アニーリングの課題

- アニーリングは敷居が高い
  - 問題をアニーリングマシンに適した形に変形する必要がある
- アニーリングマシンに特化したノウハウが必要
  - 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

本研究の目的

- 解きたい問題の制約条件等を記述するプログラミング言語の開発
- 言語で記述されたソースコードを解釈し、アニーリングによって解を求め、その解が問題の制約条件を満たすことを保証する処理系の開発

アニーリングが対象とする問題の例

一般に

アニーリングにおける問題

＝制約条件 ＋ 最適化関数

(=本言語で記述し解くことができる問題)

アニーリングが対象とする問題の例

一般に

アニーリングにおける問題

＝制約条件 ＋ 最適化関数

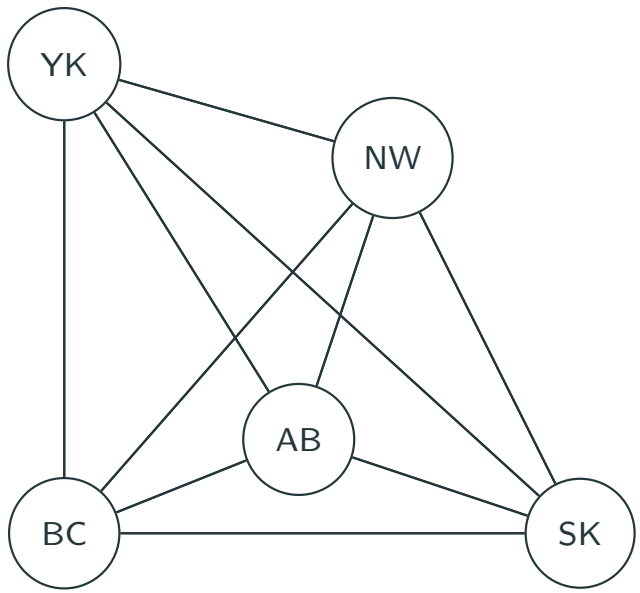
(=本言語で記述し解くことができる問題)

例: 巡回セールスマン問題

与えられたすべての都市を1度ずつ訪問し戻ってくるルートの中で距離が最短なものを求める。

巡回セールスマン問題

W_{uv}	BC	YK	NW	AB	SK
BC	–	5.0	5.0	3.0	4.5
YK	5.0	–	3.5	5.0	7.0
NW	5.0	3.5	–	3.0	4.5
AB	3.0	5.0	3.0	–	2.5
SK	4.5	7.0	4.5	2.5	–



既存の解き方

グラフ $G = (V, E)$ において都市 u, v 間の距離を W_{uv} で与える。$x_{v,j}$ $(v \in V, 1 \leq j \leq N)$ をバイナリ変数とし、都市 v を j 番目に訪問するとき $x_{v,j} = 1$, そうでないとき $x_{v,j} = 0$ とする。

$$H = A_1 \sum_{v \in V} \left(1 - \sum_{j=1}^{N} x_{v,j}\right)^2 + A_2 \sum_{j=1}^{N} \left(1 - \sum_{v \in V} x_{v,j}\right)^2$$

1 つの都市は 1 度のみ訪問　　同時に訪問できる都市は 1 箇所

$$+ A_3 \sum_{(u,v) \notin E} \sum_{j=1}^{N} x_{u,j} x_{v,j+1} + A_4 \sum_{(uv) \in E} W_{uv} \sum_{j=1}^{N} x_{u,j} x_{v,j+1}$$

辺のない都市間は移動不可　　最適化関数

困難な点

- 制約条件をバイナリ変数を用いた式で表す必要がある
- 式をイジングモデルの形 (次数 2 まで) に変形する必要がある
- アニーリングマシンの解が制約条件を満たす正しい解であるとは限らない
- 正しい解を求めるためパラメータ $(A_1, A_2, \cdots)$ を探索する必要がある

