想時点での図です。変更される場合があります

組合せ最適化事例のご紹介

量子アニーリング・イジングマシンと組合せ最適化問題

膨大な選択肢から、**制約条件**を満たし、**ベスト**な選択肢を探索する（組合せ最適化問題）

スケジューリング

配送計画

スマートシティー

集積回路設計

参考: 慶應義塾大学 田中宗 准教授 「量子コンピュータ最前線とイジングマシンの可能性」

量子アニーリング・イジングマシン ⇒ 組合せ最適化問題を解くための**専用マシン**

FIXSTARS Amplify

組合せ最適化問題を解く統一的なフレームワーク

問題設定

どのような状況で何を最小化 (最大化) したいのか
膨大な解候補（組合せ）から最適解を選ぶ

定式化

決定変数を使って目的関数と制約条件を数式で表現

- **決定変数**: 0か1を取る変数
- **目的関数**: これが最小になる解 (決定件数の組合せ) を探したい
- **制約条件**: 解が必ず満たすべき条件

実装

数式をPythonのプログラムで記述

解を取得

Fixstars Amplify AE（イジングマシン）が
最適解を探索

組合せ最適化問題の例

バイキングで**最も安く**
必要な栄養が取れる組合せは？

(10g単位)	炭水化物	タンパク質	脂質	金額
ごはん	8g	1g	1g	10円
パン	7g	1g	2g	12円
ハンバーグ	1g	5g	4g	50円
焼き魚	1g	8g	1g	35円

目的関数: 合計金額（最小化）

制約条件:
炭水化物：300g以上
タンパク質：150g以上
脂質：50g以上

最適メニュー → ごはん：380g
焼き魚：140g
金　額：870円

組合せ最適化の取り組み事例

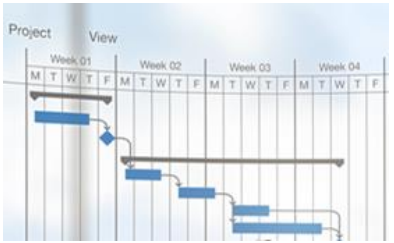
シフト作成自動化

人の直観で時間をかけて行っていた生産ラインや物流倉庫の業務シフト作成を、スキルや勤務時間などの条件をもとに最適化します

本日のセミナーのテーマ

生産計画最適化

製造工場の設備の利用割り当て(ジョブショップスケジューリング) を、納期や段取り時間などを考慮して最適化します



4/24のセミナーのテーマ

経路指示リアルタイム制御

倉庫を走行する多数の搬送ロボット (AGV) が効率よく動作するよう、最適経路だけでなく迂回や交差点での待機などリアルタイムに指示します

TBD

最適配置自動化サービス（物流梱包業務のDX）

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

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

FIXSTARS Amplify

生産計画最適化 (電気機器製造メーカー A社様)

複数の製品事業部から様々なプリント基板の注文を受け、生産を行う部門

課題

生産する基板に応じて製造装置の部品や材料を交換する「段取り時間」が必要。段取り時間を考慮した効率的な生産スケジュールを作成したい
従来は、専任者が、一日数回・毎回数十分かけて経験に基づいてスケジュールを作成。更なる生産性向上やノウハウ継承のため、生産スケジュール作成の自動化に着手

FIXSTARS Amplify

効果

生産スケジュール作成の時間・コストの大幅な削減！
(一日あたり数時間 → 数分)

段取りのための製造装置の停止回数の削減！
(10%以上削減)

最適化未経験のご担当者様1人がプログラム試作開始から約1～2ヵ月間取り組んでこの効果を実現
現在は試作段階で、実運用に向けてモデルを改良中！

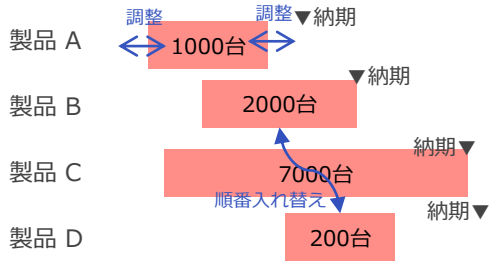
次期フェーズでは、Amplifyの活用領域の拡大を検討中！

生産計画最適化 (機器製造メーカー B社様)

各製品の生産数に応じて製造人員数の計画をする部門

課題

少量多品種の機器を製造している。それぞれの製品は生産が開始できる時期や納期が決まっていて、各製品をどの生産ラインでいつ製造するかによって日々の必要人員数が変化する。人員数と日毎の人員数の増減を最小化したい。
従来は、専任者が2名で日々計画を立案・修正していたが、効率的な生産計画の立案、業務負荷削減、属人化解消のため自動化に着手。

