量子コンピュータ時代のプログラミングセミナー

～ Fixstars Amplify で実装する生産計画最適化 ～

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 ではさまざまな専門性を持つエンジニアを募集しています
詳細は https://amplify.fixstars.com/ja/careers まで

FIXSTARS Amplify

量子コンピュータ時代のプログラミングセミナー

～ Fixstars Amplify で実装する生産計画最適化 ～

本日のAgenda

第一部 14:00-14:45

- はじめに
- 会社紹介
- 組合せ最適化問題・イジングマシン・事例のご紹介
- Fixstars Amplifyの紹介
- ワークショップ事前準備のご連絡

第二部 14:50-16:00

- Fixstars Amplifyを用いたワークショップ
  - 生産計画最適化

- 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（量子アニーリング・イジングマシン）があることを知り、そららの専用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

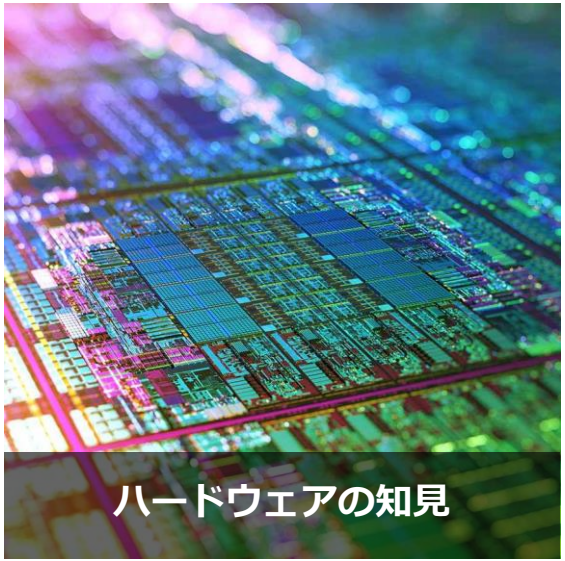
(株)Sider
完全子会社
開発支援SaaS「Sider」を運営

(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強み

コンピュータの性能を最大限に引き出す、**ソフトウェア高速化**のエキスパート集団



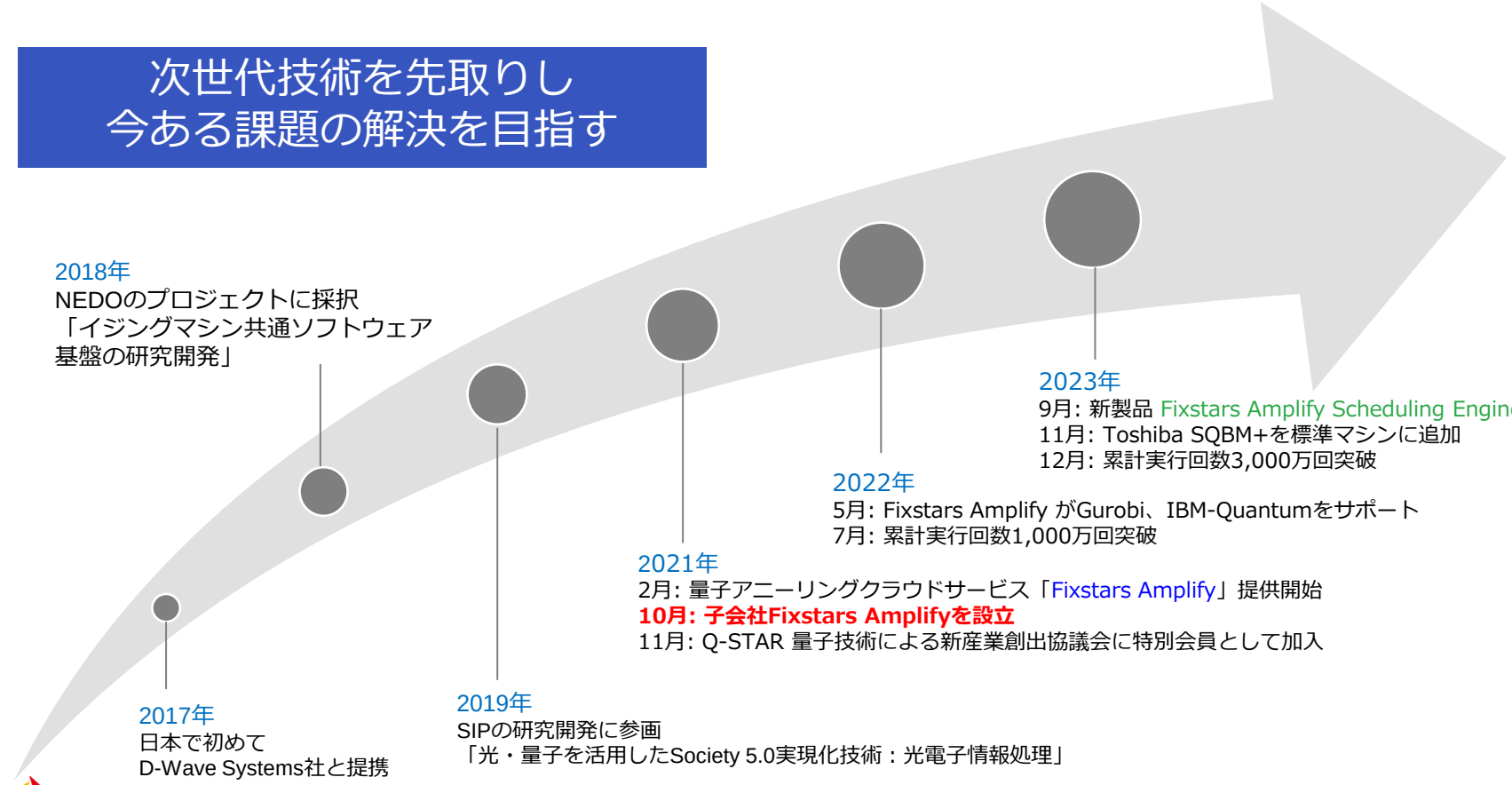
目的の製品に最適なハードウェアを見抜き、その性能をフル活用するソフトウェアを開発します。

ハードウェアの特徴と製品要求仕様に合わせて、プログラムを改良して高速化を実現します。

開発したい製品に使える技術を見抜き、実際に動作する実装までトータルにサポートします。

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

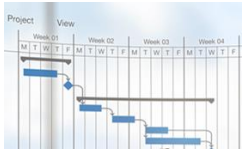
FIXSTARS Amplify

Fixstars Amplify 製品ポートフォリオ

New!

