\Orchestrating a brighter world NEC

NEC Vector Annealing Python API (x86版) リファレンスガイド

第2版

輸出する際の注意事項

本製品（ソフトウェアを含む）は、外国為替および外国貿易法で規定される規制貨物（または役務）に該当することがあります。

その場合、日本国外へ輸出する場合には日本国政府の輸出許可が必要です。

なお、輸出許可申請手続きにあたり資料等が必要な場合には、お買い上げの販売店またはお近くの当社営業拠点にご相談ください。

はしがき

- 商標、著作権について
  - Linux はアメリカ合衆国及びその他の国におけるLinus Torvalds の商標です。
  - その他、記載されている会社名、製品名は、各社の登録商標または商標です。

用語定義・略語

用語・略語	説　明
アニーリング (Annealing)	ここではシミュレーテッドアニーリングを指し、徐々に「温度」を小さくすることで最適に近い解を確率的に求める手法になります。

用語定義・略語

◆ 用語の定義

用語	データ型	記述例	定義
スピン名	文字列	'x[0][0]', 'x[0][1]', …	各スピンの名前(必ずuniqueであること) ※スピン名に、'x'とb'x'のように文字列型とバイト列型を混在して指定した場合、別のスピンとして扱われるので注意すること。
スピンの状態	整数	0 or 1	各スピンの状態を0/1で記述する
スピン	[スピン名, スピンの状態]	['x[0][0]', 0], …	スピン名とスピンの状態の組み合わせで表したもの
収束エネルギー	実数*		モデル内のエネルギーの総和
制約条件の状態	ブール変数		制約条件を満たす=True, 満たさない=False
演算時間	実数*		アニーリングの計算時間

* :アニーリングエンジンは単精度浮動小数点のデータで受け付けます。
このため、Python APIで指定可能な実数のパラメータは単精度浮動小数点で表現可能な範囲
(1.175494×10^{-38} から 3.402823×10^{38})で指定する必要があります。
上記の範囲外の値を指定した場合、infinityとして扱われ、エラーとなります。

用語定義・略語

◆ 用語の定義

用語	データ型	記述例	定義
num_threads	符号無し整数	0, 1, 2, …	並列化のスレッド数 0以下の値、もしくは、Noneを指定した場合、デフォルト数(ライセンス数の上限もしくはプロセッサ数の小さい方の値)で並列実行
seed	符号無し整数	0, 1, 2, …	疑似乱数のシード, Noneを指定した場合、実行時刻からシードを生成
β	[start β , end β , nsteps]		アニーリングの温度変化の区間 - start β (実数*) = アニーリングの開始温度のβ値 (β値 = 1 / 温度) - end β (実数*) =アニーリングの終了温度のβ値 - nsteps(整数) = start β から end β までの区間の分割数(省略可能)

qubo	辞書型	{
		('x[0][0]', 'x[0][1]'):0,
		('x[0][0]', 'x[0][1]'):7,
		…
		}
			スピン間のエネルギーを記述したデータ構造
			各スピン毎にエネルギーを定義するため、スピン数がNの場合、
			N×Nのデータが必要、但し、エネルギーが0となるスピンの組は省略可能

num_reads	整数	0,1,2, …	サンプリング回数(アニーリングの実行回数)
num_sweeps	整数		sweepの回数

* :アニーリングエンジンは単精度浮動小数点のデータで受け付けます。
このため、Python APIで指定可能な実数のパラメータは単精度浮動小数点で表現可能な範囲
(1.175494×10^{-38} から 3.402823×10^{38})で指定する必要があります。
上記の範囲外の値を指定した場合、infinityとして扱われ、エラーとなります。

目次

1. NEC Vector Annealing モジュールの使い方
2. デバッグ方法
3. Python API
4. 環境変数

NEC Vector Annealing モジュールの使い方

QUBOに定義したスピンのモデルに対して、アニーリングを実行

```
...

# quboを作成
param_C = max_len * 0.25
qubo, offset = model.to_qubo(feed_dict={'num_a': param_C})


# ------------------------------------------------
# Vector Annealing のモジュールをimport
import VectorAnnealing

# Vector Annealing のモデル作成
va_model = VectorAnnealing.model(qubo, offset)

# サンプラー作成
sa = VectorAnnealing.sampler()

# アニーリングを実行(実行回数に5回を指定)
result = sa.sample(va_model, num_reads=5)
```

結果は、1回のアニーリング結果を以下のresultクラスの配列として返します

```
-----------------------------------------
class result:
  def __init__(self, spin=None, energy=None, time=None, constraint=None, memory_usage=None):
    self.spin         = spin          # Spin state of QUBO. Ex) spin = { 'x':0, 'y':0, 'z':1 }
    self.energy       = energy        # Total energy of QUBO.
    self.time         = time          # Simulation time of this result.
    self.constraint   = constraint    # Satisfy constraint if constraint is 'True', Broken constraint if constraint is 'False'.
    self.memory_usage = memory_usage  # Memory usage of simulation.
-----------------------------------------
```

NEC Vector Annealing モジュールの使い方

pyquboでQUBOを作成

- 巡回セールスマン問題のケースより

```
# pyqubo を import
from pyqubo import solve_qubo,Array,Placeholder

# quboを2次元で定義 : x[point_num][point_num]
x = Array.create('x', shape=(point_num, point_num), vartype='BINARY')

# 式を記述
param_a = Placeholder('num_a')
Hd = sum(sum((dlength[j][k] - d_min[j]) * x[i, j] * x[(i+1)%point_num, k] for i in range(point_num)) for (j,k) in pairs)
Ha = param_a * sum((sum(x[i, j] for i in range(point_num)) - 1)**2 for j in range(point_num))
Hb = param_a * sum((sum(x[i, j] for j in range(point_num)) - 1)**2 for i in range(point_num))
H = Hd + Ha + Hb

# モデルを作成
model = H.compile()

# quboを作成
param_C = max_len * 0.25
qubo, offset = model.to_qubo(feed_dict={'num_a': param_C})
```

NEC Vector Annealing モジュールの使い方

フリップオプションの使い方

- フリップオプションを指定することにより、効率よく解を求めることが可能となります
- **※ただし、いずれのオプションもハミルトニアンの定式にも別途含める必要があります**
- 記述例(巡回セールスマン問題のケースより)

```
# create one hot constraint rule.
onehot = [0] * (2 * point_num)
for i in range(point_num):
    onehot1 = [0] * point_num
    onehot2 = [0] * point_num
    for j in range(point_num):
        onehot1[j] = 'x[%d][%d]' % (i, j)
        onehot2[j] = 'x[%d][%d]' % (j, i)
    onehot[2*i  ] = onehot1
    onehot[2*i+1] = onehot2

# create fixed spin constraint rule.
fixed = { }
for i in range(point_num):
    for j in range(point_num):
        if i == 0:
            fixed['x[0][%d]' % j] = 0
        if j == 0:
            fixed['x[%d][0]' % i] = 0
    fixed['x[0][0]'] = 1

va_model = VectorAnnealing.model(qubo, offset, onehot=onehot, fixed=fixed)
```

Onehot 制約条件を定義

Fixed spin 制約条件を定義

Vector Annealing のモデル作成時に
フリップオプションの制約条件を指定

NEC Vector Annealing モジュールの使い方

フリップオプションの記述方法

- One hot 制約条件
  - スピンのグループを定義し、指定したグループ内の1個のスピンのみ、状態が 1 となる制約条件

```
one_hot_list = [
  [ 'x[0][0]', 'x[0][1]',  'x[0][2]',  'x[0][3]',  'x[0][4]' ],
  [ 'x[1][0]', 'x[1][1]',  'x[1][2]',  'x[1][3]',  'x[1][4]' ],
  [ 'x[2][0]', 'x[2][1]',  'x[2][2]',  'x[2][3]',  'x[2][4]' ],
]

va_model = VectorAnnealing.model(qubo, offset, onehot=one_hot_list)
```

One hot 制約条件のグループ

複数のone hot 制約条件を定義

onehot に、定義したone hot制約条件を指定

NEC Vector Annealing モジュールの使い方

フリップオプションの記述方法

- Fixed spin 制約条件
  - スピンの状態を指定した値(0/1)に固定する制約条件

スピン名とスピンの状態(0/1)の組で指定

```
fixed_spin_list = {
  'x[0][0]':1,
  'x[0][1]':0,
  'x[0][2]':0,
  'x[0][3]':0
}

va_model = VectorAnnealing.model(qubo, offset, fixed=fixed_spin_list)
```

複数のスピンの状態を定義

スピンの値を格納した辞書型のデータをfixedに指定

  - 従来のリスト形式の指定方法

```
fixed_spin_list = [
  ['x[0][0]', 1],
  ['x[0][1]', 0],
  ['x[0][2]', 0],
  ['x[0][3]', 0]
]

va_model = VectorAnnealing.model(qubo, offset, fixed=fixed_spin_list)
```

