量子アニーリングの課題

- 量子アニーリングは敷居が高い
  - 問題を量子アニーリングマシンに適した形に変形する必要がある
- 量子アニーリングマシンに特化したノウハウが必要
  - 量子アニーリングマシンの解が最適であるという保証がないため、制約条件を満たしているか等、結果を手動で検証する必要がある

量子アニーリングへのアプローチ

1. 解きたい問題を制約条件等を用いて表す **(定式化)**
2. 式を最小化問題 $H(x_1, \ldots, x_N)$ の形で表す

$$(x_1, \ldots, x_N) = \underset{x_1,\ldots,x_N \in \{0,1\}}{\operatorname{argmin}} \quad H(x_1, \ldots, x_N)$$

$$H(x_1, \ldots, x_N) = H_0(x_1, \ldots, x_N) + \lambda_1 H_1(x_1, \ldots, x_N) + \cdots$$

3. 式をイジングモデルの形に変形する (J_{ij}, h_i を求める)

$$H(x_1, \ldots, x_N) = \sum_{i<j}^{N} J_{ij} x_i x_j + \sum_{i=1} h_i x_i$$

4. パラメータ $\lambda_1, \lambda_2, \ldots$ を決め、J_{ij}, h_i を数値で表す
5. 量子アニーリングを用いて $x_1, \ldots, x_N$ の最適解を求める
6. 解 $(x_1, \ldots, x_N)$ が **1.** の制約条件を満たすことを検査する

量子アニーリングへのアプローチ

1. 解きたい問題を制約条件等を用いて表す **(定式化)**
2. 式を最小化問題 $H(x_1, \ldots, x_N)$ の形で表す

$$(x_1, \ldots, x_N) = \underset{x_1,\ldots,x_N \in \{0,1\}}{\operatorname{argmin}} \; H(x_1, \ldots, x_N)$$

$$H(x_1, \ldots, x_N) = H_0(x_1, \ldots, x_N) + \lambda_1 H_1(x_1, \ldots, x_N) + \cdots$$

3. 式をイジングモデルの形に変形する (J_{ij}, h_i を求める)

PyQUBO, ThreeQ.jl で省略可能

$$\sum_{i<j}^{N} \qquad \sum_{i=1}^{N}$$

4. パラメータ $\lambda_1, \lambda_2, \ldots$ を決め、J_{ij}, h_i を数値で表す
5. 量子アニーリングを用いて $x_1, \ldots, x_N$ の最適解を求める
6. 解 $(x_1, \ldots, x_N)$ が **1.** の制約条件を満たすことを検査する

量子アニーリングへのアプローチ

1. 解きたい問題を制約条件等を用いて表す **(定式化)**
2. 式を最小化問題 $H(x_1, \ldots, x_N)$ の形で表す

$$(x_1, \ldots, x_N) = \operatorname{argmin} \quad H(x_1, \ldots, x_N)$$

$$1\}$$

$$(x_1, \ldots, x_N) = H_0(x_1, \ldots, x_N) + \lambda_1 H_1(x_1, \ldots, x_N) + \cdots$$

3. 式をイジングモデルの形に変形する (J_{ij}, h_i を求める)

$$H($$

$$\sum_{i<j}$$

$$\sum_{i=1}^{N}$$

4. パラメータ $\lambda_1, \lambda_2, \ldots$ を決め、J_{ij}, h_i を数値で表す
5. 量子アニーリングを用いて $x_1, \ldots, x_N$ の最適解を求める
6. 解 $(x_1 \ldots, x_N)$ が **1.** の制約条件を満たすことを検査する