Fixstars Amplify (2021/2～)

- **<u>汎用的な問題</u>**に対応するSDKと実行環境 (Annealing Engine)
- 2次のQUBO問題が得意
- 自分で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



本日のセミナーはこちら

Fixstars Amplify Scheduling Engine (2023/9～)

- **<u>スケジューリング問題に特化</u>**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、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 (定式化不要)

製品紹介のウェブサイトを開設し、お試しのための無料トークンの配布も開始しています。無料セミナーの準備も進めています！

組合せ最適化問題・イジングマシン・事例の紹介

量子アニーリング・イジングマシンと組合せ最適化問題

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**

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」

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

膨大な解候補（組合せ）から最適解を選ぶ
・解候補一つ一つの計算は可能
・候補数が膨大ですべての解候補を計算できない

定式化

目的関数: これを最小化（最大化）する解が最適

制約条件: 解が必ず満たすべき条件

数式で表現

実装

数式をPythonのプログラムで記述

解を取得

Fixstars Amplify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

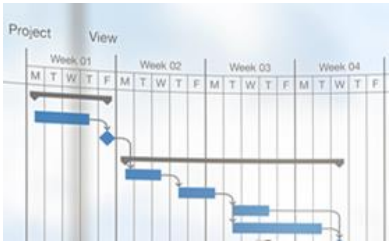
組合せ最適化の取り組み事例

シフト最適化

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

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



本日のセミナーのテーマ

経路最適化

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

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！

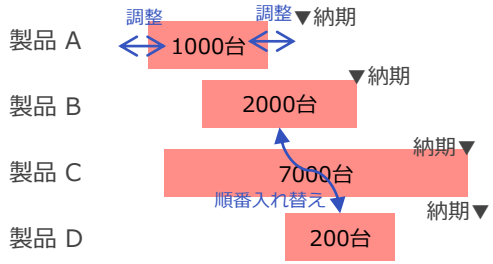
次期フェーズでは、Amplifyの活用領域の拡大を検討中！

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。

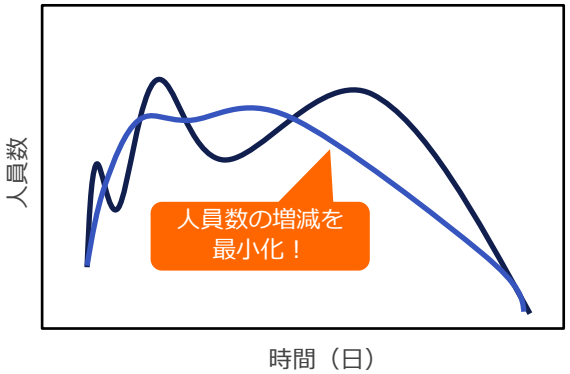


FIXSTARS Amplify

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)

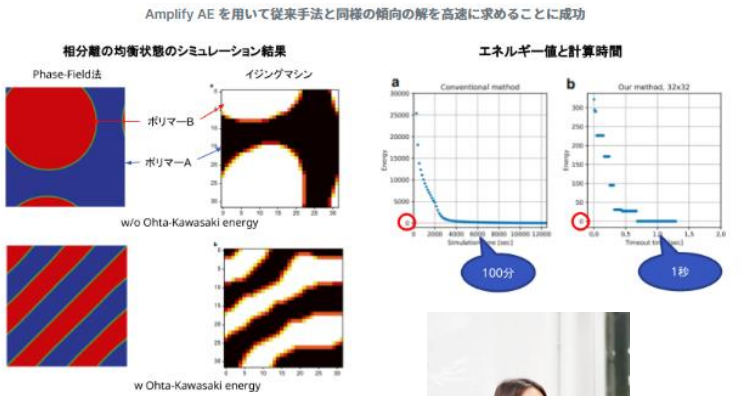
BBO

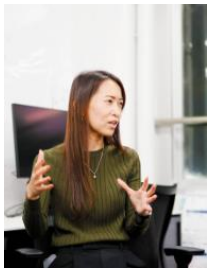
アカデミアにおける応用事例: インタビュー記事

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

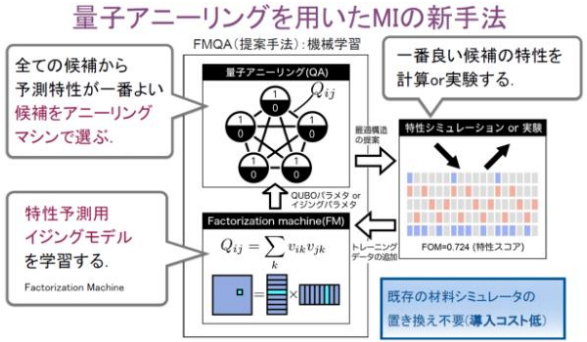
Fixstars Amplifyを使って、次世代技術を活用した高速な解析手法の開発に成功





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）

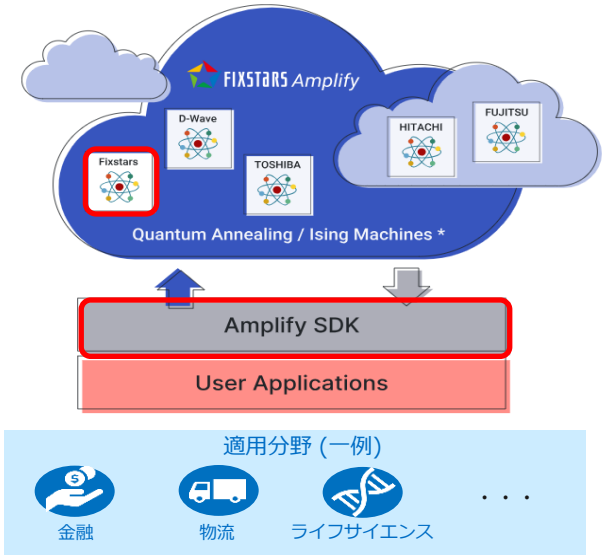
Fixstars Amplify のご紹介

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

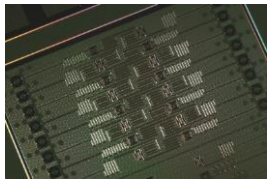
量子技術とFixstars Amplifyの対応領域

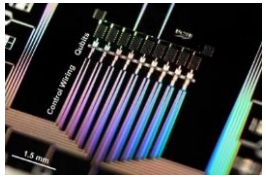
1. 量子コンピュータ

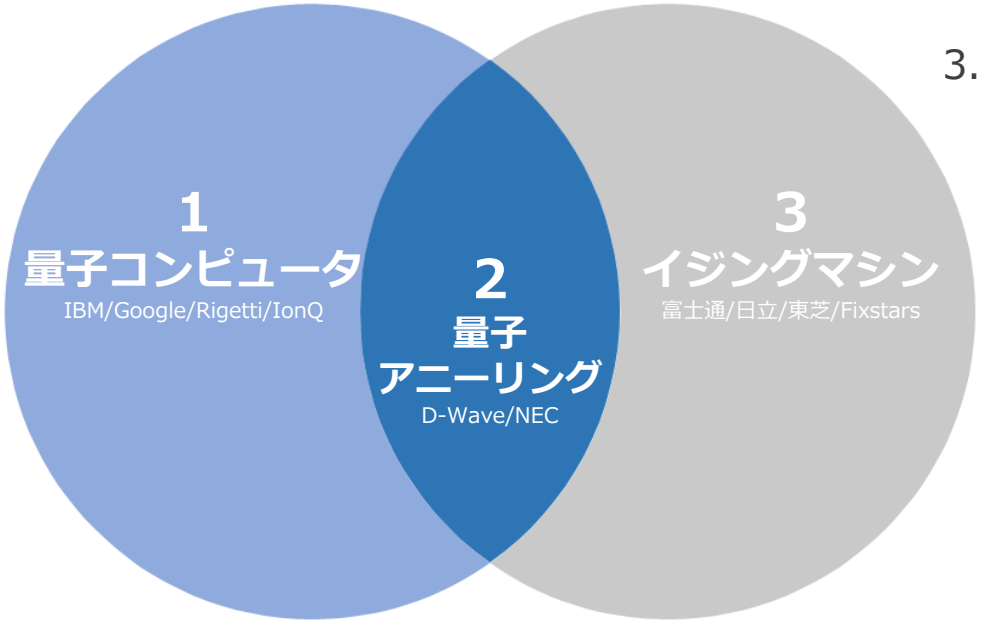
量子ゲート方式

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)