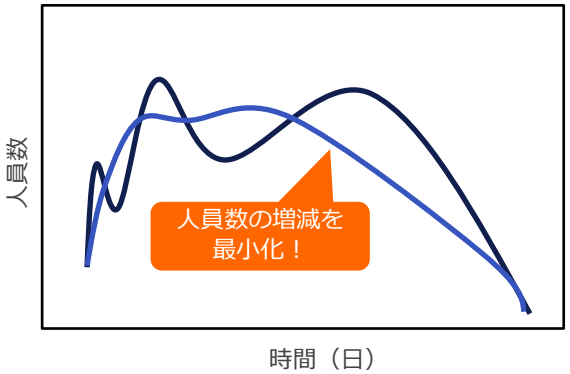


FIXSTARS Amplify

効果

必要人員数と日毎の人員数の増減の最小化！

計画立案の業務負荷削減・属人化解消！



Fixstars Amplify を活用した研究事例

	大学	研究室	論文タイトル
量子アニーリング/イジングマシンに関する研究	早稲田大学	戸川研究室	イジングマシンによる制約付きグラフ彩色問題の彩色数最小化手法 (リンク)
	東京大学	Prof. Codognet	Modeling the Costas Array Problem in QUBO for Quantum Annealing (リンク)
	名古屋大学	片桐研究室	Amplifyを用いたCMOSアニーリングマシンの特性の分析 (リンク)
	東北大学	小松研究室	組み合わせクラスタリングによるアニーリングマシンの評価 (リンク)
応用研究	慶應大学	村松研究室 (材料工学)	Phase-fieldモデルの量子アニーリングシミュレータ (リンク)
	東京大学	長谷川研究室 (量子ゲート)	ISAAQ:イジングマシンを活用した量子コンパイラ (リンク)
	山梨大学	鈴木研究室 (情報工学)	量子アニーリングによる疎行列直接解法向けフィルイン削減オーダリング (リンク)
	東京大学	津田研究室 (MI)	Designing metamaterials with quantum annealing and factorization machines (リンク)
	京都大学	野田研究室 (電子工学)	量子アニーリングを活用したフォトニック結晶レーザーの構造最適化 (リンク)
	東京大学	津田研究室 (MI)	Chemical Design with GPU-based Ising Machines (リンク)

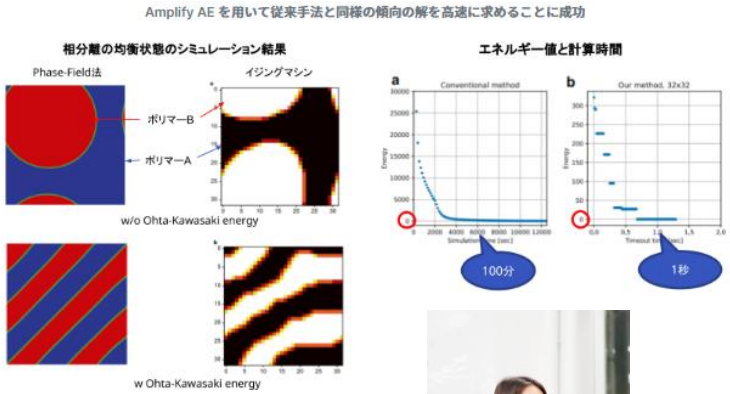
ブラックボックス最適化

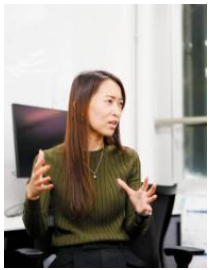
アカデミアにおける応用事例: インタビュー記事

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

慶應義塾大学 理工学部 機械工学科 村松研究室

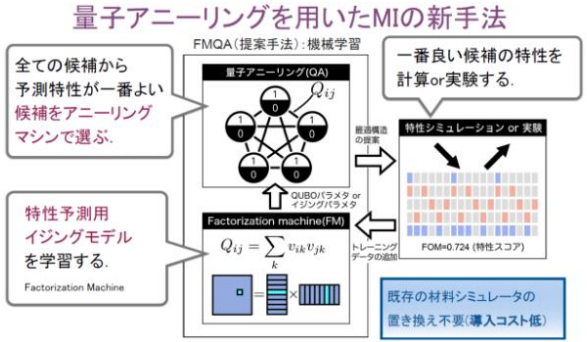
Fixstars Amplifyを使って、次世代技術を活用した高速な解析手法の開発に成功

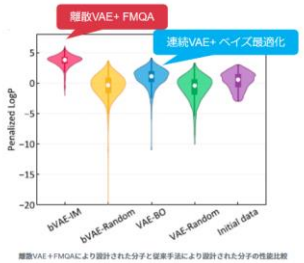




東京大学大学院 新領域創成科学研究科

ブラックボックス最適化手法（FMQA）の開発に成功





Fixstars Amplify：メディア掲載

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

Software Design

「はじめての量子プログラミング体験」
（2021年6月号～2022年1月号まで連載）

Interface

「Pythonで体験！量子コンピュータ」
（2022年6月号）

Fixstars Amplify のご紹介

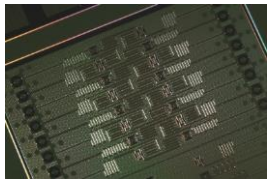
量子技術とFixstars Amplifyの対応領域

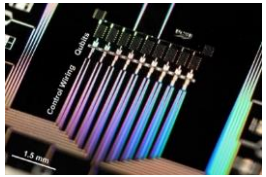
1. 量子コンピュータ

量子ゲート方式

古典汎用コンピュータの上位互換。量子力学の重ね合わせ状態を制御する量子ゲートを操作し、特定の問題を汎用的かつ高速に処理する。

QAOAにより組合せ最適化問題(**QUBO**)を取り扱うことが可能。





2. 量子アニーリング方式

量子焼きなまし法

イジングマシンの一種であり、量子焼きなまし法の原理に基づいて動作する。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する。自然計算により低エネルギー状態が出力される。組合せ最適化問題(**QUBO**)を扱う専用マシン。