新しい解き方

```
  const cities := [@BC, @YK, @NW, @AB, @SK]
  type order := 1 .. len(cities)
  order cities
  define distance @A @B := (@A <-> @B || (@A == 1,
    @B == len(cities)) || (@A == len(cities), @B == 1))

different(cities)

minimize {
    5.0 : distance @BC @YK
    5.0 : distance @BC @NW
    3.5 : distance @YK @NW
    3.0 : distance @BC @AB
    5.0 : distance @YK @AB
    3.0 : distance @NW @AB
    4.5 : distance @BC @SK
    7.0 : distance @YK @SK
    4.5 : distance @NW @SK
    2.5 : distance @AB @SK
}

  solve(cities)
  print(cities)
```

制約条件

最適化関数

変数の宣言

```
const cities := [ @BC, @YK, @NW, @AB, @SK]
type order := 1 .. len(cities)
order cities
```

リスト　変数　型

@BC,@YK,@NW,@AB,@SK は値 $1, 2, \ldots, 5 (=$ len(cities)$)$ を取る変数

$\leftrightarrow$ これらをバイナリ変数で表したい

$\Rightarrow$ 言語中の変数を 2 値にエンコードする必要がある

Encoding

Encoding の例 $\texttt{@A} \in \{1,2,3,4,5,6,7\}, x_i \in \{0,1\}$ とする。

- One-hot encoding

  $$\texttt{@A} = 1 \times x_1 + 2 \times x_2 + 3 \times x_3 + 4 \times x_4 + 5 \times x_5 + 6 \times x_6 + 7 \times x_7$$

  $$x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_8 = 1$$

- Binary encoding

  $$\texttt{@A} = 1 \times x_1 + 2 \times x_2 + 4 \times x_3$$

今回の言語では今のところ One-hot encoding を採用している

マクロ

マクロの定義

`define` マクロ名 変数 [変数 ...] `:=` 式

例

```
define distance @A @B := (@A <-> @B || (@A == 1,
  @B == len(cities)) || (@A == len(cities), @B == 1))
```

以降出てくる `distance @A @B` のパターンを`:=`の右辺の式で置き換える

条件式

条件式は 1(偽) または 0(真) の値をとる。

基本的な演算子

==, !=, >, >=, <, <= (実装は Encoding に依存)

名前	表記	実装
AND	$exp1\,, exp2$	$1-(1-exp1)(1-exp2)$
NOT	$!exp$	$1-exp$
OR	$exp1 \mid\mid exp2$	$exp1 \times exp2$

$H_{\text{制約}} =$ 条件式 1 + 条件式 2 + $\cdots$

特殊な演算子

```
@a <- @b ⇔@a == @b + 1
@a -> @b ⇔@a + 1 == @b
@a <-> @b⇔@a <- @b || @a -> @b
```

演算子の使用例

```
type color := red | yellow | green
color [@A, @B, @C, @D, @E] : p
@A != @B
@C != @E
@D != @E
@B != @D
solve(p)
print(p)
```

結果

```
@A = green
@B = red
@C = yellow
@D = yellow
@E = green
```

演算子の使用例

```
1 type color := 1 .. 3
2 color [@A, @B, @C, @D] : p
3 @A > @B
4 @B >= @C
5 @B >= @D
6 solve(p)
7 print(p)
```

結果

```
1 @A = 3
2 @B = 2
3 @C = 1
4 @D = 1
```

different

`different(cities)`

`cities` に含まれる変数が互いに異なる値を持つことを要求する。

`const cities := [@BC, @YK, @NW, @AB, @SK]` なので、

```
@BC != @YK, @BC != @NW, @BC != @AB, @BC != @SK
@YK != @NW, @YK != @AB, @YK != @SK
@NW != @AB, @NW != @SK
@AB != @SK
```

と書いても同じ

minimize(最適化関数)

```
minimize {
    5.0 : distance @BC @YK
    5.0 : distance @BC @NW
    3.5 : distance @YK @NW
    3.0 : distance @BC @AB
    5.0 : distance @YK @AB
    3.0 : distance @NW @AB
    4.5 : distance @BC @SK
    7.0 : distance @YK @SK
    4.5 : distance @NW @SK
    2.5 : distance @AB @SK
}
```

