2020年度未踏ターゲット事業　成果報告会

アプリケーション特化型イジングマシンの設計とFPGAへの実装

2021/2/11 (木)

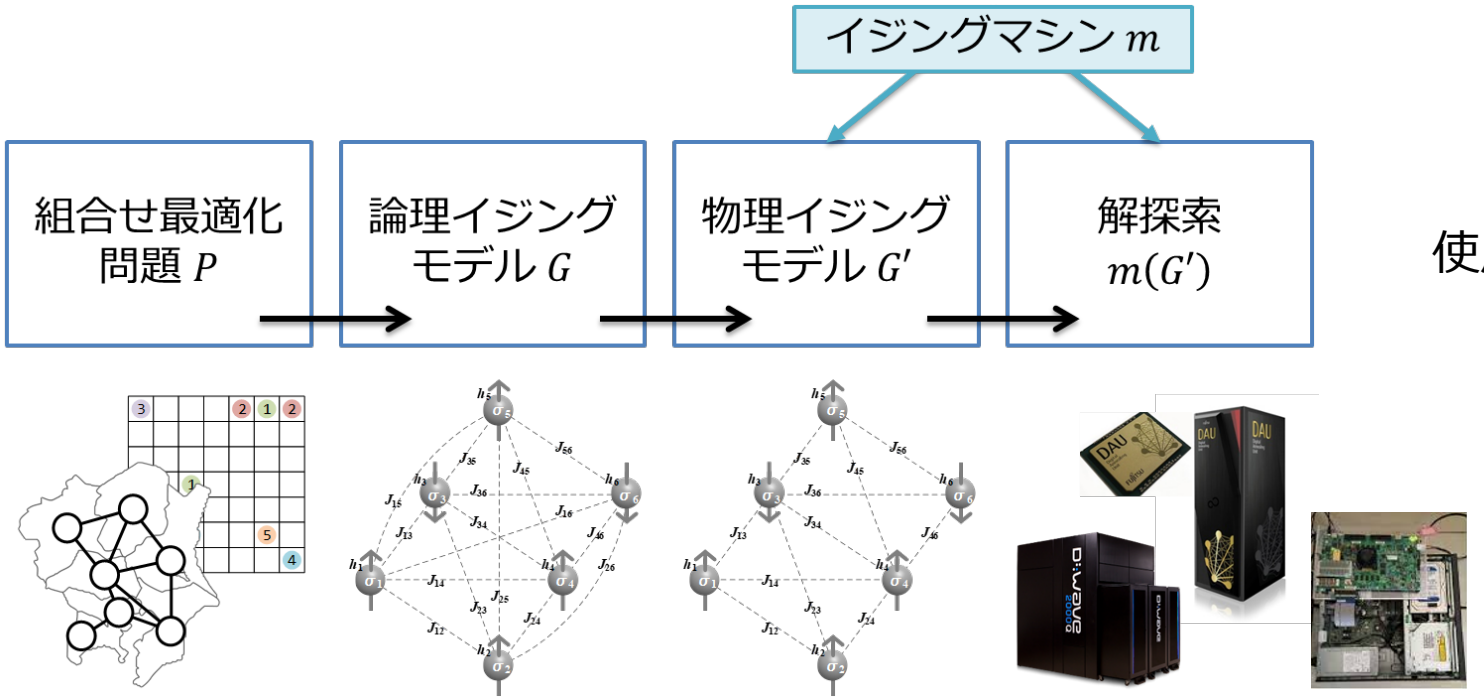
川村一志 †

（担当PM：棚橋耕太郎）

†東京工業大学 科学技術創成研究院 AIコンピューティング研究ユニット 特任助教

背景：イジングマシンを用いた組合せ最適化問題の解法

問題解法フロー



モチベーション

マシンサイド

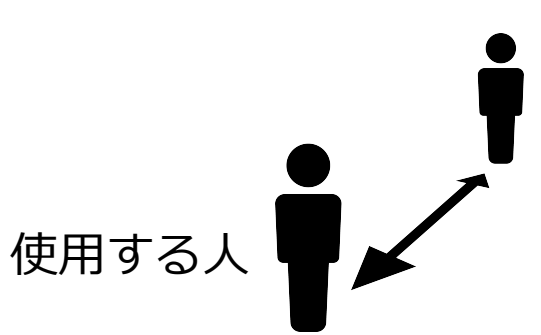
高性能なマシンを開発したい！

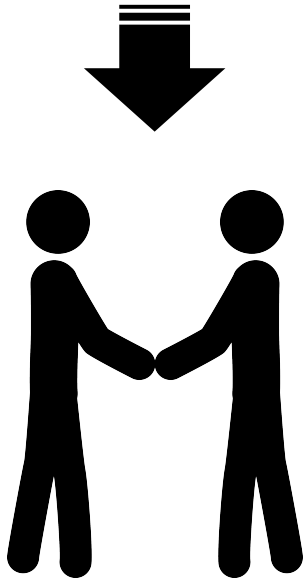
色々な場面で使われて欲しい！

こんな問題が解きたい！

このマシンはどうやったらうまく使える？

アプリサイド



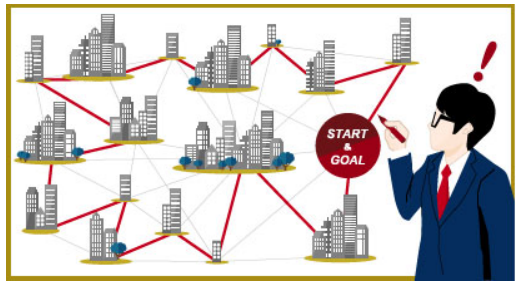


アプリ目線を取り入れた
マシンの開発

背景：イジングコンピューティングのボトルネック

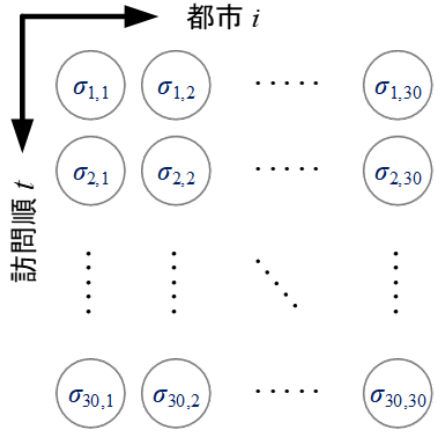
例）巡回セールスマン問題

N都市



$N!$通り

二値表現に変換



$2^{N\times N}$通り

$$H = \sum_t \sum_{i_1} \sum_{i_2} d_{i_1 i_2} x_{t,i_1} x_{t+1,i_2} + \alpha \sum_t \left(1 - \sum_i x_{t,i}\right)^2 + \alpha \sum_i \left(1 - \sum_t x_{t,i}\right)^2$$

