プログラマの為の量子コンピュータ入門

Part 3:
量子アニーリング型のプログラミング

LangEdge,Inc.
有限会社 ラング・エッジ

宮地直人（miyachi@langedge.jp）

Ver1.0 2019年9月26日

属性：  @le_miyachi

技術：古典PKIプログラマ

仕事：ぼっち有限会社（電子署名系）

量子：独自に勉強（書籍・勉強会）

趣味：勉強会の開催、OSS開発

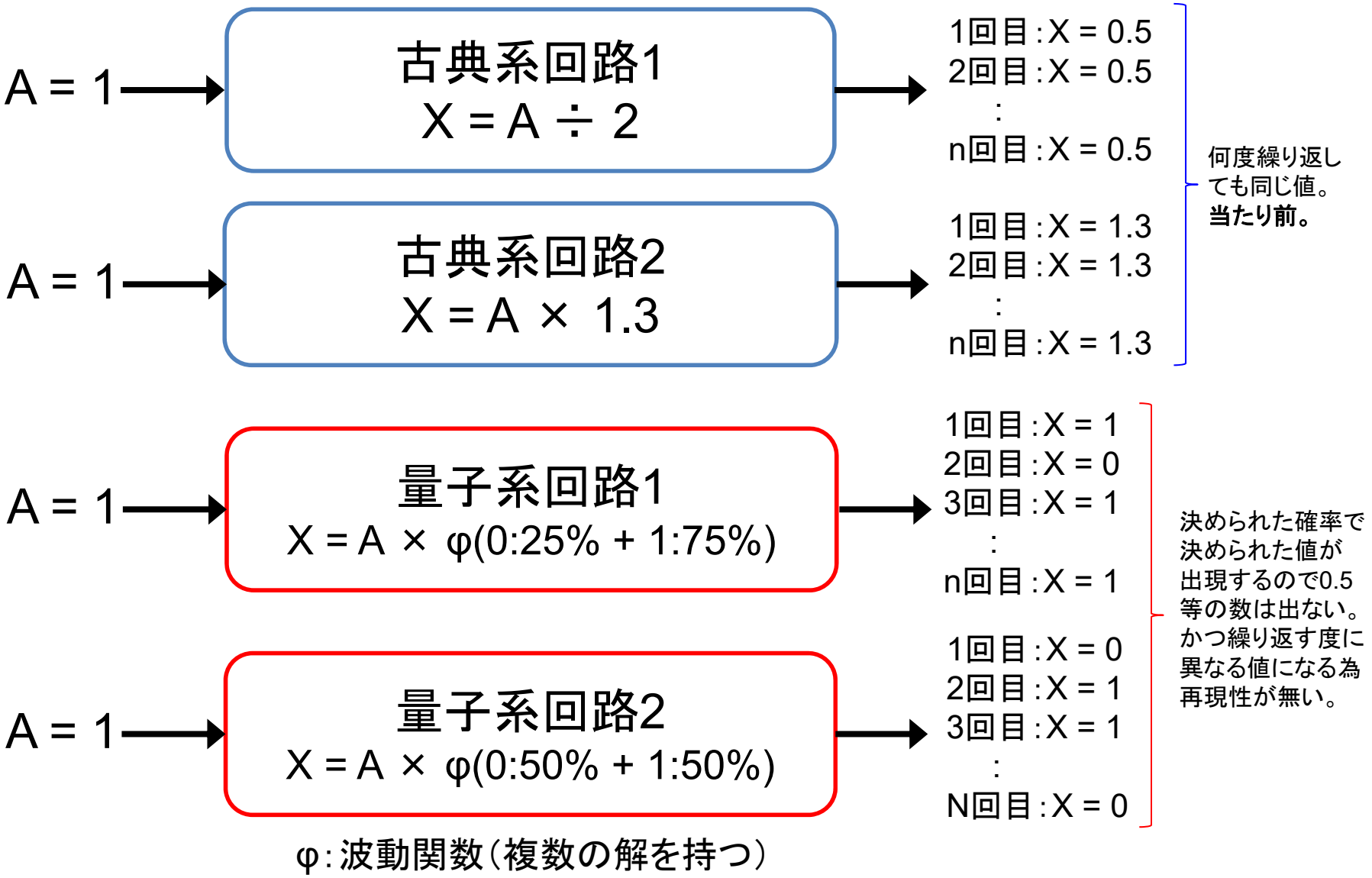
活動：OsSAL.org （オッサル）他

オープンソース署名＆認証ラボ

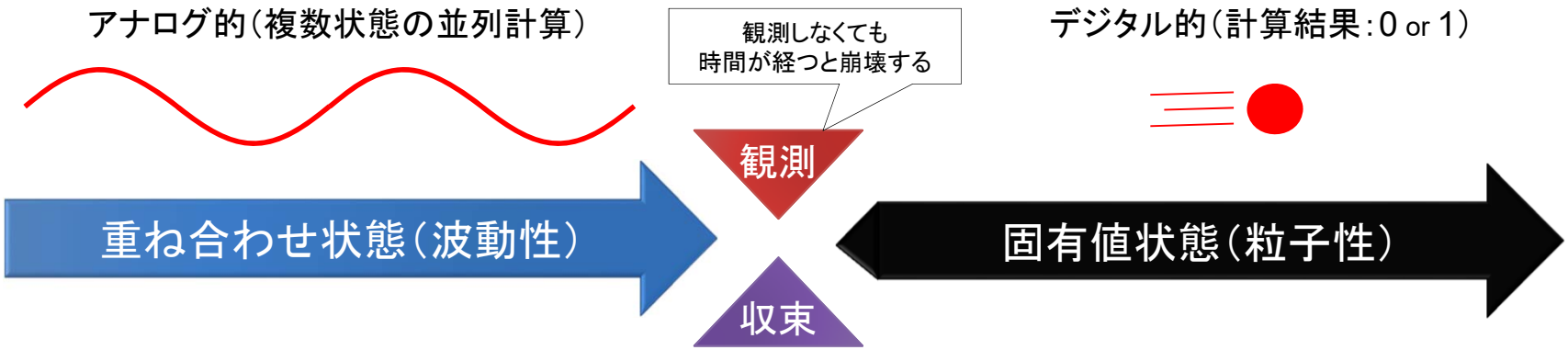
Part 0：イントロダクション

（振り返り短縮版）

古典系と量子系の測定


A = 1
古典系回路1
X = A ÷ 2
1回目 : X = 0.5
2回目 : X = 0.5
:
n回目 : X = 0.5
A = 1
古典系回路2
X = A × 1.3
1回目 : X = 1.3
2回目 : X = 1.3
:
n回目 : X = 1.3
何度繰り返しても同じ値。
当たり前。
A = 1
量子系回路1
X = A × φ(0:25% + 1:75%)
1回目 : X = 1
2回目 : X = 0
3回目 : X = 1
:
n回目 : X = 1
A = 1
量子系回路2
X = A × φ(0:50% + 1:50%)
1回目 : X = 0
2回目 : X = 1
3回目 : X = 1
:
N回目 : X = 0
決められた確率で決められた値が出現するので0.5等の数は出ない。かつ繰り返す度に異なる値になる為再現性が無い。
φ : 波動関数（複数の解を持つ）

波動と粒子の二重性（量子重ね合わせ）



複数の状態が同時に存在して
量子干渉を生じている
（コペンハーゲン解釈）

波動関数（列ベクトル）を
$|\psi>$ と表現すると

$$|\psi> = c_0|\psi_0> + c_1|\psi_1>$$
$$|c_0|^2 + |c_1|^2 = 1.0$$

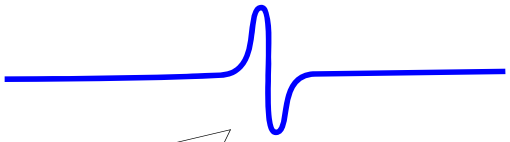
c_0 と c_1 は複数数

量子デコヒーレンス

固有状態のうち1つの状態

$|\psi_0\rangle$ または $|\psi_1\rangle$ のいずれか

波動性は失われる
複素数虚数部は意味を持たない



コペンハーゲン解釈では、粒子状態
は波動がパルス状になったと考える。
位置が決まると波長は定まらなくなる。

※ コヒーレンス時間（状態の量子干渉が失われるまでの時間）は通常短い。

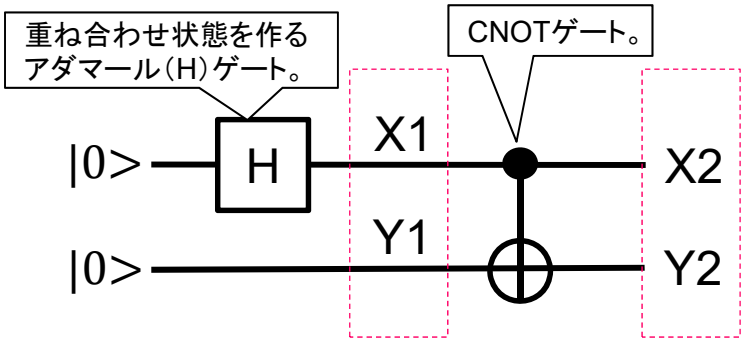
量子ビット（重ね合わせの実現）

- 0と1の量子重ね合わせの状態が可能な単位。
- 観測により2つの固有値（0か1）に収束する。
- 物理的に実現するには幾つかの方法がある。

方式	概要	開発
超伝導量子ビット	現在主流となっている超伝導状態のシリコン回路で量子ビットを実現する方式（極低温）	IBM, Google, D-Wave
イオントラップ	捕獲したイオンをレーザーで冷却して利用（室温） 理論的には量子ビット間の全結合が可能	IonQ
量子ドット	原子10～50個で構成した微小半導体を利用（極低温?）	Intel
トポロジカル	超電導体とトポロジカル絶縁体による量子ビット（極低温?）	Microsoft
NVセンター ダイヤモンド窒素-空孔中心	ダイヤモンドの炭素を窒素に置き換えて生じる欠損部に電子を捕獲して量子ビットに利用（室温）	（研究レベル）
光子 光量子コンピュータ	光子パルス群を量子ビットとして利用（室温） 全結合によるアニーリング型の計算が可能	NTT/NII/東大（ImPACT）

量子もつれ（量子エンタングルメント）

- 量子もつれは、2つの粒子（量子ビット）が互いに影響をおよぼし合い、一方を測定すると、もう一方の値が確定する現象である。
- 複数の量子ビット間を、量子もつれにより関連付けることで量子回路を構築する。以下HとCNOTによる例。



$X1 = |0\rangle : 50\% + |1\rangle : 50\%$
$Y1 = |0\rangle$

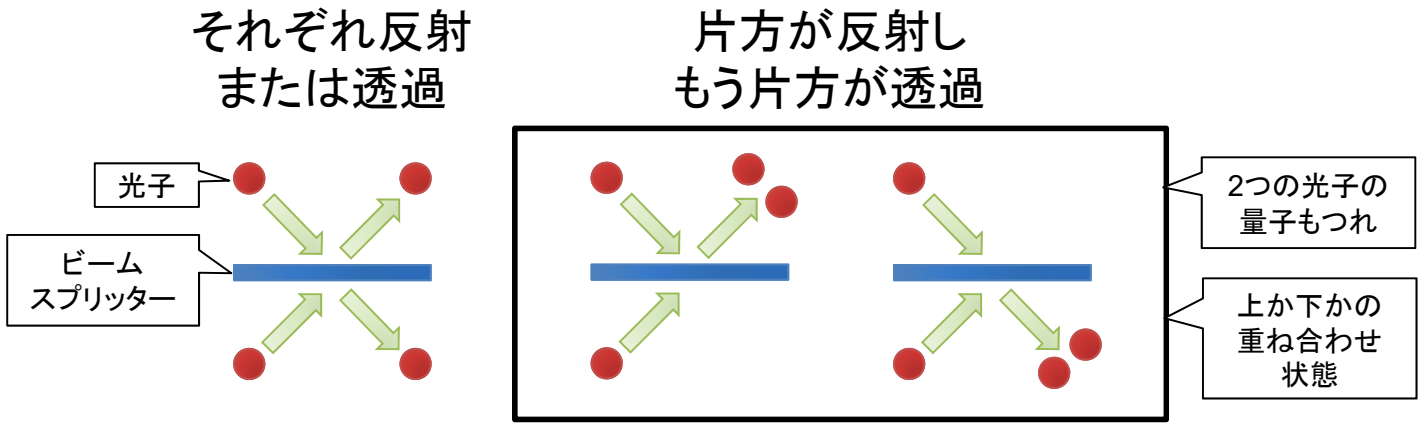
$X2 = X1 = |0\rangle : 50\% + |1\rangle : 50\%$
X1が $|0\rangle$ なら Y2も $|0\rangle$ （Y1そのまま）
X1が $|1\rangle$ なら Y2も $|1\rangle$ （Y1を反転する）

$X2Y2 = |00\rangle : 50\% + |11\rangle : 50\%$
※ $|01\rangle$ や $|10\rangle$ は 0%