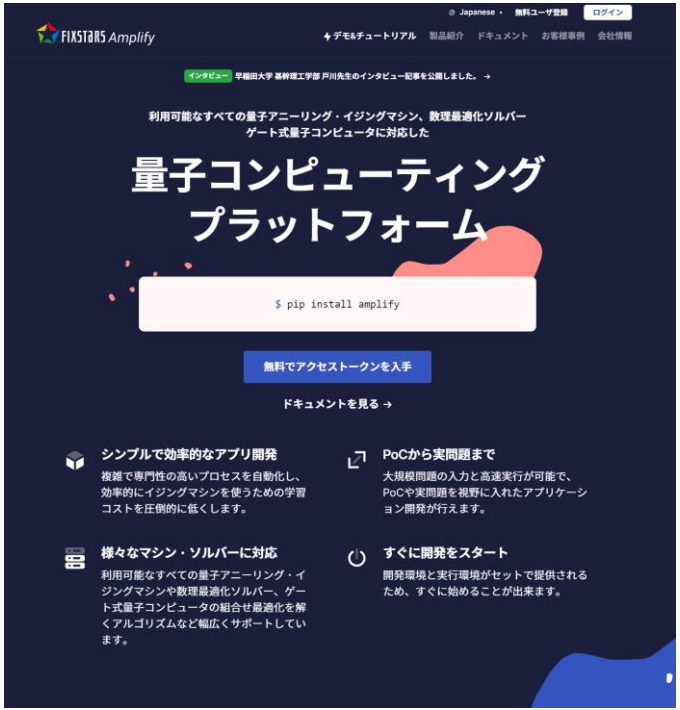
$$f(\boldsymbol{q}) = \sum_{i<j} Q_{ij} q_i q_j + \sum_i Q_{ii} q_i$$

$f(\boldsymbol{q})$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

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 の対応マシン

https://news.fixstars.com/4025/ ：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）