目的関数：移動距離の最小化　　制約：1都市ずつ訪問　　制約：各都市は1回のみ訪問

イジングモデルが表現する解の内訳

	$N=5$	$N=10$	$N=15$	$N=20$	$N=25$
解の総数($=2^{N\times N}$)	3.4×10^{7}	1.3×10^{30}	5.4×10^{67}	2.6×10^{120}	1.4×10^{188}
制約充足解の総数($=N!$)	1.2×10^{2}	3.6×10^{6}	1.3×10^{12}	2.4×10^{18}	1.6×10^{25}
制約充足解の割合	$\frac{1}{2.8\times10^{5}}$	$\frac{1}{3.6\times10^{23}}$	$\frac{1}{4.2\times10^{55}}$	$\frac{1}{1.1\times10^{102}}$	$\frac{1}{8.8\times10^{162}}$

ほとんどは
制約を**満たさない**解

解探索冗長性

川村PJ：**アプリケーション特化型**イジングマシンの開発

目標

QUBOソルバ

（短~中期）特定の性質を持つ問題(群)を効率的に解くマシンを提供

↳ 本プロジェクトでは**「One-hot制約」**に注目

（中~長期）カスタムイジングコンピューティングの概念を広める

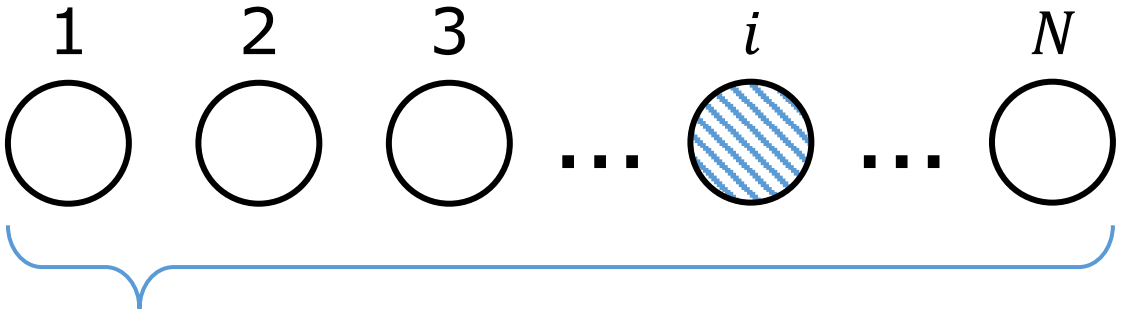
具体的な取り組み

1. One-hot制約に対応した基底状態探索アルゴリズムの構築
2. 提案アルゴリズムを実現するディジタル回路アーキテクチャの設計
3. FPGAへの実装、評価
4. イジングマシン利用環境の提供

１．One-hot制約に対応した基底状態探索アルゴリズムの構築

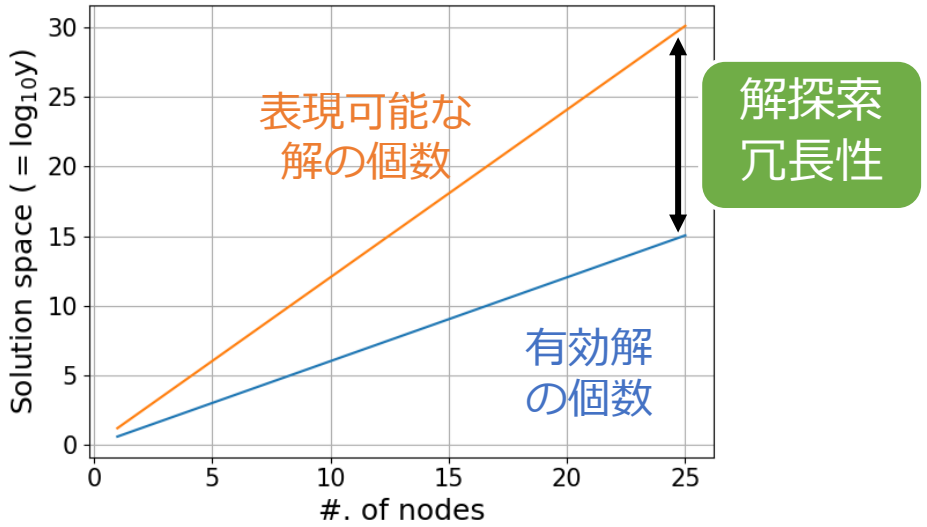
One-hot制約は「頻出」かつ「厄介」

One-hot encoding ＝ N種の情報をN個の二値変数を用いて表現



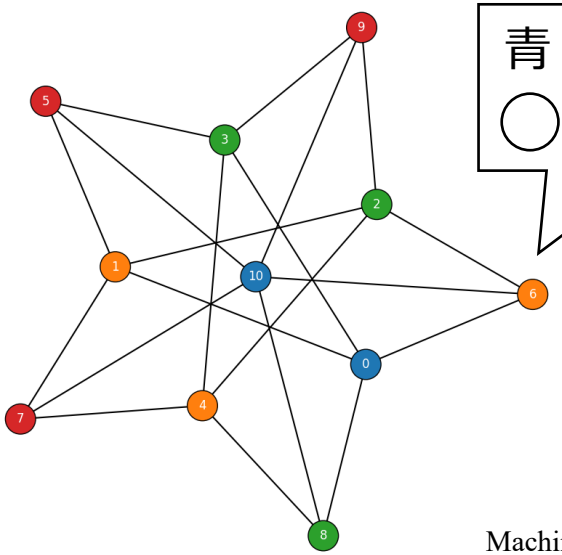
One-hot制約 ＝ N変数のうち1個のみが1（残りは0）

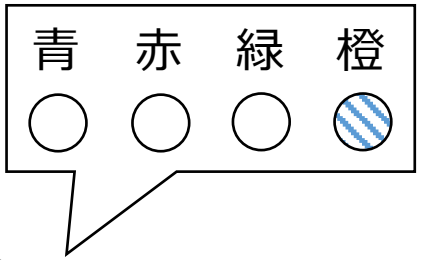
解空間の広がり方：グラフ頂点彩色問題(4色)