光量子コンピュータ

上下から光子対を入射する。

・ビームスプリッターとはハーフミラーのこと。

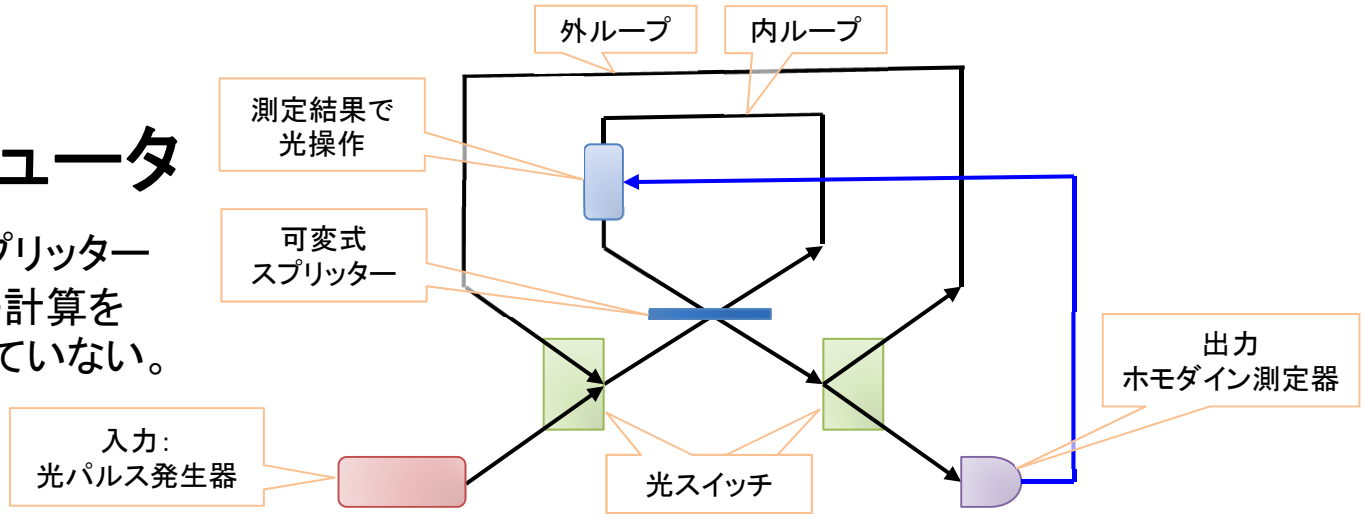


※ 測定するとこちらのいずれかになる

ループ型光量子コンピュータ

1つの可変式ビームスプリッターを何度も使うことで量子計算を行っていく。まだ完成していない。

複数のビームスプリッターを使うことで量子計算を行っていくタイプもある。



量子コンピュータの種類

量子ゲート型（狭義の量子コンピュータ）

方式：量子ゲートの量子回路による量子計算

対象問題：汎用（ただし量子アルゴリズムの範囲内）

開発企業：IBM/Google/Intel/Alibaba/Microsoft等

量子アニーリング型（正確には量子シミュレータ）

方式：イジングモデルを使った量子シミュレーション

対象問題：最適化問題特化（深層学習等への応用）

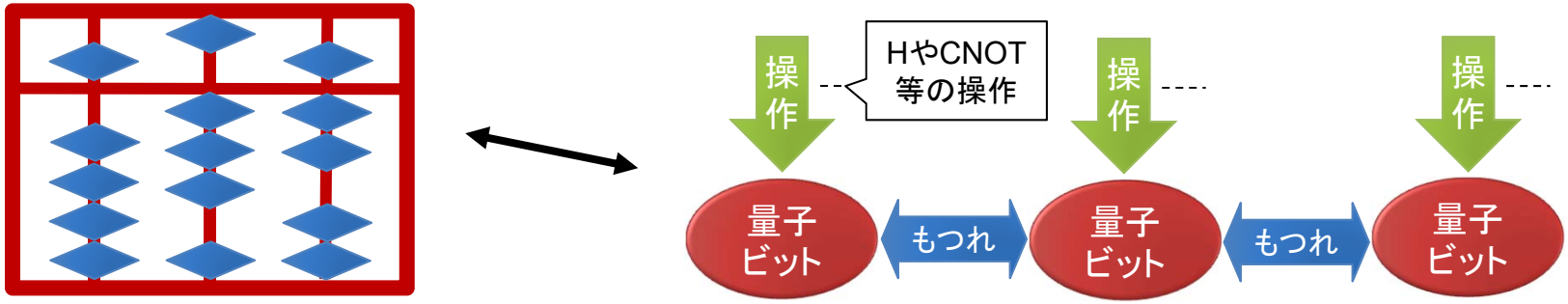
開発企業：D-Wave（非量子型では富士通と日立）

※ 非量子：富士通「デジタルアニーラ」、日立「CMOSアニーリングマシン」。

※ 他に光を使ったCIM（コヒーレントイジングマシン）もあるがここでは省略。

量子ゲート型とソロバン

ソロバンは珠(たま)を配置したハードウェアを、指で弾いて行くことで計算を進めて行く。各珠の間には桁上がり等の関連性がある。

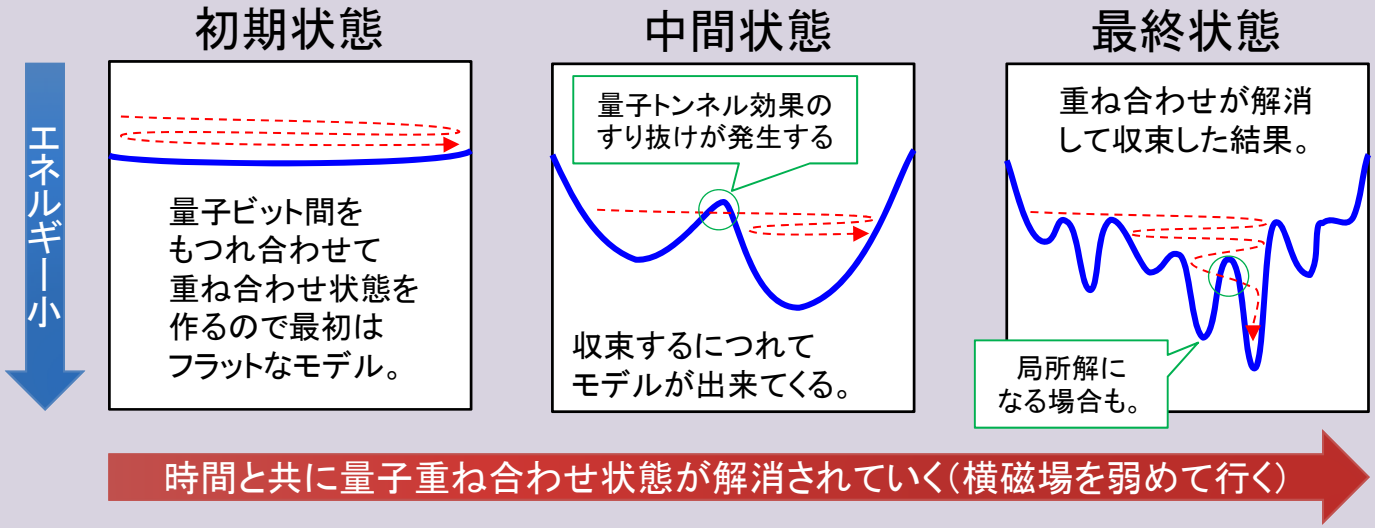


量子ゲート型は量子ビットを配置したハードウェアを、レーザーや電界で弾いて行くことで計算を進めて行く。各量子ビット間には量子もつれによる関連性がある。

ソロバンは可逆回路でもある。またソロバンは同じ操作をすれば毎回同じ値になるが量子では異なる。

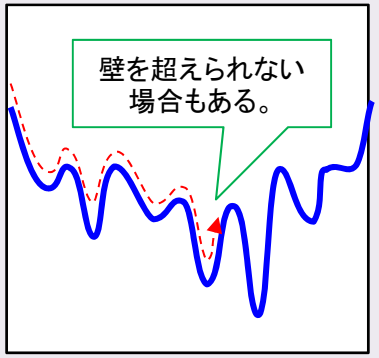
量子アニーリング型の計算

量子アニーリング



量子アニーリングでは量子ビット間の全結合(もつれ)が理想だが…

シミュレーテッドアニーリング



シミュレーテッドアニーリングは先にモデルをセットしてから最適解を求めて行く。

シミュレーテッドアニーリングは古くから使われており、現在でも富士通や日立が専用ハードウェアを使ったアニーリングを実現(例:デジタルアニーラ)している。

NISQの時代（今後5～10年程度）

Q2B: QUANTUM FOR BUSINESS 2017

物理学者 John Preskill による基調講演の論文

「*Quantum Computing in the NISQ era and beyond*」

https://arxiv.org/abs/1801.00862

NISQ: Noisy Intermediate-Scale Quantum

ノイズエラーがあり中規模量子ビット数の時代

50～数百量子ビット程度

現在は標準ハードウェアと標準ソフトウェア（アルゴリズム）を確立する時期で、特にソフトウェアは量子シミュレータを使って勉強しておくことで量子プログラミング時代に備える。

量子コンピュータの未来予想

量子超越
（古典コンピュータでは解けない問題）

RSA暗号解読

どこかでブレークスルーを生じる可能性はゼロではないが…

2015　2018　2023～2030　2040～2060くらい（予測困難）

	NISQ *ノイズあり中規模*	*量子加速* *（Quantum Speedup）*	*万能（Universal）量子コンピュータ*
実用性 汎用性	ほぼない （一部を除き古典機の方が早い）	一部あり （古典機よりも早い分野がある）	あり （ただし得意不得意はある）
信頼性	ノイズ（エラー）あり	（部分的な）エラー訂正	エラー訂正あり
量子数	$10 \sim 10^2$	（エラー訂正あり）$10^2 \sim 10^6$	10^6（数百万）以上
利用形態	専用機	専用機・外部接続装置	単独・古典機と融合

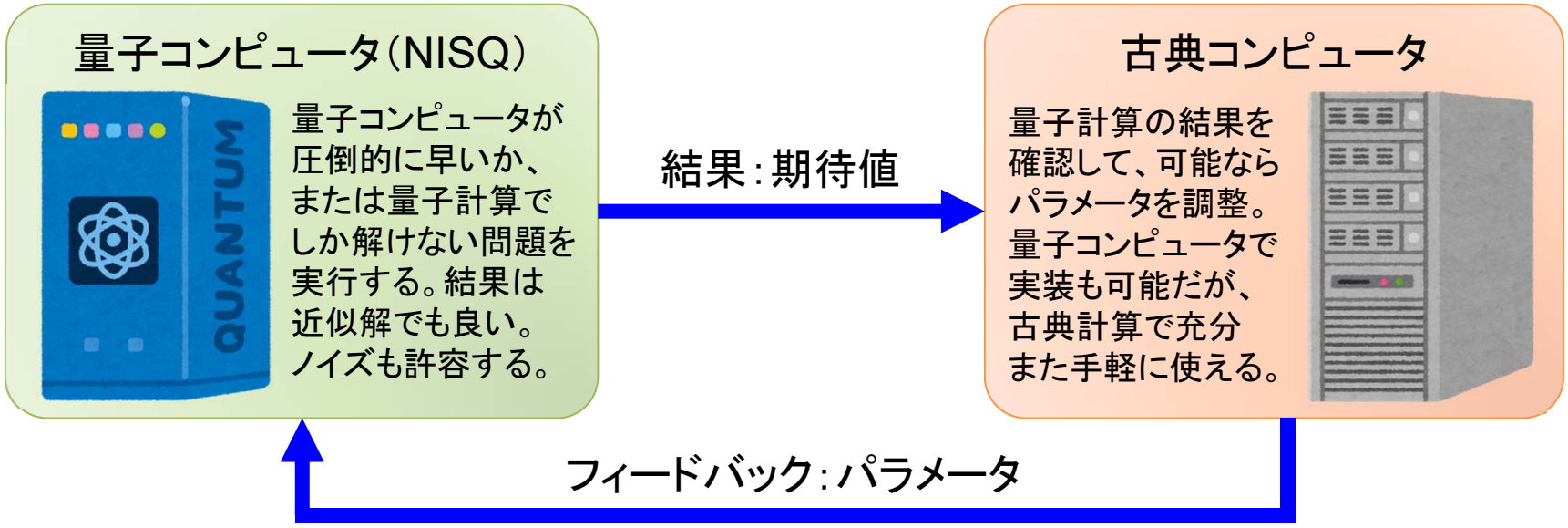
参考：（ノイマン型）古典コンピュータの歴史

1950　1955　1970

第1世代 *真空管・試用や研究*	*第2世代* *トランジスタ・商用化*	*第3世代* *集積回路・パーソナル化*

古典量子ハイブリッドアルゴリズム

現実的な利用方法として古典コンピュータとの組合せ利用が進んでいる。
第2部で説明するショアのアルゴリズムもある意味ハイブリッド計算である。



主な用途:

- 近似最適化:QAOA(Quantum Approximate Optimization Algo)
- 基底状態探索:VQE(Variational Quantum Eigensolver)
- 機械学習:QCL(Quantum Circuit Learning)

量子計算フレームワーク（量子ゲート型）

IBMやGoogleは自社量子コンピュータを使う為の量子計算フレームワークを公開している。実機だけではなくシミュレーション機能を持っているので、量子プログラミングの勉強用として最適だが小規模の量子回路のみとなる。

項目	IBM Qiskit	Google Cirq
ロゴ	 Quantum Information Science Kit	 Cirq
構成	Terra: 量子計算の基盤部（Python） Aqua: 量子アルゴリズムのライブラリ OpenQASM: 量子低レベル中間言語	Cirq:量子計算基盤Pythonライブラリ OpenFermion: 量子化学ライブラリ
提供	オープンソース（GitHub）	オープンソース（GitHub）
取得	https://qiskit.org/ https://github.com/Qiskit	https://github.com/quantumlib/Cirq
情報	https://qiskit.org/documentation/ja/	https://cirq.readthedocs.io/en/latest/
その他	IBM Q Experience: GUI利用 ※ GUIからOpenQASMに展開し実行 https://quantumexperience.ng.bluemix.net/	2018年夏に正式公開されたライブラリ 量子コンピュータ実機はまだ使えない

量子計算フレームワーク（量子アニーリング型）

日本のベンチャーが開発した量子計算フレームワークも公開されている。
D-Wave社のOceanもQiskit/Cirqと同じく基本的には自社ハード用SDKである。

項目	Blueqat	D-Wave Ocean SDK
ロゴ	Blueqat	D-Wave The Quantum Computing Company™
構成	量子ゲート型の計算（本来ゲート型） 量子アニーリング計算も可能	量子アニーリング型の計算
提供	オープンソース（GitHub）	オープンソース（GitHub）
取得	https://github.com/Blueqat	https://github.com/dwavesystems/dwave-ocean-sdk
開発	株式会社 MDR https://mdrft.com/?hl=ja	D-Wave Systems, Inc.（カナダ） https://www.dwavesys.com/
その他	Qiskit/Cirqよりも回路記述が簡単。 量子アニーリング用のWildqatも統合。 日本語のSlackも参加可能。 勉強会・ブログ等で積極的に日本語で情報発信をしている。	D-Waveマシン用のSDK。 シミュレーテッドアニーリング計算も可能。 計算に実機を使う為には登録が必要。 D-Wave Leap https://www.dwavesys.com/take-leap

Part 3：量子アニーリング型のプログラミング

アニーリング計算には数学的な考え方が必須。
数式が一杯出て来ますのでご覚悟を…

アニーリングは本来量子を使わない計算です。

組み合わせ最適化問題

様々な制約の下で多くの選択肢の中から、指標(コスト)を最も良くする結果(組み合わせ)を得る問題が、組み合わせ最適化問題。

例:巡回セールスマン問題

複数都市を1回ずつ訪問するセールスマンの最適(最短距離)な巡回順番を求める問題。
各都市間の距離がコストとして与えられる。

組み合わせ爆発

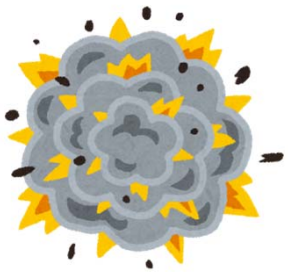
巡回セールスマン問題において、n 都市を巡回する経路の全組み合わせは $(n-1)!$ 個あるが、逆方向は同じとして半分の $(n-1)!/2$ 個となる。

$$n! = n \times (n-1) \times (n-2) \times (n-3) \times \ldots \times 2 \times 1$$

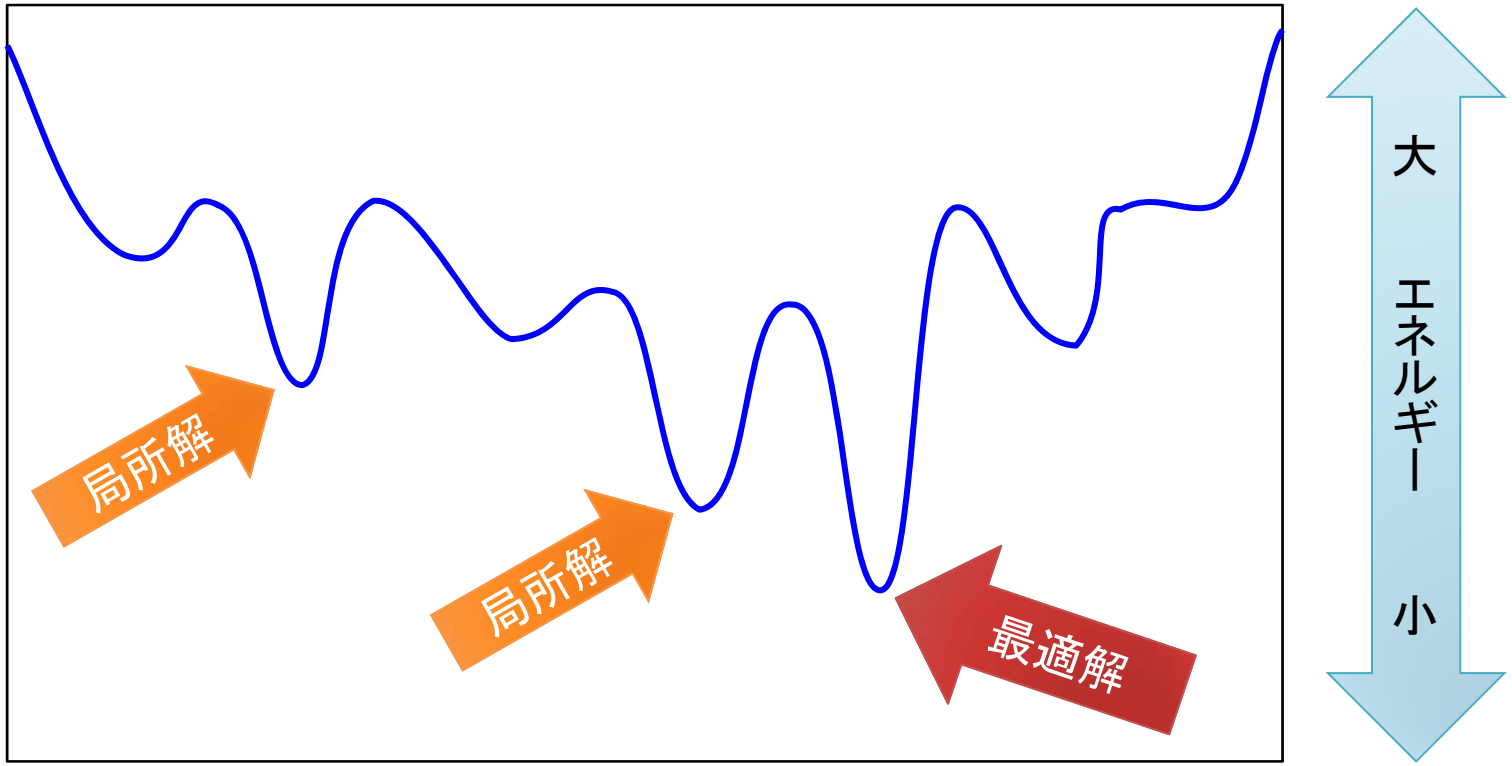
4都市だと $(4-1)\times(3-1)\div 2=3$ 通り、
20都市では 6.0×10^{16} 通り、
40都市では 1.0×10^{46} 通りとなり、
組み合わせ数が爆発的に増加する。

都市数	経路数
4	3
5	12
:	:
20	6.0E16
40	1.0E46
80	4.5E116

$(n-1)!/2 =$



最適解と局所解



最適解：最も条件を満たした解。上例では最小エネルギーの箇所。
局所解：局所的に最も条件を満たした解。他に最適解がある。

※ 問題によっては局所解で良い場合もある。
※ アニーリング計算で必ず最適解が出るわけでもない。

アニーリング（Annealing）：焼きなまし

焼きなましは、金属材料を熱した後徐々に冷やすことで、原子の内部エネルギーが極小になる状態（欠陥が減る）を得る方法。
この状態変化をシミュレーションすることで最適な解を得ることがアニーリング計算。

- **シミューレテッドアニーリング（SA）法**
  古典計算によりアニーリング計算を行う。
- **量子アニーリング法**
  量子計算によりアニーリング計算を行う。

アニーリング計算の手順（非量子も共通）

Step1 解きたい課題を組み合わせ最適化問題に変換する。

Step2 問題の定式化（上三角QUBOモデルのハミルトニアン式へ変換）。

$$H = \sum_{i<j} Q_{ij} x_i x_j + \sum_i Q_{ii} x_i$$

Step3 必要ならQUBO行列を入力イジング行列に変換する。

Step4 

イジングマシンにより最適解（局所解）を得る。 SA

Step5 必要なら結果イジング行列をQUBO行列に変換して確認する。

3-1：ハミルトニアンとQUBO

アニーリングではハミルトニアン（全エネルギー）を最小化（or最大化）する計算を行います。

まずバイナリ変数を使った上三角QUBO行列を用意して入力します。

ハミルトニアン（Hamiltonian）

ハミルトニアンとは、物理学における**エネルギー**に対応する**物理量**である。

※ ある状態にある時の全エネルギーと言っても良い。

古典力学：Hをハミルトニアン、Tを運動エネルギー、Vをポテンシャルエネルギー（例：位置エネルギー）として、全エネルギーを、$\boldsymbol{H = H(q, p; t) = T + V}$（一般化座標=$q$ / 一般化運動量=p / 時間=t ）、によって表した関数となる。