NEC Vector Annealing モジュールの使い方

フリップオプションの記述方法

- And zero 制約条件
  - スピンのグループを定義し、指定したグループ内の少なくとも1個のスピンの状態が 0 となる制約条件

```
and_zero_list = [
  [ 'x[0][0]', 'x[0][1]',  'x[0][2]',  'x[0][3]',  'x[0][4]' ],
  [ 'x[1][0]', 'x[1][1]',  'x[1][2]',  'x[1][3]',  'x[1][4]' ],
  [ 'x[2][0]', 'x[2][1]',  'x[2][2]',  'x[2][3]',  'x[2][4]' ],
]

va_model = VectorAnnealing.model(qubo, offset, andzero=and_zero_list)
```

And zero 制約条件のグループ

複数のand zero 制約条件を定義

andzero に、定義したand zero制約条件を指定

NEC Vector Annealing モジュールの使い方

フリップオプションの記述方法

- Or one 制約条件
  - スピンのグループを定義し、指定したグループ内の少なくとも1個のスピンの状態が 1 となる制約条件

```
or_one_list = [
  [ 'x[0][0]', 'x[0][1]',  'x[0][2]',  'x[0][3]',  'x[0][4]' ],
  [ 'x[1][0]', 'x[1][1]',  'x[1][2]',  'x[1][3]',  'x[1][4]' ],
  [ 'x[2][0]', 'x[2][1]',  'x[2][2]',  'x[2][3]',  'x[2][4]' ],
]

va_model = VectorAnnealing.model(qubo, offset, orone=or_one_list)
```

Or one 制約条件のグループ

複数のor one 制約条件を定義

orone に、定義したor one制約条件を指定

NEC Vector Annealing モジュールの使い方

フリップオプションの記述方法

- Cubic supplement 制約条件
  - スピン x_1, x_2, y_1 が、必ず、式 $y_1 = x_1 x_2$ を満たす状態をもつ制約条件

y_1, x_1, x_2 の順でcubic supplement 制約条件のスピンを記述

```
spl_list = [
  [ 'y[0]', 'x[0][0]',  'x[0][1]'],
  [ 'y[1]', 'x[1][0]',  'x[1][1]'],
  [ 'y[2]', 'x[2][0]',  'x[2][1]'],
]

va_model = VectorAnnealing.model(qubo, offset, spl=spl_list)
```

複数のcubic supplement 制約条件を定義

spl に、定義したcubic supplement制約条件を指定

※以下の例のような $y_1 = x_1 x_2$ を満たす制約をハミルトニアンに定義する必要があります

定式化例)

```
Hp1 = x[0,0]*x[0,1]-2*x[0,0]*y[0]-2*x[0,1]*y[0]+3*y[0]
Hp2 = x[1,0]*x[1,1]-2*x[1,0]*y[1]-2*x[1,1]*y[1]+3*y[1]
Hp3 = x[2,0]*x[2,1]-2*x[2,0]*y[2]-2*x[2,1]*y[2]+3*y[2]
```

NEC Vector Annealing モジュールの使い方

フリップオプションの記述方法

- Max one count 制約条件
  - スピンのグループを定義し、指定したグループ内で、状態が1となるスピンが閾値以下となる制約条件

```
max_one_list = [
  [ 2, [ 'x[0][0]', 'x[0][1]',  'x[0][2]',  'x[0][3]',  'x[0][4]' ] ],
  [ 1, [ 'x[1][0]', 'x[1][1]',  'x[1][2]',  'x[1][3]',  'x[1][4]' ] ],
  [ 3, [ 'x[2][0]', 'x[2][1]',  'x[2][2]',  'x[2][3]',  'x[2][4]' ] ],
]

va_model = VectorAnnealing.model(qubo, offset, maxone=max_one_list)
```

状態=1となるスピンの閾値(最大値)を指定

Max one count制約条件に指定するスピンのグループを記述

Max one count 制約条件の定義

複数のmax one count制約条件を定義

maxone に、定義したmax one count制約条件を指定

NEC Vector Annealing モジュールの使い方

フリップオプションの記述方法

- Min max one 制約条件
  - 指定したスピンのグループに対して、状態が1となるスピン数の範囲を指定する制約条件

  - Min max one 制約条件

```
constraint = [
  { 'min':2, 'max':4, 'spin_set':{ 'a', 'b', 'c', 'd', 'e', 'f' } },
  { 'min':2, 'max':4, 'spin_set':{ 'g', 'h', 'i', 'j', 'k', 'l' } }
]
model = VectorAnnealing.model(qubo, offset, minmaxone=constraint)
```

Min閾値の指定

Max閾値の指定

スピンの指定

spin_setに指定したスピンにおいて、状態が1となるスピンの個数をmin閾値以上、max閾値以下にする制約条件

» min閾値 ≦ max閾値の場合、制約条件に指定されたスピンが以下の状態を満たす
min閾値 ≦ 状態が1のスピン数 ≦ max閾値

» min閾値 > max閾値の場合、制約条件に指定されたスピンが以下の状態を満たす
状態が1のスピン数 ≦ max閾値 もしくは min閾値 ≦ 状態が1のスピン数

NEC Vector Annealing モジュールの使い方

フリップオプションの記述方法

- Min max one 制約条件
  - Conditioned min max one 制約条件
    - 条件付きのmin max one制約条件はvector_modeがconstraintか、constraint_onlyの時のみ使用可能
    - ‘condition’に指定したスピンが指定した値である場合、min max one制約条件が有効となる制約
      指定した値でない場合は、min max one制約条件は無効となる

条件の定義
スピン(cnd0)が1の場合に
制約が有効となる

スピンの指定

```
constraint = [
  {'condition':('cnd0', 1), 'min':2, 'max':4, 'spin_set':{'a', 'b', 'c', 'd', 'e', 'f'}}
    {'condition':('cnd1', 0), 'min':2, 'max':4, 'spin_set':{'g', 'h', 'i', 'j', 'k', 'l'}}
]
model = VectorAnnealing.model(qubo, offset, minmaxone=constraint)
```

Min閾値の指定

Max閾値の指定

スピン(cnd0)が指定した値である場合、
min max one 制約条件が有効となる制約条件

» min閾値 ≦ max閾値の場合、制約条件に指定されたスピンが以下の状態を満たす
min閾値 ≦ 状態が1のスピン数 ≦ max閾値

» min閾値 > max閾値の場合、制約条件に指定されたスピンが以下の状態を満たす
状態が1のスピン数 ≦ max閾値 もしくは min閾値 ≦ 状態が1のスピン数

» min閾値=max閾値=1の場合
条件付きのOne hot制約条件として動作

NEC Vector Annealing モジュールの使い方

フリップオプションの記述方法

- Min max one 制約条件(旧フォーマット)
  - 従来のフォーマットでも min max one 制約条件を指定可能
    - 各制約条件に対して最初に定義する値(左側)をmin、２つ目(右側)をmaxとします

スピンの指定

```
minmax_one_list = [
   [ 1, 3, [ 'x[0][0]', 'x[0][1]', 'x[0][2]', 'x[0][3]', 'x[0][4]' ] ],
   [ 3, 1, [ 'x[1][0]', 'x[1][1]', 'x[1][2]', 'x[1][3]', 'x[1][4]' ] ],
   [ 2, 2, [ 'x[2][0]', 'x[2][1]', 'x[2][2]', 'x[2][3]', 'x[2][4]' ] ],
]
va_model = VectorAnnealing.model(qubo, offset, minmaxone=minmax_one_list)
```

Min閾値　Max閾値

複数のmin max one 制約条件を定義

» min閾値 ≦ max閾値の場合、制約条件に指定されたスピンが以下の状態を満たす
min閾値 ≦ 状態が1のスピン数 ≦ max閾値

» min閾値 > max閾値の場合、制約条件に指定されたスピンが以下の状態を満たす
状態が1のスピン数 ≦ max閾値 もしくは min閾値 ≦ 状態が1のスピン数

フリップオプションの記述方法

- Pattern 制約条件
  - Patterm制約条件はvector_modeがconstraintか、constraint_onlyの時のみ指定可能
  - 以下に示すスピンのパターンを制約条件として定義可能

  - Pattern 制約条件
    - パターン制約は実装されていません。FixedSpin制約条件を使用して記述してください

  - Conditioned pattern 制約条件
    - ‘condition’に指定したスピンが指定した値である場合、必ずスピンがpatternに指定した状態となる制約条件
      指定した値でない場合、patternに指定した制約条件は無効となる

条件の定義
スピン(cnd0)が1の場合に
制約が有効となる

スピンのパターンを定義
(a,b,c) = (0,0,0)

```
constraint = [
  { 'condition':('cnd0', 1), 'pattern':{ 'a':0, 'b':0, 'c':0 } },
  { 'condition':('cnd1', 0), 'pattern':{ 'd':1, 'e':0, 'f':1 } }
]
model = VectorAnnealing.model(qubo, offset, pattern=constraint)
```