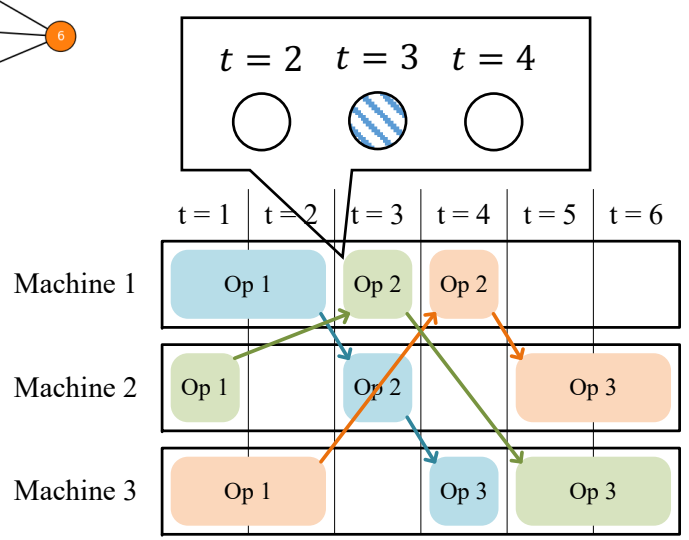
One-hot制約を含む問題

グラフ頂点彩色問題



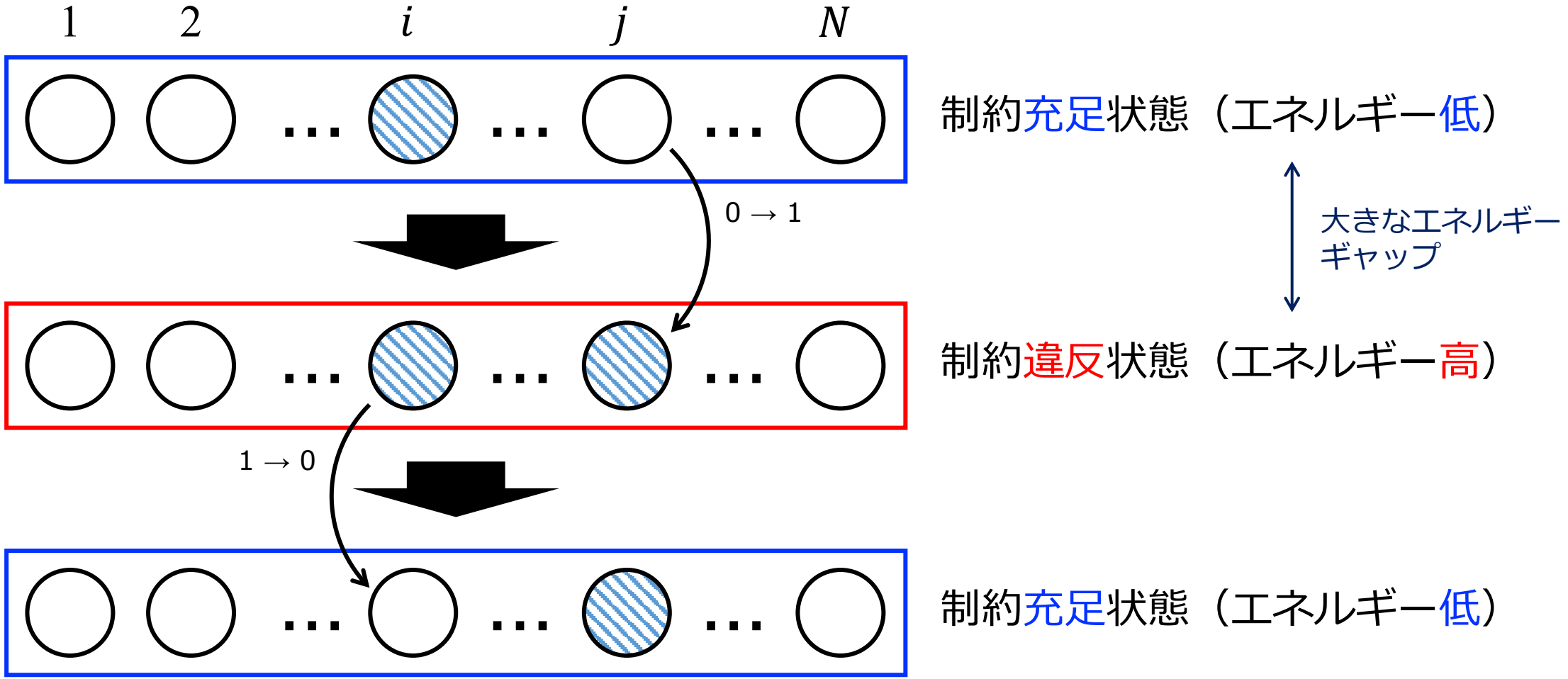


ジョブショップ
スケジューリング問題
(JSP)



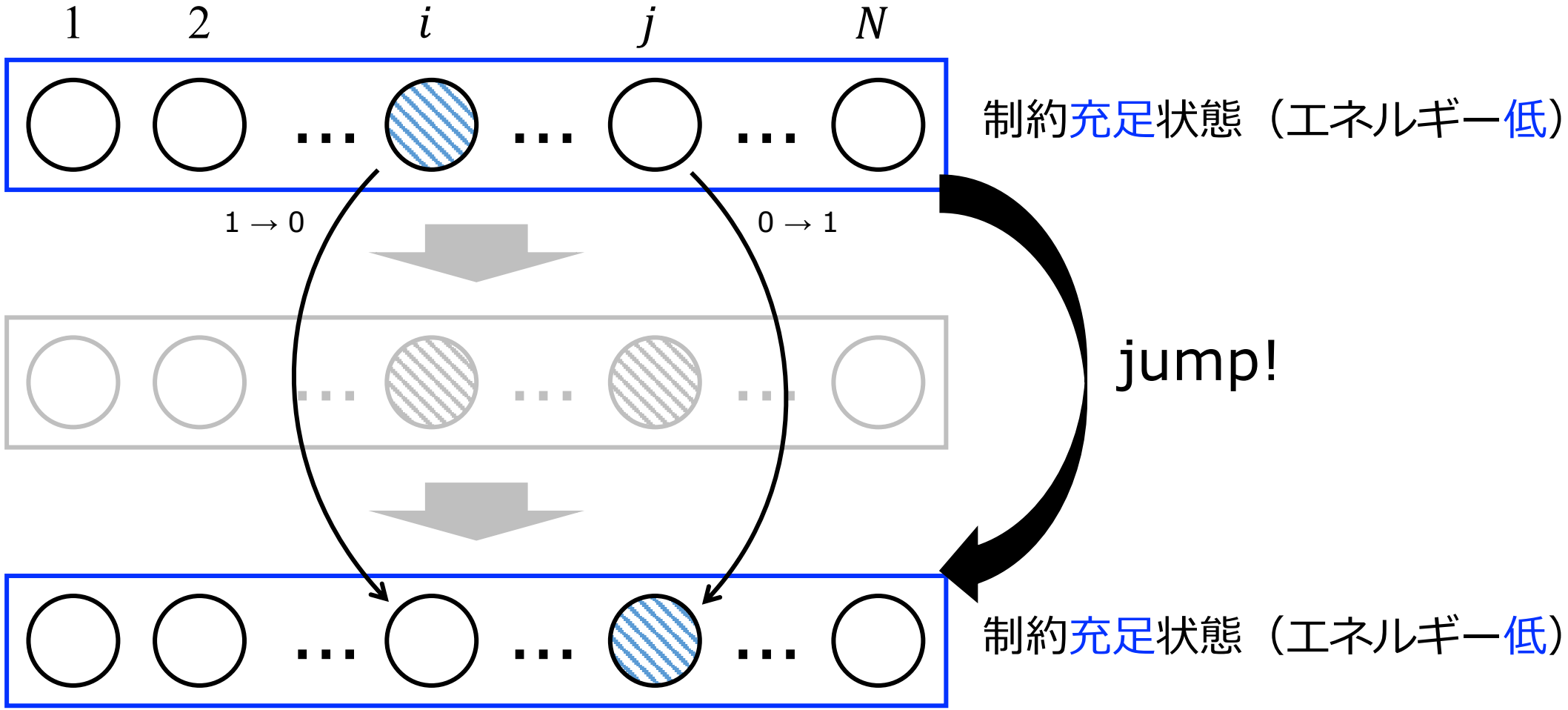
Simulated Annealing (SA)【従来の基底状態探索手法】

- SAでは1変数の更新を繰り返し、基底状態を探索



2変数同時更新SA【提案手法】

- One-hot制約を満たすように2変数を同時更新



ソフトウェアシミュレーション for JSP

- ft06（#jobs = 6）from JSPLIB (https://github.com/tamy0612/JSPLIB)