3. イジングマシン

二値二次多項式模型

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**)を扱う専用マシン。

変数は0,1または±1。統計物理学におけるイジング模型(磁性体の性質を表す模型)に由来。様々な実装により実現されている。

Amplify AE

二次計画問題

組合せ最適化問題

- 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)
- 連続最適化問題
  - 決定変数が連続値（実数など）

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模型 (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

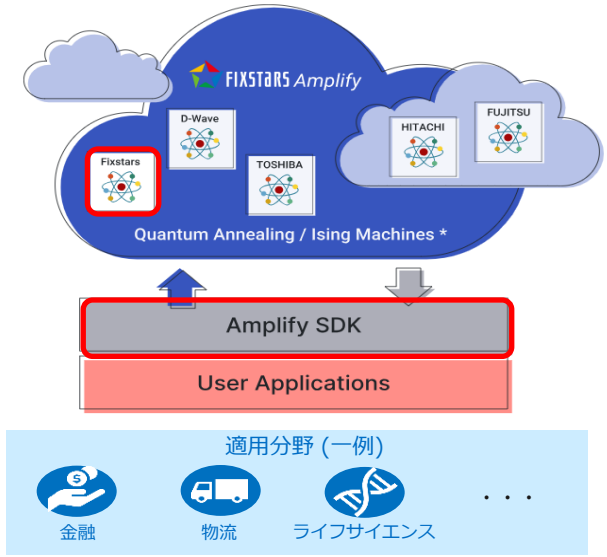
目的関数　決定変数　係数

クラウドサービス: 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独自開発のGPUアニーリングマシンなど、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

イジングマシンの実行手順

1. 数理モデル検討　解きたい課題の「目的関数」「決定変数」「制約条件」を検討する

2. QUBO定式化 (論理)　「 2値決定変数+二次形式」で「目的関数」と「決定変数」を記述 (変換) する
「制約条件」は直接扱えないので「ペナルティ関数」で表現する

3. QUBO定式化 (物理)　各マシンの仕様や制限に準拠した形式にQUBO模型を変換する
(例: 二次項に制約がある場合は「グラフマイナー埋め込み」問題を解く)

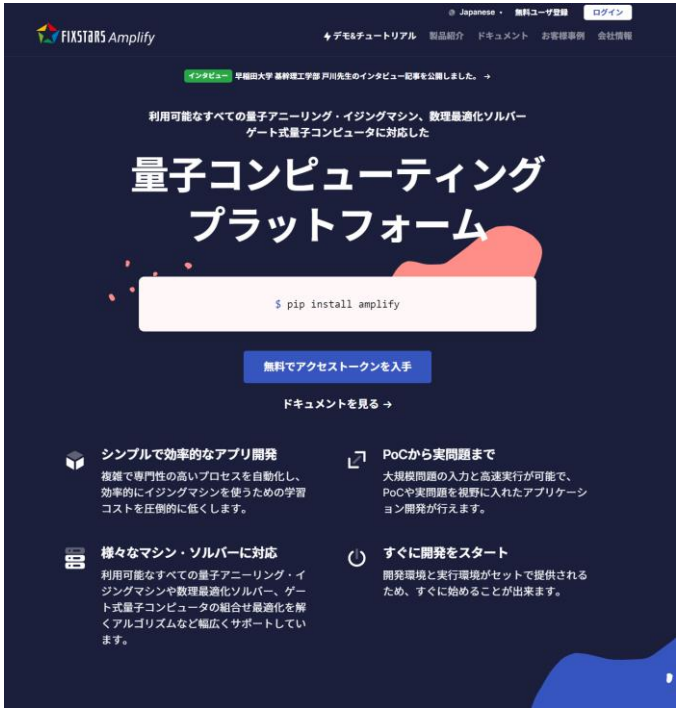
4. 入力データの準備　各マシンのSDKやAPI仕様に合わせてQUBO模型 (物理) をデータ化する

5. マシンの実行　マシンを実行して出力の変数値やエネルギー値(コスト値)を解析する
上記の逆の手順を辿り解きたい課題の「決定変数」を解釈する

FIXSTARS Amplify
SWによる
支援と自動化

Fixstars Amplify の特長

- **いつでも** 開発環境と実行環境がセットのため
  すぐにプログラミングと実行が出来る
- **誰でも** ハードウェアや専門的な知識が不要
  無料で開発がスタート可能
  多くの解説、サンプルコード
- **高速に** 26万ビットクラスの大規模問題の
  高速処理と高速実行が可能
- **あらゆる** 一般に公開されている全てのイジング
  マシンを利用可能