量子力学：ハミルトニアンは、系の全エネルギーを表す演算子として示される。Hをハミルトニアン行列、Eをエネルギー固有値とした場合に、時間発展しないシュレディンガー方程式を使い、$\boldsymbol{H\varphi(x) = E\varphi(x)}$ の固有値問題として表すことができる。

参考：シュレディンガー方程式と波動関数

※ 理解できなくても量子プログラミングでの大きな問題にはなりません。

（時間依存する）シュレディンガー方程式

$\hbar$：プランク定数h / 2π

m：質量

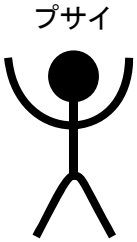
$$i\hbar\frac{d}{dt}\psi(x,t) = H\psi(x,t) = \left(-\frac{\hbar^2}{2m}\frac{d^2}{dx^2} + V(x)\right)\psi(x,t)$$

H:ハミルトニアン

時間から見た全エネルギー

ψ:波動関数

運動エネルギー　ポテンシャルエネルギー



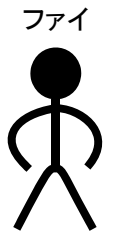
シュレディンガー方程式は、波動関数の値を求める方程式ではなく、
波動関数の式そのものを求める方程式となっている。

古典力学では粒子のエネルギーは保存される（時間依存が無い）ので、
時間発展しないシュレディンガー方程式を導くことができる。

$$E\phi(x) = H\phi(x) = \left(-\frac{\hbar^2}{2m}\frac{d^2}{dx^2} + V(x)\right)\phi(x)$$

エネルギー固有値

φ:波動関数
（時間依存無し）



http://scienceinoh.jp/schrodinger/

ハミルトニアンの計算例（総当たり計算）

例題：以下ハミルトニアン式が最小値を取る x_1 と x_2 を求めよ。

$$H = 4x_1^2 - 2x_2^2 + 2$$

※ x_1 と x_2 は 0 or 1 のバイナリ変数

計算：ここでは全てのケースの計算表を作成して確認。

x_1	x_2	H
0	0	2
0	1	0
1	0	6
1	1	4

アニーリングで計算することが最終目標ですが、
ここでは計算をイメージする為に手計算します。

$(x_1, x_2) = (0, 1)$ の時に、
最小値 $H = 0$ となっている。

答：$x_1 = 0$ と $x_2 = 1$ の時に最小値 $H = 0$（最適解）となる。

Blueqatを使う（MDR）

本来Blueqatはゲート型計算用のSDKですがアニーリング計算も可能。

環境：**Anaconda3**（**Python3.7**）

以下より環境に合わせてダウンロードとインストール

https://www.anaconda.com/distribution/

ライブラリ：**Blueqat**（ブルーキャット）

Windows版：Anaconda Prompt

MacOS版：ターミナル

インストール

```
pip install blueqat
```

バージョン指定インストール

```
pip install blueqat=0.3.9
```

アンインストール

```
pip uninstall blueqat
```

※ Blueqatのバージョン確認：

In:	import blueqat blueqat.__version__
Out:	'0.3.9‘

本資料のソースは 0.3.9 と表示される環境にて確認しています。

Blueqat でアニーリング計算（Wildqat）

Wildqat：アニーリング計算用のSDK（MDR社）
https://github.com/Blueqat/Wildqat

現在は**Blueqat**に組み込まれている。

```
from blueqat import opt     # Wildqatの機能
```

ドキュメント（日本語）：
https://wildqat.readthedocs.io/ja/latest/

- QUBOによるアニーリング計算が可能。
- D-Wave への接続も可能。

QUBOによるハミルトニアン計算

例題：以下ハミルトニアン式のQUBO行列から解を求めよ。

$$H = 4x_1^2 - 2x_2^2 + 2$$

x_1 と x_2 の2次式なので
QUBO行列が計算可能

※ x_1 と x_2 は 0 or 1 のバイナリ変数

全体にかかる係数
や倍数は省略可能

$$= \begin{pmatrix} x_1 & x_2 \end{pmatrix} \underbrace{\begin{pmatrix} 4 & 0 \\ 0 & -2 \end{pmatrix}}_{\text{QUBO行列}} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + 2$$

Blueqatによるアニーリング計算

計算：

```
from blueqat import opt                    # Wildqatのオプションインポート
q = opt.Opt().add([[4,0],[0,-2]])          # QUBO行列（２次元配列）のセット
print(q.run())                             # アニーリング計算の実行と表示
```

```
[0, 1]
```

総当たりの計算結果と同じ

答：$x_1 = 0$ と $x_2 = 1$ の時に最小値（最適解）となる。

QUBO によるアニーリング計算

QUBO: Quadratic Unconstrained Binary Optimization
（2次制約なし2値最適化）

QUBO化可能なハミルトニアン式の条件：

- バイナリ変数：変数が取る値は0または1のみ
- 2次式：変数の最高次数が2である多項式

※ 多項式：「＋」または「－」の記号によって2つ以上の項を結びつけた式。

○ 可能	$H = A^2 + 2AB + B^2 + 1$	$H = A + 2AB - 4BC + C - 2$
× 不可	$H = A^3 + 2A^2B + B + 1$	$H = 2A^4 - 3A^2 + B^3 - 4$

- 2体問題：1つの項が2変数間の関係まで

※ ただし多体問題を制約により2体問題に変換できれば計算可能となる（後述）。

○ 可能	$H = A^2 + 2AB + B + 1$	$H = A + 2AB - 4CD + 3BE$
× 不可	$H = A^2 - ABC + BC + 1$	$H = BCE - ABCD - 4$

ハミルトニアン式とQUBOの一般化

２体問題の一般式として以下が成り立つ。

$$H = \sum_{i,j} Q_{ij} x_i x_j$$

以下変換により（上）三角行列化できる。

$$Q'_{ij} = \begin{cases} Q_{ij} + Q_{ji} & (i < j) \\ Q_{ij} & (i = j) \\ 0 & (i > j) \end{cases}$$

上三角行列化されたQUBOを一般化する。

$$H = \sum_{i<j} Q_{ij} x_i x_j + \sum_{i} Q_{ii} x_i$$

QUBO の上三角行列化

$$H = \sum_{i,j} Q_{ij} x_i x_j$$

4×4の場合

$Q_{00}x_0x_0$	$Q_{01}x_0x_1$	$Q_{02}x_0x_2$	$Q_{03}x_0x_3$
$Q_{10}x_1x_0$	$Q_{11}x_1x_1$	$Q_{12}x_1x_2$	$Q_{13}x_1x_3$
$Q_{20}x_2x_0$	$Q_{21}x_2x_1$	$Q_{22}x_2x_2$	$Q_{23}x_2x_3$
$Q_{30}x_3x_0$	$Q_{31}x_3x_1$	$Q_{32}x_3x_2$	$Q_{33}x_3x_3$

$Q'_{ij} = Q_{ij} + Q_{ji}$　　$x_i = x_i^2$

$$H = \sum_{i<j} Q'_{ij} x_i x_j + \sum_{i} Q_{ii} x_i^2$$

変換

$$Q'_{ij} = \begin{cases} Q_{ij} + Q_{ji} & (i < j) \\ Q_{ij} & (i = j) \\ 0 & (i > j) \end{cases}$$

$x_i x_j = x_j x_i$

$Q_{00}x_0^2$	$(Q_{01}+Q_{10})x_0x_1$	$(Q_{02}+Q_{20})x_0x_2$	$(Q_{03}+Q_{30})x_0x_3$
0	$Q_{11}x_1^2$	$(Q_{12}+Q_{21})x_1x_2$	$(Q_{13}+Q_{31})x_1x_3$
0	0	$Q_{22}x_2^2$	$(Q_{23}+Q_{32})x_2x_3$
0	0	0	$Q_{33}x_3^2$

上三角行列に一般化されたQUBO

QUBOを使った計算例

例題：以下式をハミルトニアン式とQUBO行列を作成して解け。

$$x_1 + x_2 = 1$$ （※ x_1 と x_2 は 0 or 1 のバイナリ変数）

ハミルトニアン式：条件を満たす時に0（最小値）になるようにする。

$$H = (1 - (x_1 + x_2))^2$$

定式化

$$= -x_1^2 - x_2^2 + 2x_1x_2 + 1$$

変換詳細は次頁

QUBO行列による式：

$$H = \begin{bmatrix} x_1 & x_2 \end{bmatrix} \begin{bmatrix} -1 & 2 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

条件を満たしたハミルトニアン式はQUBO化できる

計算：

```
from blueqat import opt                    # Wildqatのオプション
q = opt.Opt().add([[-1,2],[0,-1]])         # QUBOのセット
print(q.run(shots=8))                      # アニーリング計算を8回実行
```

```
[[1, 0], [0, 1], [0, 1], [1, 0], [1, 0], [0, 1], [1, 0], [0, 1]]
```

(x_1, x_2) が $(0,1)$ と $(1,0)$ の確率50%で発生している

参考:前頁ハミルトニアン式の変形

$$H = (1 - (x_1 + x_2))^2$$
$$= 1 - 2(x_1 + x_2) + (x_1 + x_2)^2$$
$$= 1 - 2x_1 - 2x_2 + x_1^2 + x_2^2 + 2x_1x_2$$
$$= 1 - 2x_1^2 - 2x_2^2 + x_1^2 + x_2^2 + 2x_1x_2$$
$$= -x_1^2 - x_2^2 + 2x_1x_2 + 1$$

バイナリ変数なので
1乗＝2乗（$0^2=0$, $1^2=1$）
とできるので、
$2x_1 = 2x_1^2$ / $2x_2 = 2x_2^2$

※ 変数が4つの場合の例:

$$H = (1 - (x_1 + x_2 + x_3 + x_4))^2$$
$$= 1 - 2(x_1 + x_2 + x_3 + x_4) + (x_1 + x_2 + x_3 + x_4)^2$$
$$= 1 - 2x_1 - 2x_2 - 2x_3 - 2x_4 + x_1^2 + x_2^2 + x_3^2 + x_4^2$$
$$+2x_1x_2 + 2x_1x_3 + 2x_1x_4 + 2x_2x_3 + 2x_2x_4 + 2x_3x_4 + 1$$
$$= 1 - 2x_1^2 - 2x_2^2 - 2x_3^2 - 2x_4^2 + x_1^2 + x_2^2 + x_3^2 + x_4^2$$
$$+2x_1x_2 + 2x_1x_3 + 2x_1x_4 + 2x_2x_3 + 2x_2x_4 + 2x_3x_4 + 1$$
$$= -x_1^2 - x_2^2 - x_3^2 - x_4^2$$
$$+2x_1x_2 + 2x_1x_3 + 2x_1x_4 + 2x_2x_3 + 2x_2x_4 + 2x_3x_4 + 1$$

参考:前頁ハミルトニアン式のQUBO

$$H = (1 - (x_1 + x_2))^2$$

$$\mathrm{QUBO} = \begin{pmatrix} -1 & 2 \\ 0 & -1 \end{pmatrix}$$

$$H = (1 - (x_1 + x_2 + x_3))^2$$

$$\mathrm{QUBO} = \begin{pmatrix} -1 & 2 & 2 \\ 0 & -1 & 2 \\ 0 & 0 & -1 \end{pmatrix}$$

$$H = (1 - (x_1 + x_2 + x_3 + x_4))^2$$

$$\mathrm{QUBO} = \begin{pmatrix} -1 & 2 & 2 & 2 \\ 0 & -1 & 2 & 2 \\ 0 & 0 & -1 & 2 \\ 0 & 0 & 0 & -1 \end{pmatrix}$$

$x_1 x_2 x_3 x_4$
のうち1つだけ
1になるQUBO

ハミルトニアンとQUBOのまとめ

- ハミルトニアンは全エネルギーを示す。
- ハミルトニアン式からQUBO行列を得ることができる。
  変数は0か1のバイナリ変数となる。

  実はバイナリ変数なので1乗＝2乗（$0^2=0$, $1^2=1$）とできる。

  例: $\boldsymbol{H = 5x^2 - 2x + 2}$

  $\boldsymbol{= 5x^2 - 2x^2 + 2 = 3x^2 + 2}$

重要!

- QUBOモデルは以下の式に一般化できる。

$$H = \sum_{i<j} Q_{ij} x_i x_j + \sum_i Q_{ii} x_i$$

内部でイジングモデルへの変換が必要（後述）

- QUBO行列で最小値状態をアニーリング計算可能。

Ocean SDKを使う（D-Wave）

環境：**Anaconda3**（**Python3.7**）

以下より環境に合わせてダウンロードとインストール

https://www.anaconda.com/distribution/

ライブラリ：**D-Wave Ocean**（オーシャン）

Windows版：Anaconda Prompt

MacOS版：ターミナル

インストール

```
pip install dwave-ocean-sdk
```

アンインストール

```
pip uninstall dwave-ocean-sdk
```

バージョン確認

```
pip show dwave-ocean-sdk
```

※ Oceanのバージョン：
本資料のソースは
Version: 1.4.0
環境にて確認しています。

Ocean：QUBOで最適化問題を解く

例題：以下式をハミルトニアン式とQUBO行列を作成して解け。

$$x_1 + x_2 = 1 \qquad H = \begin{bmatrix} x_1 & x_2 \end{bmatrix} \begin{bmatrix} -1 & 2 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + 1$$

オフセット値

Oceanのdimod（SAシミュレーテッドアニーリング）による計算：

```
from dimod import *                                          # dimodインポート
Q = {(0, 0):-1, (0, 1):2, (1, 1):-1}                         # QUBO行列(dict)
b = BinaryQuadraticModel.from_qubo(Q, 1.0)                   # QUBO設定(オフセット値は1)
r = SimulatedAnnealingSampler().sample(b, num_reads=8)       # SAを8回実行
print(r)                                                     # 結果表示
```

省略時：10回

```
   0  1 energy num_oc.
0  0  1    0.0       1
2  1  0    0.0       1
3  1  0    0.0       1
4  0  1    0.0       1
5  0  1    0.0       1
6  0  1    0.0       1
7  0  1    0.0       1
1  1  1    1.0       1
['BINARY', 8 rows, 8 samples, 2 variables]
```

エネルギーの低い順に出力される
$E = 0.0$ の時は $(0,1)$ と $(1,0)$ が
ほぼ確率50%で発生している
1回目の $(1,1)$ 等では $E = 1.0$

3-2：イジングモデル

イジングモデルとは2つの状態を持つ格子点の近接点同士の相互作用を考慮する格子モデル。

アニーリング計算はイジングモデルにより最小のハミルトニアン状態を得る計算方式である。

イジング(Ising)モデル

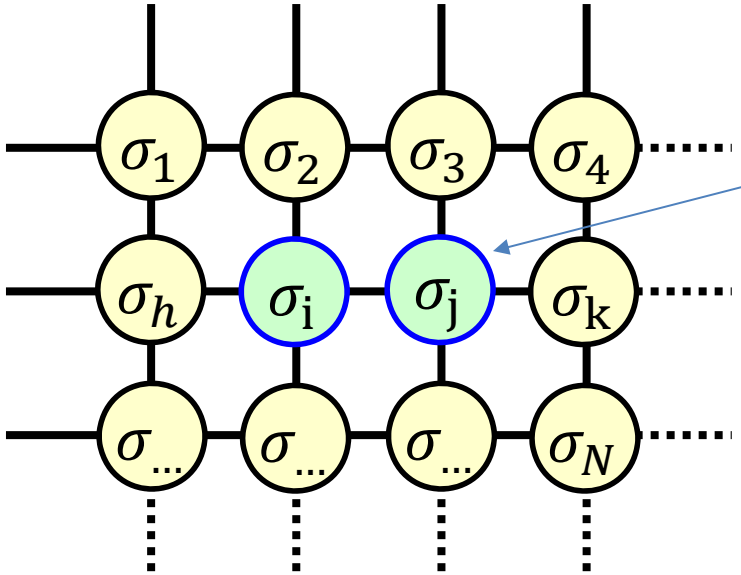
磁性体ではイジング変数はイジングスピンとなる。

イジング変数 σ：+1 または −1 のどちらかの値を取る。
イジングモデル：複数のイジング変数で構成するモデル。

$$\boldsymbol{\sigma_i = \pm 1 \quad (i = 1, 2, 3, \ldots, N)}$$

σ：シグマ

格子上にイジング変数を配置した2次元イジングモデル：



各 σ_i の持つエネルギーは $-h_i\sigma_i$ である。

左図において隣あった i と j は
最接近格子点である。

格子点間には相互作用がある2次元モデル。
I と j は積 $\sigma_i\sigma_j$ が相互作用を示す。
相互作用エネルギーは $-J_{ij}\sigma_i\sigma_j$ となる。

※ なお h_i と J_{ij} はそれぞれ定数（パラメータ）。

イジングモデルの次元

1次元イジングモデル：隣合った間の相互作用

$$H = -\sum_i J_i \sigma_i \sigma_{i+1} - \sum_i h_i \sigma_i$$

2次元イジングモデル：2体間の相互作用

$$H = -\sum_{i<j} J_{ij} \sigma_i \sigma_j - \sum_i h_i \sigma_i$$

アニーリング計算では任意の2体間の相互作用を扱う**2次元イジングモデル**となる。

※ 3体間以上の問題には適用できない（QUBOモデルと同じ）。

2次元イジングモデルのハミルトニアン

イジング変数 $\sigma_i = \pm 1$ （$i = 1,2,3,\ldots,N$）の時、イジングモデルのハミルトニアン（全エネルギー）は、以下の式で求められる。

$$H = -\sum_{i<j} J_{ij}\sigma_i\sigma_j - \sum_i h_i\sigma_i$$

最小化と最大化は方向（符号）が異なるだけなので以下と書ける。

全エネルギー

$$\boldsymbol{H = \sum_{i<j} J_{ij}\sigma_i\sigma_j + \sum_i h_i\sigma_i}$$

QUBOモデルのハミルトン式と同じになった！

自分自身への相互作用は無いので $i \neq j$ である。
また ij と ji は同じなので $i < j$ として重複を避ける。

相互作用エネルギー
（2体間のエネルギー）

局所磁場エネルギー
（1体にかかるエネルギー）

※ J_{ij} と h_i は係数（パラメータ）。

QUBOモデルとイジングモデル

重要!