		【従来SA】1変数ずつ更新			【提案SA】2変数同時更新		
T	変数の個数	One-hot制約充足率	JSP制約違反数 最小	平均	One-hot制約充足率	JSP制約違反数 最小	平均
55 (opt)	834	1.0	3	3.7	1.0	1	1.0
58	942	1.0	2	3.1	1.0	1	1.0
60	1014	1.0	1	2.5	1.0	0	0.1
70	1374	1.0	1	2.0	1.0	0	0.0
80	1734	1.0	2	2.2	1.0	0	0.0

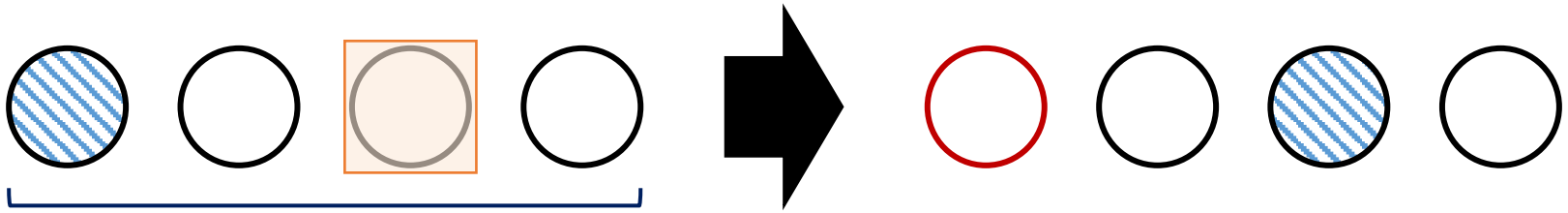
JSP制約
1. 実行順序制約
2. マシン同時使用禁止制約

シミュレーション条件
- 従来SA：#loops = 5.0×10^7
- 提案SA：#loops = 1.0×10^7

- 2変数同時更新SAを用いた場合のみ、制約充足解を得ることに成功

2変数同時更新SAのアルゴリズム

A) ランダムに選択された変数が 0 の場合

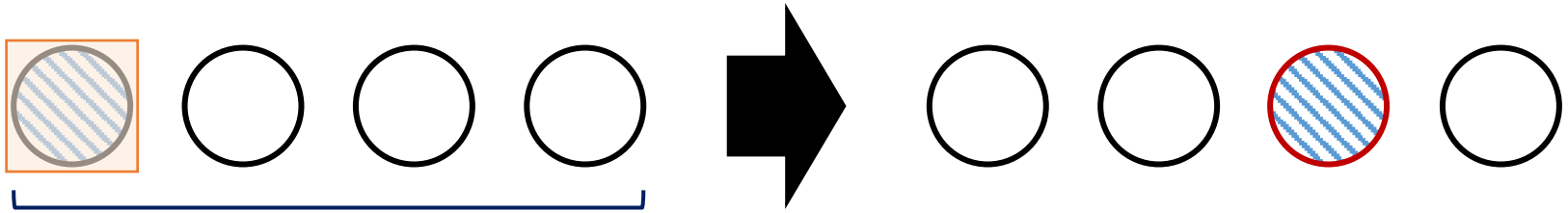


1-1. 選択された変数を含む One-hot制約**範囲を特定**

1-2. 特定された変数の範囲から1の**変数を特定**

新規に導入すべき動作

B) ランダムに選択された変数が 1 の場合



2-1. 選択された変数を含む One-hot制約**範囲を特定**

2-2. 特定された変数の範囲から1に変更する**変数をランダム選択**

(A)と(B)は実質的に**等価**の動作

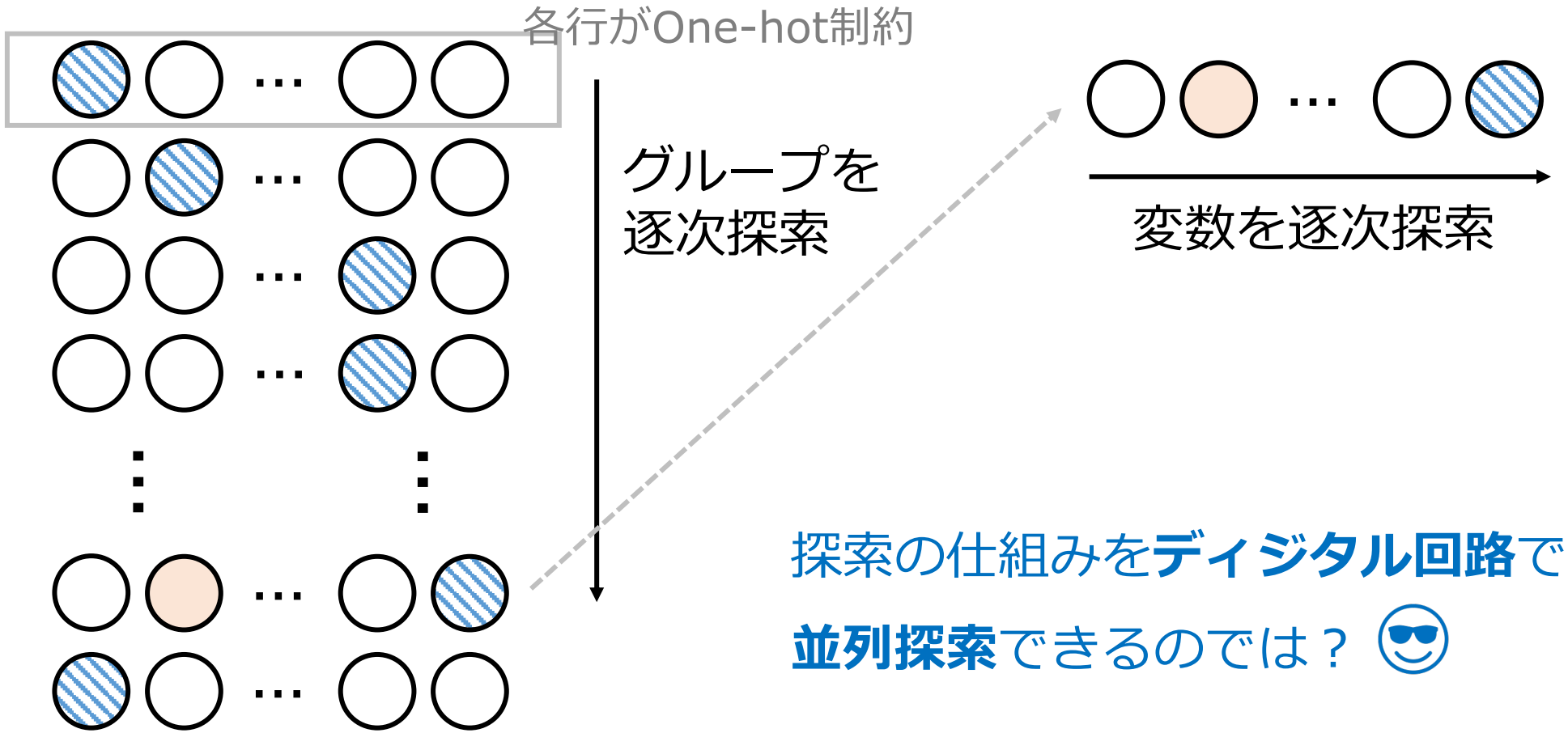
実際には**(A)のみ**を実装すれば十分

発生確率は(A)≫(B)