Amplify SDK

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。

実行方法

```python
# 入力モデルの構築
q = gen_symbols(BinaryPoly, 2)
f = 1 - q[0] * q[1]

# 実行マシンの設定
client = FixstarsClient()
client.url = "http://xxx.xxx.xxx.xxx"

# アニーリングの実行
s = Solver(client)
result = s.solve(f)
values = result.solutions[0].values

# 結果の解釈
solution = decode_solution(q, values)

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

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

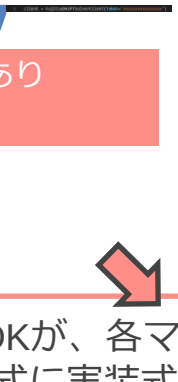
富士通・デジタルアニーラの設定用コード

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

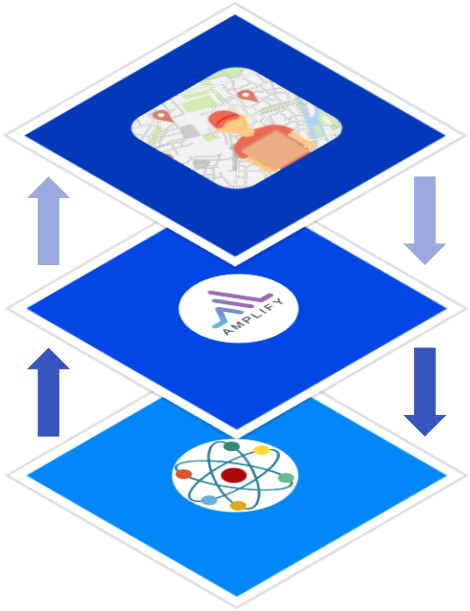
SDKが、各マシンに対して最適な形式に実装式を多段変換！

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プログラム
- 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



Fixstars Amplify Annealing Engine (Amplify AE)

NVIDIA GPU V100/A100 で動作

- 独自の並列化シミュレーテッド
  アニーリングプログラム

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

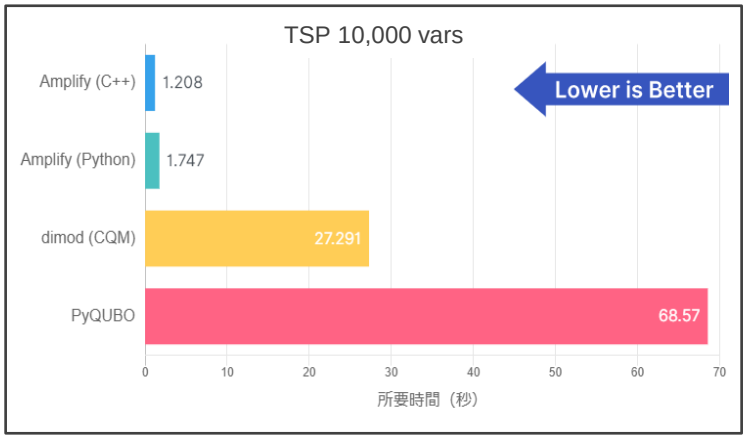
- 120,000 ビット（全結合）
- 260,000 ビット超（疎結合）

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

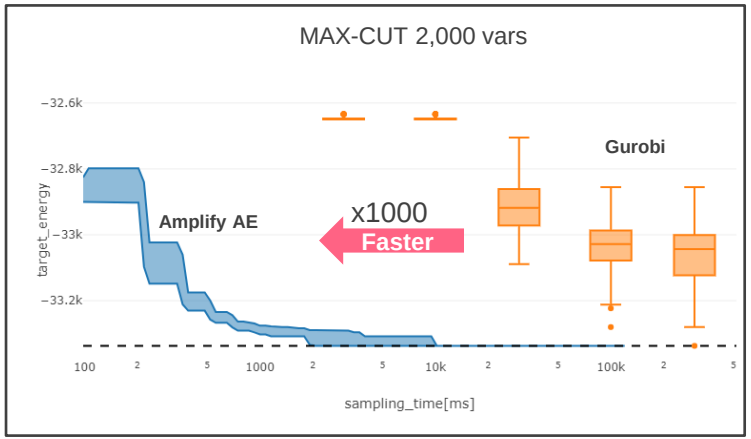
※標準マシンは評価・検証用途では無償提供

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

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

チュートリアル基礎編

はじめてのAmplify

プログラミング難易度 ★

Amplify SDK による基本的なプログラミングフローを説明したチュートリアルです。

チュートリアル

チュートリアル基礎編

組合せ最適化問題とは

プログラミング難易度 ★★

イジングマシンを用いて数の分割問題を解くためのプログラムを作成し、組合せ最適化問題について理解を深めます。

チュートリアル

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

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

FIXSTARS Amplify

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: 3,000万回超

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

	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	
SQBM+オプション	無料	月額30万円（税込33万円）	月額30万円（税込33万円）	月額90万円（税込99万円）
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）

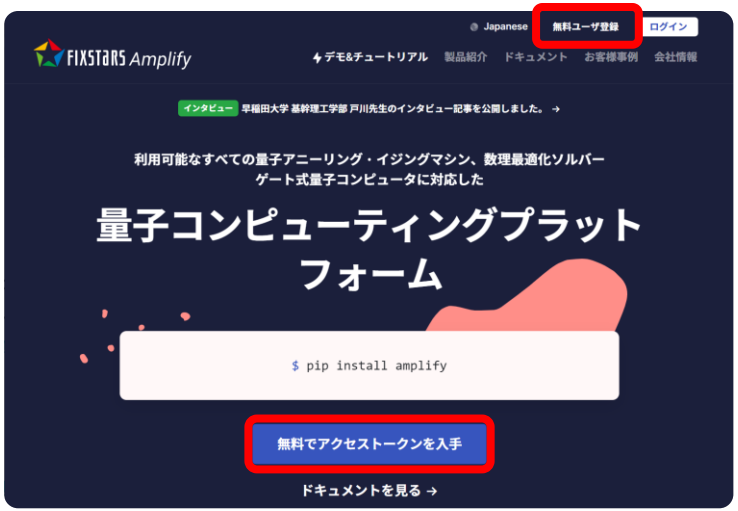
Google Colab 検索

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)

- 入手いただいたAmplifyのトークンを用いて、以下のURLにあるサンプルコードが動くか確認をお願いします。サンプルコードは閲覧のみ可能ですので「ドライブにコピー」の上、ご自身のトークンを入力し、Shift+Enterで実行をしてください
  (警告が出る場合がありますが、「このまま実行」を選択下さい)
  - https://colab.research.google.com/drive/1evYBKqKfVrEzrQOa-SWwciROfvqjL8qm?usp=sharing

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

この部分に、ご自身のトークン番号(32桁)を入力の上、Shift+Enterで実行下さい。
ご自身のトークン番号は、Amplify HPよりご確認いただけます

- ご自身のトークンを入力の上、2つのセルをShift+Enterで実行し、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

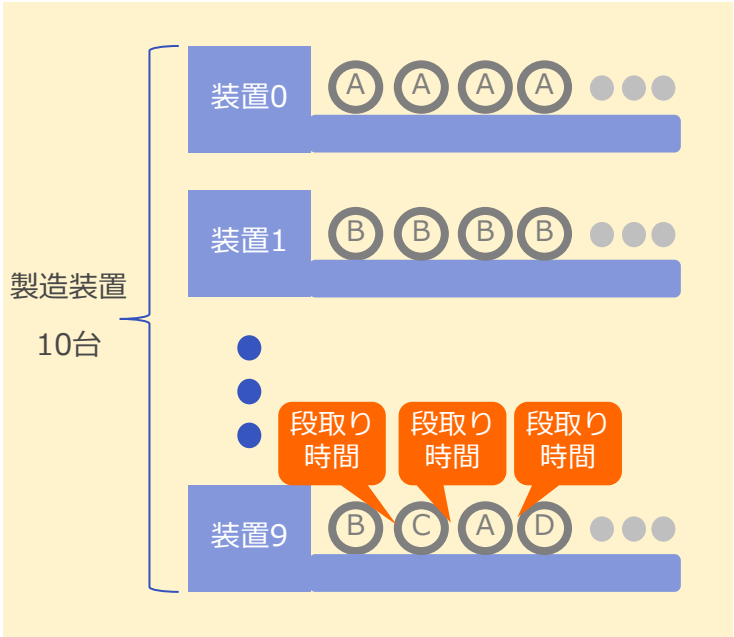
Step1	https://colab.research.google.com/drive/1oAaEPnj5F71jnQ3_ELOI4NCe-6im6luw#scrollTo=ubFqJ2XEL76u
Step2	https://colab.research.google.com/drive/171migOXlXlhXWhZ4uAhCW2WjJ5Vmaqn9#scrollTo=seeNN5uEK52D
Step3	https://colab.research.google.com/drive/1fqZn0Jz8JdzgCmpml2DWguUIuZOwNxEm#scrollTo=dNtZPM-jJetC