QUBOモデル: x は 0 or 1 のバイナリ変数

$$H = \sum_{i<j} Q_{ij} x_i x_j + \sum_i Q_{ii} x_i$$

イジングモデル: σ は －1 or ＋1のどちらかの値

$$H = \sum_{i<j} J_{ij} \sigma_i \sigma_j + \sum_i h_i \sigma_i$$

違いは変数が 0/1 か -1/+1 かだけ、以下変換式で変換が可能。

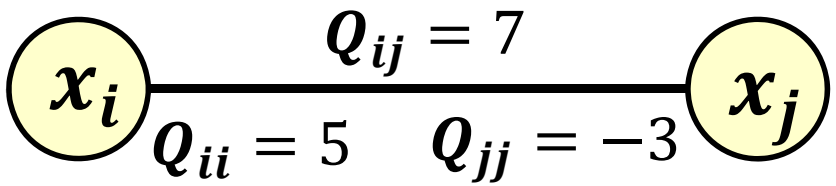
$$x_i = \frac{\sigma_i + 1}{2} \quad (x = 0 \text{ なら } \sigma = -1 \,/\, x = 1 \text{ なら } \sigma = +1)$$

ゆえにQUBOモデルとイジングモデルは相互変換が可能である。

係数（coefficient）

モデル	1次係数 （Linear Coefficient）	2次係数 （Quadratic Coefficient）	変数 （Variable）
QUBO	Q_{ii}	Q_{ij}	x_i (Binary:0/1)
イジング	h_i	J_{ij}	σ_i (Spin:±1)

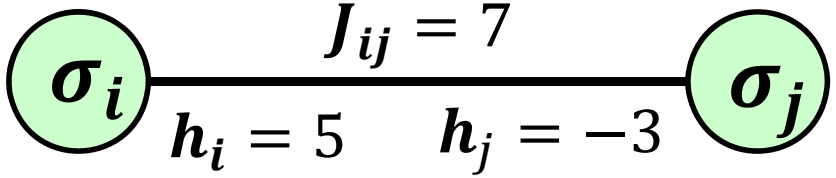
QUBO例：$H = 5x_i + 7x_ix_j - 3x_j$



$Q_{ii} = 5$	$Q_{ij} = 7$
0	$Q_{jj} = -3$

意味は異なる

イジング例：$h = 5\sigma_i - 3\sigma_j$ ，$J = 7\sigma_i\sigma_j$



$h_i = 5$	$J_{ij} = 7$
0	$h_j = -3$

イジングマシン

※ イジングマシンはイジングモデルを解くシステム。

シミュレーテッド アニーリング（SA）	既存コンピュータ上のシミュレータ（遅い）	古典汎用機による計算
量子アニーリングマシン	量子イジング専用機 （実機により実証実験中）	D-Wave NEC 等
量子ゲート型コンピュータ	量子断熱計算 （計算可能だが小規模）	IBM Q Google 等
半導体アニーリングマシン	既存技術で高速化 （実機により実証実験中）	富士通 日立
コヒーレントイジングマシン （光量子コンピュータ）	光子を使った計算 （実装に向けた実験中）	東大・NTT・NII
シミュレーテッド分岐（SB） SBM: Simulated Bifurcation Machine	並列計算可能マシン （GPU利用やFPGA化も可）	東芝デジタルソリューションズ

3-3：グラフ理論

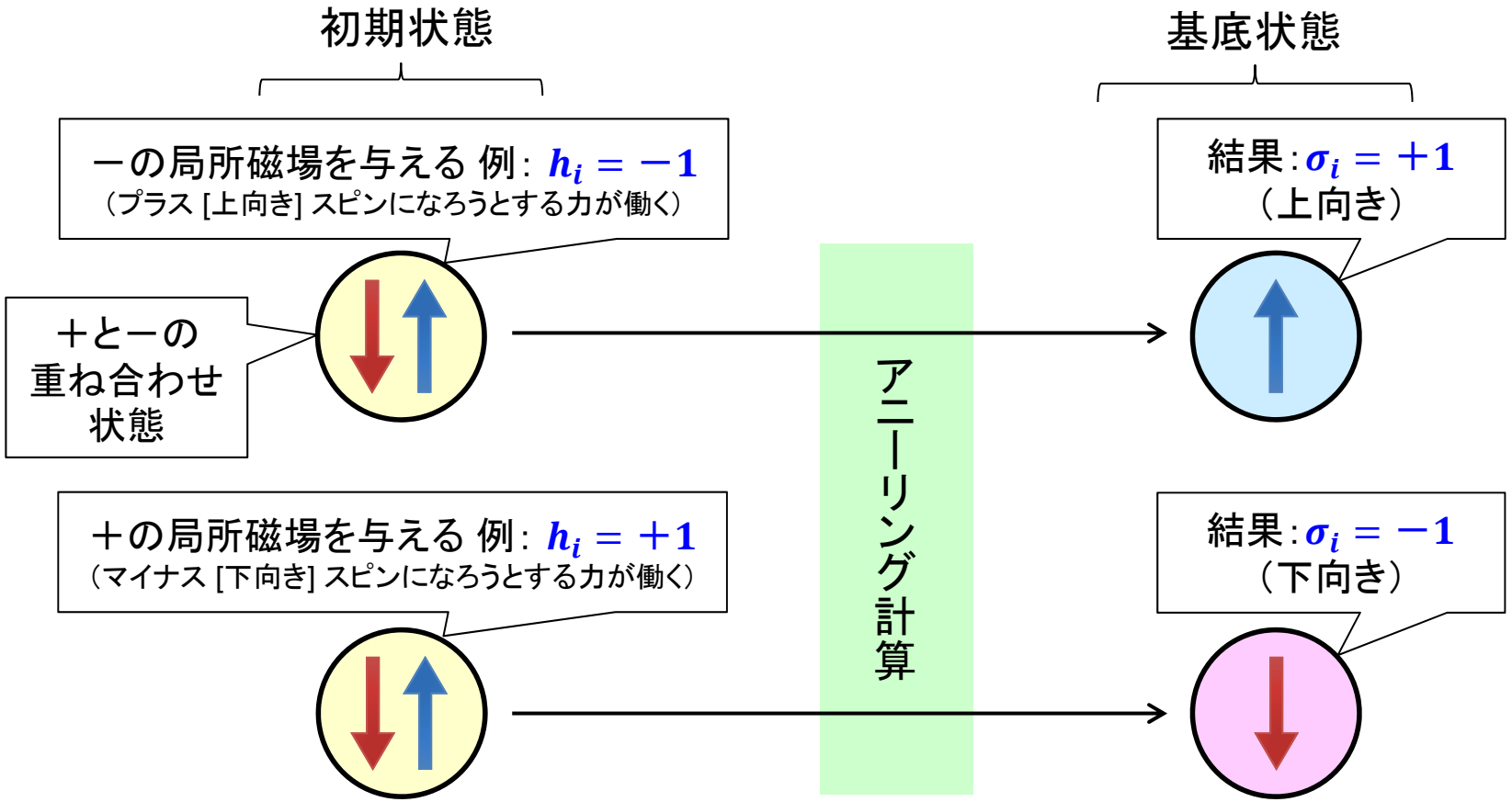
QUBOモデルやイジングモデルを使って問題を解く為に問題をグラフ化する。

グラフは、ノード（スピンや量子ビット）と、エッジ（2点間の相互作用）で構成される。

スピンの局所磁場（1体）

$$H = \sum_{i<j} J_{ij}\sigma_i\sigma_j + \sum_i h_i\sigma_i$$

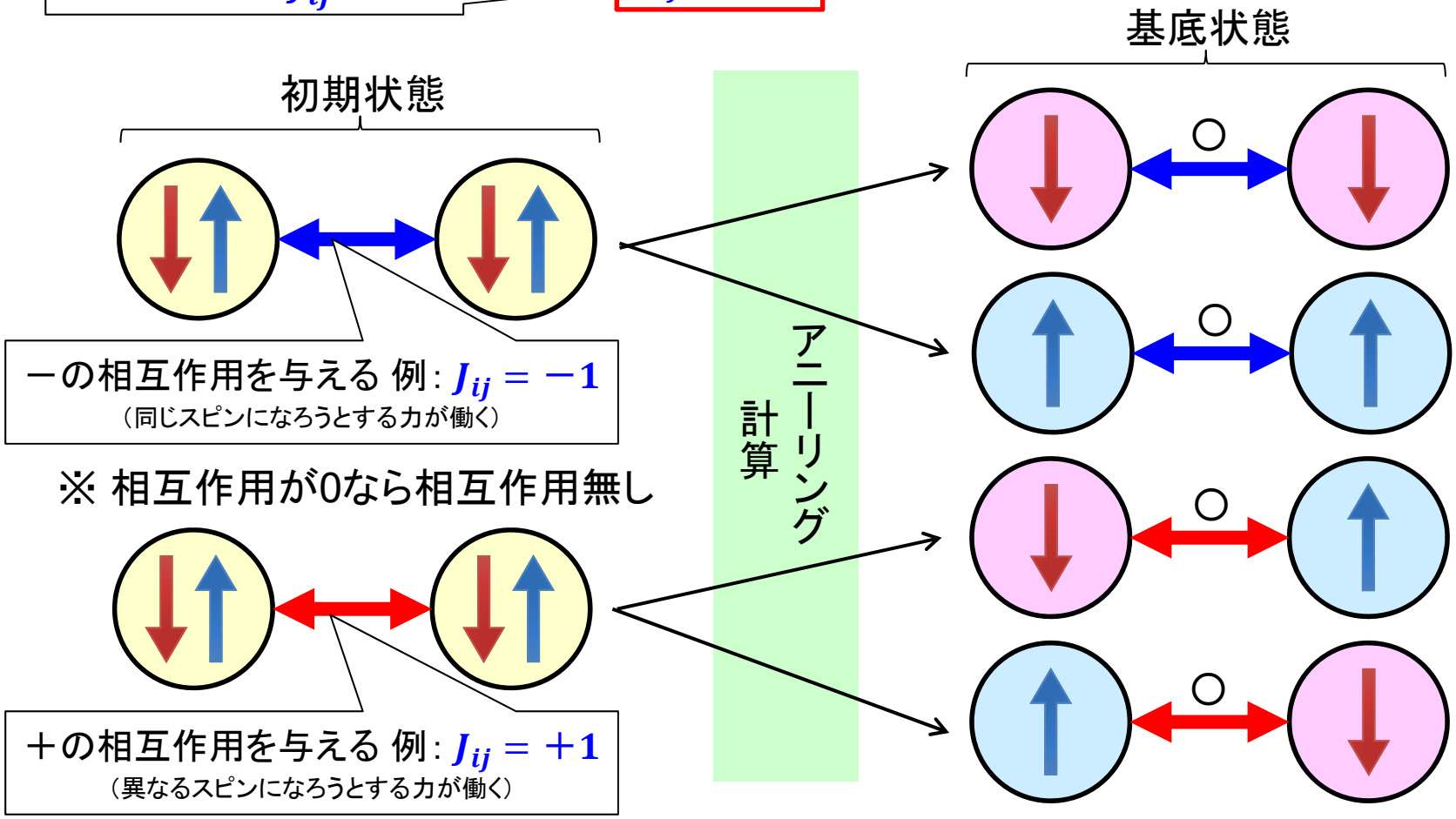
この部分の h_i 係数



2スピンの相互作用（2体間）

$$H = \sum_{i<j} J_{ij}\sigma_i\sigma_j + \sum_i h_i\sigma_i$$

この部分の J_{ij} 係数



イジング(Ising)で2スピン相互作用を解く

例題:2スピン間の相互作用がプラス(反発)の時の結果を得る。

$$h_0 = 0, h_1 = 0 \ / \ J_{01} = 1, (J_{10} = 0)$$

$h_0 = 0$	$J_{01} = 1$
0	$h_1 = 0$

Oceanのdimod(SA シミュレーテッドアニーリング)による計算:

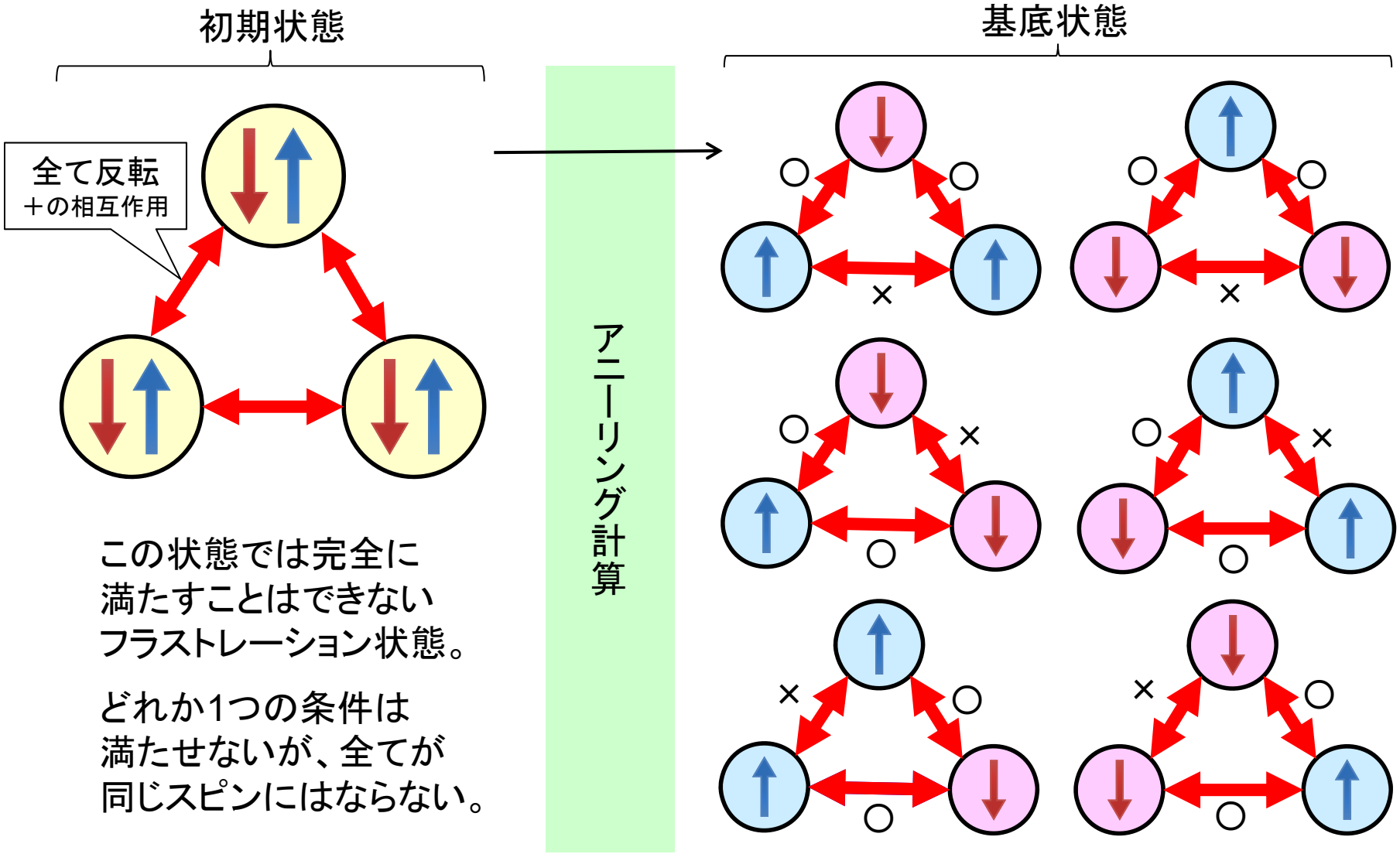
```
from dimod import *                                          # dimodインポート
h = {0:0, 1:0}                                               # h1/h2の設定、空辞書 {} も可
J = {(0, 1): 1}                                              # J12をマイナスに設定
b = BinaryQuadraticModel.from_ising(h, J, 0.0)               # Ising設定(オフセット値は0)
r = SimulatedAnnealingSampler().sample(b, num_reads=8)       # SAを8回実行
print(r)
```

Isingでセットする場合は h と J と offset を与える

```
   0  1 energy num_oc.
0 -1 +1   -1.0       1
1 +1 -1   -1.0       1
2 +1 -1   -1.0       1
3 +1 -1   -1.0       1
4 -1 +1   -1.0       1
5 -1 +1   -1.0       1
6 -1 +1   -1.0       1
7 +1 -1   -1.0       1
['SPIN', 8 rows, 8 samples, 2 variables]
```

−1 と +1 か、+1 と −1 のいずれか同じ値にはならい

3スピン(2体間)のフラストレーション



イジング(Ising)で3スピン相互作用を解く

例題：3スピン間の各相互作用がプラス(反発)の時の結果を得る。

$$h_0 = h_1 = h_2 = 0 \ / \ J_{01} = J_{02} = J_{12} = 1$$

$h_0 = 0$	$J_{01} = 1$	$J_{02} = 1$
0	$h_1 = 0$	$J_{12} = 1$
0	0	$h_2 = 0$

Oceanのdimod（SAシミュレーテッドアニーリング）による計算：

```
from dimod import *                                          # dimodインポート
h = {}                                                       # h1/h2/h3の設定
J = {(0, 1): 1, (0, 2): 1, (1, 2): 1}                        # J12/J13/J23をマイナスに設定
b = BinaryQuadraticModel.from_ising(h, J, 0.0)               # Ising設定(オフセット値は0)
r = SimulatedAnnealingSampler().sample(b, num_reads=8)       # SAを8回実行
print(r)                                                     # 結果表示
```

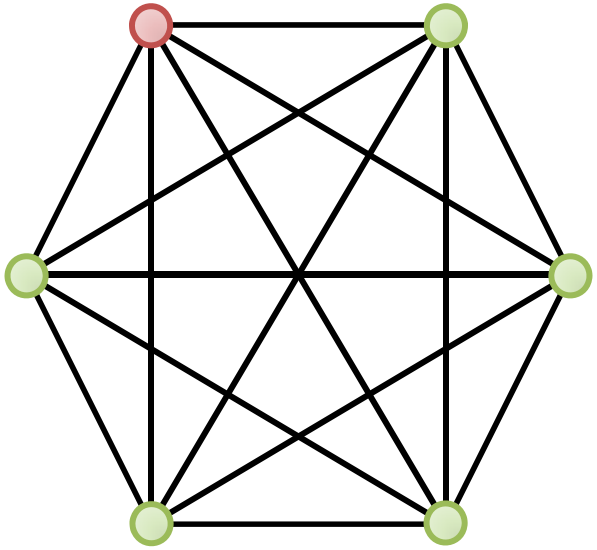
```
   0  1  2 energy num_oc.
0 -1 +1 -1   -1.0       1
1 -1 +1 -1   -1.0       1
2 +1 +1 -1   -1.0       1
3 -1 +1 +1   -1.0       1
4 +1 -1 -1   -1.0       1
5 +1 -1 -1   -1.0       1
6 +1 -1 -1   -1.0       1
7 +1 +1 -1   -1.0       1
['SPIN', 8 rows, 8 samples, 3 variables]
```

実行をする度に結果が異なっているが、
どれか1つが反転する結果となっていて、
全て −1 と +1 の組み合わせは無い

理想的な結合方式:完全グラフ

重要!