:の右辺の式 *exp* が成り立つとき、左辺の値 *w* が加算される。

実装

$$H_{\text{最適化}} = w1 \times exp1 + w2 \times exp2 + \cdots$$

※ `maximize` もある

solve と print

- solve([@A, @B, . . .])
  - 指定した変数 (@A,@B,· · ·)(および制約条件によってそれらと関係づけられた変数) について解く
  - 求めたい変数のみを指定する (不要な変数は指定しない) ことで、問題のサイズを小さくできアニーリング時間が短くなる
- print([@A, @B, . . .])
  - solve で求まった変数の値のうち、指定された変数 (@A,@B,· · ·) の値を表示
  - 表示したい変数をフィルタできる

巡回セールスマン問題の例 (再掲)

```
const cities := [@BC, @YK, @NW, @AB, @SK]
type order := 1 .. len(cities)
order cities
define distance @A @B := (@A <-> @B || (@A == 1,
  @B == len(cities)) || (@A == len(cities), @B == 1))

different(cities)

minimize {
    5.0 : distance @BC @YK
    5.0 : distance @BC @NW
    3.5 : distance @YK @NW
    3.0 : distance @BC @AB
    5.0 : distance @YK @AB
    3.0 : distance @NW @AB
    4.5 : distance @BC @SK
    7.0 : distance @YK @SK
    4.5 : distance @NW @SK
    2.5 : distance @AB @SK
}

solve(cities)
print(cities)
```

他の問題の例

本言語を用いることで、以下に示すアニーリングの有名問題を解くことができる。

- グラフ分割問題
- 集合詰め問題
- 頂点被覆問題
- 最小最大マッチング問題
- グラフ彩色問題
- ハミルトン閉路問題
- ...

上記のようなよく知られた問題だけでなく、実社会における現実的な問題を簡潔に記述できるようにすることが目標

イジングモデルへの変形

$$H(x_1,\ldots,x_N) = H_{\text{制約}}(x_1,\ldots,x_N) + H_{\text{最適化}}(x_1,\ldots,x_N)$$

N 個の変数　　$x\in\{0,1\}$ より一般に N 次の式で表される

$$\longrightarrow H_{Ising}(x_1,\ldots,x_{N+M}) = \sum_{i\neq j} J_{ij}x_ix_j + \sum_i h_ix_i + C_{offset}$$

式変形
変数の追加

$N+M$ 個の変数

高々2 次の式

ただし、

$$\underset{x_1,\cdots,x_N}{\text{argmin}} H(x_1,\ldots,x_N) = \underset{x_1,\cdots,x_{N+M}}{\text{argmin}} H_{Ising}(x_1,\ldots,x_{N+M})\Big|_{x_1,\cdots,x_N}$$

$$\left(\text{一般に} \min_{x_1,\cdots,x_N} H(x_1,\ldots,x_N) \neq \min_{x_1,\cdots,x_{N+M}} H_{Ising}(x_1,\ldots,x_{N+M})\right)$$

イジングモデルへの変形[1]