提案手法で抽象化

PyQUBO, ThreeQ.jl で省略可能

未踏での提案

数式処理を用いたアニーリングマシン向け
プログラミング言語及びその処理系の開発

この言語を使用することにより

- 簡単な文法で問題を記述するだけで
- 量子アニーリングマシンのハードウェアを意識することなく
- 制約条件を満たす、正しい解を得ることができる

ことを目指す

ターゲット

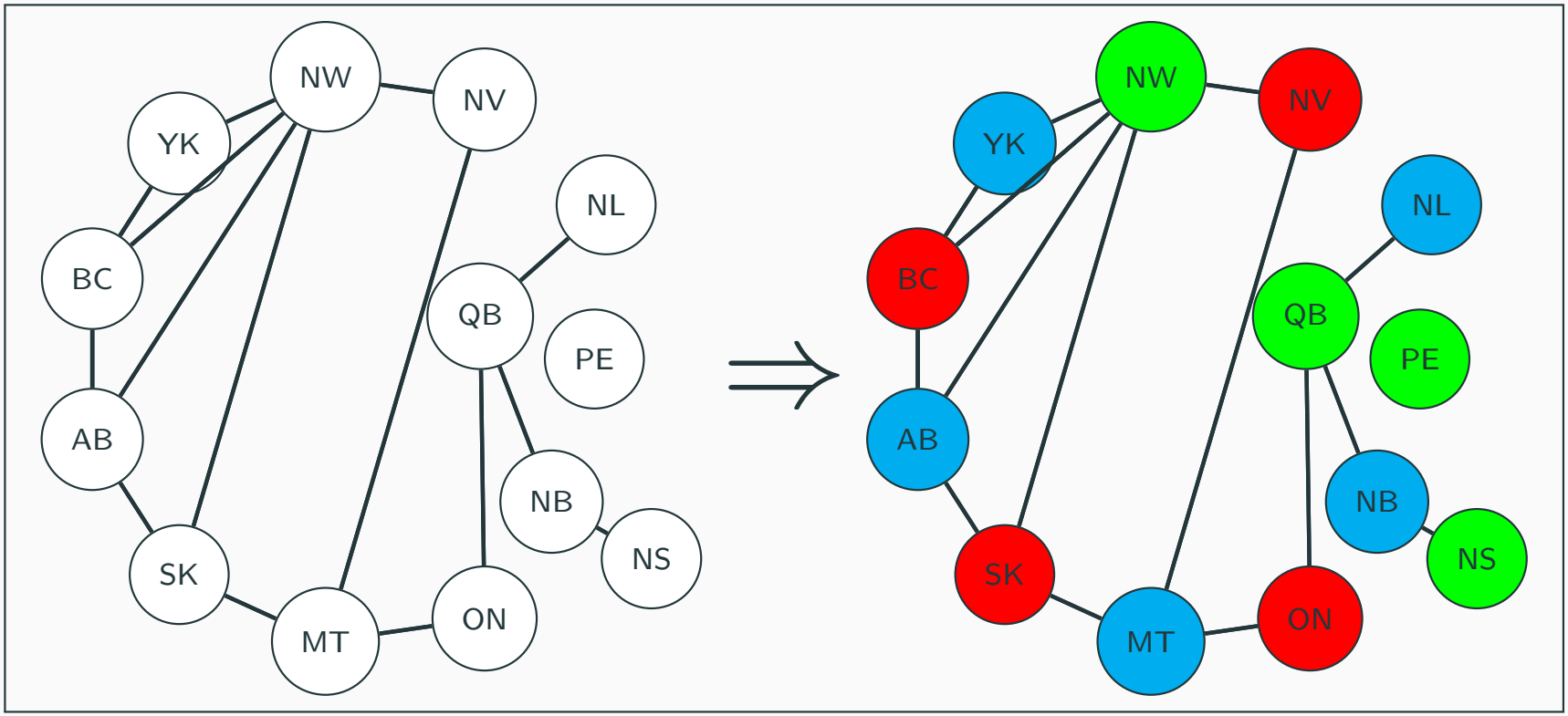
- Java 等の従来のプログラミング言語を扱ったことはあるが、量子アニーリングに関する知識はない技術者
- 量子アニーリング専門の技術者を雇用する余裕はないが、上記技術者を擁し、量子アニーリングを用いて業務を改善したいと考える中小企業

例: 3 色問題

https://commons.wikimedia.org/wiki/File:Political_map_of_Canada.png

例: 3色問題

以下のようなグラフを、線でつながった丸が異なる色になるように3色で塗り分けたい



3 色問題を解く

無向グラフ (V, E) について、$v \in V$ は州を表し、$u, v \in V$ に対して $(u, v) \in E$ は州 u と州 v が地続きになっているとき、またこのときに限り成立するとする。

1. 問題を定式化する: $x_{ui} \in \{0, 1\}$ は州 $u \in V$ を色 $i \in \{1, 2, 3\}$ で塗られるとき 1、そうでないとき 0 であるとする。

$$^{\forall} u \in V \quad \sum_{i=1}^{3} x_{ui} = 1$$

1 つの州は複数の色では塗れない

$$^{\forall} i \in \{1, 2, 3\} \; ^{\forall} (u, v) \in E \quad x_{ui} \neq x_{vi}$$