スピン(cnd0)が1の場合に、
(a,b,c) = (0,0,0) となる制約

NEC Vector Annealing モジュールの使い方

フリップオプションの記述方法

- Pattern 制約条件
  - Prohibited pattern 制約条件
    - スピンがpatternに指定した状態以外の組み合わせとなる制約条件

```
constraint = [
  { 'prohibit':True, 'pattern':{ 'a':0, 'b':0, 'c':0 } },
  { 'prohibit':True, 'pattern':{ 'a':1, 'b':1, 'c':1 } }
]
model = VectorAnnealing.model(qubo, offset, pattern=constraint)
```

'pattern'が禁止パターンであることを指定

スピンのパターンの定義 (a,b,c) = (0,0,0)

2個の禁止パターン制約で、(a,b,c)が(0,0,0), (1,1,1)以外の状態を取る制約を定義

  - Conditioned prohibited pattern 制約条件
    - 'condition'に指定したスピンが指定した値である場合、prohibited pattern制約が有効となる
      指定した値でない場合、prohibited pattern制約が無効となる

```
constraint = [
  { 'condition':('cnd0', 1), 'prohibit':True, 'pattern':{ 'a':0, 'b':0, 'c':0 } },
  { 'condition':('cnd1', 0), 'prohibit':True, 'pattern':{ 'd':1, 'e':1, 'f':1 } }
]
model = VectorAnnealing.model(qubo, offset, pattern=constraint)
```

条件の定義 スピン(cnd0)が1の場合に制約が有効となる

'pattern'が禁止パターンであることを指定

スピンのパターンの定義 (a,b,c) = (0,0,0)

スピン(cnd0)が1の場合に、(a,b,c) が (0,0,0) 以外となる制約

NEC Vector Annealing モジュールの使い方

フリップオプションの記述方法

- Pattern 制約条件
  - Pattern equi spin 制約条件
    - スピンがPatternに指定した状態である場合、equi_spinに指定したスピンの状態が1となり、それ以外の場合は0となる制約条件

スピンがpatternに指定した状態を満たす場合、スピン(equi)が1となり、満たさない場合、スピン(equi)が0となる

```
constraint = [
  { 'equi_spin':'equi', 'pattern':{ 'a':0, 'b':0, 'c':0 } },
  { 'equi_spin':'cnd1', 'pattern':{ 'd':1, 'e':1, 'f':1 } }
]
model = VectorAnnealing.model(qubo, offset, pattern=constraint)
```

- Conditioned min max one 制約等の条件付き制約条件と組み合わせることで、スピンが特定のpatternを満たす時に、別の制約条件を有効とする記述が可能

スピンがpatternに指定した状態を満たす場合、スピン(cnd0)が1となり、conditioned min max oneが有効となる

```
constraint1 = [
  { 'equi_spin':'cnd0', 'pattern':{ 'a':0, 'b':0, 'c':0 } }
]
constraint2 = [
  { 'condition':('cnd0', 1), 'min':2, 'max':4, 'spin_set':{ 'g', 'h', 'i', 'j', 'k', 'l' } }
]
model = VectorAnnealing.model(qubo, offset, pattern=constraint1, minmaxone=constraint2)
```

NEC Vector Annealing モジュールの使い方

フリップオプションの記述方法

- Weighted sum 制約条件
  - Weighted sum 制約条件はvector_modeがconstraintか、constraint_onlyの時のみ指定可能
  - スピンと重みのグループを定義し、指定したグループ内で、状態が1となるスピンの重みの総和の範囲を指定する制約条件

  - Weighted sum制約条件
    - 重み付き総和の範囲を指定するパラメータをweighted_sum[“comparison”]で指定
      - » 重み付き総和 ≦ 比較対象値である場合
        “comparison”:(VectorAnnealing.COMPARISON_OPERATOR_LESS_OR_EQUAL, 比較対象値)で指定
      - » 重み付き総和 = 比較対象値である場合
        “comparison”:(VectorAnnealing.COMPARISON_OPERATOR_EQUAL, 比較対象値)で指定
      - » 重み付き総和 ≧ 比較対象値である場合
        “comparison”:(VectorAnnealing.COMPARISON_OPERATOR_GREATER_OR_EQUAL, 比較対象値)で指定

スピン名1:係数1, スピン名2:係数2, … の重み付き総和

1*a + 2*b <= 1

1*c + 2*d = 2

1*e + 2*f >=2

```
constraint = [
    {'weight':{'a':1, 'b':2}, 'comparison':(VectorAnnealing.COMPARISON_OPERATOR_LESS_OR_EQUAL, 1) },
    {'weight':{'c':1, 'd':2}, 'comparison':(VectorAnnealing.COMPARISON_OPERATOR_EQUAL, 2) },
    {'weight':{'e':1, 'f':2}, 'comparison':(VectorAnnealing.COMPARISON_OPERATOR_GREATER_OR_EQUAL, 2) }
]
model = VectorAnnealing.model(qubo, 0, weighted_sum=constraint)
```

- 重みの係数と比較対象値は整数のみで実数は扱えません。ただし負の値は扱えます。

NEC Vector Annealing モジュールの使い方

フリップオプションの記述方法

- Weighted sum 制約条件
  - Conditioned weighted sum 制約条件
    - –‘condition’に指定したスピンが指定した値である場合、weighted sum制約が有効となる
      指定した値でない場合、weighted sum制約が無効となる

条件の定義
スピン(cnd0)が1の場合に
制約が有効となる

比較演算子、比較対象値

```
constraint = [
    {'condition':('cnd0', 1), 'weight':{'a':1, 'b':2}, 'comparison':(VectorAnnealing.COMPARISON_OPERATOR_LESS_OR_EQUAL, 1)},
    {'condition':('cnd1', 0), 'weight':{'c':1, 'd':2}, 'comparison':(VectorAnnealing.COMPARISON_OPERATOR_EQUAL, 2)},
    {'condition':('cnd2', 1), 'weight':{'e':1, 'f':2}, 'comparison':(VectorAnnealing.COMPARISON_OPERATOR_GREATER_OR_EQUAL, 2)}
]
model = VectorAnnealing.model(qubo, 0, weighted_sum=constraint)
```

スピン名1:係数1, スピン名2:係数2,
… の重み付き総和

スピン(cnd0)が指定された値の場合に、
weighted sum制約条件が有効となる

NEC Vector Annealing モジュールの使い方

vector_modeのconstraintとconstraint_onlyについて

- フリップオプションを指定することにより、制約を満たすパターンを見つけます。
  （ただし、矛盾した設定のために制約を満足する解が無い場合は、無視されます）
  そのため、このモードを使用する場合はフリップオプションに関する定式を
  ハミルトニアンに含める必要はございません。
- 目的関数はなく、フリップオプションで指定する制約条件しかない問題では、constraint_only
  をご使用ください。
  制約条件を満たした時点で探索を終了します（アニーリングは行いません）。
  この場合、ハミルトニアンは不要になりますので、model(qubo, offset)で指定する
  quboとoffsetの入力は以下のようにしてください。

```
qubo = {}
offset = 0
va_model = VectorAnnealing.model(qubo, offset, onehot=onehot_list, weighted_sum=weight_sum_list)
sampler = VectorAnnealing.sampler()
```

補足）

- constraint_only：アニーリングしないため、目的関数などのハミルトニアンを入れなくても動きます
- constraint：目的関数などのハミルトニアンを入れないと動きません

- 本モードでしか使えない以下のフリップオプションがあります。
  - Pattern：禁止するスピンのパターンを制約に追加できます
  - Weighted sum：重み付き総和の値の条件を制約に追加できます
  - Conditioned min max one:min閾値 ≦ 状態が1のスピン数 ≦ max閾値を満たす制約を追加できます。

NEC Vector Annealing モジュールの使い方

QUBOでは2次項までしか扱えませんでしたが、3次以上の高次項もそのまま扱えるようになりました

- QUBOで3次項以上がハミルトニアンに出てきた場合、補助スピンと制約式を追加して2次項に変換してからアニーリングしていましたが、高次項のままアニーリングできるようになりました。
  高次項のスピン間のエネルギーはハミルトニアンではなく、以下のhigh_orderオプションで指定してください。

x[0][0]*x[0][1]*x[0][2]の係数は3.0

x[1][0]*x[1][1]*x[1][2]*x[1][3]の係数は4.0

```
high_order_list = {
    ('x[0][0]', 'x[0][1]', 'x[0][2]') : 3.0,
    ('x[1][0]', 'x[1][1]', 'x[1][2]', 'x[1][3]') : 4.0,
    ('x[2][0]', 'x[2][1]', 'x[2][2]', 'x[2][3]', 'x[2][4]') : -5.0
}

va_model = VectorAnnealing.model(qubo, offset, high_order=high_order_list)
```