6スピンの完全グラフ



11	12	13	14	15	16
21	22	23	24	25	26
31	32	33	34	35	36
41	42	43	44	45	46
51	52	53	54	55	56
61	62	63	64	65	66

完全グラフ:全スピン間を結合する。

左例は6スピンだがこの関係を示す為
6×6の行列(下図)が必要となる。
スピン数が増えると量子コンピュータの
接続数が増える為に難易度が高くなる。

接続方向性は無いので12と21の接続が
同じとなる為に、下三角行列 $Q_{ij}(i > j)$
の部分は使われない。

上三角行列が $Q_{ij}(i < j)$ を示す。

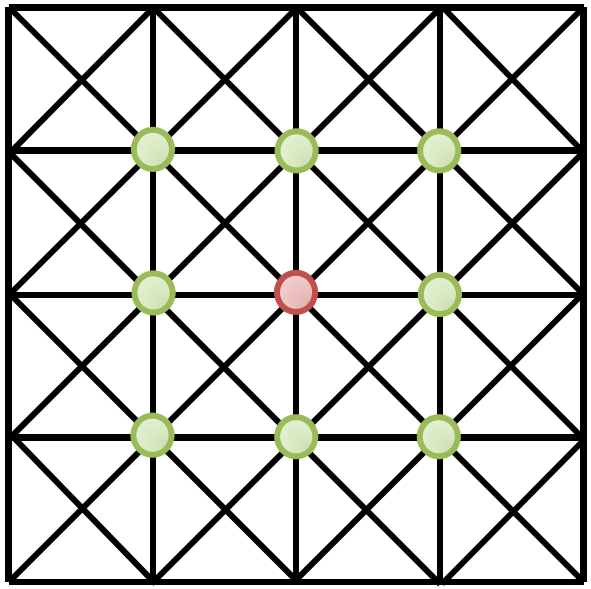
対角 11、22…等が $Q_{ij}(i = j)$ を示す。

※ 完全グラフはQUBO/イジングモデルの利用が容易。

キンググラフとキメラグラフ

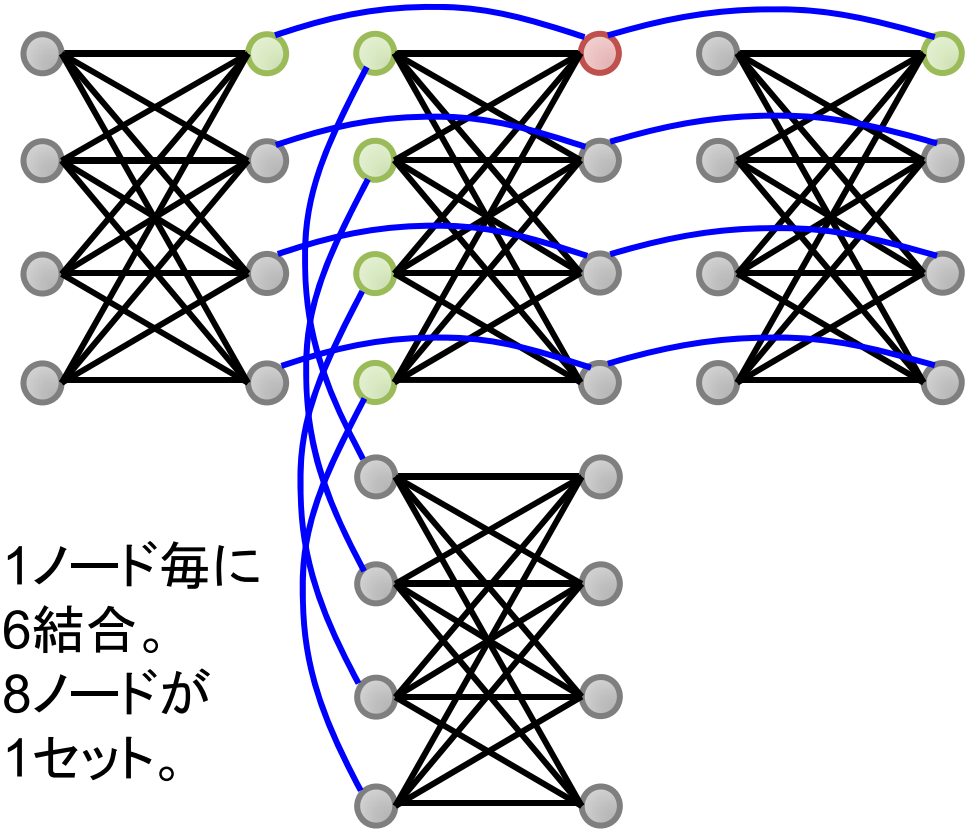
完全グラフの実現が困難 or 問題に依存。

キンググラフ



1ノード毎に8結合。
チェスのキングの動きと同じ
縦横斜めの隣のスピンへの
接続がされている。

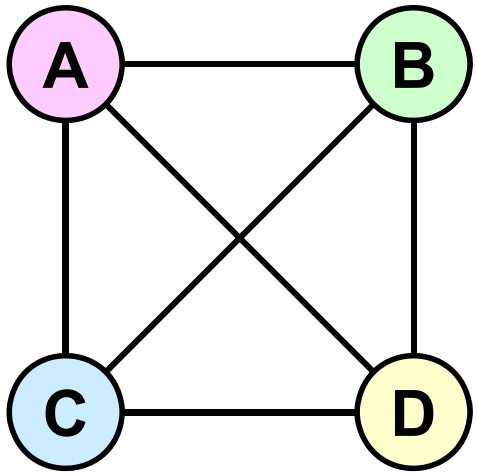
キメラグラフ

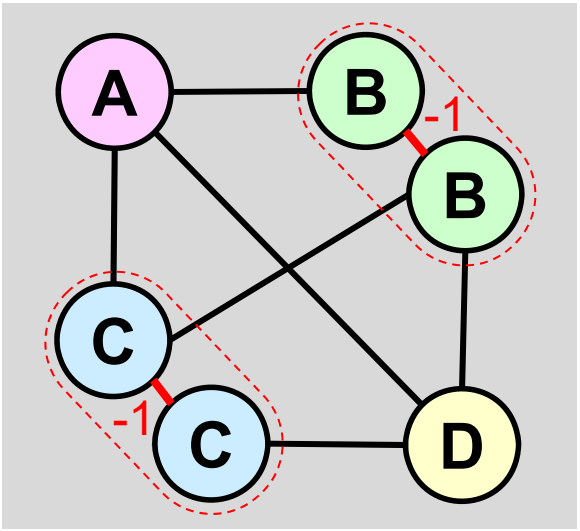


1ノード毎に
6結合。
8ノードが
1セット。

キメラグラフから完全グラフへの変換

4スピンの完全グラフ（理想）

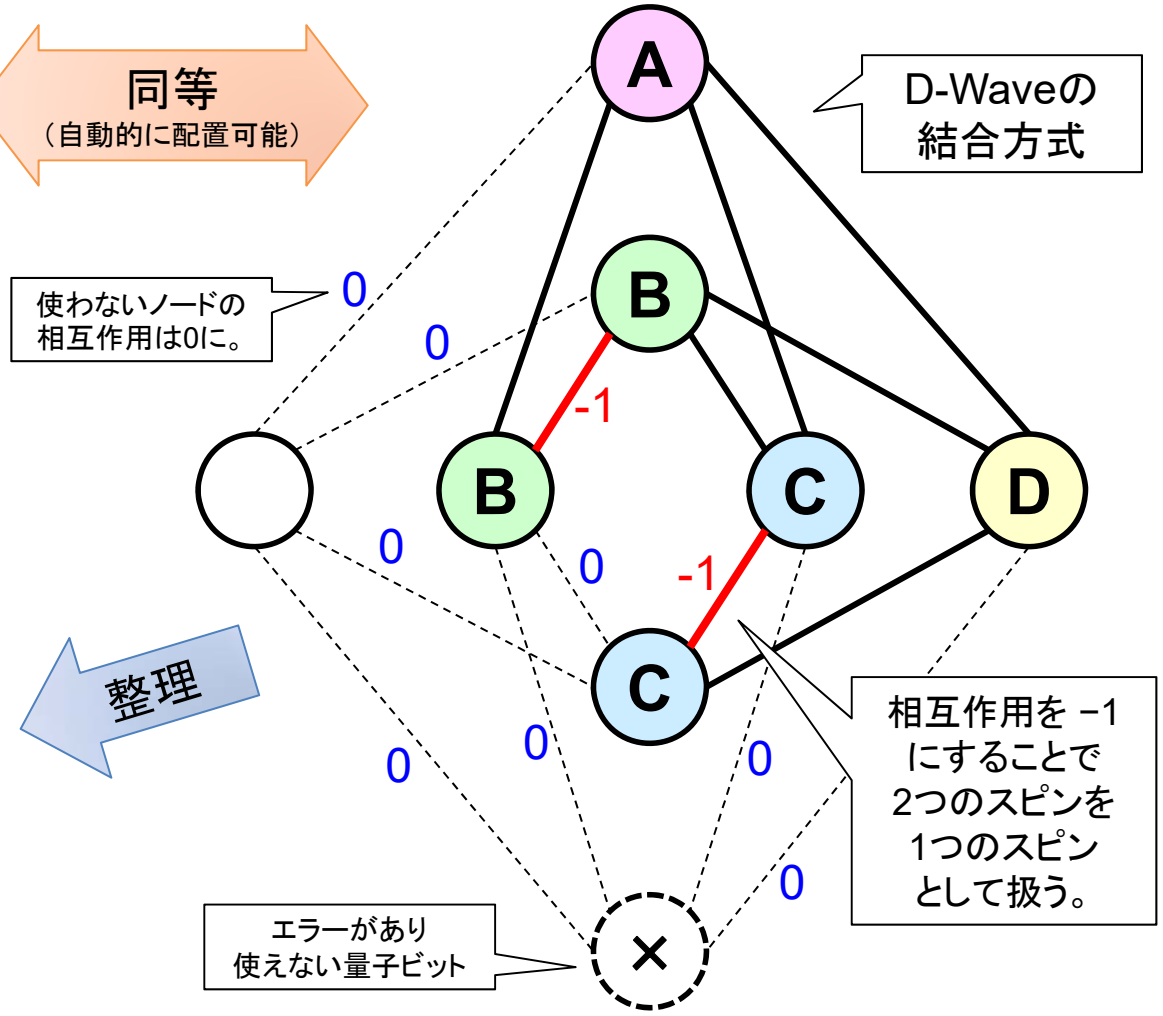




同等
（自動的に配置可能）

整理

4スピンの完全グラフと
同等のキメラグラフ



2019年現在の主なイジングマシン比較

	D-Wave 2000Q	日立 CMOSアニーラ	富士通 デジタルアニーラ	東芝DS SBM
方式	量子	非量子	非量子	非量子
グラフ	キメラグラフ（6結合）	キンググラフ（8結合）	完全グラフ（全結合）	完全グラフ（全結合）
スピン数（ビット数）	2048（ただし欠損あり）	2500 (50×50)	1024	10000 公開中（実証中）
諧調（係数）	4～5 bits（16～32諧調）	8 bits（256諧調）	16 bits（65536諧調）	64 bits ？（ソフトウェア）
補足	次世代 スピン数：5000 ペガサスグラフ（15結合）	現行機：3諧調 正式販売前	次世代 スピン数：8192 諧調：64 bits	FPGA版にて10万スピンの実績もある？ クラウドサービス化予定

3-4: 巡回セールスマン問題

それでは実際に巡回セールスマン問題を解いてみよう。

今回の巡回セールスマン問題と定式化

巡回セールスマン問題:

- 全都市を1度だけ訪問して出発都市に戻る。
- 最短経路(今回は距離)のルートを得る。

今回の前提と定式化:

1. 都市はA/B/C/Dの4つ(距離は別途説明)とする。
2. 4都市×4移動(距離×時間)の16スピンを使う。
3. 都市間接続(移動)は完全グラフ(全結合)とする。
4. QUBOモデル(都市にいる1、いない0)を作成する。
5. 距離(コスト関数)はスピン間の相互作用で指定。
6. 巡回する為の制約関数を用意する。

4都市の場合のスピンモデルの設計

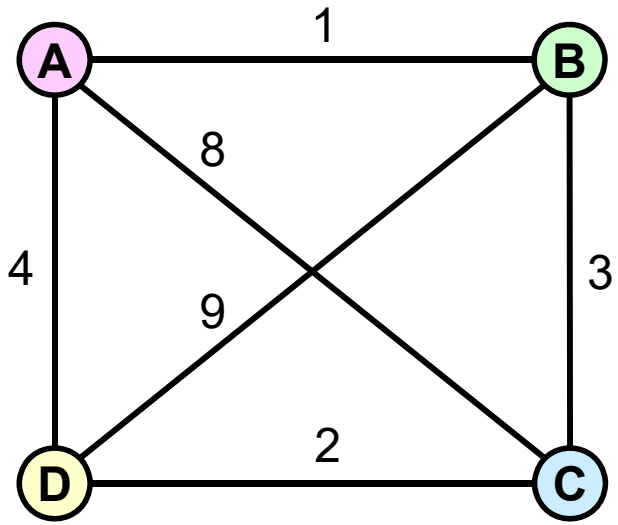
全部で4×4の16スピンを利用	都市A c=1	都市B c=2	都市C c=3	都市D c=4
1番目 t=1	x_1 1	x_2 0	x_3 0	x_4 0
2番目 t=2	x_5 0	x_6 0	x_7 0	x_8 1
3番目 t=3	x_9 0	x_{10} 1	x_{11} 0	x_{12} 0
4番目 t=4	x_{13} 0	x_{14} 0	x_{15} 1	x_{16} 0
1番目に戻る	x_1 1	x_2 0	x_3 0	x_4 0