どう実現する？

One-hot制約範囲の特定 & 1の変数の特定

- ソフトウェアプログラムとして実装すると…

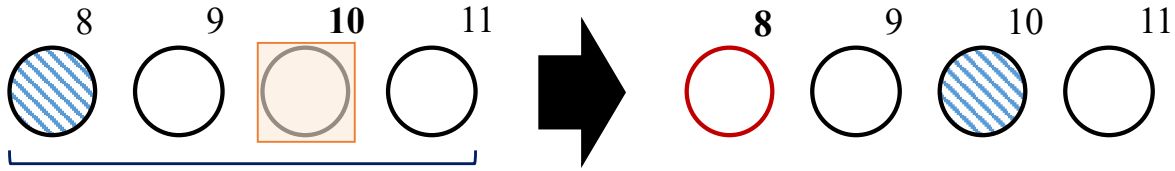


探索の仕組みを**ディジタル回路**で構成すれば

並列探索できるのでは？

２．提案アルゴリズムを実現するディジタル回路アーキテクチャの設計

アルゴリズムの実現



1-1. 選択された変数を含む One-hot制約**範囲を特定**

1-2. 特定された変数の範囲から1の**変数を特定**

1.1

変数index

10

範囲特定モジュール

グループ番号

3

One-hot制約範囲管理テーブル

One-hot制約範囲管理テーブル

グループ番号	開始index	終了index
1	0	3
2	4	7
3	8	11
4	12	15
⋮	⋮	⋮
$N-1$	36	39
N	40	43
$N+1$	44	44
⋮	⋮	⋮
32	44	44