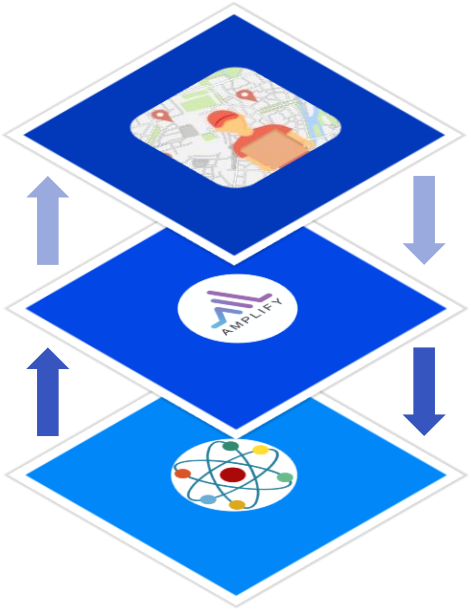
Fixstars Amplify クラウドの構成

アプリケーション

組合せ最適化問題を含むアプリケーションは
Amplify SDK を用いることで**効率的かつ簡便**に組合せ最適化の定式化が可能
評価・検証ではAmplifyクラウドは**無償提供**されるため**直ぐに開発をスタートできる**

量子アニーリング・イジングマシン (Amplify AE)

Amplify クラウドが量子アニーリングマシン・イジングマシンの**大規模実行環境を提供**
Amplify SDK で書かれた最適化エンジンは
他社製も含め**全ての商用イジングマシンで実行可能**



Amplify SDK

組合せ最適化問題の定式化やマシンを高度に操るための**最先端の技術や知見をライブラリ化**したシンプルなインターフェースを提供
専門知識が不要で量子アニーリングイジングマシンを用いた最適化エンジンの開発に取り組める

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」

https://news.fixstars.com/4025/ ：2024/11/28 SQBM+を標準マシンとして追加

Fixstars Amplify の技術

- Fixstars Amplify SDK（開発環境）
- Fixstars Amplify Annealing Engine（実行環境の1つ）

開発環境：Fixstars Amplify SDK

Fixstars Amplify SDK ならアニーリングのプログラミングが圧倒的に短縮されます

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。

開発環境インストール

```
$ pip install amplify
```

最適化コード例

```python
from amplify import VariableGenerator, FixstarsClient, solve

# 入力モデルの構築
q = VariableGenerator().array("Binary", 2)
f = 1 - q[0] * q[1]

# 実行マシンの設定
client = FixstarsClient()
client.parameters.timeout = 1000

# アニーリングの実行
result = solve(f, client)

# 結果の解釈
solution = q.evaluate(result.best.values)

print(f"result: {q} = {solution}")
# result: [q_0, q_1] = [1. 1.]
```

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

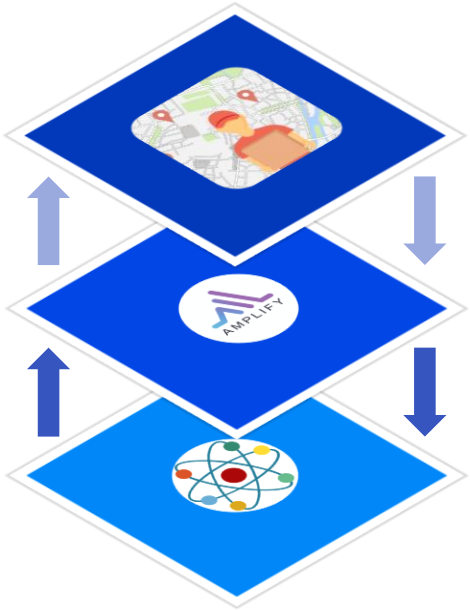
SDKが、各マシンに対して最適な形式に実装式を多段変換！

FIXSTARS Amplify

Amplify Annealing Engine

- NVIDIA GPU V100/A100 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
  - 社会課題への取り組み・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かつ最高速レベル
  - 128Kビット (全結合) / 256Kビット超 (疎結合)

Amplify Cloud



実行環境：Fixstars Amplify Annealing Engine (AE)