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
～生産計画最適化～

生産計画最適化

【問題】10台の製造装置を使い4種の製品を合計100個作りたい。製造する品種を変更するには、部品の交換するための段取り時間が必要。全数の生産が完了するまでの時間を最短にするには、どの製品を、どの製造装置で、どういう順番で製造するのが最適か



4品種、合計100個

品種	製品数	処理時間(時間/ロット)
A	30	2時間
B	10	3時間
C	40	1時間
D	20	1時間

段取り時間	
A→A	0時間
B→A	2時間
C→A	1時間
D→A	2時間

段取り時間がかかるから、同じ品種の製品をできるだけまとめて生産したら良さそうだけど、全部はできないし、どうするのが一番いいんだろう・・・

生産計画最適化

問題設定
↓
定式化
↓
実装
↓
解を取得

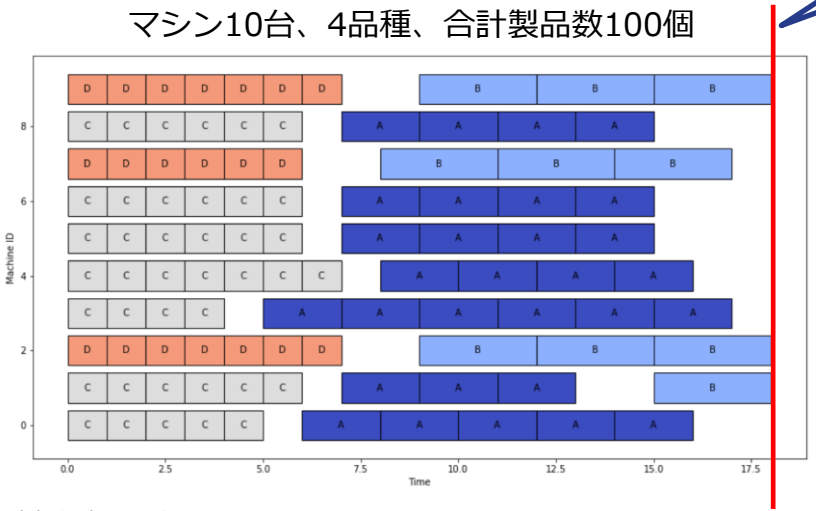
段取り時間を含めた総生産時間を最小化する

目的関数: 生産時間と段取りの総生産時間の最小化

制約条件: ① 各装置が同時に作れるのは1品種のみ
② 各品種の合計生産数が予定数通り

Amplifyが
10秒で計算

マシン10台、4品種、合計製品数100個

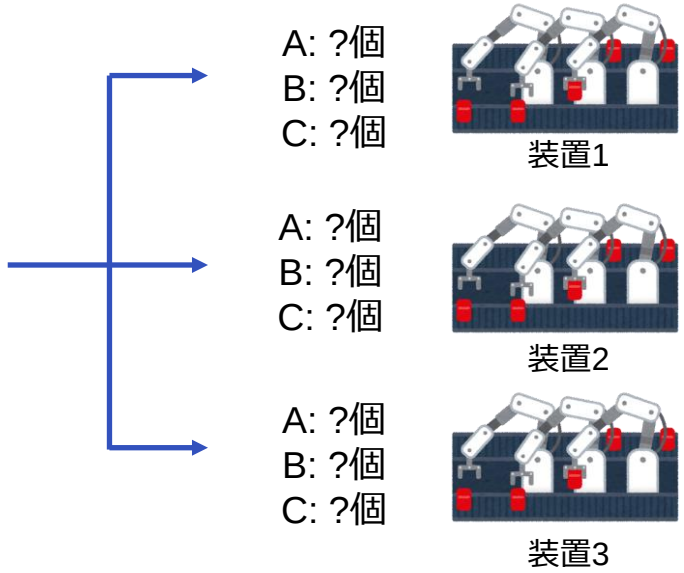


ワークショップ: 問題設定

【問題】 3台の製造装置で3品種、合計45個の生産計画を作成します。各品種の生産台数、生産時間、及び、生産する品種を交換する際に生じる段取り時間は以下の通りです。全ての製品の生産ができるだけ早く完了するよう、生産にかかる総所要時間 (各製品の生産完了までにかかる所要時間の合計) が少なく、総段取り時間も少ない、バランスの取れた生産計画の作成を目指します。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

品種	生産時間	生産台数
A	1	10
B	2	15
C	3	20

段取り時間			
切替元品種	切替先品種		
	A	B	C
A	0	2	4
B	1	0	3
C	2	4	0



ワークショップ: 3 Step

【問題】 3台の製造装置で3品種、合計45個の生産計画を作成します。各品種の生産台数、生産時間、及び、生産する品種を交換する際に生じる段取り時間は以下の通りです。全ての製品の生産ができるだけ早く完了するよう、生産にかかる総所要時間 (各製品の生産完了までにかかる所要時間の合計) が少なく、総段取り時間も少ない、バランスの取れた生産計画の作成を目指します。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

まず、2つの制約を守るだけのプログラムを作ります
制約①: 各装置が同時に作れるのは1品種のみ
制約②: 各品種の合計生産数が予定数通り

解の候補多数あり

Step2

Step1に「総所要時間の最小化」という目的①を追加し、複数の解の候補から目的①を実現する解を求めるプログラムを作ります

Step3

Step2に「総段取り時間の最小化」という目的②を追加して、目的①と目的②を同時に実現するプログラムを作ります

Step1

まず、2つの制約を守るだけのプログラムを作ります
制約①: 各装置が同時に作れるのは1品種のみ
制約②: 各品種の合計生産数が予定数通り

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つの制約を守るだけのプログラムを作ります

制約①: 各装置が同時に作れるのは1品種のみ
制約②: 各品種の合計生産数が予定数通り

決定変数の準備

1つの装置で全製品を生産する可能性があるので最大45個のスロットを用意

Machine (m)	Type (t)	Slot (i) 0	1	・・・	44
0	A	q_0 (0 or 1)	q_4 (0 or 1)	・・・	q_176 (0 or 1)
	B	q_1 (0 or 1)	q_5 (0 or 1)	・・・	q_177 (0 or 1)
	C	q_2 (0 or 1)	q_6 (0 or 1)	・・・	q_178 (0 or 1)
	X(ダミー)	q_3 (0 or 1)	q_7 (0 or 1)	・・・	q_179 (0 or 1)
1	A	q_180 (0 or 1)	q_184 (0 or 1)	・・・	q_356 (0 or 1)
	B	q_181 (0 or 1)	q_185 (0 or 1)	・・・	q_357 (0 or 1)
	C	q_182 (0 or 1)	q_186 (0 or 1)	・・・	q_358 (0 or 1)
	X(ダミー)	q_183 (0 or 1)	q_187 (0 or 1)	・・・	q_359 (0 or 1)
2	A	q_360 (0 or 1)	q_364 (0 or 1)	・・・	q_336 (0 or 1)
	B	q_361 (0 or 1)	q_365 (0 or 1)	・・・	q_537 (0 or 1)
	C	q_362 (0 or 1)	q_366 (0 or 1)	・・・	q_538 (0 or 1)
	X(ダミー)	q_363 (0 or 1)	q_367 (0 or 1)	・・・	q_539 (0 or 1)

装置ごとに4品種
(A~C+ダミー)

BinaryPoly型 (3×45×4) = 540 [qbit]
1は割当、0は非割当を表す

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

Machine (m)	Type (t)	Slot (i) 0	1	2	3	・・・	44
0	A	0	0	0	0	・・・	0
	B	1	1	1	1	・・・	0
	C	0	0	0	0	・・・	0
	X(ダミー)	0	0	0	0	・・・	1
1	A	1	1	1	1	・・・	0
	B	0	0	0	0	・・・	0
	C	0	0	0	0	・・・	0
	X(ダミー)	0	0	0	0	・・・	1
2	A	0	0	1	0	・・・	0
	B	0	0	0	1	・・・	0
	C	1	1	0	0	・・・	0
	X(ダミー)	0	0	0	0	・・・	1

装置#2では最初に製品Cを作る

※ 説明の都合上、ダミー製品が後ろに並ぶような図となっていますが、実際は前に並ぶよう定式化しています。

実装

決定変数