連番で管理

1.2

8

開始index

終了index

11

変数特定モジュール

8

変数index

変数レジスタ

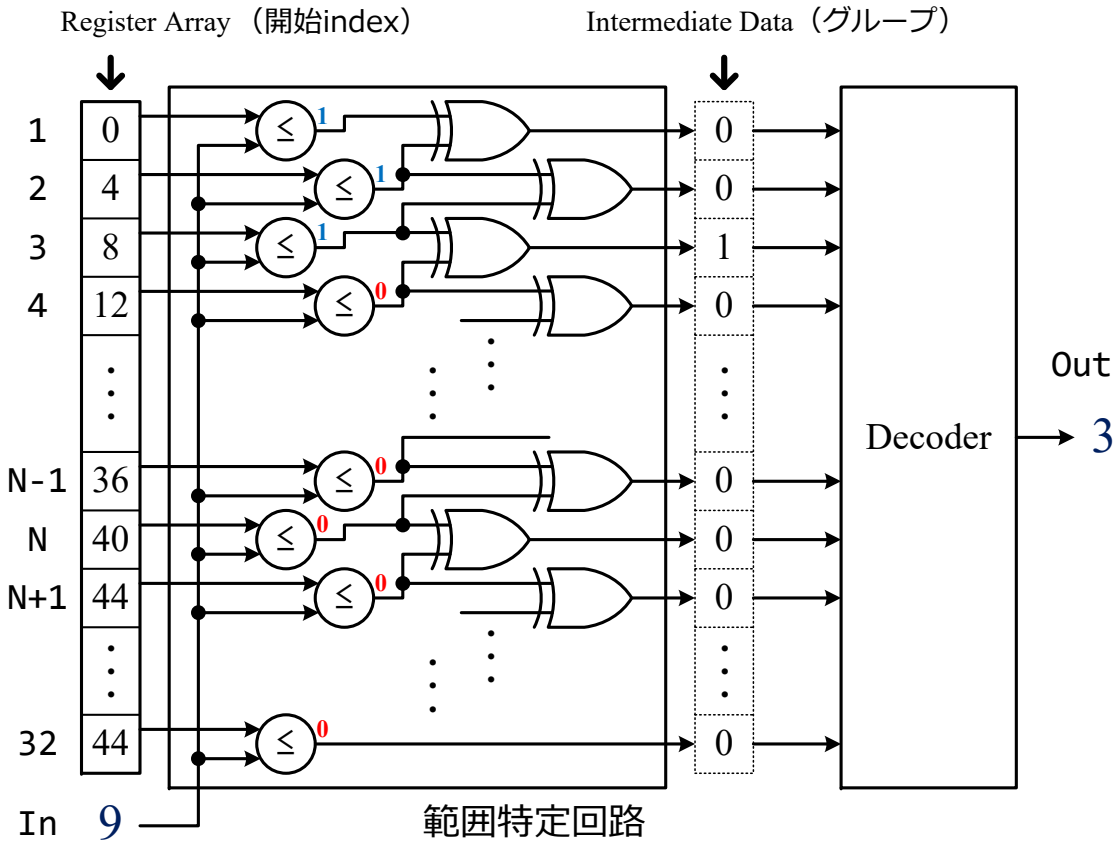
変数レジスタ

0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	1	0	1	0	0	1	0	0	0	0

・・・

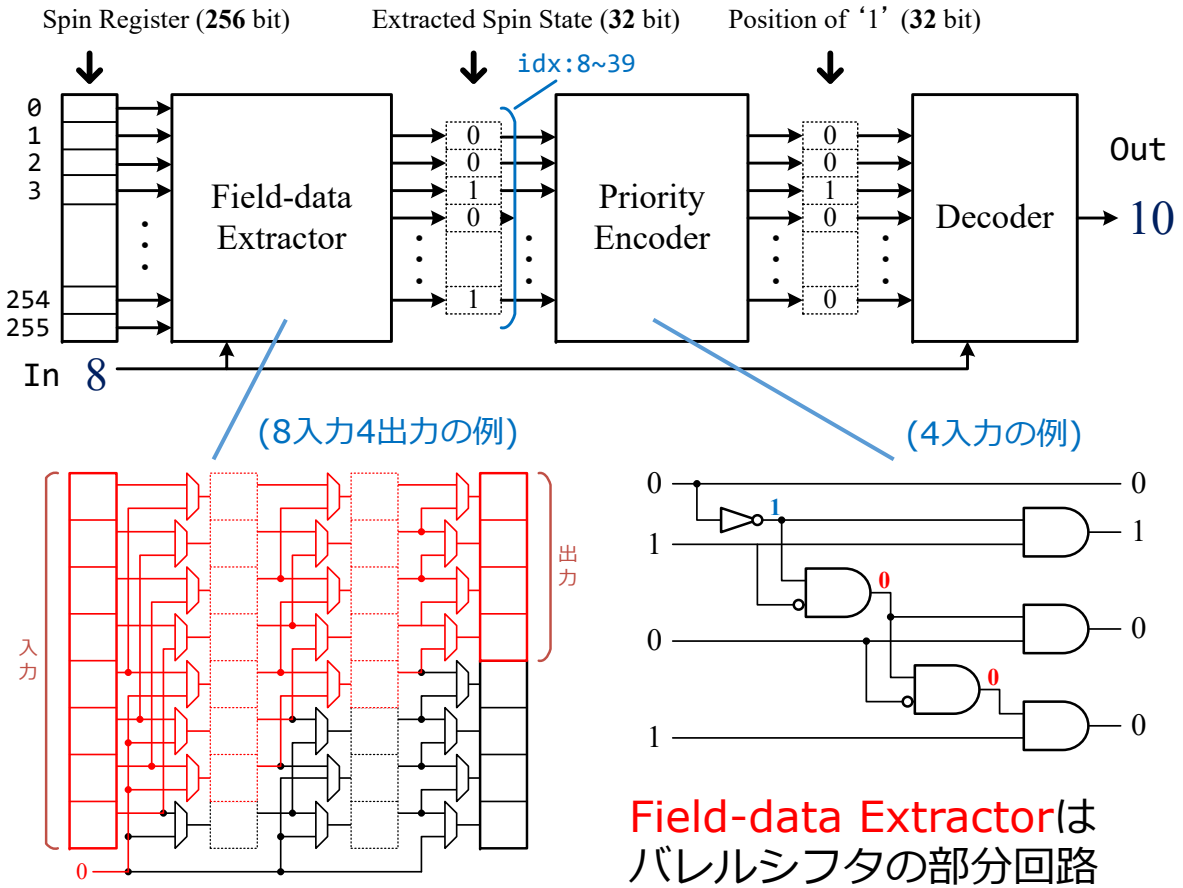
ディジタル回路アーキテクチャ

範囲特定モジュール



マージソータ[1]を参考に構成

変数特定モジュール



Field-data Extractorは
バレルシフタの部分回路
として構成される[2]

[1] https://ieeexplore.ieee.org/document/7966636

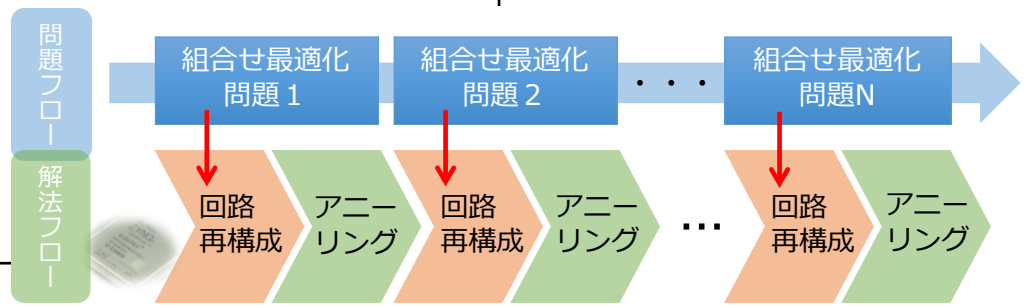
[2] https://search.ieice.org/bin/summary.php?id=e100-a_4_1015

３．FPGAへの実装、評価

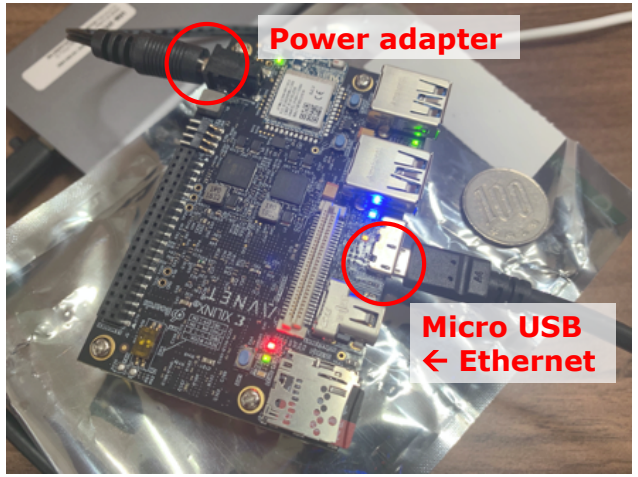
FPGAベース イジングマシン

- なぜFPGA？ → イジングマシンを自作してみたかった 😉
  自作のハードウェアが手元で動いたら楽しい 😆
- 強みは？？

vs CPU	• 演算処理のカスタム化、並列化 → 高速化
vs GPU	• 追加機能（e.g. 同時更新変数の選択）を低コストで実装可能 • 低消費電力な小型デバイスの存在 → IoT機器として活用？
vs ASIC	• 同一デバイスで機能の切り替えが可能 • 開発コストが低い • 開発から実働までの時間が短い



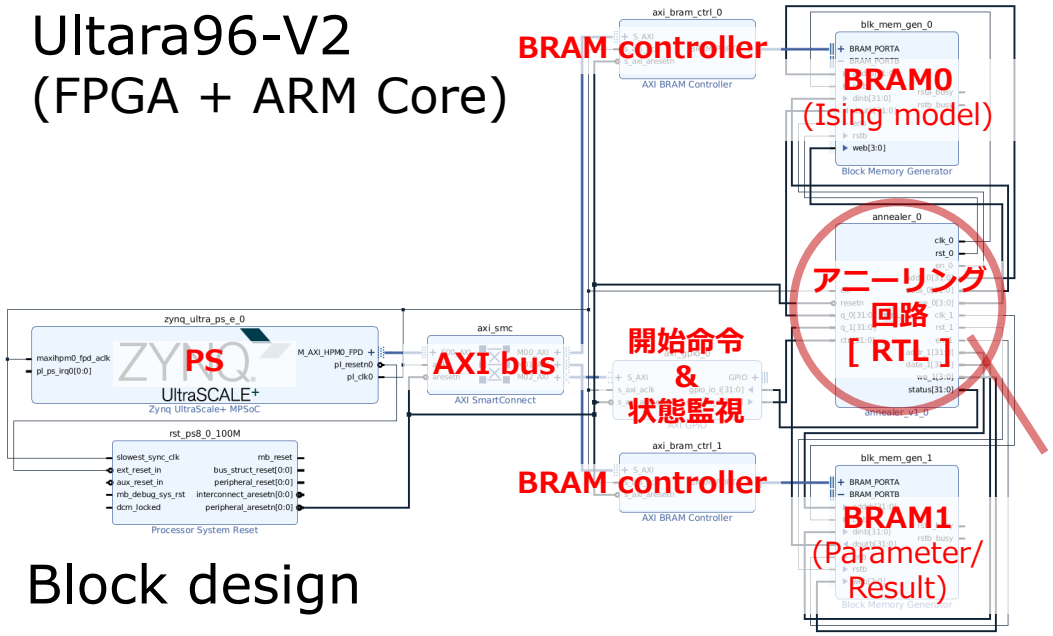
FPGAへの実装 → 実行



QUBOソルバ

- 16bit, 512変数、全結合
- 32bit, 256変数、全結合

Ultara96-V2
(FPGA + ARM Core)