NVIDIA GPU V100/A100 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

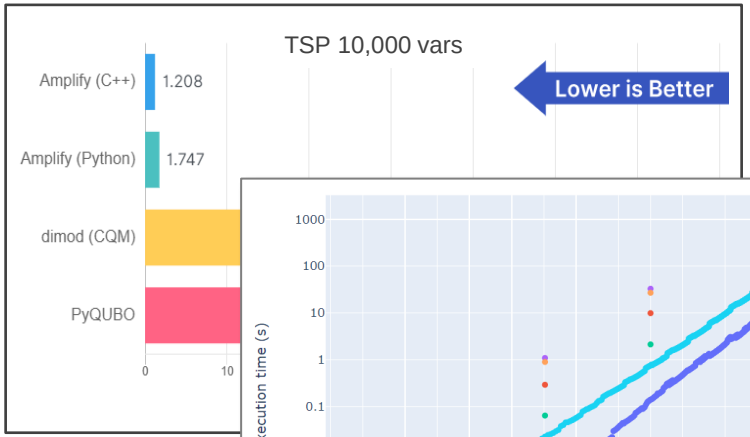
FIXSTARS Amplify

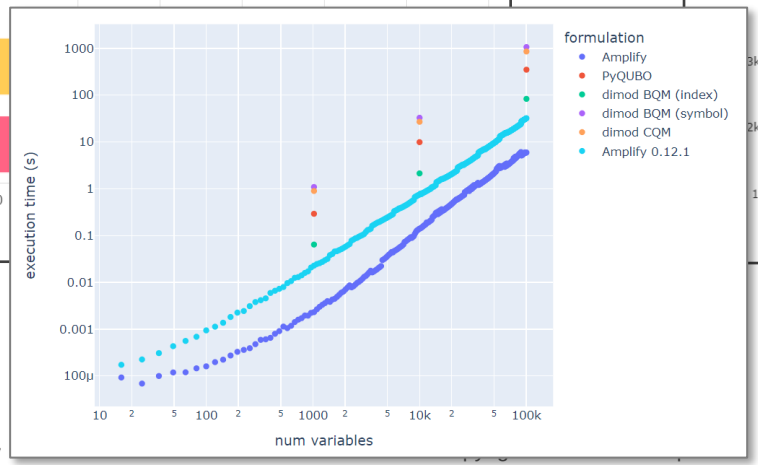
Amplify SDK/AE パフォーマンス

Amplify は最速レベルの定式化・求解速度を達成しています

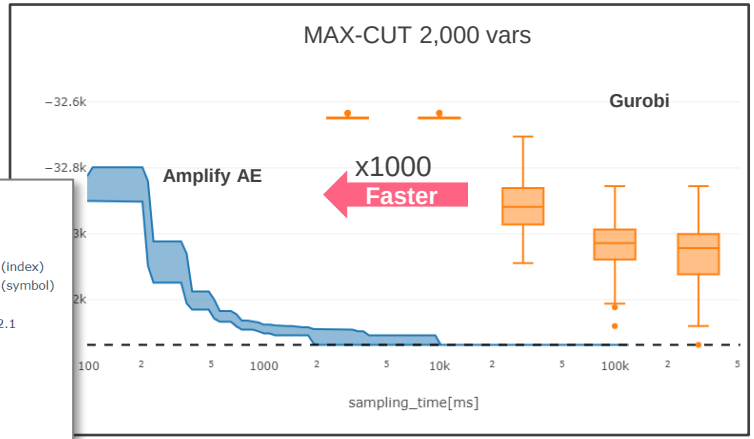
SDK 定式化処理速度

AE 求解性能・速度





2024/1: SDK v1
でさらに高速に



FIXSTARS Amplify

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

デモアプリケーション

ピクロスパズルの求解

プログラミング難易度 ★★★★

複雑な定式化の例として、数字で与えられるヒントを元にマスを塗り、絵を完成させるパズルゲーム、ピクロスを解くアプリを開発します。

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 (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

様々な分野で利用が拡大しています

豊田通商株式会社 TOYOTA TSUSHO CORPORATION

NTT DATA

NRI 野村総合研究所 Nomura Research Institute

住友商事株式会社

IPA Better Life with IT

登録社・組織数: 約600

累計実行回数: 3,500万回超

Fusic Co.,Ltd.

慶應義塾 Keio University

早稲田大学

株式会社ネクスティ エレクトロニクス

豊田中央研究所

Fixstars Amplify ご利用プラン

料金のご紹介

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

Fixstars Amplifyクラウド利用料

→ 近日中に新プラン発表予定

	ベーシックプラン 評価・検証用の無料プラン 使い始める	スタンダードプラン 実運用レベルで使いたい人は 見積もりを依頼する	プレミアムプラン 最高性能で計算したい人は 見積もりを依頼する	
利用料金	無料	月額10万円（税込11万円）	月額20万円（税込22万円）	月額60万円（税込66万円）
ユーザー数	1	1	1	5
計算環境	スモール	ミディアム	ラージ	
D-Waveマシンの無料実行	3分/月	3分/月	3分/月	
サポート	ベーシック	スタンダード	プレミアム	
評価・検証フェーズでの利用	✓	✓	✓	
実運用フェーズでの利用		✓	✓	

定式化や実装を
手厚く
支援します！

開発支援サービス(個別見積り)

コンサル・システム開発等
数百万円～数千万円

＋

月額利用料
百万円～

セミナー・トレーニングのご紹介

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

お客様の実際の課題解決をご支援するために、無料セミナーや有償トレーニングを提供しています。

無料セミナー・ワークショップ

ビジネス向け、エンジニア向けに分けて
開催しています！

ビジネス向け

製造業向け量子コンピュータ時代のDXセミナー

見える化、予測・分析、その先の最適化へ

組合せ最適化問題や量子アニーリング・イジングマシンの概要をご紹介したのち、製造業における組合せ最適化を活用したDX推進の一例として、生産計画最適化や生産ラインのシフト最適化などの事例とデモをご紹介いたします。「Fixstars Amplify」を通じて量子アニーリング・イジングマシンを活用することで、どのようなビジネス上の効果が期待できるのかを感じていただきたいと思います。