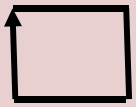
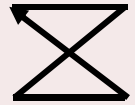
上例の答え：‘1000000101000010’ = ADBC（A)

ab = 28 ac = 36 ad = 81 bc = 50 bd = 67 cd = 63

巡回セールスマン問題1

距離
AB = 1
AC = 8
AD = 4
BC = 3
BD = 9
CD = 2



	**ABCDA** = 1+3+2+4 = **10**
	**ABDCA** = 1+9+2+8 = **20**
	**ACBDA** = 8+3+9+4 = **24**

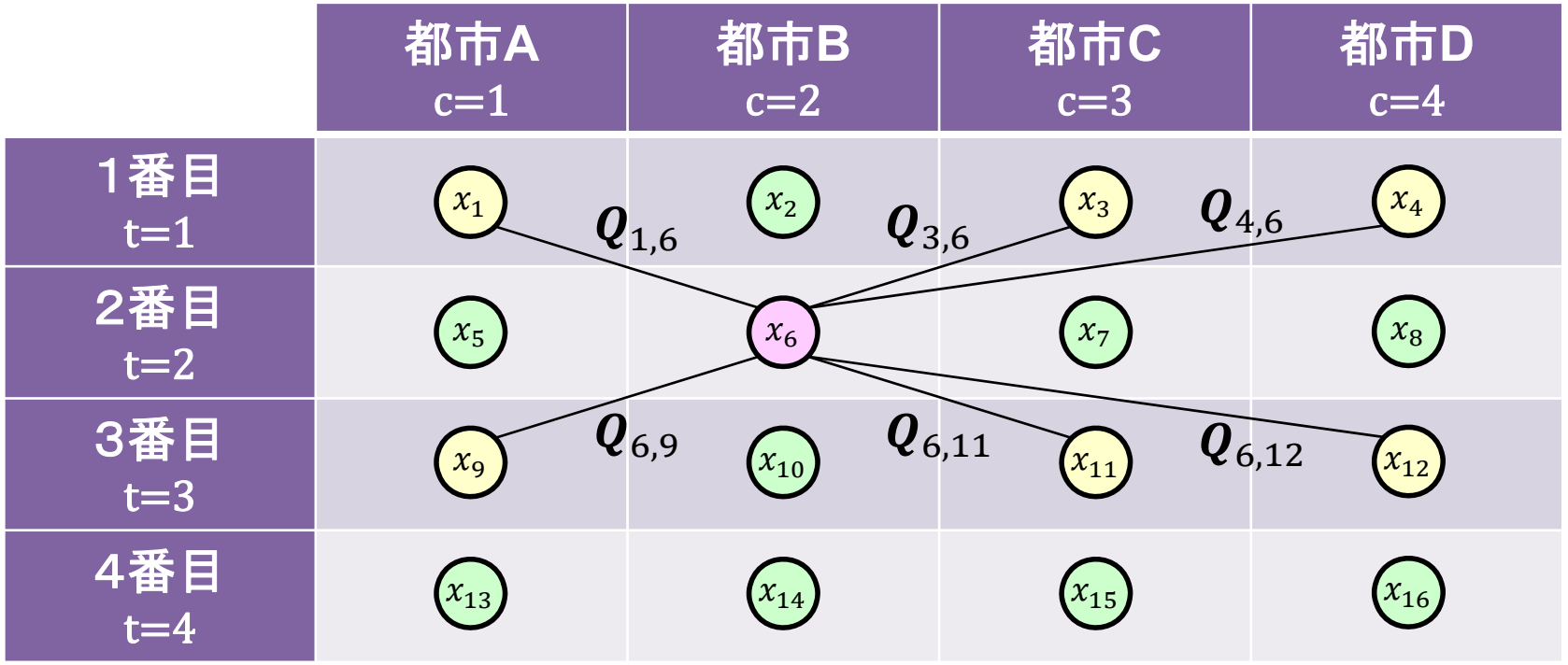
明らかに周辺を順に回るABCDAの順番の距離が短い。
逆順もあるのでADCBAでも良い（距離は同じ）。
まずこの問題をQUBO化してBlueqatで解いてみる。

コスト関数（都市間の距離と総距離）

上下行の自分以外の都市への距離が必要。最上段と最下段は循環しているとする。

総距離 $Hd = \sum_{i,j} Q_{i,j} x_i x_j$

都市 x_i と x_j が1の時（通過時）のみ
その都市間の距離 $Q_{i,j}$ が加算される。



※ $Q_{1,6} = Q_{6,9} = \mathrm{AB} = 1$ / $Q_{3,6} = Q_{6,11} = \mathrm{BC} = 3$ / $Q_{4,6} = Q_{6,12} = \mathrm{CD} = 2$

ij 間の距離QUBO行列（Hd）

$i \setminus j$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1		0	0	0	0	AB:1	AC:8	AD:4	0	0	0	0	0	AB:1	AC:8	AD:4
2			0	0	AB:1	0	BC:3	BD:9	0	0	0	0	AB:1	0	BC:3	BD:9
3				0	AC:8	BC:3	0	CD:2	0	0	0	0	AC:8	BC:3	0	CD:2
4					AD:4	BD:9	CD:2	0	0	0	0	0	AD:4	BD:9	CD:2	0
5						0	0	0	0	AB:1	AC:8	AD:4	0	0	0	0
6							0	0	AB:1	0	BC:3	BD:9	0	0	0	0
7								0	AC:8	BC:3	0	CD:2	0	0	0	0
8									AD:4	BD:9	CD:2	0	0	0	0	0
9										0	0	0	0	AB:1	AC:8	AD:4
10											0	0	AB:1	0	BC:3	BD:9
11												0	AC:8	BC:3	0	CD:2
12													AD:4	BD:9	CD:2	0
13														0	0	0
14															0	0
15																0
16																

距離
AB = 1
AC = 8
AD = 4
BC = 3
BD = 9
CD = 2

	A	B	C	D
1st	x_1	x_2	x_3	x_4
2nd	x_5	x_6	x_7	x_8
3rd	x_9	x_{10}	x_{11}	x_{12}
4th	x_{13}	x_{14}	x_{15}	x_{16}

ij 間の距離QUBO行列（*Hd*）

```
Hd = np.array([
[0,0,0,0,0,1,8,4,0,0,0,0,0,1,8,4],
[0,0,0,0,1,0,3,9,0,0,0,0,1,0,3,9],
[0,0,0,0,8,3,0,2,0,0,0,0,8,3,0,2],
[0,0,0,0,4,9,2,0,0,0,0,0,4,9,2,0],
[0,0,0,0,0,0,0,0,1,8,4,0,0,0,0],
[0,0,0,0,0,0,0,0,1,0,3,9,0,0,0,0],
[0,0,0,0,0,0,0,0,8,3,0,2,0,0,0,0],
[0,0,0,0,0,0,0,0,4,9,2,0,0,0,0,0],
[0,0,0,0,0,0,0,0,0,0,0,0,0,1,8,4],
[0,0,0,0,0,0,0,0,0,0,0,0,1,0,3,9],
[0,0,0,0,0,0,0,0,0,0,0,0,8,3,0,2],
[0,0,0,0,0,0,0,0,0,0,0,0,4,9,2,0],
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],
])
```

1次のスピン列を2次元行列へ変換

都市インデックス cと、時間インデックス tの2つを使いメインインデックス iを変換する。

$$x_i = x_{c,t}$$
$$i = (t-1) * 4 + c$$

都市を巡回する為の
制約を作りやすくする

	都市A c=1	都市B c=2	都市C c=3	都市D c=4
1番目 t=1	$x_1 = x_{1,1}$	$x_2 = x_{2,1}$	$x_3 = x_{3,1}$	$x_4 = x_{4,1}$
2番目 t=2	$x_5 = x_{1,2}$	$x_6 = x_{2,2}$	$x_7 = x_{3,2}$	$x_8 = x_{4,2}$
3番目 t=3	$x_9 = x_{1,3}$	$x_{10} = x_{2,3}$	$x_{11} = x_{3,3}$	$x_{12} = x_{4,3}$
4番目 t=4	$x_{13} = x_{1,4}$	$x_{14} = x_{2,4}$	$x_{15} = x_{3,4}$	$x_{16} = x_{4,4}$

制約関数（ペナルティ関数）

	都市A	都市B	都市C	都市D
1番目	1	0	0	0
2番目	0	0	0	1
3番目	0	1	0	0
4番目	0	0	1	0

巡回例

全都市を巡回する条件（2次元行列）:

1. 横軸（X軸）各行ではどれか1つだけが1になる。

条件: $\sum_c x_{c,t} = 1 \rightarrow Hc = \sum_t \left(1 - \sum_c x_{c,t}\right)^2$

- 同時間では1都市だけにいることができる
- 各行の総和が1になる（1つだけが1）

2. 縦軸（Y軸）各列ではどれか1つだけが1になる。

条件: $\sum_t x_{c,t} = 1 \rightarrow Ht = \sum_c \left(1 - \sum_t x_{c,t}\right)^2$

- 毎回異なる都市を訪問する
- 各列の総和が1になる（1つだけが1）

横軸の制約ハミルトニアン式（Hc）

各行に関する制約ハミルトニアン式は以下となる（既出）。

$$x_{1,t} + x_{2,t} + x_{3,t} + x_{4,t} = 1$$

ハミルトニアン式の変形：

$$\begin{aligned} H &= \left(1 - \left(x_{1,t} + x_{2,t} + x_{3,t} + x_{4,t} \right) \right)^2 \\ &= -{x_{1,t}}^2 - {x_{2,t}}^2 - {x_{3,t}}^2 - {x_{4,t}}^2 \\ &\quad +2x_{1,t}x_{2,t} + 2x_{1,t}x_{3,t} + 2x_{1,t}x_{4,t} \\ &\quad +2x_{2,t}x_{3,t} + 2x_{2,t}x_{4,t} + 2x_{3,t}x_{4,t} + 1 \end{aligned}$$

4×4のQUBO行列：

$$\begin{pmatrix} -1 & 2 & 2 & 2 \\ 0 & -1 & 2 & 2 \\ 0 & 0 & -1 & 2 \\ 0 & 0 & 0 & -1 \end{pmatrix}$$

全横軸制約のQUBO行列（Hc）

$i \setminus j$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	−1	2	2	2												
2		−1	2	2												
3			−1	2												
4				−1												
5					−1	2	2	2								
6						−1	2	2								
7							−1	2								
8								−1								
9									−1	2	2	2				
10										−1	2	2				
11											−1	2				
12												−1				
13													−1	2	2	2
14														−1	2	2
15															−1	2
16																−1

同じ時刻に1都市のみに存在する為の制約

縦軸の制約ハミルトニアン式（Ht）

各列に関する制約ハミルトニアン式は以下となる。

$$x_{c,1} + x_{c,2} + x_{c,3} + x_{c,4} = 1$$

ハミルトニアン式の変形：

$$\begin{aligned} H &= \left(1 - \left(x_{c,1} + x_{c,2} + x_{c,3} + x_{c,4} \right) \right)^2 \\ &= -{x_{c,1}}^2 - {x_{c,2}}^2 - {x_{c,3}}^2 - {x_{c,4}}^2 \\ &\quad + 2x_{c,1}x_{c,2} + 2x_{c,1}x_{c,3} + 2x_{c,1}x_{c,4} \\ &\quad + 2x_{c,2}x_{c,3} + 2x_{c,2}x_{c,4} + 2x_{c,3}x_{c,4} + 1 \end{aligned}$$

4×4のQUBO行列：

$$\begin{pmatrix} -1 & 2 & 2 & 2 \\ 0 & -1 & 2 & 2 \\ 0 & 0 & -1 & 2 \\ 0 & 0 & 0 & -1 \end{pmatrix}$$

全縦軸制約のQUBO行列（Ht）

$i \setminus j$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	−1				2				2				2			
2		−1				2				2				2		
3			−1				2				2				2	
4				−1				2				2				2
5					−1				2				2			
6						−1				2				2		
7							−1				2				2	
8								−1				2				2
9									−1				2			
10										−1				2		
11											−1				2	
12												−1				2
13													−1			
14														−1		
15															−1	
16																−1

全都市を1回ずつ
訪問する為の制約

全制約のQUBO行列（$Hp = Hc + Ht$）

$i \setminus j$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	−2	2	2	2	2				2				2			
2		−2	2	2		2				2				2		
3			−2	2			2				2				2	
4				−2				2				2				2
5					−2	2	2	2	2				2			
6						−2	2	2		2				2		
7							−2	2			2				2	
8								−2				2				2
9									−2	2	2	2	2			
10										−2	2	2		2		
11											−2	2			2	
12												−2				2
13													−2	2	2	2
14														−2	2	2
15															−2	2
16																−2

都市を巡回する為の
制約QUBO

全制約のQUBO行列（$Hp = Hc + Ht$）

```
Hp = np.array([
[-2, 2, 2, 2, 2, 0, 0, 0, 2, 0, 0, 0, 2, 0, 0, 0],
[0, -2, 2, 2, 0, 2, 0, 0, 0, 2, 0, 0, 0, 2, 0, 0],
[0, 0, -2, 2, 0, 0, 2, 0, 0, 0, 2, 0, 0, 0, 2, 0],
[0, 0, 0, -2, 0, 0, 0, 2, 0, 0, 0, 2, 0, 0, 0, 2],
[0, 0, 0, 0, -2, 2, 2, 2, 2, 0, 0, 0, 2, 0, 0, 0],
[0, 0, 0, 0, 0, -2, 2, 2, 0, 2, 0, 0, 0, 2, 0, 0],
[0, 0, 0, 0, 0, 0, -2, 2, 0, 0, 2, 0, 0, 0, 2, 0],
[0, 0, 0, 0, 0, 0, 0, -2, 0, 0, 0, 2, 0, 0, 0, 2],
[0, 0, 0, 0, 0, 0, 0, 0, -2, 2, 2, 2, 2, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, -2, 2, 2, 0, 2, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, -2, 2, 0, 0, 2, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, -2, 0, 0, 0, 2],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, -2, 2, 2, 2],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, -2, 2, 2],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, -2, 2],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, -2],
])
```

全体のハミルトニアン式（定式化）

ここまでで計算したエネルギー計算のまとめ：

$$H = Hd + k \times Hp \quad (Hp = Hc + Ht)$$

H　：全エネルギー
Hd　：総距離（都市間距離のコストQUBO）
Hp　：総制約（都市を巡回させる為の制約QUBO）
k　：制約に対する重み付けの補正係数（調整用）
Hc　：横軸制約（同時に1都市のみ）
Ht　：縦軸制約（同時刻に1都市のみ）

Hd と Hp の2つのQUBOは導出済みなので計算が可能。

巡回セールスマン問題1の計算

```
# 左右は繋がっている
import numpy as np
from blueqat import opt
Hd = np.array([
[0,0,0,0,0,1,8,4,0,0,0,0,0,1,8,4],
[0,0,0,0,1,0,3,9,0,0,0,0,1,0,3,9],
[0,0,0,0,8,3,0,2,0,0,0,0,8,3,0,2],
[0,0,0,0,4,9,2,0,0,0,0,0,4,9,2,0],
[0,0,0,0,0,0,0,0,0,1,8,4,0,0,0,0],
[0,0,0,0,0,0,0,0,1,0,3,9,0,0,0,0],
[0,0,0,0,0,0,0,0,8,3,0,2,0,0,0,0],
[0,0,0,0,0,0,0,0,4,9,2,0,0,0,0,0],
[0,0,0,0,0,0,0,0,0,0,0,0,1,8,4],
[0,0,0,0,0,0,0,0,0,0,0,0,1,0,3,9],
[0,0,0,0,0,0,0,0,0,0,0,0,8,3,0,2],
[0,0,0,0,0,0,0,0,0,0,0,0,4,9,2,0],
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],
])
```

```
Hp = np.array([
[-2,2,2,2,2,0,0,0,2,0,0,0,2,0,0,0],
[0,-2,2,2,0,2,0,0,0,2,0,0,0,2,0,0],
[0,0,-2,2,0,0,2,0,0,0,2,0,0,0,2,0],
[0,0,0,-2,0,0,0,2,0,0,0,2,0,0,0,2],
[0,0,0,0,-2,2,2,2,2,0,0,0,2,0,0,0],
[0,0,0,0,0,-2,2,2,0,2,0,0,0,2,0,0],
[0,0,0,0,0,0,-2,2,0,0,2,0,0,0,2,0],
[0,0,0,0,0,0,0,-2,0,0,0,2,0,0,0,2],
[0,0,0,0,0,0,0,0,-2,2,2,2,2,0,0,0],
[0,0,0,0,0,0,0,0,0,-2,2,2,0,2,0,0],
[0,0,0,0,0,0,0,0,0,0,-2,2,0,0,2,0],
[0,0,0,0,0,0,0,0,0,0,0,-2,0,0,0,2],
[0,0,0,0,0,0,0,0,0,0,0,0,-2,2,2,2],
[0,0,0,0,0,0,0,0,0,0,0,0,0,-2,2,2],
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,-2,2],
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,-2],
])
k = 4
q = opt.Opt().add(Hd+k*Hp)
opt.counter(q.run(shots=100))
```

巡回セールスマン問題1の計算結果例

```
Counter({'1000000100100100': 11,
         '0100100000010010': 14,
         '0100001000011000': 15,
         '0010010010000001': 14,
         '0001100001000010': 16,
         '0001001001001000': 10,
         '1000010000100001': 8,
         '0010000110000100': 12})
```

```
ADCB = ADCBA
BADC = ADCBA
BCDA = ABCDA
CBAD = ADCBA
DABC = ABCDA
DCBA = ADCBA
ABCD = ABCDA
CDAB = ABCDA
```

全部で8パターンが出力されたが、内容を確認すると開始点が違うだけで、最短経路の ABCDA か、逆順の ADCBA になっている。なお回数は計算毎に異なる。

※ つまり最適解が出力された。

巡回セールスマン問題1の計算(改)

```
# 左右は繋がっている
import numpy as np
from blueqat import opt
Hd = np.array([
[0,0,0,0,0,1,8,4,0,0,0,0,0,1,8,4],
[0,0,0,0,1,0,3,9,0,0,0,0,1,0,3,9],
[0,0,0,0,8,3,0,2,0,0,0,0,8,3,0,2],
[0,0,0,0,4,9,2,0,0,0,0,0,4,9,2,0],
[0,0,0,0,0,0,0,0,0,1,8,4,0,0,0,0],
[0,0,0,0,0,0,0,0,1,0,3,9,0,0,0,0],
[0,0,0,0,0,0,0,0,8,3,0,2,0,0,0,0],
[0,0,0,0,0,0,0,0,4,9,2,0,0,0,0,0],
[0,0,0,0,0,0,0,0,0,0,0,0,1,8,4],
[0,0,0,0,0,0,0,0,0,0,0,0,1,0,3,9],
[0,0,0,0,0,0,0,0,0,0,0,0,8,3,0,2],
[0,0,0,0,0,0,0,0,0,0,0,0,4,9,2,0],
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],
])
```

```
Hp = np.array([
[-8,2,2,2,2,0,0,
[0,-2,2,2,0,2,
[0,0,-2,2,0,0,2,
[0,0,0,-2,0,0,0,
[0,0,0,0,-2,2,2,
[0,0,0,0,0,-2,2,2,0,2,0,0,0,2,0,0],
[0,0,0,0,0,0,-2,2,0,0,2,0,0,0,2,0],
[0,0,0,0,0,0,0,-2,0,0,0,2,0,0,0,2],
[0,0,0,0,0,0,0,0,-2,2,2,2,2,0,0,0],
[0,0,0,0,0,0,0,0,0,-2,2,2,0,2,0,0],
[0,0,0,0,0,0,0,0,0,0,-2,2,0,0,2,0],
[0,0,0,0,0,0,0,0,0,0,0,-2,0,0,0,2],
[0,0,0,0,0,0,0,0,0,0,0,0,-2,2,2,2],
[0,0,0,0,0,0,0,0,0,0,0,0,0,-2,2,2],
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,-2,2],
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,-2],
])
k = 4
q = opt.Opt().add(Hd+k*Hp)
opt.counter(q.run(shots=100))
```