```
# 装置IDごとに(処理順序,品種)のarrayを作成
from amplify import BinarySymbolGenerator

gen = BinarySymbolGenerator()
allocation_variables = gen.array(num_mac, num_slots, num_types)
print(allocation_variables)
```

Step1

まず、2つの制約を守るだけのプログラムを作ります
制約①: 各装置が同時に作れるのは1品種のみ
制約②: 各品種の合計生産数が予定数通り

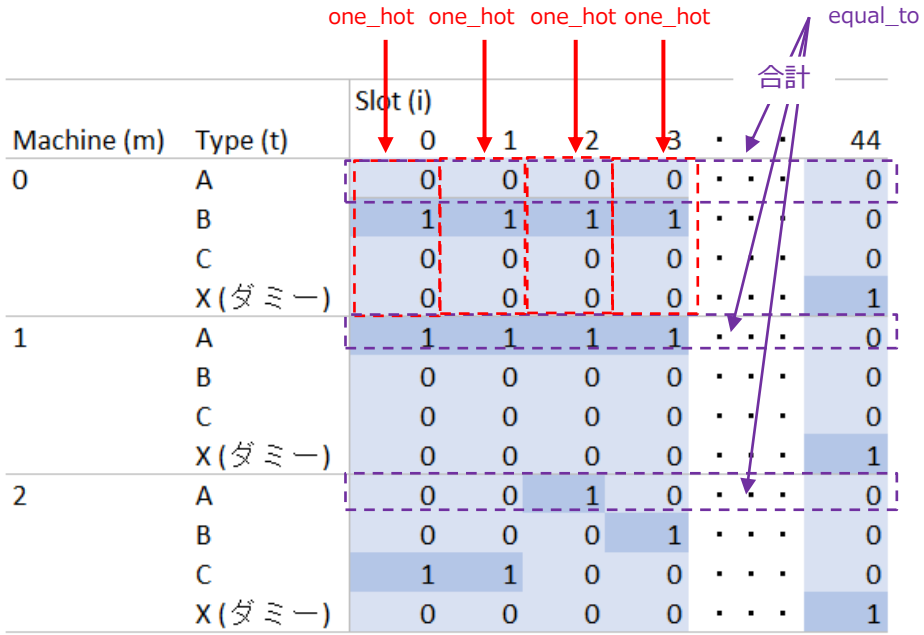
定式化

制約①: 各装置が同時に作れるのは1品種のみ
→ one_hot制約

$$\sum_{t=1}^{4} q_{m,i,t} = 1$$

制約②: 各品種の合計生産数が予定数通り
→ equal_to制約 (等式制約)

$$prod\_num_t = \sum_{m=1}^{3}\sum_{i=1}^{45} q_{m,i,t}$$

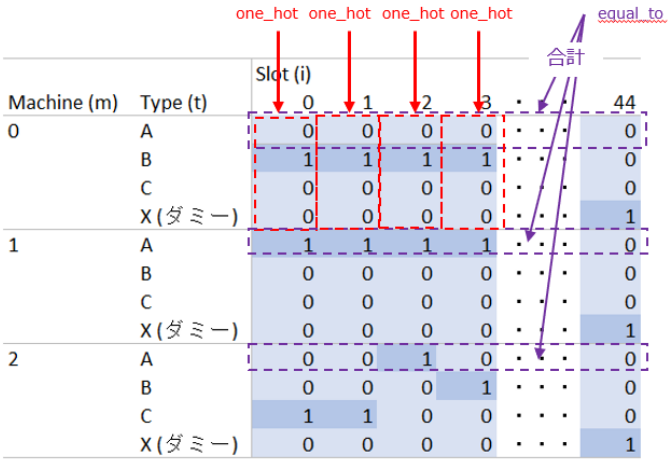


Machine (m)	Type (t)	Slot (i) 0	1	2	3	・・・	44
0	A	0	0	0	0	・・・	0
	B	1	1	1	1	・・・	0
	C	0	0	0	0	・・・	0
	X (ダミー)	0	0	0	0	・・・	1
1	A	1	1	1	1	・・・	0
	B	0	0	0	0	・・・	0
	C	0	0	0	0	・・・	0
	X (ダミー)	0	0	0	0	・・・	1
2	A	0	0	1	0	・・・	0
	B	0	0	0	1	・・・	0
	C	1	1	0	0	・・・	0
	X (ダミー)	0	0	0	0	・・・	1

Step1

まず、2つの制約を守るだけのプログラムを作ります

制約①: 各装置が同時に作れるのは1品種のみ
制約②: 各品種の合計生産数が予定数通り



【補足】
制約条件の取り扱いに関する詳細は、こちらにあるチュートリアルも合わせてご参照下さい

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

実装

```python
##################
# 制約条件
##################

from amplify.constraint import one_hot, equal_to
from amplify import sum_poly

# 制約① 各装置が同時に作れるのは1品種のみ（one-hot制約）
loc_constraints = sum(
    [
        one_hot(allocation_variables[m][i])
        for m in range(num_mac)
        for i in range(num_slots)
    ]
)


# 制約② 各品種の合計生産台数が予定数通り（equal_to制約）
unit_constraints = sum(
    [
        equal_to(allocation_variables[:, :, t].sum(), units[t])
        for t in range(num_types - 1)  # dummy品種は除く
    ]
)


# 制約
constraints = loc_constraints + unit_constraints
```

Step1

まず、2つの制約を守るだけのプログラムを作ります
制約①: 各装置が同時に作れるのは1品種のみ
制約②: 各品種の合計生産数が予定数通り

求解

```python
##########
# 求解
##########

from amplify.client import FixstarsClient
from amplify import Solver

# 実行マシンクライアントの設定
client = FixstarsClient()
client.token = token
client.parameters.timeout = 10 * 1000  # タイムアウト10秒

# モデル化
model = constraints

# アニーリングマシンの実行
solver = Solver(client)  # ソルバーに使用するクライアントを設定
result = solver.solve(model)  # 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

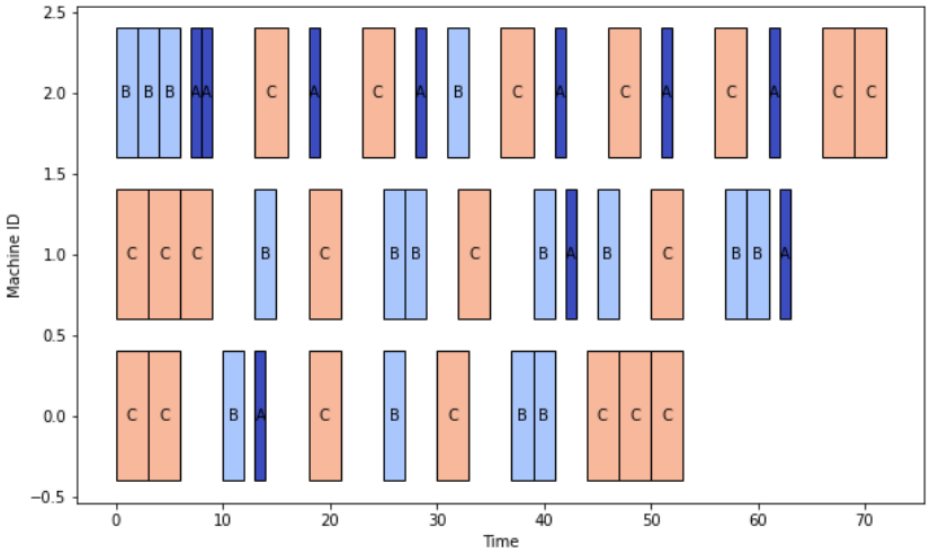
まず、2つの制約を守るだけのプログラムを作ります
制約①: 各装置が同時に作れるのは1品種のみ
制約②: 各品種の合計生産数が予定数通り

結果の取得

```
values = result[0].values  # 解を格納
allocation_solutions = allocation_variables.decode(values)
print(allocation_solutions)
```