例) $H(x,y,z) = x \underbrace{y\ z}$ $\qquad (x,y,z,w \in \{0,1\})$

$\hookrightarrow w$ とおく

関数 $D(w,y,z) = yz - 2yw - 2zw + 3w$ を用いると

$$w = yz \Leftrightarrow (w,y,z) = \mathrm{argmin}\ D(w,y,z)$$

$$\therefore H_{Ising}(x,y,z,w) = xw + \lambda D(w,y,z)$$

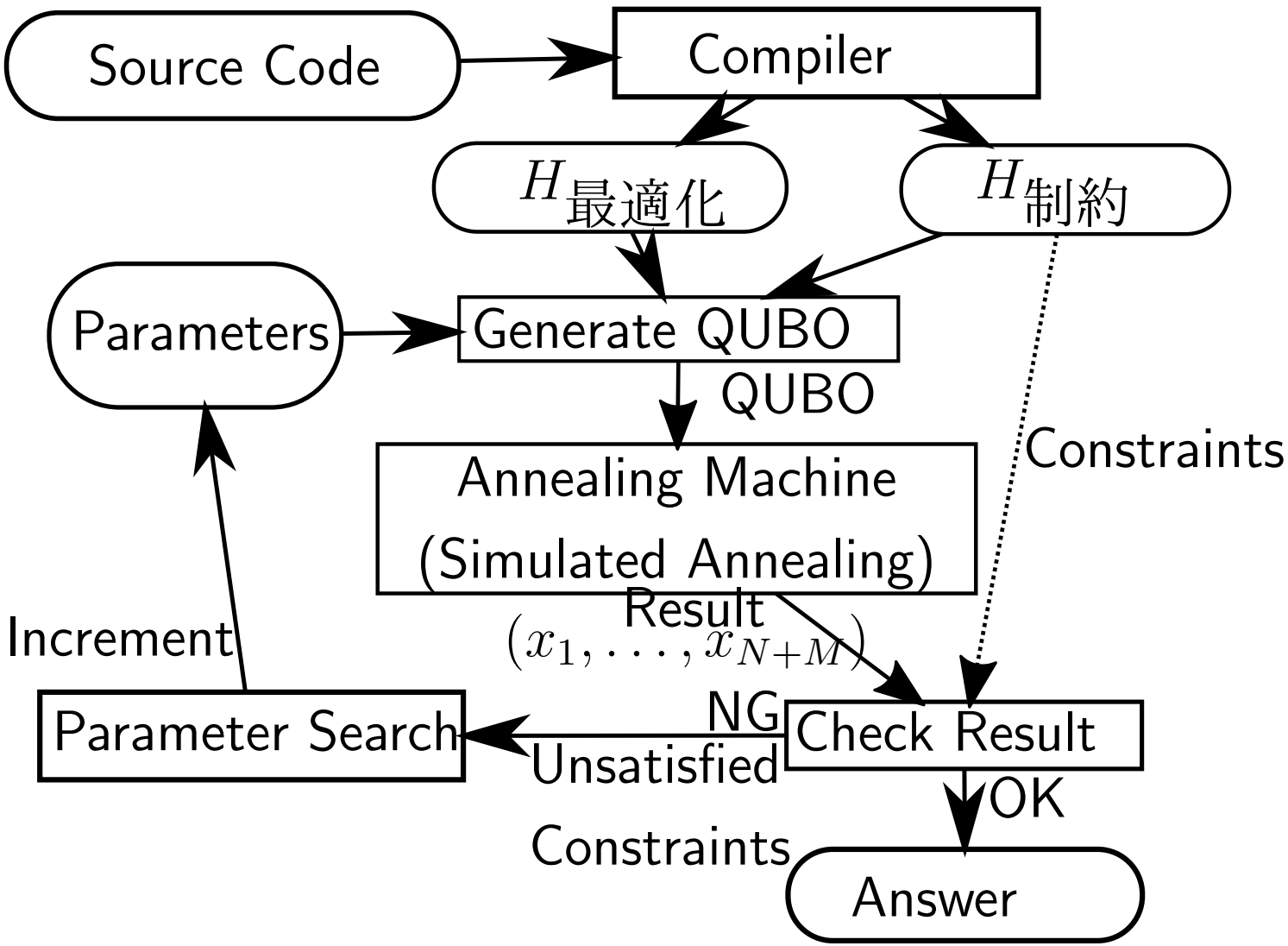
(λ は 1 より十分大きい)

$$\underset{x,y,z}{\mathrm{argmin}}\ H(x,y,z) = \underset{x,y,z,w}{\mathrm{argmin}}\ H_{Ising}(x,y,z,w)\Big|_{x,y,z}$$