$x_{1,1}$(都市A)の1次係数を大きくして最初に都市Aから開始させる。

巡回セールスマン問題1(改)計算結果

```
Counter({'1000010000100001': 57,
         '1000000100100100': 43})
```

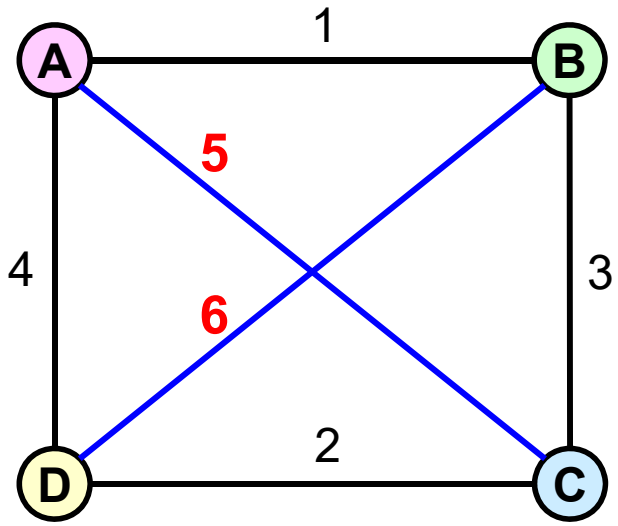
ABCD = ABCDA
ADCB = ADCBA

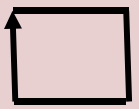
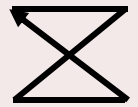
都市Aを開始点として、ABCDA とその逆順 ADCBA の2パターンのみが出力された。確率はほぼ50%ずつとなっている。

なお補正係数 k に今回の計算では 4 をセットした。
2つのQUBO、Hd と Hp のバランスが取れるようにする必要があるが、この辺りは結果を見て調整が必要。
今回は Hd の各値が 0～9 で、Hp の各値が -2～2 であったので、Hp に補正係数 4 倍を適用したところ良い感じで計算結果が得られた。

巡回セールスマン問題2

距離
AB = 1
AC = 5
AD = 4
BC = 3
BD = 6
CD = 2



	**ABCDA** = 1+3+2+4 = **10**
	**ABDCA** = 1+6+2+5 = **14**
	**ACBDA** = 5+3+6+4 = **18**

問題1に比較してAC間とBD間の距離を縮めた。

AC間：8→5 , BD間：9→6

周辺を順に回るABCDAの順番の距離が最短だが、
2番目のABDCAとの差は問題1より小さくなった。

ABCDA：10→10 , ABDCA：20→14 , ACBDA：24→18

巡回セールスマン問題2の計算

```
# 左右は繋がっている
import numpy as np
from blueqat import opt
Hd = np.array([
[0,0,0,0,0,1,5,4,0,0,0,0,0,1,5,4],
[0,0,0,0,1,0,3,6,0,0,0,0,1,0,3,6],
[0,0,0,0,5,3,0,2,0,0,0,0,5,3,0,2],
[0,0,0,0,4,6,2,0,0,0,0,0,4,6,2,0],
[0,0,0,0,0,0,0,0,0,1,5,4,0,0,0,0],
[0,0,0,0,0,0,0,0,1,0,3,6,0,0,0,0],
[0,0,0,0,0,0,0,0,5,3,0,2,0,0,0,0],
[0,0,0,0,0,0,0,0,4,6,2,0,0,0,0,0],
[0,0,0,0,0,0,0,0,0,0,0,0,1,5,4],
[0,0,0,0,0,0,0,0,0,0,0,0,1,0,3,6],
[0,0,0,0,0,0,0,0,0,0,0,0,5,3,0,2],
[0,0,0,0,0,0,0,0,0,0,0,0,4,6,2,0],
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],
])
```

```
Hp = np.array([
[-8,2,2,2,2,0,0,0,2,0,0,0,2,0,0,0],
[0,-2,2,2,0,2,0,0,0,2,0,0,0,2,0,0],
[0,0,-2,2,0,0,2,0,0,0,2,0,0,0,2,0],
[0,0,0,-2,0,0,0,2,0,0,0,2,0,0,0,2],
[0,0,0,0,-2,2,2,2,2,0,0,0,2,0,0,0],
[0,0,0,0,0,-2,2,2,0,2,0,0,0,2,0,0],
[0,0,0,0,0,0,-2,2,0,0,2,0,0,0,2,0],
[0,0,0,0,0,0,0,-2,0,0,0,2,0,0,0,2],
[0,0,0,0,0,0,0,0,-2,2,2,2,2,0,0,0],
[0,0,0,0,0,0,0,0,0,-2,2,2,0,2,0,0],
[0,0,0,0,0,0,0,0,0,0,-2,2,0,0,2,0],
[0,0,0,0,0,0,0,0,0,0,0,-2,0,0,0,2],
[0,0,0,0,0,0,0,0,0,0,0,0,-2,2,2,2],
[0,0,0,0,0,0,0,0,0,0,0,0,0,-2,2,2],
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,-2,2],
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,-2],
])
k = 4
q = opt.Opt().add(Hd+k*Hp)
opt.counter(q.run(shots=100))
```

巡回セールスマン問題2の計算結果

```
Counter({'1000000100100100': 35,
         '1000010000100001': 47,
         '1000001000000100': 5,
         '1000001000010100': 6,
         '1000010000000010': 6,
         '1000010000010010': 1})
```

ADCB = ABCDA
ABCD = ABCDA
AC-B = [計算ミス]
ACDB = ABDCA
AB-C = [計算ミス]
ABDC = ABDCA

巡回しないケース(計算ミス)が発生してしまった。
補正係数を 4 から 5 に変更して再計算してみる。
つまり巡回させる為の制約を強くしてみる。

```
k = 5
q = opt.Opt().add(Hd+k*Hp)
opt.counter(q.run(shots=100))
```

巡回セールスマン問題2(改)計算結果

```
Counter({'1000000100100100': 34,
         '1000001000010100': 12,
         '1000010000100001': 41,
         '1000010000010010': 11,
         '1000001001000001': 2})
```

```
ADCB = ABCDA
ACDB = ABDCA
ABCD = ABCDA
ABDC = ABDCA
ACBD = ACBDA
```

補正係数を 4 から 5 に変更したことで正しく巡回するようになった。

結果を見ると、最短距離(最適解)の ABCDA が75%、2番目に短い距離(局所解？)の ABDCA が23%で、最も悪い(長い距離)の ACBDA は2%となっている。

最適解だけではなく局所解を求めることもできそうだ。

3-5: 多体相互作用

現在のイジングマシンでは通常2体の相互作用のみ対応しています。3体以上の相互作用を必要とする計算では2体表現に変換する必要があります。

3体問題の2体問題への変換例

例題：以下ハミルトニアン式が最小値を取る $x_1\ x_2\ x_3$を求めよ。

3変数が影響している　QUBO化できない

$$H = x_1 - x_1x_2x_3$$

※ $x_1\ x_2\ x_3$は 0 or 1 のバイナリ変数

x_2x_3を x_4 として式を置き換える（2体問題にする）。

$$H = x_1 - x_1x_4 \quad (x_4 = x_2x_3)$$

$x_4 = x_2x_3$ が成り立つ制約ハミルトニアン H' を
ハミルトニアン H に追加することで2体問題に変換。

$$H = x_1 - x_1x_4 + H'$$

前頁制約 H' のハミルトニアン式

$x_4 = x_2x_3$ が成り立つ係数 $c_1 \sim c_6$ を求める。

$$H' = c_1x_2 + c_2x_3 + c_3x_4 + c_4x_2x_3 + c_5x_3x_4 + c_6x_2x_4$$

真理値表を作って上の式から条件を求める。

$x_2x_3 = x_4$ の時

x_2	x_3	x_4	H'
0	0	0	0
0	1	0	0
1	0	0	0
1	1	1	0

$$c_2 = 0$$
$$c_1 = 0$$
$$\sum_i c_i = 0$$

$x_2x_3 \neq x_4$ の時

x_2	x_3	x_4	H'
0	0	1	d_1
0	1	1	d_2
1	0	1	d_3
1	1	0	d_4

$$c_3 = d_1$$
$$c_3 + c_5 = d_2$$
$$c_3 + c_6 = d_3$$
$$c_4 = d_4$$

係数 c_1～c_6 を求める

$$c_1 = 0$$
$$c_2 = 0$$
$$c_3 = d_1$$
$$c_3 + c_5 = d_2$$
$$c_3 + c_6 = d_3$$
$$c_4 = d_4$$

※ d_iはゼロより大きい

$$d_1 = 3$$
$$d_2 = 1$$
$$d_3 = 1$$
$$d_4 = 1$$

$$c_1 = 0$$
$$c_2 = 0$$
$$c_3 = 3$$
$$c_4 = 1$$
$$c_5 = -2$$
$$c_6 = -2$$

c_1～c_6 を代入して制約 H' のハミルトニアン式を得る。

$$H' = 3x_4 + x_2x_3 - 2x_3x_4 - 2x_2x_4$$

3体問題を2体問題へ変換する制約 H'

$$H = x_1 - x_1x_4 + H'$$
$$H' = 3x_4 + x_2x_3 - 2x_3x_4 - 2x_2x_4$$

より2体問題の以下ハミルトニアン式が得られた。

※ k は調整用の補正係数。

$$H = x_1 - x_1x_4 + k * (3x_4 + x_2x_3 - 2x_3x_4 - 2x_2x_4)$$

コストQUBO　　制約QUBO

$$H = \begin{pmatrix} 1 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} + k \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -2 \\ 0 & 0 & 0 & -2 \\ 0 & 0 & 0 & 3 \end{pmatrix}$$

3体問題を2体問題へ変換して計算

```
import numpy as np
from blueqat import opt
H1 = np.array([
[1,0,0,-1],
[0,0,0, 0],
[0,0,0, 0],
[0,0,0, 0],
])
H2 = np.array([
[0,0,0, 0],
[0,0,1,-2],
[0,0,0,-2],
[0,0,0, 3],
])
k = 1
q = opt.Opt().add(H1+k*H2)
opt.counter(q.run(shots=100))
```

```
Counter({'1111': 19,
         '0111': 22,
         '0000': 26,
         '0010': 16,
         '0100': 17})
```

$$H = x_1 - x_1x_4 + (3x_4 + x_2x_3 - 2x_3x_4 - 2x_2x_4)$$

$$\mathbf{1111}: H = 0$$
$$= 1 - 1*1 + (3*1 + 1*1 - 2*1*1 - 2*1*1)$$
$$\mathbf{0111}: H = 0$$
$$= 0 - 0*1 + (3*1 + 1*1 - 2*1*1 - 2*1*1)$$
$$\mathbf{0000}: H = 0$$
$$= 0 - 0*0 + (3*0 + 0*0 - 2*0*0 - 2*0*0)$$
$$\mathbf{0010}: H = 0$$
$$= 0 - 0*0 + (3*0 + 0*1 - 2*1*0 - 2*0*0)$$
$$\mathbf{0100}: H = 0$$
$$= 0 - 0*0 + (3*0 + 1*0 - 2*0*0 - 2*1*0)$$

※ 上記以外のパターンでは $H \neq 0$ となる。

3-6: アニーリング計算まとめ

ここまで巡回セールスマン問題と多体相互作用について見て来ました。

全体の流れのまとめと次のステップの為の推薦図書等をまとめます。

Re:組み合わせ最適化問題

様々な**制約**の下で多くの選択肢の中から、**指標**(コスト)を最も良くする**結果**(組み合わせ)を得る問題が、組み合わせ最適化問題。

1. アニーリングで解く為には、制約と指標(コスト)を、QUBO行列にする必要がある。
2. その為には与えられた問題の定式化を行う必要がある。

組み合わせ最適化問題の定式化

課題となっている組み合わせ最適化問題を解く為のハミルトニアン式を定めること。

QUBO用ハミルトニアン式の要件：

- **バイナリ変数**：変数が取る値は0または1のみ
- **2次式**：変数の最高次数が2である多項式
  - 多項式：「＋」または「－」の記号によって2つ以上の項を結びつけた式。
- **2体問題**：1つの項が2変数間の関係まで
  - ただし多体問題を制約により2体問題に変換できれば計算可能となる。

※文章問題を数学問題に変換する必要があり、数学的素養が求められる。

アニーリング計算の解き方

重要!

全エネルギー（QUBO） コスト関数（QUBO） 補正係数 制約関数（QUBO）

$$H = H_{cost} + k \times H_{penalty}$$

1. 問題に合わせたスピンモデルを用意する。
2. 問題に合わせて定式化（コスト関数と制約関数の用意）。

H	全エネルギー （ハミルトニアン）	求めるべきエネルギーをQUBO形式で得ることで、 イジングマシンへの入力とする。
H_{cost}	コスト関数 （目的条件）	指標をコスト値に変換する式からQUBOを用意する。
$H_{penalty}$	制約関数 （制約条件）	問題を成立させる為の制約式からQUBOで用意する。 複数の制約の組み合せが必要な場合もある。
k	補正係数	コスト関数と制約関数のバランスを取る為の係数。

3. 補正係数を調整して正しい最適解や局所解が得られるようにする。

アニーリング計算の精度

1. フラストレーションを生じる問題では補正係数の調整が結構微妙で面倒。
   - 結果を見て調整して実行の繰り返しが必要。
2. フラストレーションを生じる問題では最適解を得る確率は結構低く、局所解も多い。
   - 機械学習のサンプリングには使えそう。
   - 最適解と他の解の間の差が少ないと良い結果が得難いような気がする。
3. アニーリング計算に向いている問題と向いていない問題がありそうだ…

有名な組み合わせ最適化問題

ナップサック問題:

- 異なる価値とコストを持つ荷物を上限の中で最高コストになるようナップサックに詰合せる組み合わせを求める。

グラフ彩色問題:

- 隣り合った領域が同じ色にならないように塗り分ける問題。

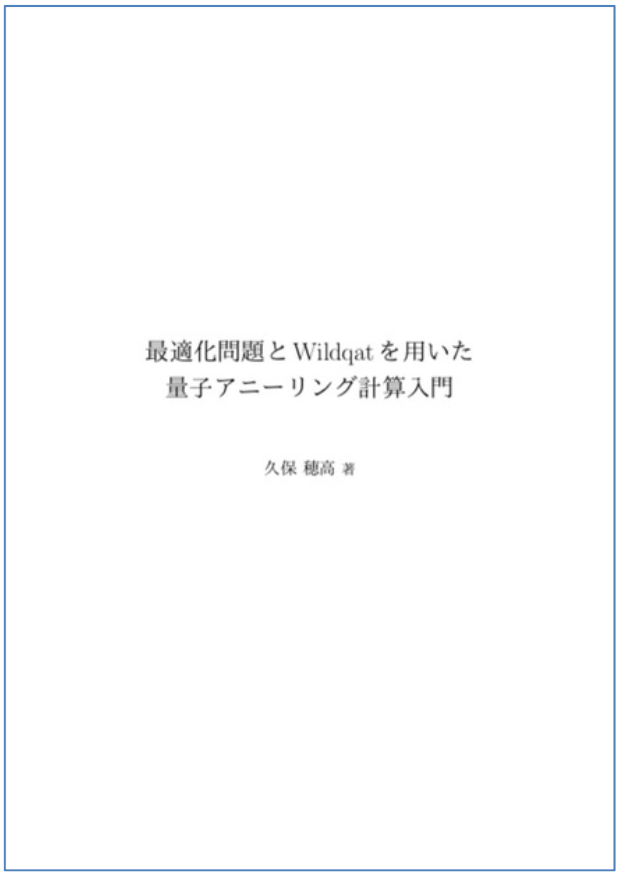
クラスタリング(クラスタ分割問題):

- 与えられたデータ集合を部分集合に分割する問題。

配送計画問題:

- 巡回セールスマン問題を一般化して配送を効率良く行う問題。
- アニーリング計算の実証実験等で良く使われている問題。

量子アニーリング：推薦図書



最適化問題とWildqatを用いた
量子アニーリング計算入門
(BOOTH：ダウンロード商品)
久保 穂高 (著) - 143 ページ

ダウンロードPDF版：980円
https://hodaka.booth.pm/items/1415833

入門だけでなく、豊富な問題の例が載っているので　本資料を見た後に勉強するには最適な書籍です。問題の定式化の勉強になります。

問題例：分割問題、カバー・パッキング問題、不等式問題、彩色問題、ハミルトン路

量子ボルツマン機械学習

制限ボルツマン機械学習にてパラメータ更新による解候補の再計算をモンテカルロ法では無く、量子アニーリング計算で行う。

この場合に量子アニーリング計算は最適解では無く近似解で良い。

つまり教師データ生成用のサンプリングマシンとして量子アニーリングを使って平均値を求める。

現在は機械学習と量子アニーリング計算の連携が模索されている。

量子アニーリング：参考図書

量子アニーリングの基礎

基本法則から読み解く物理学最前線 18

西森 秀稔 (著), 大関 真之 (著),

須藤 彰三 (監修), 岡 真 (監修) - 160 ページ

出版社:共立出版 (2018/5/19)

単行本（ソフトカバー）版：2160円

https://www.amazon.co.jp/gp/product/4320035380/

比較的アカデミックな内容ですが1冊持っておきたい書籍です。量子アニーリングに関して幅広い範囲を説明しています。量子ボルツマン機械学習についての記述もあります。

3-7：D-Wave / D-Wave Leap

量子アニーリングマシンとして最初の商用マシンが D-Wave One（2011年：256量子ビット）でした。

D-Wave Systems, Inc.