複数のhigh_orderを定義

x[2][0]*x[2][1]*x[2][2]*x[2][3]*x[2][4]の係数は-5.0

- { ('x', 'y', 'z', 'z') : -1 } のような項の指定は、同じスピンの積はz*z = z の関係があるため、{ ('x', 'y', 'z') : -1 } と書く必要があります。なお3次項以上が設定可能です。

NEC Vector Annealing モジュールの使い方

初期スピン状態の指定

- 以下の3通りの方法で、アニーリング開始時のスピンの状態を指定することが可能です
  - モデルに対してFixedSpin制約条件を指定
    - Fixed Spin 制約条件は、スピンが固定であることを指定する制約条件であるため、指定したスピンの状態を初期スピン状態にも適用します
  - モデルに対して初期スピン状態を指定
  - アニーリング実行時に初期スピン状態を指定

- 動作
  - モデル・アニーリング実行時に指定した初期スピン状態を初期値として使用します
  - 同一スピンに対して異なる値を指定した場合、以下の優先順位で値を設定してください
    1. Fixed spin 制約条件に指定した状態
    2. アニーリング実行時に指定した初期スピン状態
    3. モデルに対して指定した初期スピン状態

NEC Vector Annealing モジュールの使い方

初期スピン状態の指定

- スピンのモデルに対して初期スピン状態を指定する場合

```
spin_state = {
  'x[0][0]':0,
  'x[1][1]':1,
  'x[2][3]':1,
  …
}

va_model = VectorAnnealing.model(qubo, offset, init_spin=spin_state)
```

スピン名とスピンの状態(0,1)の組み合わせでスピン状態を指定
指定しないスピンは初期値なしとして扱い、アニーリング開始時に乱数で値を決定します

スピンの初期値を格納した辞書型のデータをinit_spinに指定

- アニーリング実行時に指定する場合

```
spin_state = {
  'x[0][0]':0,
  'x[1][1]':1,
  'x[2][3]':1,
  …
}

sa = VectorAnnealing.sampler()
result = sa.sample(va_model, num_reads=5, init_spin=spin_state)
```

スピン名とスピンの状態(0,1)の組み合わせでスピン状態を指定
指定しないスピンは初期値なしとして扱い、アニーリング開始時に乱数で値を決定します

スピンの初期値を格納した辞書型のデータをinit_spinに指定

デバッグ方法

内部処理の詳細を出力します

- 環境変数「VA_LOG_LEVEL=LOG_INFO_DETAIL」を指定して実行

【実行例】

```
$ export VA_LOG_LEVEL=LOG_INFO_DETAIL
$ cd samples/TSP/python
$ python3.6m ./tsp.py ../data/eil51.tsp
2022-07-06 19:39:51.726 [Log] VA_LOG_LEVEL = 5
2022-07-06 19:39:51.726 [Log] VA_LOG_FILE = [stderr]
...

2022-07-06 19:40:28.628 [va_qubo:OUT]   Absolute weight range : 1.000000e+00 - 8.000000e+01
2022-07-06 19:40:28.628 [va_qubo:OUT]   Max weight            : 8.000000e+01
2022-07-06 19:40:28.628 [va_qubo:OUT]   Scale                 : 1.000000e+00
2022-07-06 19:40:28.628 [va_qubo:OUT]   Offset                : 1.315800e+03
2022-07-06 19:40:28.628 [va_qubo:OUT] Matrix:sparse  beta:[10.000000 - 100.000000 / 200]  iteration:1000  threads:8
ASL(seed):718363418
2022-07-06 19:40:28.628 [va_qubo:OUT] 50: 204.000122 beta 17.629141 broken 0  accept 1.485006%
2022-07-06 19:40:28.628 [va_qubo:OUT] 100: 133.000122 beta 21.711180 broken 0  accept 0.177816%
2022-07-06 19:40:28.942 [va_qubo:OUT] 150: 117.000122 beta 29.331663 broken 0  accept 0.081699%
2022-07-06 19:40:29.253 [va_qubo:OUT] 200: 117.000122 beta 37.834625 broken 0  accept 0.146578%
2022-07-06 19:40:29.565 [va_qubo:OUT] 250: 117.000122 beta 41.504047 broken 0  accept 0.108131%
2022-07-06 19:40:29.876 [va_qubo:OUT] 300: 117.000122 beta 38.720387 broken 0  accept 0.068483%
2022-07-06 19:40:30.185 [va_qubo:OUT] 350: 117.000122 beta 33.700642 broken 0  accept 0.130959%
2022-07-06 19:40:30.494 [va_qubo:OUT] 400: 117.000122 beta 27.364401 broken 0  accept 0.075692%
2022-07-06 19:40:30.803 [va_qubo:OUT] 450: 117.000122 beta 23.816856 broken 0  accept 0.139369%
2022-07-06 19:40:31.112 [va_qubo:OUT] 500: 117.000122 beta 24.945082 broken 0  accept 0.109333%
2022-07-06 19:40:31.417 [va_qubo:OUT] 550: 117.000122 beta 31.440355 broken 0  accept 0.068483%
2022-07-06 19:40:31.721 [va_qubo:OUT] 600: 117.000122 beta 27.364401 broken 0  accept 0.115340%
2022-07-06 19:40:32.028 [va_qubo:OUT] 650: 117.000122 beta 35.297073 broken 0  accept 0.091311%
2022-07-06 19:40:32.336 [va_qubo:OUT] 700: 117.000122 beta 47.686115 broken 0  accept 0.129758%
...
```

Sweep数毎に左からエネルギー値、β値、制約を満たしているか(0なら満たしている)、スピン反転率を表示

デバッグ方法

va_qubo.exe の入出力ファイルを確認します

- 環境変数「DEBUG=true」を指定して実行してください
  - 中間ファイルを残します

- 環境変数「VA_QUBO_FILE_TEXT=true」を指定して実行してください
  - 中間ファイルをテキスト形式で出力します

【実行例】

```
$ export VA_LOG_LEVEL=LOG_INFO_DETAIL
$ export DEBUG=true
$ export VA_QUBO_FILE_TEXT=true
$ cd samples/TSP/python
$ python3.6 ./tsp.py ../data/eil51.tsp

  ...

2020-05-13 16:48:27.388 [VectorAnnealing.sampler] Output qubo file time: 0.133 sec
2020-05-13 16:48:27.388 [VectorAnnealing.sampler] run : ['/home/user/.local/libexec/VectorAnnealing/bin/wrap',
 '/home/user/.local/libexec/VectorAnnealing/python/VectorAnnealing/../../../../bin/va_qubo.exe', '-f',
 './qubo-27668.txt', '-o', './out-27668.txt', '-B', '-r', '5', '-i', '1000', '-s', '0.2', '-e', '1000']
2020-05-13 16:48:33.510 [va_qubo] trix:sparse  beta:[0.200000 - 1000.000000 / 200]  iteration:1000  threads:8
ASL
2020-05-13 16:48:33.510 [va_qubo] 1.62866 broken 1
2020-05-13 16:48:36.592 [va_qubo] 100: 493 beta 13.8427 broken 0
2020-05-13 16:48:38.352 [va_qubo] 150: 254 beta 17.8956 broken 0
```

コマンド名(va_qubo.exe)

入力ファイル

出力ファイル

デバッグ方法

スピン名とスピン番号の対応を固定で設定します

- Modelの初期化で、spin_listにスピン名の配列を指定します
  - spin_listに指定した配列に格納した順で、スピン名にスピン番号を０から順に割り振ります

```
spin_name_list = None
if debug :
    spin_name_list = [ ]
    for i in range(point_num):
        for j in range(point_num):
            spin_name_list.append('x[%d][%d]' % (i, j))

va_model = VectorAnnealing.model(qubo, offset, onehot=onehot, fixed=fixed, spin_list=spin_name_list)
```

デバッグ方法

乱数のシードの設定

- va_qubo.exe は疑似乱数を用いてアニーリングの処理を行うため、毎回結果が異なります
- spin_listの指定と合わせて、疑似乱数のシード(正の整数)を設定することで、同じデータ・パラメータに対して、必ず同じ結果を得ることが可能になります
  - va_qubo.exe のオプション(-S) を指定

```
$ va_qubo.exe -f ./tsp_qubo-142640.bin -o ./out-142640.bin -i 10000 -s 0.200000 -e 1000.000000 -S 0
```

  - アニーリング実行時に seed パラメータを指定する(python)

```
spin_list = [ スピン1, スピン2, スピン3, … ]
va_model = VectorAnnealing.model(qubo, offset, onehot=onehot, fixed=fixed, spin_list=spin_list)
sa       = VectorAnnealing.sampler()

result   = sa.sample(va_model, num_reads=5, seed=0)
```