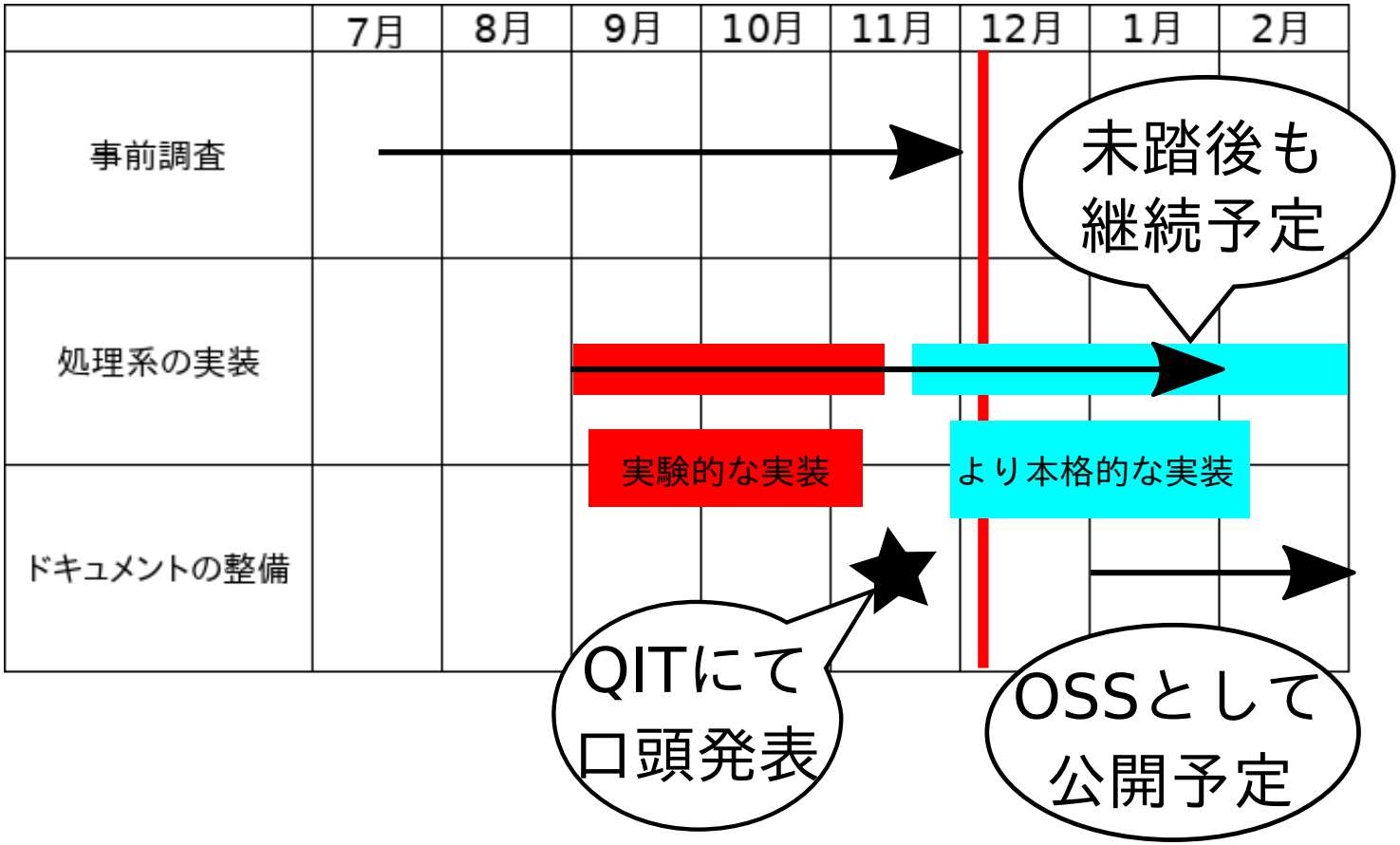
w	y	z	D
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	3
1	0	1	1
1	1	0	1
1	1	1	0

[1] 石川 博, 高階グラフカット, "画像の認識・理解シンポジウム", 2009

パラメータ調整と解の検証



進捗状況と今後の計画


7月
8月
9月
10月
11月
12月
1月
2月
事前調査
処理系の実装
ドキュメントの整備
未踏後も
継続予定
実験的な実装
より本格的な実装
QITにて
口頭発表
OSSとして
公開予定