D-Wave The Quantum Computing Company™

D-Wave Systems, Inc.（カナダ：英語）サイト：
https://www.dwavesys.com/
設立：1999年　本社：カナダブリティッシュコロンビア州バーナビー市

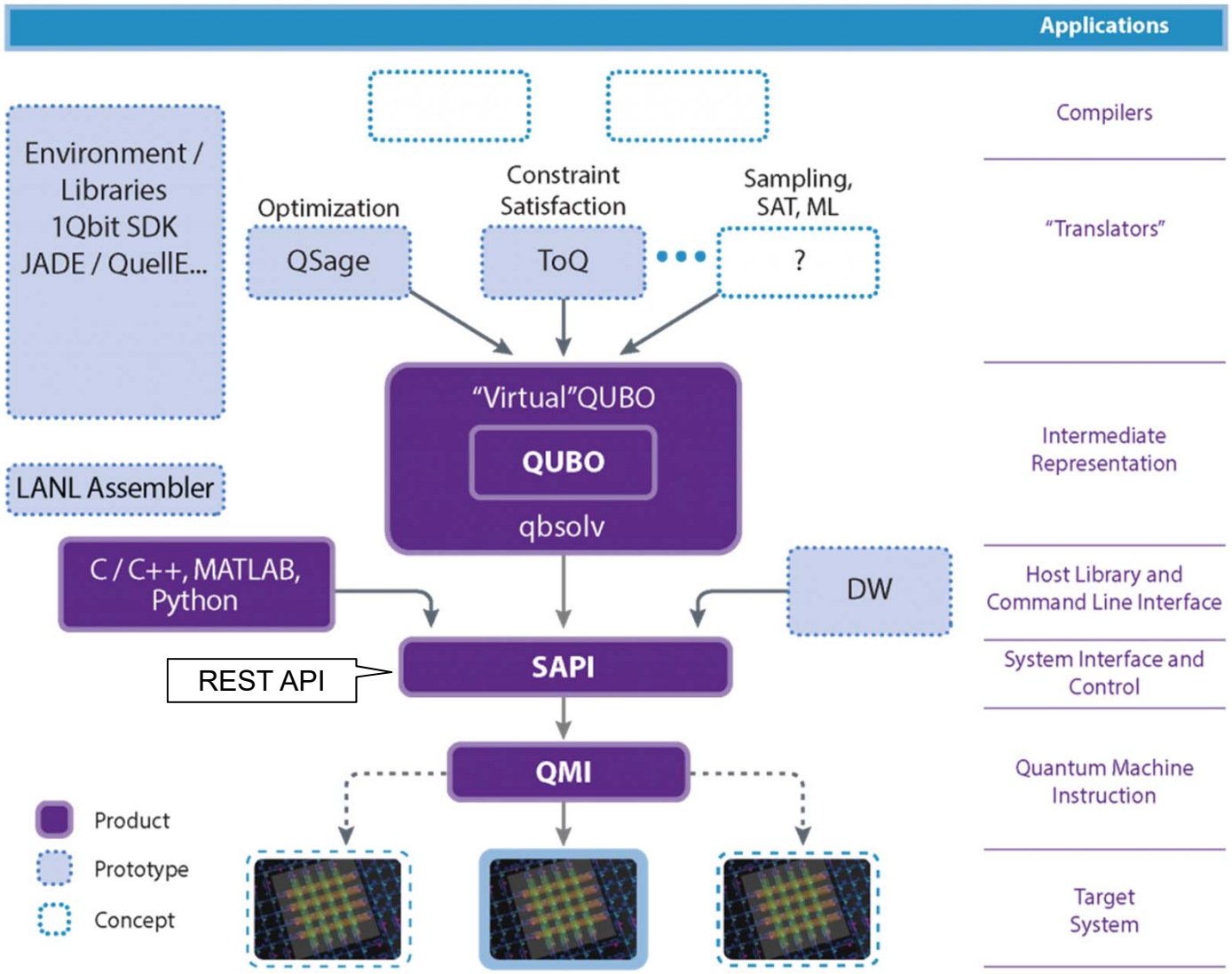
D-Wave社（日本）サイト：
http://dwavejapan.com/

D-Wave 2000Q（D-Wave社サイトから）

現行機：D-Wave 2000Q	
量子ビット数	2048 qubit
グラフ	キメラグラフ（6結合）
諧調（係数）	4～5 bits（16～32諧調）
価格（非公開）	17億円？
出荷開始	2017年

次世代機：D-Wave 5000Q？	
量子ビット数	5000 qubit
グラフ	ペガサスグラフ（15結合）
諧調（係数）	4～5 bits？（16～32諧調）
価格（非公開）	不明？
出荷開始（予定）	2020年

D-Wave Software.（D-Wave社サイトから）



D-Wave Leap（クラウドサービス）

https://cloud.dwavesys.com/leap/

登録することで、クラウド上のD-Wave 2000Qが使えるサービス。

- **Trial Plan**：
  1カ月有効（初回登録直後のプラン）
- **Free Developer Access**：※ 要GitHubリポジトリ
  自動更新、1カ月あたり1分間の利用、成果はOSS化必要
  - 1時間2000ドルの有料プランも選べる。

※クラウドを使わない利用方法であれば Leap への登録無しでも D-Wave Ocean SDK をインストールすればシミュレータ等は使うことができる。

D-Wave Ocean SDK 実機の設定

API endpoint URL と API Token を dwave.conf に設定する。
作成は「dwave config create」で行える。
入力が必要な情報は Leap にログインすると取得可能。

- API endpoint URL: 画面下方の「Solver API endpoint」を利用。
- API Token: 画面左にある「API Token」の copy ボタンで取得。

実行例:

```
(base) > dwave config create <改行>
Configuration file not found; the default location is:
C:¥Users¥myuser¥AppData¥Local¥dwavesystem¥dwave¥dwave.conf
Configuration file path [C:¥Users¥myuser¥AppData¥Local¥dwavesystem¥dwave¥dwave.conf]: <改行>
Profile (create new) [prod]: <改行>
API endpoint URL [skip]: <ログインダッシュボード下にあるSolver API endpointのURL>
Authentication token [skip]: <ログインダッシュボードのAPI Tokenをコピーペースト>
Default client class (qpu or sw) [qpu]: <改行>
Default solver [skip]: <改行>
Configuration saved.
(base) >
```

Ocean：D-Wave 実機で最適化問題を解く

例題：以下式をハミルトニアン式とQUBO行列を作成して解け。

$$x_1 + x_2 = 1 \qquad H = \begin{bmatrix} x_1 & x_2 \end{bmatrix} \begin{bmatrix} -1 & 2 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + 1$$

オフセット値

D-Wave実機による計算：

```
from dimod import *                                                  # dimodインポート
from dwave.system.samplers import DWaveSampler                       # 実機用インポート1
from dwave.system.composites import EmbeddingComposite               # 実機用インポート2
Q = {(0, 0):-1, (0, 1):2, (1, 1):-1}                                 # QUBO行列(dict)
b = BinaryQuadraticModel.from_qubo(Q, 1.0)                           # QUBO設定(オフセット値は1)
r = EmbeddingComposite(DWaveSampler()).sample(b, num_reads=8)  # 実機で8回実行
print(r)                                                             # 結果表示
```

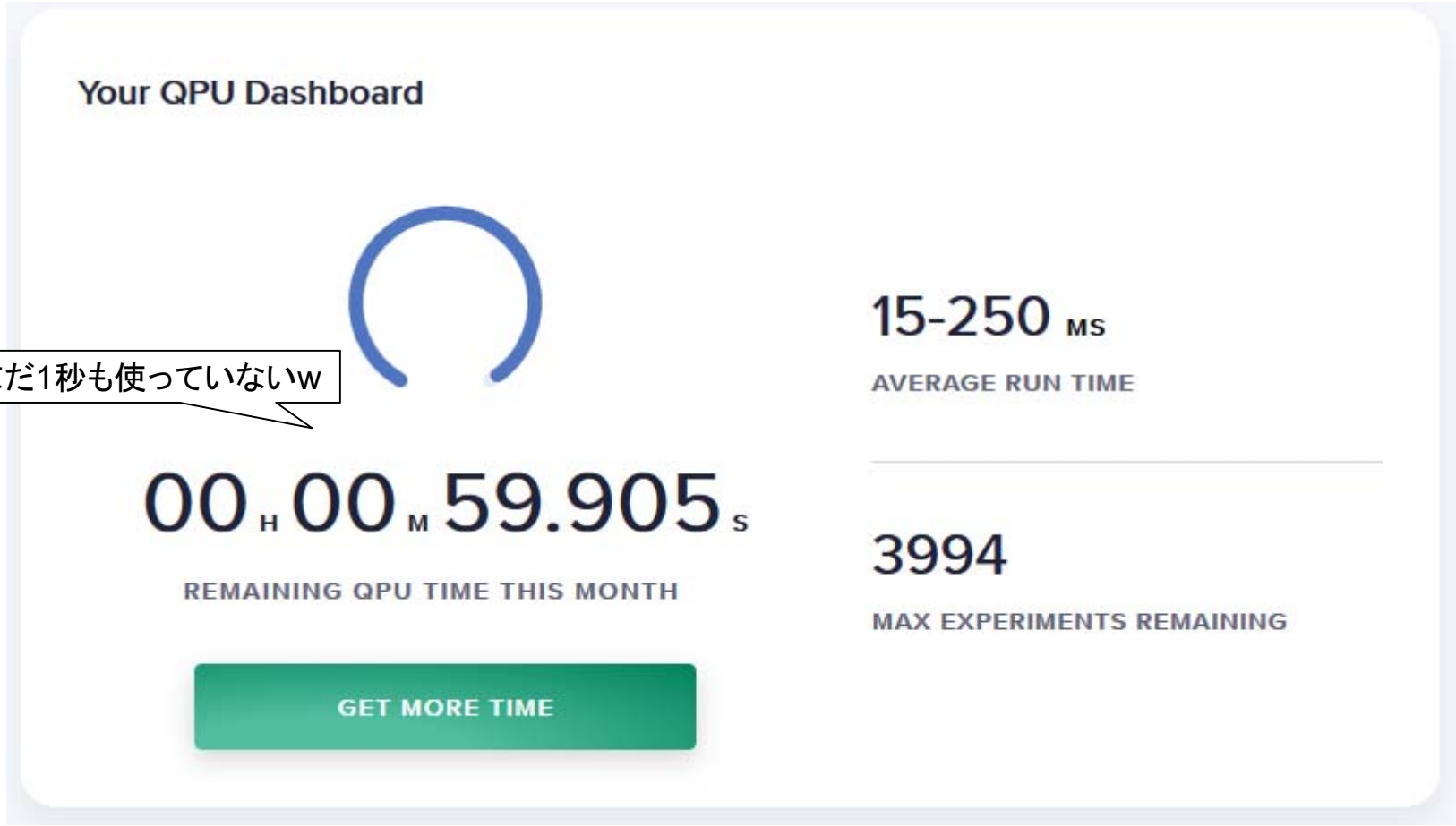
```
   0  1 energy num_oc.  chain_.
0  1  0    0.0       4      0.0
1  0  1    0.0       4      0.0
['BINARY', 2 rows, 8 samples, 2 variables]
```

E = 0.0 の (0,1) と (1,0) が
4回ずつ発生している

※ 実行するとだいたい数秒～10秒程度で結果が返ってくる。

Leap：D-Wave 実機の残り時間

自分のダッシュボードに情報が出ている。
（1分使い切るのはなかなか大変そうです。）



3-8: 量子アニーリング編 付録

PyQUBO : QUBO生成SDK

QUBO行列があればアニーリング計算できることは分かった。
でもこの資料ではQUBOを自分で作成してたよね、他の方法は？
答: **PyQUBO を使えばハミルトニアン式からQUBOを得られる。**

PyQUBO:（dwave-ocean-sdk にも含まれている）

https://github.com/recruit-communications/pyqubo

利用例: $H = -x_1^2 - x_2^2 + 2x_1x_2 + 1$

```
from pyqubo import *                          # PyQUBOのインポート
q0, q1 = Binary("0"), Binary("1")             # バイナリ変数の準備
H = - q0 - q1 + 2*q0*q1 + 1                   # ハミルトニアン式
model = H.compile()                           # PyQUBOコンパイル
qubo, offset = model.to_qubo()                # 結果のQUBO化
print(qubo, 'offset=', offset)                # 結果表示
print(model.to_qubo(index_label=True))        # 結果をインデックスで
```

```
{('q0', 'q1'): 2.0, ('q0', 'q0'): -1.0, ('q1', 'q1'): -1.0} offset= 1.0
({(0, 1): 2.0, (0, 0): -1.0, (1, 1): -1.0}, 1.0)
```

QUBO= $\begin{pmatrix} -1 & 2 \\ 0 & -1 \end{pmatrix}$

※ 多体問題も対応しているようです。

QUBOとイジングモデル

Blueqat / Wildqat（ARRAY/配列形式）	D-Wave Ocean SDK（DICT/辞書形式）	
QUBO	**QUBO**	**Ising Model**

```
Q = [
  [-1, 0, 3],
  [0, -2, 0],
  [0,  0, 1]
]
# 全ての要素を記述
```

同じ

```
Q = {
  (0, 0): -1,  # Q00
  (0, 2):  3,  # Q02
  (1, 1): -2,  # Q11
  (2, 2):  1   # Q22
}
# 値のある要素のみ記述
```

```
h = {
  0:-1,      # h0
  1:-2,      # h1
  2: 1       # h2
}
J = {
  (0, 2):3,  # J02
}
# 局所磁場hと
# 相互作用Jに分ける
```

```
h, J, offset1 = qubo_to_ising(Q, offset2)
Q, offset2 = ising_to_qubo(h, J, offset1)
```

変換

PythonのQUBO形式変換（dictとarray）

```
# 配列から辞書へ変換
def array2dict(ary):
    dct = dict(((i,j), ary[i][j])
        for i in range(len(ary))
        for j in range(len(ary[0]))
        if ary[i][j] != 0)
    return dct
# 辞書から配列へ変換(配列サイズをszで指定)
def dict2array(dct, sz):
    ary = [[0] * sz for i in range(sz)]
    for i in dct:
        ary[i[0]][i[1]] = dct[i]
    return ary
# 試験配列の初期化
org = [[-1,2],[0,-1]]
print('org=',org)
# 配列から辞書へ変換
dictQubo = array2dict(org)
print('dict=',dictQubo)
# 辞書から配列へ変換(配列サイズを指定)
arrayQubo = dict2array(dictQubo, sz=2)
print('array=',arrayQubo)
```

もっと良い実装もあると思いますが、とりあえず使えると言うことで(^^;

```
org= [[-1, 2], [0, -1]]
dict= {(0, 0): -1, (0, 1): 2, (1, 1): -1}
array= [[-1, 2], [0, -1]]
```

量子ゲート型で最適化問題を解く

例題：以下式をハミルトニアン式とQUBO行列を作成して解け。

$$x_1 + x_2 = 1 \qquad H = \begin{pmatrix} x_1 & x_2 \end{pmatrix} \begin{pmatrix} -1 & 2 \\ 0 & -1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$$

アニーリング計算（既出）：

```
from blueqat import opt                 # Blueqatのオプション
q = opt.Opt().add([[-1,2],[0,-1]])      # QUBOのセット
print(q.run(shots=8))                   # アニーリング計算を8回実行
```

```
[[1, 0], [0, 1], [0, 1], [1, 0], [1, 0], [0, 1], [1, 0], [0, 1]]
```

(x_1, x_2) が $(0,1)$ と $(1,0)$ の確率50%で発生

量子ゲート型QAOA（量子断熱）計算：

```
from blueqat import opt                 # Blueqatのオプション
q = opt.Opt().add([[-1,2],[0,-1]])      # QUBOのセット
print(q.qaoa().most_common(4))          # QAOA計算から上位4値を表示
```

```
(((1, 0), 0.4992877208471856), ((0, 1), 0.4992877208471856),
 ((0, 0), 0.0007122791528145044), ((1, 1), 0.0007122791528145044))
```

(x_1, x_2) が $(0,1)$ と $(1,0)$ のほぼ確率50%で発生している $(0,0)$ と $(1,1)$ も僅かな確率で発生

※ BlueqatではQAOAの入力としてQUBOも使えるので比較が楽。

- 量子ゲート型でも組み合わせ最適化計算は可能である。

量子アニーリング型の状況は…

製品としてはD-Waveしか無いが、近々NECが参入する？D-Wave 新型の投入も近い。

富士通・日立・東芝等は非量子のアニーリングマシンをリリース。プログラミングは量子と同じ。

量子ゲート型でアニーリング（量子断熱）計算を行うケースも増えているようだ。

アニーリング計算を何で解くかは別として、計算をする為の定式化やQUBO等の知識は共通。

今まずはアニーリングプログラミングする為の手法を勉強しましょう！

http://scienceinoh.jp/schrodinger/