Python API

実装

- Module構成

パス(インストールパスからの相対パス)	処理
./libexec/VectorAnnealing/python/VectorAnnealing/__init__.py	
./libexec/VectorAnnealing/python/VectorAnnealing/log.py	ログ出力
./libexec/VectorAnnealing/python/VectorAnnealing/model.py	QUBOファイルの出力、結果ファイルの入力処理
./libexec/VectorAnnealing/python/VectorAnnealing/parameter.py	パラメータの定義
./libexec/VectorAnnealing/python/VectorAnnealing/resource.py	リソース使用量取得
./libexec/VectorAnnealing/python/VectorAnnealing/result.py	アニーリングの結果格納用クラス
./libexec/VectorAnnealing/python/VectorAnnealing/sampler.py	アニーリング実行関数
./libexec/VectorAnnealing/python/VectorAnnealing/valib.py	C++ APIの呼び出し

Python API：log.py

logクラス (log.py)

- VA_LOG_LEVELやVA_LOG_FILEを環境変数で設定することで、ログの出力を制御できます

- 環境変数

  - VA_LOG_LEVEL
    - Default は LOG_WARNING とします

値	数値	動作
LOG_NONE	0	ログ出力無し
LOG_MESSAGE	1	メッセージを出力
LOG_ERROR	2	LOG_MESSAGE + ERRORを出力
LOG_WARNING	3	LOG_ERROR + WARNINGを出力
LOG_INFO	4	LOG_WARNING + 実行時の情報を出力
LOG_INFO_DETAIL	5	LOG_INFO + 詳細情報を出力 ※障害が発生した際の原因調査等に本ログレベルを使用いたします

  - VA_LOG_FILE
    - ログを指定したファイルに出力します
    - Default は標準エラー出力とします

Python API：model.py

model クラス (model.py)

- 関数一覧

関数	動作
__init__()	モジュールの初期化を行います
__del__()	内部変数の解放を行います
get_num_spins()	スピン数を取得します
get_energy()	エネルギー値を計算します

Python API：model.py

model クラス (model.py)

- __init__(self, qubo,
  offset, onehot=None, fixed=None, andzero=None, orone=None,
  supplement=None, maxone=None, minmaxone=None,
  pattern=None, weighted_sum=None, high_order=None,
  init_spin=None, spin_list=None)

※　=のないオプションは必須

- 動作
  - 指定したパラメータでmodelクラスを初期化します

- パラメータ
  - qubo (辞書型)
    - » QUBO 形式のデータを指定します

スピン名*1	=	スピンの名称(文字列型)
スピンの組	=	(スピン名1，スピン名2)*2
相互エネルギー	=	スピン間の相互エネルギー値(浮動小数点型)
QUBO	=	**{ スピンの組1:相互エネルギー1，スピンの組2:相互エネルギー2，… }**

  - offset (実数型)
    - » QUBO に格納した正規化された重み情報の offset を指定します

*1) スピン名に、'x'とb'x'のように文字列型とバイト列型を混在して指定した場合、別のスピンとして扱われるので注意してください。
*2) スピンの組が対角項の場合は、(スピン名1, スピン名1)のように同じスピン名を指定してください。

Python API：model.py

model クラス (model.py)

- __init__(self, qubo, offset …)
  - パラメータ
    - onehot
      - » ONE HOT 制約条件のパラメータを以下の形式で設定します
        (グループ内のスピンのうち、1個のスピンのみ状態が1となり、それ以外のスピンの状態は0となります)
      - » 制約条件を設定しない場合は None を指定します(省略した場合は制約なし(None)となります)

```
スピン名*1          = スピンの名称(文字列型)
制約                = [ スピン名1, スピン名2, スピン名3, … ]
ONE HOT 制約条件    = [ 制約1, 制約2, 制約3, … ]
```

    - fixed
      - » FIXED 制約条件のパラメータを以下の形式で指定します
        (スピンの状態は、必ず、指定した値(0 or 1)を取ります)
      - » 制約条件を設定しない場合は None を指定します(省略した場合は制約なし(None)となります)

```
スピン名*1              = スピンの名称(文字列型)
スピンの状態            = 0 or 1 (整数型)
FIXED制約条件(辞書型)   = { スピン名1:スピンの状態1, スピン名2:スピンの状態2, … }

制約                    = [ スピン名, スピンの状態 ]
FIXED制約条件(配列)     = [ 制約1, 制約2, … ]
```

*1) スピン名に、'x'とb'x'のように文字列型とバイト列型を混在して指定した場合、別のスピンとして扱われるので注意してください。

Python API：model.py

model クラス (model.py)

- __init__(self, qubo, offset …)
  - パラメータ
    - andzero
      - » AND ZERO 制約条件のパラメータを以下の形式で指定します
        （グループ内のスピンの状態の論理積が必ず0となります(少なくとも、どれか1個のスピンは状態が0となります)）
      - » 制約条件を設定しない場合は None を指定します(省略した場合は制約なし(None)となります)

```
スピン名*1            = スピンの名称(文字列型)
制約                  = [ スピン名1, スピン名2, … ]
AND ZERO 制約条件     = [ 制約1, 制約2, … ]
```

    - orone
      - » OR ONE 制約条件のパラメータを以下の形式で指定します
        （グループ内のスピンの状態の論理和が必ず1となります(少なくとも、どれか1個のスピンは状態が1となります)）
      - » 制約条件を設定しない場合は None を指定します(省略した場合は制約なし(None)となります)

```
スピン名*1            = スピンの名称(文字列型)
制約                  = [ スピン名1, スピン名2, … ]
OR ONE 制約条件       = [ 制約1, 制約2, … ]
```

*1) スピン名に、'x'とb'x'のように文字列型とバイト列型を混在して指定した場合、別のスピンとして扱われるので注意してください。

Python API：model.py

model クラス (model.py)

- __init__(self, qubo, offset …)
  - パラメータ
    - supplement
      - » supplement(CUBIC SUPPLEMENT制約条件) は、スピン y_1, x_1, x_2 に対して、スピンの状態が以下の条件を満たすことを制約とします

        $y_1 = x_1 x_2$
      - » CUBIC SUPPLEMENT 制約条件のパラメータを以下の形式で指定します
      - » 制約条件を設定しない場合は None を指定します(省略した場合は制約なし(None)となります)
      - » **本パラメータは、high_orderと同時に指定できません**

```
スピン名*1                  = スピンの名称(文字列型)
制約                        = [ スピン名(y1), スピン名(x1), スピン名(x2) ]
CUBIC SUPPLEMENT 制約条件   = [ 制約1, 制約2, 制約3, … ]
```

    - maxone
      - » MAX ONE COUNT 制約条件のパラメータを以下の形式で指定します
        (グループ内のスピンにおいて、状態が1となるスピンの個数が閾値以下となります)
      - » 制約条件を設定しない場合は None を指定します(省略した場合は制約なし(None)となります)

```
スピン名*1          = スピンの名称(文字列型)
閾値                = 状態が1となるスピンの個数の最大値
制約                = [ 閾値, [ スピン名1, スピン名2, スピン名3 ] ]
MAX ONE 制約条件    = [ 制約1, 制約2, 制約3, … ]
```

*1) スピン名に、'x'とb'x'のように文字列型とバイト列型を混在して指定した場合、別のスピンとして扱われるので注意してください。

Python API：model.py

model クラス (model.py)

- __init__(self, qubo, offset …)
  - パラメータ
    - minmaxone
      - » MIN MAX ONE制約条件のパラメータを以下の形式で指定します
      - » ‘condition‘にスピン名・スピンの状態を指定した場合、スピンが指定した状態を満たす場合にその制約が有効となります。指定した状態を満たさない場合は、制約は無効となります。’condition’を省略した場合は、常に制約が有効となります。
        ‘condition’の指定は、vector_modeがconstraintか、constraint_onlyの時のみ使用でき、辞書型のminmaxone制約条件でのみ指定可能です。
      - » 閾値1 < 閾値2 の場合、以下の制約となります
        グループ内のスピンにおいて、状態が1となるスピンの個数が閾値1以上、閾値2以下
      - » 閾値1 > 閾値2 の場合、以下の制約となります
        グループ内のスピンにおいて、状態が1となるスピンの個数が閾値2以下、もしくは、閾値1以上
      - » 制約条件を設定しない場合は None を指定します(省略した場合は制約なし(None)となります)

```
スピン名*1                    = スピンの名称(文字列型)
スピンの状態                  = 0 or 1(整数型)
条件                          = (スピン名, スピンの状態)          スピンが指定した状態をを満たす時、制約が有効となる条件
閾値1                         = 状態が1となるスピンの個数の最小値
閾値2                         = 状態が1となるスピンの個数の最大値
スピングループ                = (スピン名1, スピン名2, スピン名3)
制約                          = { 'condition':条件, ¥             constraintか、constraint_onlyの時のみ使用可能、省略可能
                                  'min':閾値1, 'max':閾値2, ¥
                                  'spin_set':(スピン名1, スピン名2, スピン名3) }
MIN MAX ONE 制約条件(辞書型)  = [ 制約1, 制約2, 制約3, … ]
MIN MAX ONE 制約条件(辞書型)  = [ 制約1, 制約2, 制約3, … ]

スピン名*1                    = スピンの名称(文字列型)
閾値1                         = 状態が1となるスピンの個数の最小値
閾値2                         = 状態が1となるスピンの個数の最大値
制約                          = [ 閾値1, 閾値2, [ スピン名1, スピン名2, スピン名3 ] ]
MIN MAX ONE 制約条件(配列)    = [ 制約1, 制約2, 制約3, … ]
```