Block design

合成ツール

ハードウェア記述言語
Verilog-HDLで記述

Jupyter notebook
で実行

Hardware Configuration

```
In [21]: from pynq import Overlay
         from pynq import MMIO

         base = Overlay("/home/xilinx/pynq/overlays/201210_QUBO_Solv

         bram0_addr = base.ip_dict['axi_bram_ctrl_0']['phys_addr']
         bram1_addr = base.ip_dict['axi_bram_ctrl_1']['phys_addr']
         print(hex(bram0_addr))
         print(hex(bram1_addr))
         bram0 = MMIO(base_addr = bram0_addr, length = 256 * 1024)
         bram1 = MMIO(base_addr = bram1_addr, length =  16 * 1024)

         ctrl_addr = base.ip_dict['axi_gpio_0']['phys_addr']
         print(hex(ctrl_addr))
         ctrl = MMIO(base_addr = ctrl_addr, length = 0x1000)
```

```
0xa0000000
0xa0040000
0xa0050000
```

Transfer Data

```
In [22]: ## J & h
         for y in range(MAX_N):
             for x in range(MAX_N):
                 idx = y * MAX_N + x
                 if x != y:
                     bram0.write(4*idx, int(J[y][x]))
                 else:
                     bram0.write(4*idx, int(h[y]))

         ## spin states
         for i in range(pack_num):
             bram1.write(4*(i+8), int(state_compressed[i]))
```

Start Annealing

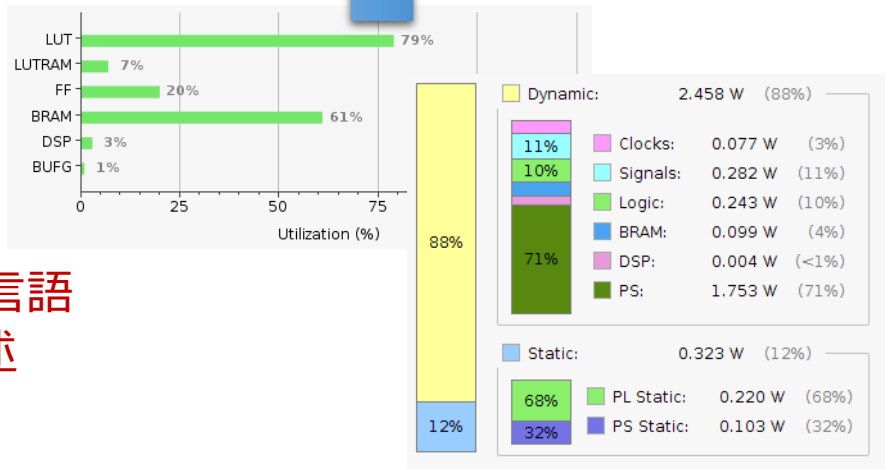
```
In [24]: import time
         ctrl.write(4, 0)
         ctrl.write(0, 1)

         t0 = time.time()
         ctrl.write(4, 0xFFFFFFFF)
         while ctrl.read(0) != 0:
             pass
         t1 = time.time()
         elapsed_time_prop = t1 - t0

         print('Elapsed time (Prop) [sec]:', elapsed_time_prop)
         print('Elapsed time (Conv) [sec]:', elapsed_time_conv)
         print('Rate:', elapsed_time_prop / elapsed_time_conv)
```

```
Elapsed time (Prop) [sec]: 0.06171679496765137
Elapsed time (Conv) [sec]: 0.02740335464477539
Rate: 2.2521620351841865
```

Confirm Result

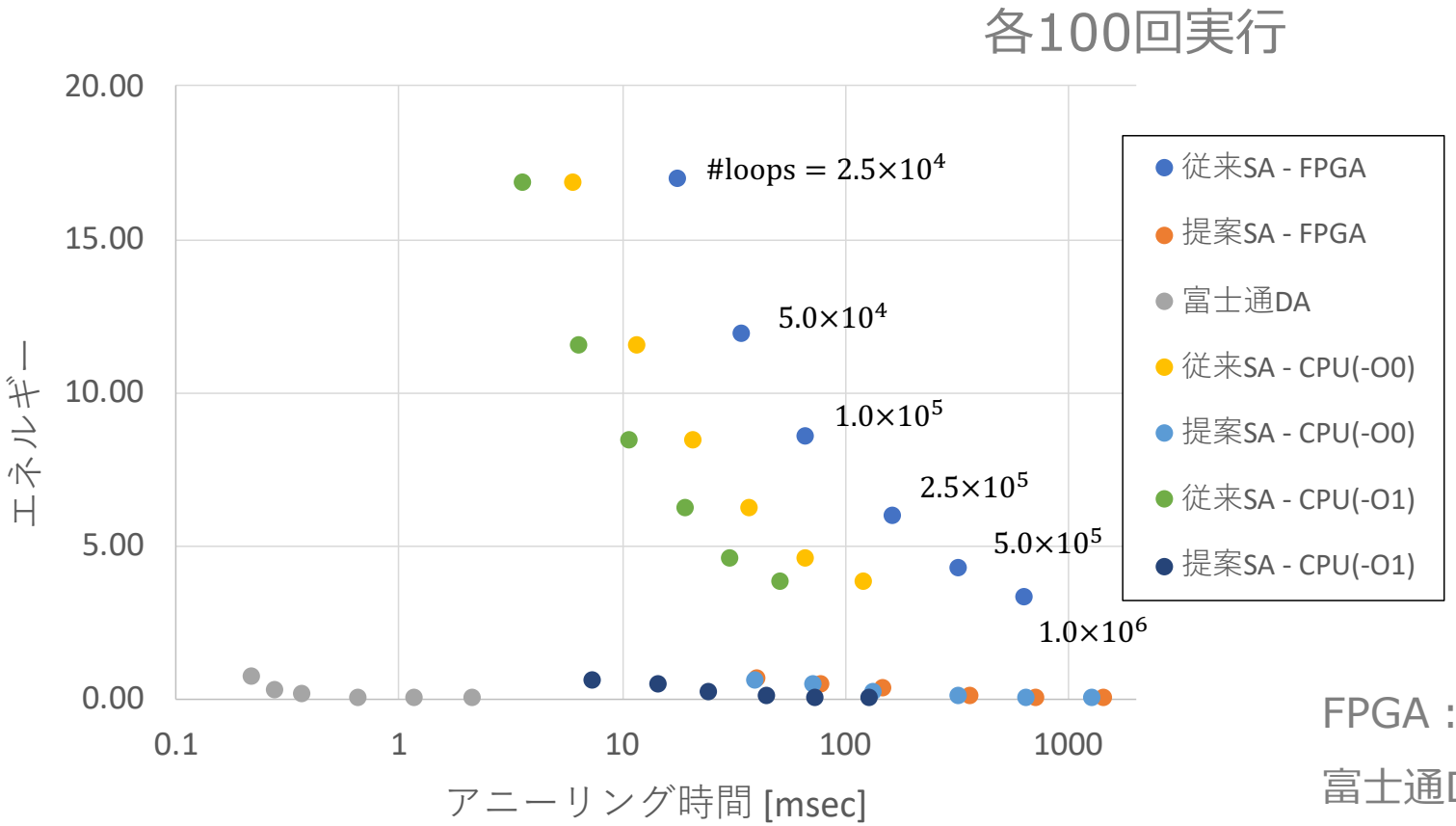
```
In [25]: read_state = np.ones((MAX_N,), dtype=np.int32)
         for i in range(pack_num):
             val = bram1.read(4*(i+48))
             val_str_list = list(format(val, '032b'))
             for j in range(32):
                 if val_str_list[31-j] == '0':
                     read_state[i*32+j] = 0
                 else:
                     read_state[i*32+j] = 1
         read_energy = np.dot(np.dot(read_state, J), read_state) /
         print('-- read energy --')
         print(read_energy)
```

```
-- read energy --
0
```