可視化

最終生産完了時間: 72



一つ目の制約 (各装置が同時に作れるのは1品種のみ)と二つ目の制約 (各品種の合計生産数が予定数通り) を満たす計画を作ることができました。但し、解の候補はたくさんある状況です。

Step2

Step1に「総所要時間の最小化」という目的①を追加し、複数の解の候補から目的①を実現する解を求めるプログラムを作ります

Step2のサンプルコードのレビュー

Step2

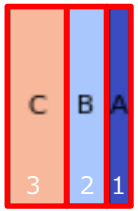
Step1に「総所要時間の最小化」という目的①を追加し、複数の解の候補から目的①を実現する解を求めるプログラムを作ります

このステップでは、各製品の生産時間のみに着目し、総所要時間の最小化を目指します (目的①)。総所要時間は、各製品の生産完了までにかかる所要時間の合計としていますので、同じ製品を作る場合でも、生産する順番によって総所要時間の合計は変わります (完成品をできるだけ早く製造する計画を作る前提です) 。

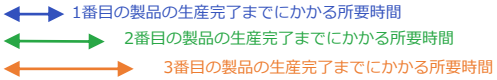
各製品の生産時間

品種	生産時間
A	1
B	2
C	3

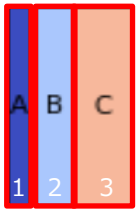
例1



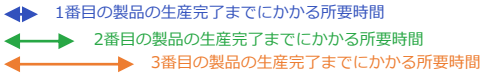
総所要時間（各製品の生産完了までにかかる所要時間の合計)
= 1番目の所要時間 + 2番目の所要時間 + 3番目の所要時間
= 3 + (3+2) + (3+2+1)
= 3*3 + 2*2 + 1*1
= **14**



例2



総所要時間
= 1 + (1+2) + (1+2+3)
= 1*3 + 2*2 + 3*1
= **10**



今回のワークショップではこちらを目指します

定式化

総所要時間の最小化

$$process = \sum_{m=1}^{3}\sum_{i=1}^{45}\sum_{t=1}^{4} a_i \cdot proc\_time_t \cdot q_{m,i,t}$$

（ a_i は順序係数を表す（N+1-i, N=45））

決定変数

赤字部分を順序係数と呼んでいます

Step2

Step1に「総所要時間の最小化」という目的①を追加し、複数の解の候補から目的①を実現する解を求めるプログラムを作ります

実装

：追加コード

```python
##################
# 目的関数
##################

# 目的① 総所要時間の最小化
process = 0
for i in range(num_slots):
    process += (num_slots - i) * (proc_time * allocation_variables[:, i]).sum()

# 目的関数
objective = process
```

```python
##################
# 制約条件
##################
        .
        .
        .
# 制約の重み
constraint_weight = 35

# 制約
constraints = constraint_weight * (loc_constraints + unit_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

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

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。

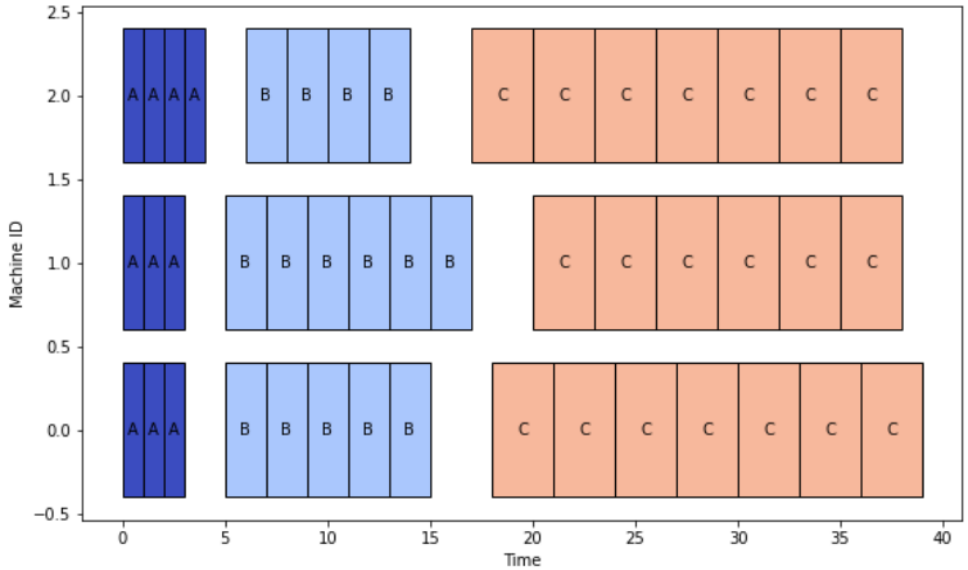
【補足】
制約条件の取り扱いに関する詳細は、こちらにあるチュートリアルも合わせてご参照下さい

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

Step2

Step1に「総所要時間の最小化」という目的①を追加し、複数の解の候補から目的①を実現する解を求めるプログラムを作ります

最終生産完了時間: 39



2つの制約を満たした上で、総所要時間の最小化する計画を作ることができました。次のステップでは、段取り時間に着目し、段取り時間の最小化のコードを追加します

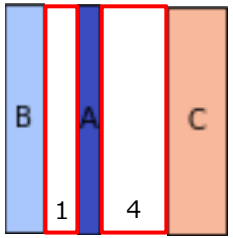
Step3

Step2に「総段取り時間の最小化」という目的②を追加して、目的①と目的②を同時に実現するプログラムを作ります

段取り時間

段取り時間			
切替元品種	切替先品種		
	A	B	C
A	0	2	4
B	1	0	3
C	2	4	0

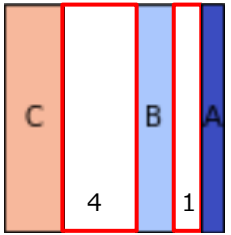
例1



総段取り時間
= 1 + (1 + 4)
= 1*2+4*1
= **6**

今回のワークショップではこちらを目指します

例2



総段取り時間
= 4 + (4 + 1)
= 4*2+1*1
= **9**

赤字部分を順序係数と呼びます

定式化

$$switch = \sum_{m=1}^{3} \sum_{i=2}^{45} \sum_{fr=1}^{4} \sum_{to=1}^{4} a_i \cdot sw\_time_{fr,to} \; q_{m,i-1,fr} \cdot q_{m,i,to}$$

（ a_i は順序係数を表す（N-i, N=45））

決定変数同士の積
(二次の項)

Step3

Step2に「総段取り時間の最小化」という目的②を追加して、目的①と目的②を同時に実現するプログラムを作ります

実装

定式化

$$switch = \sum_{m=1}^{3}\sum_{i=2}^{45}\sum_{fr=1}^{4}\sum_{to=1}^{4} a_i \cdot sw\_time_{fr,to} \cdot q_{m,i-1,fr} \cdot q_{m,i,to}$$

（ a_i は順序係数を表す (N-i, N=45)）