*1) スピン名に、‘x’とb’x’のように文字列型とバイト列型を混在して指定した場合、別のスピンとして扱われるので注意してください。

Python API：model.py

model クラス (model.py)

- __init__(self, qubo, offset …)
  - パラメータ
    - Pattern
      - » Pattern 制約条件は vector_mode がconstraintか、constraint_onlyの時のみ使用できます
      - » Pattern 制約条件のパラメータを以下の形式で指定します
      - » ‘condition‘にスピン名・スピンの状態を指定した場合、スピンが指定した状態を満たす場合にその制約が有効となります。指定した状態を満たさない場合は、制約は無効となります。’condition’を省略した場合は、常に制約が有効となります。
      - » ‘equi_spin’にスピン名を指定した場合、スピンがスピンパターン指定した状態を満たす場合、指定したスピンの状態を1に設定します。状態を満たさない場合は0 を指定します。’equi_spin’の設定は省略することができます。
      - » ‘prohibit’にスピンパターンのモードを指定します。’prohibit’にTrueを設定した場合、スピンがスピンパターンに指定した状態を満たさない制約条件(指定したスピンのパータン以外となる制約条件)となります。Falseを指定した場合、もしくは、’prohibit’を省略した場合は、スピンがスピンパターンを満たす制約条件となります。
      - » 制約条件を設定しない場合は None を指定します(省略した場合は制約なし(None)となります)

```
スピン名*1          = スピンの名称(文字列型)
スピン状態          = 0 or 1 (整数型)
条件                = (スピン名, スピンの状態)          スピンが指定した状態をを満たす時、制約が有効となる条件
モード              = True or False(Boolean)
スピンパターン      = { スピン名1:スピンの状態1, スピン名2:スピンの状態2, ... }
制約                = { 'condition':条件, ¥              省略可能
                        'equi_spin':スピン名, ¥         省略可能
                        'prohibit':モード, ¥            省略可能
                        'pattern':スピンパターン }
PATTERN 制約条件    = [ 制約1, 制約2, 制約3, ... ]
```

*1) スピン名に、‘x’とb‘x’のように文字列型とバイト列型を混在して指定した場合、別のスピンとして扱われるので注意してください。

Python API：model.py

model クラス (model.py)

- __init__(self, qubo, offset …)
  - パラメータ
    - Pattern
      - » Pattern 制約条件は以下の組合わせでパラメータを指定する必要があります。

condition	equi_spin	prohibit	pattern	動作
省略	省略	False/省略	指定	エラー(指定不可)
省略	省略	指定	指定	スピンがPatternに指定した状態意外となる制約条件
省略	指定	False/省略	指定	スピンがPatternに指定した状態を満たす場合、equi_spinに指定したスピンの状態が1となる制約条件
省略	指定	指定	指定	エラー(指定不可)
指定	省略	False/省略	指定	conditionに指定した条件を満たす場合、スピンがpatternに指定したパターンとなる制約条件
指定	省略	指定	指定	conditionに指定した条件を満たす場合、スピンがpatternに指定したパターン以外となる制約条件
指定	指定	False/省略	指定	エラー(指定不可)
指定	指定	指定	指定	エラー(指定不可)

*1) スピン名に、'x'とb'x'のように文字列型とバイト列型を混在して指定した場合、別のスピンとして扱われるので注意してください。

Python API：model.py

model クラス (model.py)

- __init__(self, qubo, offset …)
  - パラメータ
    - weighted_sum
      - » Weighted sum 制約条件は vector_mode がconstraintか、constraint_onlyの時のみ使用できます
      - » Weighted sum 制約条件のパラメータを以下の形式で指定します
      - » ‘condition‘にスピン名・スピンの状態を指定した場合、スピンが指定した状態を満たす場合にその制約が有効となります。指定した状態を満たさない場合は、制約は無効となります。’condition’を省略した場合は、常に制約が有効となります。
      - » ‘weight’にスピン名をキーとしてスピンに対する係数を辞書型で指定します。ここに指定した値は以下に示す式で重み付き総和を求め、’comparison’に指定した条件と比較します。
        重み付き総和 = Σ(スピンの状態 × スピンに対する係数)
      - » ‘comparison’には、比較演算子と比較対象値を指定します。比較演算子には以下に表のいずれかの値をしています。

比較演算子	動作
VectorAnnealing.COMPARISON_OPERATOR_LESS_OR_EQUAL	重み付き総和 ≦ 比較対象値を満たす制約条件
VectorAnnealing.COMPARISON_OPERATOR_EQUAL	重み付き総和 = 比較対象値を満たす制約条件
VectorAnnealing.COMPARISON_OPERATOR_GREATER_OR_EQUAL	重み付き総和 ≧ 比較対象値を満たす制約条件

      - » 制約条件を設定しない場合は None を指定します(省略した場合は制約なし(None)となります)

Python API：model.py

model クラス (model.py)

- __init__(self, qubo, offset …)
  - パラメータ
    - weighted_sum

```
スピン名*1              = スピンの名称(文字列型)
スピン状態              = 0 or 1 (整数型)
制約条件                = (スピン名, スピンの状態)          スピンが指定した状態をを満たす時、制約が有効となる条件
係数                    = 整数型(-2147483648以上、2147483647未満の値を指定)
比較対象値              = 整数型(-2147483648以上、2147483647未満の値を指定)
比較演算子              = 以下の値のいずれか
 VectorAnnealing.COMPARISON_OPERATOR_LESS_OR_EQUAL      重み付き総和 ≦ 比較対象値である場合
 VectorAnnealing.COMPARISON_OPERATOR_EQUAL              重み付き総和 =  比較対象値である場合
 VectorAnnealing.COMPARISON_OPERATOR_GREATER_OR_EQUAL   重み付き総和 ≧ 比較対象値である場合
制約                    = { 'condition'制約条件,          constraint, constraint_onlyモードでの仕様可能、省略可能
                            'comparison':(比較演算子, 比較対象値),
                            'weight':{ スピン名1:係数1, スピン名2:係数2, ... } }
WEIGHTED SUM 制約条件 = [ 制約1, 制約2, 制約3, … ]
```

*1) スピン名に、'x'とb'x'のように文字列型とバイト列型を混在して指定した場合、別のスピンとして扱われるので注意してください。

Python API：model.py

model クラス (model.py)

- __init__(self, qubo, offset …)
  - パラメータ
    - high_order
      - 3次以上の項(高次項--3個以上のスピン間)のエネルギーについて、スピン名をキーとした辞書型でエネルギーを指定します
      - **本パラメータは、supplementオプションと同時に指定できません。**
        supplementオプションでは、3次以上の項の関係を示すために補助スピンを用いていたが、high_orderでは、直接それらの関係を表現できます。
      - high_orderのパラメータを以下の形式で指定します。

```
スピン名*1                    = スピンの名称(文字列型)
多次項の係数                  = 値(浮動小数型)
多次項のスピンの組み合わせ    = (スピン名1, スピン名2, スピン名3, … )
high_order                    = { 多次項のスピンの組み合わせ1:多次項の係数1,
                                  多次項のスピンの組み合わせ2:多次項の係数2,
                                  … }
```

*1) スピン名に、'x'とb'x'のように文字列型とバイト列型を混在して指定した場合、別のスピンとして扱われるので注意してください。

Python API：model.py

model クラス (model.py)

- __init__(self, qubo, offset …)
  - パラメータ
    - init_spin
      - » 以下に示すように、スピン名をキーとしてスピンの状態を格納した辞書型で、初期スピン状態を指定します
      - » 初期値を指定しないスピンは、アニーリング開始時に乱数で値を初期化します
      - » sampler.sample()のパラメータでも初期スピンを設定した場合は、samplerで指定した初期スピン状態でスピンを初期化します
      - » 本パラメータを指定しない場合、すべてのスピンを乱数で初期化します

```
スピン名*1          = スピンの名称(文字列型)
スピンの状態        = 0 or 1 (整数型)
init_spin(辞書型) = { スピン名1:スピンの状態1, スピン名2:スピンの状態2, … }

スピン              = [ スピン名, スピンの状態 ]
init_spin(配列)    = [ スピン1, スピン2, … ]
```