性能評価：解品質・アニーリング時間

- グラフ頂点彩色問題[3]：頂点数93、5色（＝465変数）



- ✓ #loopsを揃えた場合、提案SAは富士通DAと同等の解品質
- ✓ 提案FPGAマシンのアニーリング時間は、CPU(最適化なし)と同等
- ✓ 現状、モデル情報の読み出しがボトルネック
- ✓ 並列読み出しにより一桁程度の改善が期待される

FPGA：Ultra96-V2

富士通DA (Digital Annealer)：第二世代

CPU：Intel Xeon W-2125 @4.00GHz

注）C++ソフトウェアコードは疎結合モデル用に最適化

[3] https://sites.google.com/site/graphcoloring/vertex-coloring

４．イジングマシン利用環境の提供

イジングマシン シミュレータ（Python）

- ハードウェア処理（乱数生成、近似計算）を忠実にシミュレート
- 内部処理の可視化、期待値検証、実行時間の見積もり

```python
# Transform to binary number
def decimal_to_bin(data, ibit, fbit):
    ...
# Transform to decimal number
def binary_to_dec(str_int, str_frc):
    ...
# binary multiplier
def binary_mul(int1, frc1, int2, frc2, ibit, fbit):
    ...
```

```
Model file: QUBO_problem/2-FullIns_3_QUBO_5.dat
One-hot table (size: 52)
[(0, 4), (5, 9), ... , (250, 254), (255, 259)]
----------------------------------------
-- Model & Parameters ------------------
----------------------------------------
 N = 260
 #Loops  = 100 x 1000
 T(set)  = 100.0 --> 0.1
 T(real) = 100.0 --> 0.10000066578831485
 seed = 12345
----------------------------------------
100%|██████████████████████████████| 100/100 [00:02<00:00, 39.72it/s]
----------------------------------------
-- Result ------------------------------
----------------------------------------
 H(init) = 201
 H(fin)  = 0
 #. One-hot constraint violations = 0
----------------------------------------
```

kawamura-labwork Update README.md　205f86e on 15 Sep　14 commits

G1_Ising.dat	Update model file	2 months ago
README.md	Update README.md	2 months ago
simulator.py	Update model file	2 months ago

README.md

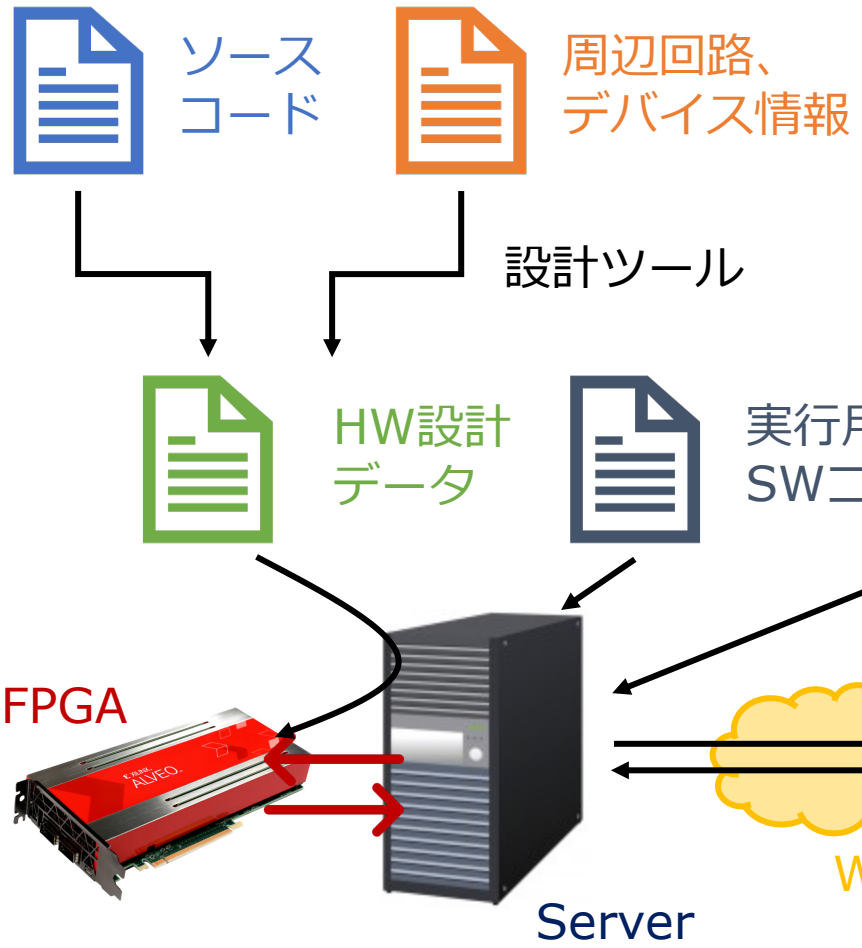
HW simulator of SA

GitHub コード公開中

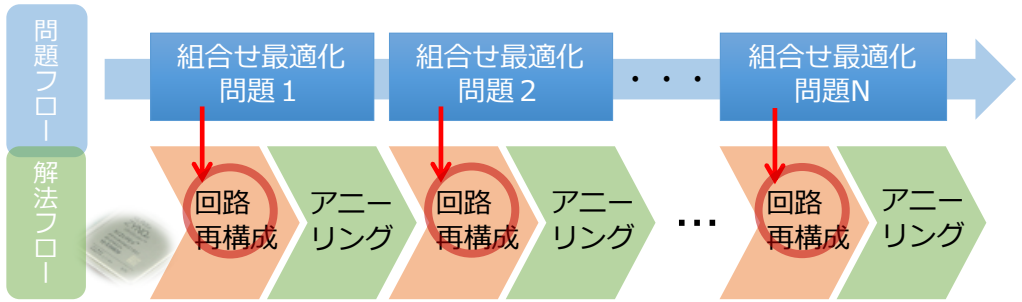
https://github.com/kawamura-labwork/SA-HW-simulator

今後の展望

a. イジングマシン利用環境の提供



b. カスタムイジングコンピューティングの普及に向けた機能拡張



① ACRiルーム

ssh

https://gw.acri.c.titech.ac.jp/wp/

② AWS

Web API

https://aws.amazon.com/jp/ec2/instance-types/f1/