```
####################
# 目的関数
####################

# 目的1 総所要時間の最小化
process = 0
for i in range(num_slots):
    process += (num_slots - i) * (proc_time * allocation_variables[:, i]).sum()

# 目的2 総段取り時間の最小化
switch = 0
for i in range(1, num_slots):
    for fr in range(num_types):
        for to in range(num_types):
            switch += (num_slots - 1 - i) * (
                sw_time[fr, to]
                * allocation_variables[:, i - 1, fr]
                * allocation_variables[:, i, to]
            ).sum()

# 目的関数
objective = process + switch
```

：追加コード

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

Step3

Step2に「総段取り時間の最小化」という目的②を追加して、目的①と目的②を同時に実現するプログラムを作ります

2つの制約を満たした上で、総所要時間を最小化しながら、総段取り時間を最小化する計画を作ることができました！

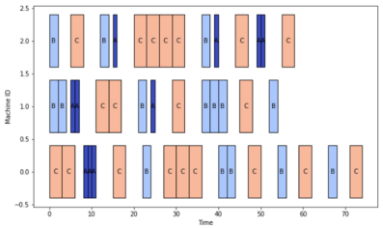
ワークショップ: おさらい

制約のみからはじめ、複数の目的を加え、最適なバランスの計画が作れるようになりました

Step1

処理完了時間: **74**

制約



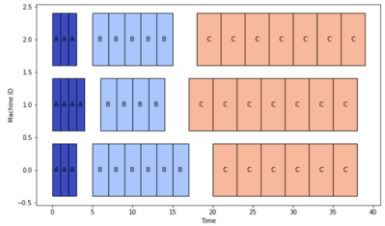
Step2

処理完了時間: **39**

制約

＋

総所要時間の最小化



Step3

処理完了時間: **37**

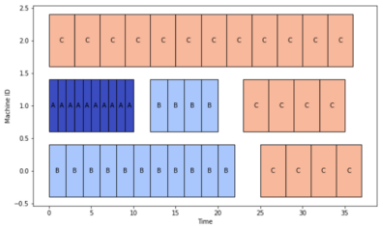
制約

＋

総所要時間の最小化

＋

総段取り時間の最小化



Wrap Up

発展的課題

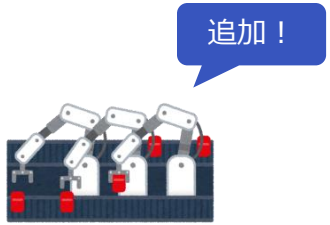
課題1

各品種の生産時間と段取り時間をそれぞれ10倍の値とした場合に、ワークショップと同じ目的を実現する最適な生産計画を作成して下さい (生産時間と段取り時間の値を変えるだけだと解は求まりません)

課題2

工場の生産能力向上のため、新しい製造装置を1台導入することになりましたが、その製造装置は、特殊な加工器具を要する品種Bを製造することはできません。この状況を考慮し、ワークショップと同じ目的を実現する最適な生産計画を作成して下さい

A: ?個
B: 0個
C: ?個



製造装置4

ご質問・ご不明点は問合せフォームでご連絡下さい
https://amplify.fixstars.com/ja/contact

今後について

- 本日の「発展的課題」にチャレンジ！
- デモ・チュートリアルにチャレンジ！

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

制約のみで定式化

チュートリアル応用編

会議室割当問題

デモアプリケーション

グラフ彩色問題

目的関数＋制約で定式化

チュートリアル応用編

タクシーマッチング問題

デモアプリケーション

巡回セールスマン問題

困った時は
ドキュメンテーションを！

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

今後のセミナーのご案内

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

2023/2/7 (水)
「ブラックボックス最適化」

第一部 14:00 - 14:35
- 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 14:40 - 16:00
- Fixstars Amplifyを用いた「ブラックボックス最適化」のワークショップ
- Q&A

2023/3 (TBD)
「シフト最適化」(TBD)

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

2023/4 (TBD)
「生産計画最適化」(TBD)

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 SE を用いた「生産計画最適化」のワークショップ
- Q&A

ご質問・ご不明点は問合せフォームでご連絡下さい
https://amplify.fixstars.com/ja/contact

FIXSTARS Amplify

本セミナーのゴール

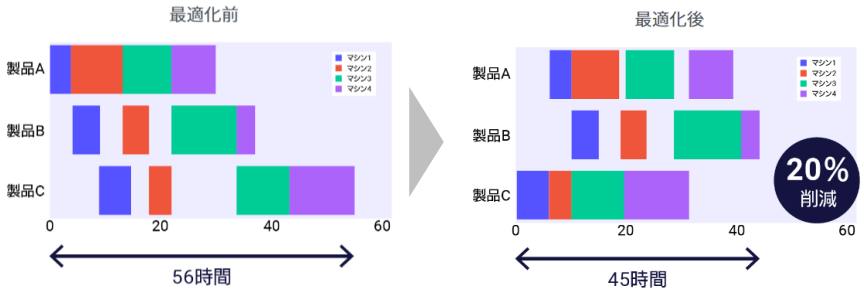
- 身の回りには組合せ最適化問題がたくさんあることを知る
- 組合せ最適化問題を解くための専用マシン（イジングマシン）があることを知り、イジングマシンを使って問題を解くための統一的なフレームワークや、問題設定の考え方、目的関数や制約条件の定式化のポイントを理解する
- ワークショップを通して、実際に量子アニーリング・イジングマシンを動かしてみることで、実問題への適用の足掛かりを得る

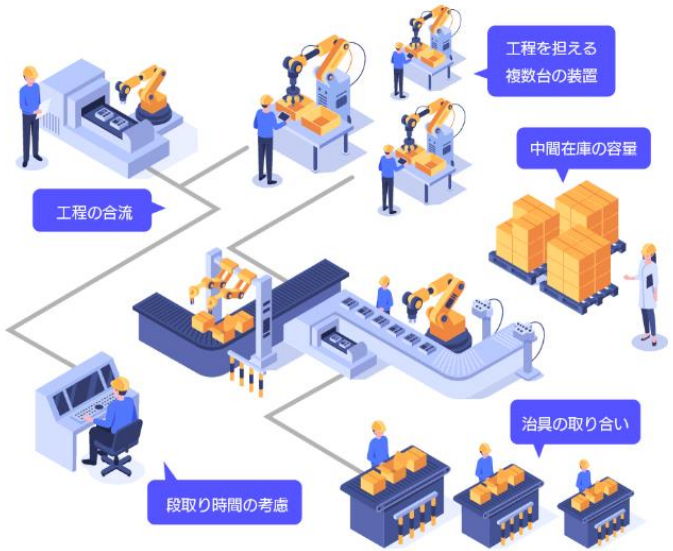
Fixstars Amplify Scheduling Engine のご紹介

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

無料トークンの配布も
開始しています！

- **スケジューリング問題に特化**したSDKと実行環境
- 最適化の目的をMakespanの最小化に絞り、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 (定式化不要)





FIXSTARS Amplify

ご清聴ありがとうございました

アンケートへのご協力お願いします！