    - spin_list
      - » スピン名の配列を指定します
      - » 指定された場合、配列に格納されたスピン名の順で、0から順にスピン番号を割り振ります
        - 実行ごとにスピン番号とスピン名の対応が異なる問題を回避するために導入(デバッグ用のパラメータ)

```
スピン名*1   = スピンの名称(文字列型)
spin_list    = [ スピン名1, スピン名2, … ]
```

*1) スピン名に、'x'とb'x'のように文字列型とバイト列型を混在して指定した場合、別のスピンとして扱われるので注意してください。

Python API：model.py

model クラス (model.py)

- __del__(self)
  - 動作
    - 内部変数の解放を行います

Python API：model.py

model クラス (model.py)

- get_num_spins(self)
  - 動作
    - インスタンスに設定されたモデルのスピン数を返す。
  - 戻り値
    - スピン数

*1) スピン名に、‘x’とb’x’のように文字列型とバイト列型を混在して指定した場合、別のスピンとして扱われるので注意してください。

Python API：model.py

model クラス (model.py)

- get_energy(self, spin)
  - 動作
    - スピンがSpinに指定された状態の時のエネルギーの総和を返す。
  - パラメータ
    - Spin (辞書型)
      - » スピンとスピンの状態を辞書型で指定する。

```
スピン名*1   = スピンの名称(文字列型)
スピン状態   = 0 or 1 (整数型)
spin         = { スピン名1:係数1, スピン名2:係数2, ... } }
```

  - 戻り値
    - エネルギーの総和
  - エラー
    - スピン名・スピンの状態のデータ型が不正な場合、TypeError例外を発行する。
    - スピンの状態が0/1以外の場合、ValueError例外を発行する。
    - 引数Spinにモデルで定義したスピンが全て設定されていない場合、KeyError例外を発行する。

*1) スピン名に、'x'とb'x'のように文字列型とバイト列型を混在して指定した場合、別のスピンとして扱われるので注意してください。

Python API：result.py

result クラス (result.py)

- 関数一覧

関数	動作
__init__()	モジュールの初期化を行う

- __init__(self, spin=None, energy=None, time=None, constraint=None, memory_usage=None)

※　=のないオプションは必須

  - 動作
    - resultクラスに結果を格納します

  - パラメータ
    - spin (辞書型)
      - スピン名をキーとして、スピンの状態を格納した辞書型のデータ

```
スピン名      = スピンの名称(文字列型)
スピンの状態 = 0 or 1 (整数型)
spin          = { スピン名1:スピン1の状態, スピン名2:スピン2の状態, … }
```

    - energy (実数型)
      - QUBOの総エネルギー値
    - time
      - アニーリングの演算時間
    - constraint
      - 制約条件の状態(スピンの状態が制約条件を満たす場合はTrue)
    - memory_usage
      - アニーリングの演算で使用したメモリの使用量(GiB単位)

Python API：sampler.py

sampler クラス (sampler.py)

- 関数一覧

関数	動作
__init__()	環境変数を取得し、モジュールの初期化を行います
__del__()	内部変数の解放を行います
sample()	va_qubo.exeを実行し、アニーリングの結果を返します

- __init__(self)
  - 動作
    - モジュールの初期化を行います

- __del__(self)
  - 動作
    - va_qubo.exe の入出力ファイルを削除します

Python API：sampler.py

sampler クラス (sampler.py)

- sample(self, model, num_reads=1, num_results=None, num_sweeps=None,
  beta_range=None, beta_list=None, init_spin=None, dense=None,
  seed=None, vector_mode=None, precision=None,
  num_threads=None, recalculate_energy=True)　※　=のないオプションは必須
  - 動作
    - va_qubo.exe を実行し、num_reads に指定した回数アニーリングを行います

Python API：sampler.py

sampler クラス (sampler.py)

- sample(self, model, …)
  - パラメータ
    - model (model クラス)
      - » VectorAnnealing.model モジュールを指定してください
    - num_reads (整数型)
      - » アニーリング回数を指定してください (0以上)
      - » 0 を指定した場合は、アニーリングを行わず、QUBO ファイルの作成のみを行います (default = 1)
    - num_results (整数型)
      - » None もしくは1を指定した場合、最適解を1個返します
      - » num_reads と同じ値を指定した場合、全アニーリング結果を返します
      - » それ以外の場合は、次のエラー表示をします　”Invalid number of result.”
    - num_sweeps (整数型)
      - » アニーリングの sweep 数を指定(1以上)してください
      - » Noneを指定した場合、default値で動作します(default = 500)
    - beta_range ([実数型(start), 実数型(end), 整数型(number of steps)])
      - » beta_range の3番目のパラメータ(number of steps)は省略可能です
        省略した場合は va_qubo.exe の default 値(200)を使用します
      - » アニーリングの温度変化の範囲をβで指定(default = None)してください
        None の場合は va_qubo.exe の default 値([10, 100, 200])で動作)
    - beta_list ([実数型1, 実数型2, 実数型3, …])
      - » Sweep毎のbetaの値を配列に格納して指定してください
      - » Noneを指定した場合、beta_rangeに指定したパラメータで動作します

beta_range と beta_list を同時に指定した場合は、beta_list に指定した値を使用します

Python API：sampler.py

sampler クラス (sampler.py)

- sample(self, model, …)
  - パラメータ
    - init_spin(リスト型)
      - » スピン名をキーとしてスピンの状態を格納した辞書型で、初期スピン状態を指定してください
      - » 初期値を指定しないスピンは、アニーリング開始時に乱数で値を初期化します
      - » 本パラメータを指定しない場合、スピンのモデル作成時に指定した初期スピン配置でスピンを初期化します
        モデルにも初期スピン配置を指定しなかった場合は、すべてのスピンを乱数で初期化します

```
スピン名*1          = スピンの名称(文字列型)
スピンの状態        = 0 or 1 (整数型)
init_spin(辞書型) = { スピン名1:スピンの状態1, スピン名2:スピンの状態2, … }

スピン              = [ スピン名, スピンの状態 ]
init_spin(配列)    = [ スピン1, スピン2, … ]
```

    - dense (boolean)
      - » True の場合、内部処理を密行列で行います
      - » False の場合、内部処理を疎行列で行います
      - » None の場合、Qubo の非ゼロ要素数の密度により、疎行列・密行列のモードを決定します
    - num_threads (整数型)
      - » アニーリングで使用するスレッド数を指定します
      - » 1以上の値を指定した場合は、指定したスレッド数でアニーリングエンジンを並列実行します
      - » 0以下もしくはNone(デフォルト)の場合、環境変数OMP_NUM_THREADSに指定した値で並列実行します
      - » 環境変数OMP_NUM_THREADSも指定されていない場合は、ライセンスで使用可能なコア数もしくはシステムで使用可能なプロセッサ数の小さいほうの値を使用します
    - seed(整数型)
      - » va_qubo.exe に、乱数のシードを指定してください
      - » None(デフォルト)の場合、 va_qubo.exe のデフォルト値で動作します (実行時の時刻から乱数のシードを作成)

*1) スピン名に、'x'とb'x'のように文字列型とバイト列型を混在して指定した場合、別のスピンとして扱われるので注意してください

Python API：sampler.py

sampler クラス (sampler.py)

- sample(self, model, …)
  - パラメータ
    - vector_mode(str型)
      - » VectorAnnealing.VECTOR_MODE_SPEED の場合、速度優先でアニーリングを実行します
      - » VectorAnnealing.VECTOR_MODE_CONSTRAINT の場合、制約優先でアニーリングを実行します
      - » VectorAnnealing.VECTOR_MODE_CONSTRAINT_ONLY の場合、上記制約優先で制約を満たした時点で終了します(アニーリングは行いません)
      - » vector_mode を指定しない場合、環境変数 VA_QUBO_VECTOR_MODE の値で動作を決定します
        環境変数の値がspeedの場合は速度優先、constraintの場合は制約優先、constraint_onlyの場合は制約限定となります
      - » 環境変数も設定されていない場合は速度優先(speed)になります
    - precision(整数型)
      アニーリングの演算精度を指定します
      - » VectorAnnealing.PRECISION_COMPUTE_SINGLEの場合、単精度でアニーリングの演算を実行します
      - » VectorAnnealing.PRECISION_COMPUTE_DOUBLEの場合、倍精度でアニーリングの演算を実行します
    - recalculate_energy(boolean)
      - » True の場合、エネルギーの再計算を行います(default)
      - » False の場合、エネルギーの再計算をスキップします

Python API：sampler.py

sampler クラス (sampler.py)