つながった州は同じ色では塗れない

3 色問題を解く

2. 問題の式を最小化問題の形で表す:

$$
\begin{aligned}
H &= w_1 \sum_{v \in V} \left(1 - \sum_{i=1}^{3} x_{vi} \right)^2 + w_2 \sum_{(uv) \in E} \sum_{i=1}^{3} x_{ui} x_{vi} \\
&= w_1 \sum_{v \in V} \left(- \sum_{j=1}^{3} x_{vj} + \sum_{j,k=1}^{3} x_{vj} x_{vk} \right) \\
&\quad + w_2 \sum_{(v,w) \in E} \sum_{i=1}^{3} x_{vi} x_{wi} + \text{const.}
\end{aligned}
$$

3 色問題を解く

3. 式をプログラムとして記述する:

```
for u ∈ V:
    for i ∈ {1, 2, 3}:
        addterm −w_1 × x_ui
        for v ∈ V:
            addterm w_1 × x_ui × x_vi
            if (u, v) ∈ E:
                addterm w_2 × x_ui × x_vi
```

4. パラメータを決定する:

$w_1 = 1; w_2 = 1$

ThreeQ.jl の例

問題の設定

```
provinces = ["BC", "YK", "NW", "AB", "SK", "NV",
    "MT", "ON", "QB", "NB", "NS", "PE", "NL"]
neighbors = Dict()
neighbors["BC"] = ["YK", "NW", "AB"]
neighbors["YK"] = ["BC", "NW"]
neighbors["NW"] = ["YK", "BC", "AB", "SK", "NV"]
neighbors["AB"] = ["BC", "NW", "SK"]
neighbors["SK"] = ["AB", "NW", "MT"]
neighbors["NV"] = ["NW", "MT"]
neighbors["MT"] = ["NV", "SK", "ON"]
neighbors["ON"] = ["MT", "QB"]
neighbors["QB"] = ["ON", "NB", "NL"]
neighbors["NB"] = ["QB", "NS"]
neighbors["NS"] = ["NB"]
neighbors["PE"] = []
neighbors["NL"] = ["QB"]
```

パラメータの設定

```
numcolors = 3; w1 = 1; w2 = 1
```

ハミルトニアンの生成

```
m = ThreeQ.Model("mapcolor", "laptop", "c4-sw_sample",
        "workingdir", "c4")
@defvar m colors[1:length(regions), 1:numcolors]
for i = 1:length(regions)
  for j = 1:numcolors
    @addterm m -w1 * colors[i, j]
    for k = 1:j - 1
      @addterm m 2 * w1 * colors[i, j] * colors[i, k]
    end
  end
end
for j = 1:length(regions)
  for k = 1:j - 1
    if regions[k] in neighbors[regions[j]]
      for i = 1:numcolors
        @addterm m w2 * colors[j, i] * colors[k, i]
      end
    end
  end
end
```

ThreeQ.jl の例

求解

```
ThreeQ.qbsolv!(m; minval=-length(regions) * w1, S=20,
        showoutput=true)
```

解の確認

```
if length(answer) < length(regions)
  println("Some regions have not a color!")
  return false
end
for r1 in regions
  for r2 in neighbors[r1]
    if answer[r1] == answer[r2]
      println("$r1 and $r2 have the same color!")
    end
  end
end
println("OK")
```

提案手法

```
type color := red | orange | yellow
color [@BC, @YK, @NW, @AB, @SK, @NV, @MT, @ON, @QB, @NB, @NS
  , @PE, @NL] : p
@BC != @AB, @BC != @YK, @BC != @NW
@YK != @BC, @YK != @NW
@NW != @YK, @NW != @BC
@AB != @BC, @AB != @NW, @AB != @SK
@SK != @AB, @SK != @NW, @SK != @MT
@NV != @NW, @NV != @MT
@MT != @NV, @MT != @SK, @MT != @ON
@ON != @MT, @ON != @QB
@QB != @ON, @QB != @NB, @QB != @NL
@NB != @QB, @NB != @NS
@NS != @NB
@NL != @QB
solve(p)
print(p)
```

その他

検討事項

- 問題の状態での最適化
- 古典コンピュータとの連携
- 制約条件を満たさなかった場合の補助的な最適化

成果物

将来的に FLOSS として公開予定

今後の計画

- 事前調査 (7-11 月)
  論文調査, 環境構築, アニーリングマシンの動作確認, 既存の実装の調査
- 設計 (9-11 月)
  実装方法の検討, テストの作成
- 開発実施 (10-1 月)
  初期実装, 構文木の生成, 実行器の生成, ハミルトニアンの生成, 計算機能の実装
- 評価・運用 (1-2 月)
  全テストコードの実行に対応, より大きな問題に対する運用方法の検討, ドキュメントの整備