エンジニア向け

製造業向け量子コンピュータ時代のDXセミナー

最適化の中身を覗いてみよう

製造業における組合せ最適化を活用したDX推進の一例として、生産計画最適化、勤務シフト最適化などの事例を用いて、問題設定の考え方、目的関数や制約条件の定式化、実装のポイントなどを実際のコードを見ながら解説します。また、サンプルコードを用いて、ご自身の環境で実際に量子アニーリング・イジングマシンを動かす体験をしていただきます。

企業向けプライベートトレーニング

お客様が抱える実際の課題やデータを使った
カスタムメイドのトレーニングです！

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1.5か月のコースです。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。後半では、お客様が抱える実際の課題やデータを使ったトレーニングを実施します。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んでみたい方に最適なコースです。

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 (ブラウザ上) でPythonプログラミングを行います。Google Colaboratoryを使うので、事前にログイン出来ることを確認をお願いします（要Googleアカウント）

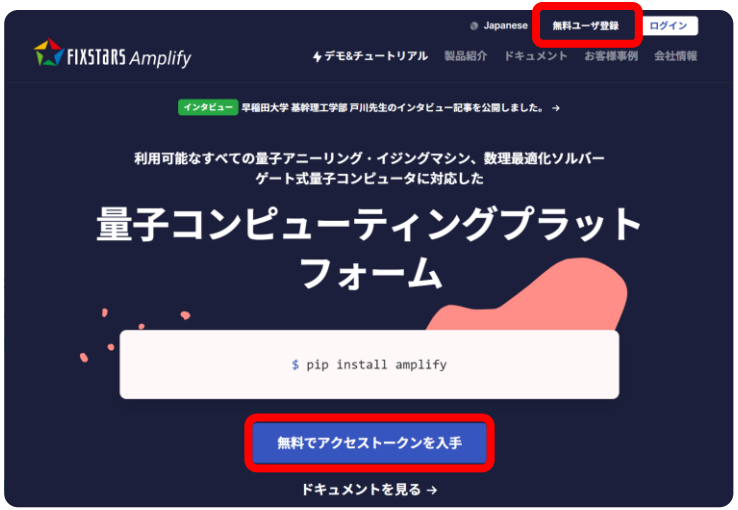
Google Colab 検索

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

- Fixstars Amplify ホームページよりユーザ登録の上、無料トークンの取得をお願いします（1分で終わります）

Fixstars Amplify 検索

https://amplify.fixstars.com/



ワークショップの事前準備 (2)

【事前メールに記載】

- 取得されたトークンを用いて、トークンチェック用サンプルコードが動くか確認をお願いします

https://colab.research.google.com/drive/1evYBKqKfVrEzrQOa-SWwciROfvqjL8qm?usp=sharing (URLはZoomのチャット欄を参照)

- サンプルコードは閲覧のみ可能な状態です。「ファイル」→「ドライブにコピーを保存」の上、ご自身のトークンを入力してください。その後、Shift + Enterで実行下さい。

```
! pip install amplify

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

「Amplify AE」のセクションにある「Amplify AE Basic」プランが今回使う無料トークンになります

- ご自身のトークン番号は、Amplifyウェブページ → よりご確認いただけます。

- 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/1M9_fty7GQ4gPVa87lkLA9jdJR2OEqvvC?usp=sharing
Step2	https://colab.research.google.com/drive/1ZSgLMVgZTjIGODDy_YIJy29zr827OI6Y?usp=sharing
Step3	https://colab.research.google.com/drive/100KyhLplCh9oZ854BEXY0aadeE3IkSWl?usp=sharing
Step4	https://colab.research.google.com/drive/1wzffv95TAr1cx7j_WiYnXEWzU1z3uIZE?usp=sharing

質問は随時、Zoomの チャット か Q&A でお願いします。
対応可能なメンバーが対応致します。

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

担当可能役職

従業員ID	責任者	主任技師	技師
0	1	1	1
1	0	1	1
2	0	0	1
⋮			
23	0	0	1
24	0	0	1

スキル値(1=初級, 2=中級, 3=上級)

前工程	後工程	組立て
3	3	3
3	3	2
2	1	0
⋮		
0	0	2
1	0	1

FIXSTARS Amplify

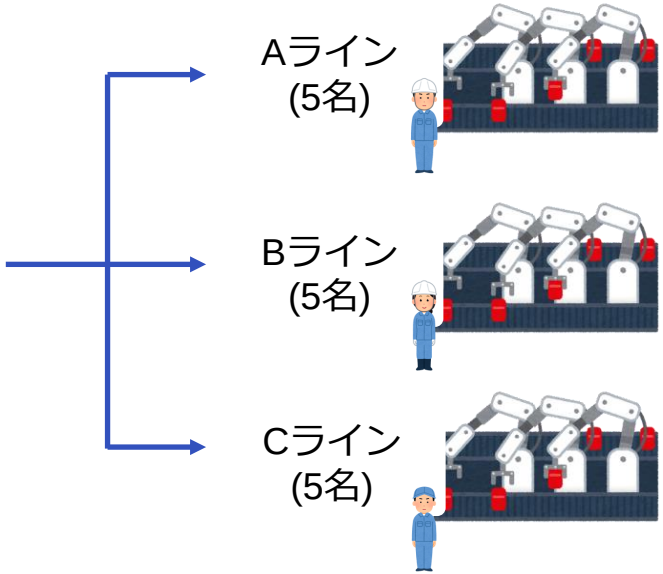
ワークショップ: 問題設定

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**、また、各ラインのラインスキル値の**ばらつきが少ない**、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

組合せは
約1,400万通り!



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る。各従業員は、各ラインに対するスキル値を持ち、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**、また、各ラインのラインスキル値の**ばらつきが少ない**、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

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

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、また、各ラインのラインスキル値のばらつきが少ない、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アルゴリズ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

Step1のサンプルコードのレビュー

(尚、本ワークショップでは、最適化のコードにフォーカスし、下準備や可視化のコードの詳細は割愛します)

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

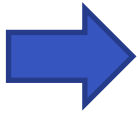
決定変数の準備

BinaryPoly型
15×3 = 45 [qbit]

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)の組合せを探す



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

実装