- sample(self, model, …)
  - 戻り値
    - va_qubo.exeを実行し、num_reads 回アニーリングを実行します
    - 制約条件を満たし収束エネルギーが小さい順で、アニーリング結果を返します
    - アニーリング結果を以下の形式で格納し、戻り値に設定します

```
スピン名          = スピンの名称(文字列型)
スピンの状態      = 0 or 1 (整数型)
アニーリング結果  = { スピン名1:スピン1の状態,スピン名2:スピン2の状態, … }
収束エネルギー    = 収束したエネルギーの総和(実数型)
制約条件の状態    = 制約条件を満たす(True), 制約条件を満たさない(False)
演算時間          = va_qubo.exeの計算時間(実数型)
メモリ使用量      = va_qubo.exeのメモリ使用量(実数型, GiB単位)
結果              = result(spin=アニーリング結果, energy=収束エネルギー, time=演算時間,
                           constraint=制約条件の状態, memory_usage=メモリ使用量)
戻り値            = [ 結果1, 結果2, … ]
```

  - エラー
    - アニーリングに失敗した場合は、エラーメッセージを Exception クラスに設定し、例外を発行します

Python API：resource.py

resourceクラス (resource.py)

- 関数一覧

関数	動作
resource_usage()	リソース使用量をログに出力

Python API：resource.py

resourceクラス (resource.py)

- resource_usage(self, log_level=VA_LOG_INFO)
  - 動作
    - 自プロセス(python)と子プロセス(アニーリングエンジン)のメモリ/CPUリソース使用量を log_level で、ログに出力します

環境変数

環境変数

- 共通
  - TMP
    - テンポラリファイルを出力するディレクトリ(default = カレントディレクトリ)
  - DEBUG
    - 環境変数が設定されていた場合はデバッグモードを有効とします。値は何でもよいです(default = 無効)
      - » **有効の場合、アニーリング終了後に中間結果を保持します(テンポラリファイルを削除しません)**
- LOG関連
  - VA_LOG_LEVEL
    - ログ出力レベルを指定してください(default = LOG_WARNING)

値	数値	動作
LOG_NONE	0	ログ出力無し
LOG_MESSAGE	1	メッセージを出力
LOG_ERROR	2	LOG_MESSAGE + ERRORを出力
LOG_WARNING	3	LOG_ERROR + WARNINGを出力
LOG_INFO	4	LOG_WARNING + 実行時の情報を出力
LOG_INFO_DETAIL	5	LOG_INFO + 詳細情報を出力 ※障害が発生した際の原因調査等に本ログレベルを使用いたします

  - VA_LOG_FILE
    - ログ出力先のファイル名を指定します(default = 標準エラー出力)
  - VALIC_LOG_LEVEL_CLIENT
    - ライセンス処理のログ出力レベルを指定してください(default = VALIC_LOG_WARNING)
      - » **VALIC_LOG_ERROR: エラーログのみを出力**
      - » **VALIC_LOG_WARNING: エラーログ、警告ログを出力**
      - » **VALIC_LOG_INFO: エラーログ、警告ログ、情報ログを出力**
  - VALIC_LOG_FILE
    - ライセンスサービスのログ出力先のファイル名を指定します(default = 標準エラー出力)

環境変数

- va_qubo.exe 関連
  - VA_QUBO_VECTOR_MODE
    - “speed”が設定されていた場合、速度優先でアニーリングを実行します(default)
    - “constraint”を設定した場合、制約優先でアニーリングを実行します
    - “constraint_only”を設定した場合、上記制約優先で求解を行い、制約を満たした時点で終了します(アニーリングは行いません)
  - VA_QUBO_FILE_TEXT
    - 環境変数が設定されていた場合(値は何でもよいです)、ファイルをテキスト形式で出力します
      (default = バイナリ形式)
  - VA_QUBO_DENSITY_THRESHOLD
    - 疎行列・密行列のモードを決定する閾値を0 ～ 1 の割合で指定してください(default：0.03)
    - QUBOの密度が指定した閾値以上の場合、va_qubo.exeの内部の行列を密行列とし、閾値未満の場合は疎行列として扱います
  - VA_QUBO_OUTPUT_STATE_CYCLE
    - 指定したsweep数毎にエネルギーの収束状況を標準出力に出力します
      - » 注）ログ出力レベルをVA_LOG_INFO_DETAILに設定しておく必要があります
        なお、出力内容についてはデバッグ方法の章をご参照ください
    - 0を指定した場合、全sweep数を20個分に区切り、各段階のエネルギーの収束状況を標準出力に出力します(default)
      - » sweep数<20の場合は、1 sweep毎のエネルギーの収束状況をsweep数分だけ出力します
    - ※VA_QUBO_OUTPUT_STATE_CUCLE>sweep数とした場合は、エネルギーの収束状況は出力されません

環境変数

- va_qubo.exe 関連
  - VA_QUBO_ENABLE_FAST_SAT
    - “enable”を指定した場合、高速版制約優先モードを使用します。”disable”を指定した場合、従来版の制約優先モードで動作します(default: enable)
    - この環境変数はconstraintか、constraint_onlyでのみ有効です
  - VA_QUBO_ENABLE_FAST_SAT_NORMAL_INIT
    - “enable”を指定した場合、従来版の制約優先モードで初期スピンの決定を行った後に、高速版制約優先モードで解を求めます。”disable”の場合は、高速版制約優先モードで初期スピンの決定を行います(default: disable)
    - この環境変数は VA_QUBO_ENABLE_FAST_SATがenableかつconstraintか、constraint_onlyでのみ有効です
  - VA_QUBO_ENABLE_PHASE_SAVING
    - “enable”を指定した場合、phase saving 機能を用いて、解の探索を高速化します。”disable”を指定した場合、従来通りの動作をします(default: enable)
    - この環境変数は VA_QUBO_ENABLE_FAST_SATがenableかつconstraintか、constraint_onlyでのみ有効です
  - VA_QUBO_SAT_LARGE_WEIGHTED_SUM_THRESHOLD
    - 制約条件の解法を大規模な問題向けに切り替える閾値を0から2147483647の数値で指定します(default: 2000)
    - weighted sum, maxonecount, minmaxone制約条件において、制約に含まれるスピンの個数が閾値より大きい場合、大規模向けの制約のアルゴリズムを用いて制約の解を求めます
    - この環境変数は VA_QUBO_ENABLE_FAST_SATがenableかつconstraintか、constraint_onlyでのみ有効です
  - VA_QUBO_SAT_ENABLE_STATIC_VAR_ORDER
    - “enable”を指定した場合、変数優先度の機能を有効にし、解の探索を高速化します。 ”disable”を指定した場合、従来通りの動作をします(default: enable)
    - この環境変数は VA_QUBO_ENABLE_FAST_SATがenableかつconstraintか、constraint_onlyでのみ有効です

環境変数

- va_qubo.exe 関連
  - VA_QUBO_SAT_WEIGHTED_SUM_LIMIT
    - 演算の高速化のため、weighted sumの係数,比較対象値の値を変形して処理を実行する場合があります
    - 以下の警告メッセージが出力された場合は、VA_QUBO_PB_ENCODER_PRINT_USED_ENCODINGS に大きな値を設定してください (default:20000)

```
SAT could not satisfy the constraints with reduced weight mode in the weighted sum constraints. You may find the sastifiable solution if you set over 20000 to VA_QUBO_PB_ENCODER_PRINT_USED_ENCODINGS.
```

    - ただしメモリ使用量が大きくなり、演算速度も低下するため、極力変更しないことを推奨いたします
    - この環境変数は VA_QUBO_ENABLE_FAST_SATがdisableかつconstraintか、constraint_onlyでのみ有効です
  - OMP_NUM_THREADS
    - アニーリング実行時のスレッド数を指定します
    - 省略時はライセンスで使用可能なコア数もしくはシステムで使用可能なプロセッサ数の小さいほうの値を使用します
  - VA_QUBO_FAST_SPARSE
    - “enable”を指定した場合、疎行列モードかつフリップオプションが設定されている場合に実行速度を高速化することができます。 ”disable”を指定した場合、 従来通りの動作をします（default:disable）
    - この環境変数を有効化した場合、高速化の可否にかかわらず疎行列モードのメモリ使用量が増加します
  - VA_QUBO_RUN_ON_CHILD_PROCESS
    - 密行列モード時に動作する環境変数です
    - ”disable”を指定した場合、新しい動作モードで動作し、メモリ使用量が V3.0.1X より減少します。“enable”を指定した場合、従来通りの動作に戻り、従来通りのログ情報が得られますが、メモリ使用量が増加します（default:disable）

発行履歴

◆ 発行履歴一覧

2024年	1月	初版
2024年	11月	2版

◆ 追加・変更点詳細

初版	新規作成
2版	V4.0.0X向けに内容を修正

NEC Vector Annealing Python API (x86版) リファレンスガイド

2024年　11月　2版

日本電気株式会社
東京都港区芝五丁目７番１号
TEL(03)3454-1111（大代表）


Orchestrating a brighter world
NEC