```
# 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)
```

決定変数

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 sum, 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 = 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 = 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
```

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 FixstarsClient, solve

# 実行マシンクライアントの設定
client = FixstarsClient()
client.token = token
client.parameters.timeout = 1 * 1000  # タイムアウト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

- 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。

無料版は1ジョブ10秒まで設定可
有料版では1分まで設定可能

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の合計スキル値: 430.0

loc	worker_id	worker_skill
line_A	0	130
line_A	4	90
line_A	6	80
line_A	10	70
line_A	13	60

line_Bの合計スキル値: 440.0

loc	worker_id	worker_skill
line_B	2	130
line_B	7	90
line_B	8	80
line_B	11	70
line_B	12	70

line_Cの合計スキル値: 395.0

loc	worker_id	worker_skill
line_C	1	60
line_C	3	55
line_C	5	120
line_C	9	90
line_C	14	70

全ラインの合計スキル値: 1265.0

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。

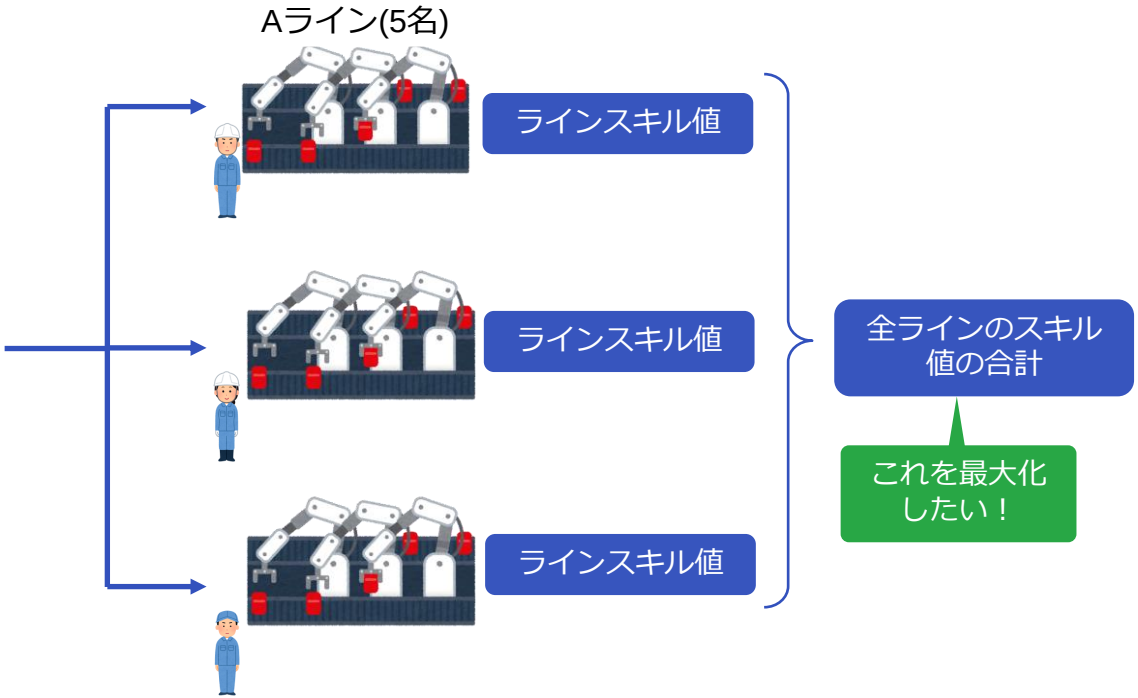
Step2

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

Step2のサンプルコードのレビュー

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

：追加コード

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

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

# 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)
```

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

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

from amplify import sum, 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 = 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 = sum(
    equal_to(q[:, l], df_req["worker"][l])
    for l in range(num_locations)
)
# display(req_constraints)

# 制約の重み
constraint_weight = 150

# 制約
constraints = constraint_weight * (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)

制約条件には適切な値の重みを設定する必要があります。典型的なスキル値より大きくしておく必要があるため 150 としました

ご参考: https://amplify.fixstars.com/ja/docs/amplify/v1/penalty.html#id1

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　line_Bのラインスキル値: 490　line_Cのラインスキル値: 530

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

loc	worker_id	worker_skill
line_B	2	130
line_B	3	120
line_B	7	90
line_B	10	80
line_B	13	70

loc	worker_id	worker_skill
line_C	4	130
line_C	5	120
line_C	6	110
line_C	9	90
line_C	12	80

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:　1475.0

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1,265) 。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

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

(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
####################

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

# 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

ばらつき全くなし！

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	5	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	6	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	4	130
line_C	7	100
line_C	9	90
line_C	11	80
line_C	12	80

全ラインの合計スキル値： 1440.0

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1,265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
####################

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

# 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: 490

loc	worker_id	worker_skill
line_B	2	130
line_B	3	120
line_B	7	90
line_B	10	80
line_B	13	70

line_Cのラインスキル値: 480

loc	worker_id	worker_skill
line_C	4	130
line_C	6	110
line_C	9	90
line_C	12	80
line_C	14	70

全ラインの合計スキル値: 1460.0

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：**1,265**

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

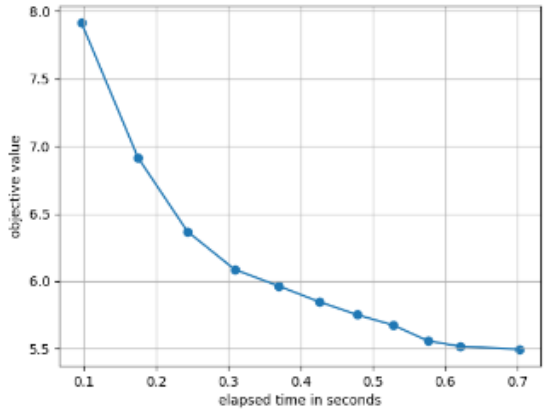
Wrap Up

Tips: 適切なタイムアウト値の設定について

実行後にヒストリー情報や num_iterations などを確認しながら適切なタイムアウト値を調整します

ヒストリー情報 (ドキュメントはこちら)

- Amplify AE は1回の実行の中で探索を繰り返し、時間の許す限り最良解を徐々に改善していきます。
- 必要な設定変更を行うことにより、1回の実行の中で、最良解を更新した時間とその解の値を確認することができます。解の収束の様子などから実行時間の過不足の判断に用いることができます。



num_iterations (ドキュメントはこちら)

- 実行後に num_iterations を確認することにより、どの程度の「探索」が行われたか確認することができます。「探索」とは、Amplify AE が実装しているアルゴリズムの実行単位であり、この値が大きいほど広く探索が行えたことを意味します。この値が一桁など小さい場合には探索が十分でない可能性があります。
- Amplify AE は初めに1回だけ探索を行い、タイムアウトまで余裕がある場合には、時間の許す限り解を徐々に改善していくという動作になっています。設定したタイムアウト値が問題規模に対して小さすぎる場合には、最初の探索が指定したタイムアウト時間内に終わらないこともあります。その場合、初めの探索だけで実行が終わり、num_iterations は 1 となります。

```
print(result.client_result.execution_parameters.num_iterations)
```

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

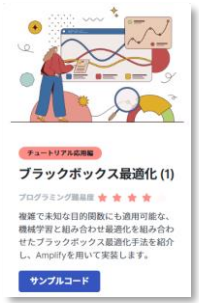
目的関数のみで定式化

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

セミナーのご案内

今後も無料セミナーを開催します！第二部からの参加も可能です

Amplify SE を使った
セミナーです！

2024/3/27 (水)
「シフト最適化」

第一部 14:00 - 14:45
- フィックスターズの紹介
- 組合せ最適化問題・イジングマシンの紹介
- Fixstars Amplifyの紹介

第二部 14:50 - 16:00
- Fixstars Amplifyを用いた「シフト最適化」のワークショップ
- Q&A

2024/4 (仮)
「生産計画最適化」

第一部 14:00 - 14:45
- フィックスターズの紹介
- 組合せ最適化問題・イジングマシンの紹介
- Fixstars Amplifyの紹介

第二部 14:50 - 16:00
- Fixstars Amplifyを用いた「生産計画最適化」のワークショップ
- Q&A

2024/5 (仮)
「搬送経路最適化」

第一部 14:00 - 14:45
- フィックスターズの紹介
- 組合せ最適化問題・イジングマシンの紹介
- Fixstars Amplifyの紹介

第二部 14:50 - 16:00
- Fixstars Amplifyを用いた「搬送経路最適化」のワークショップ
- Q&A

セミナーの内容等に関してご質問・ご不明点がありましたら、問合せフォームでご連絡下さい
https://amplify.fixstars.com/ja/contact

FIXSTARS Amplify

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！