NEC Vector Annealing
C++ API（x86 版）
ユーザーズガイド

第2版

日本電気株式会社

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。

目次

1　はしがき

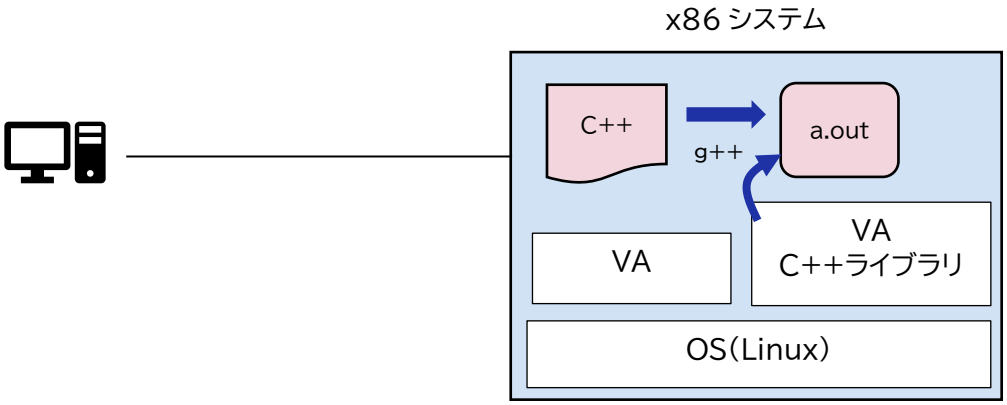
本書は、x86 システム上における、NEC Vector Annealing（以降 VA と略す）の C++ API の利用方法を説明したものです。

商標、著作権について

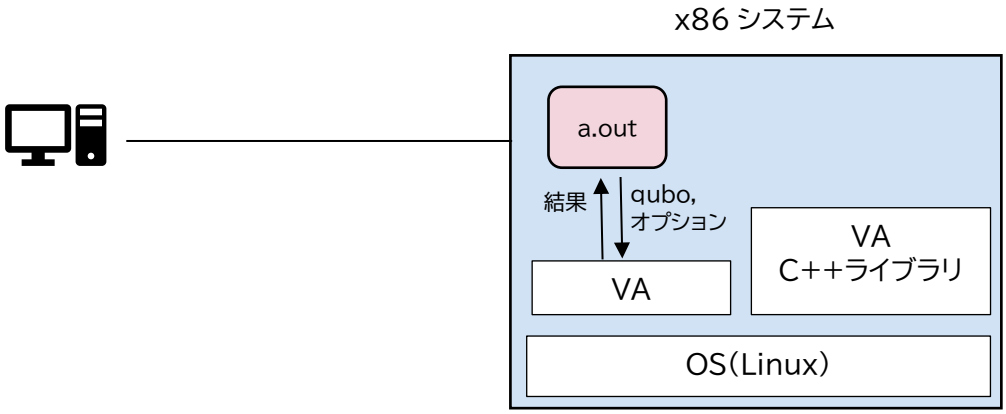
Linux はアメリカ合衆国及びその他の国における Linus Torvalds の商標です。
その他、記載されている会社名、製品名は、各社の登録商標または商標です。

2 利用環境

x86 システム上に VA および C++ API 用ライブラリがインストールされている状況を想定します。x86上で C++ API を使用した C++プログラムを、g++でコンパイルし実行ファイル(a.out)を作成します。



作成した実行ファイルを x86 システム上で実行します。問題の qubo データや VA の実行時オプションは自動的に VA に引き渡され実行されます。

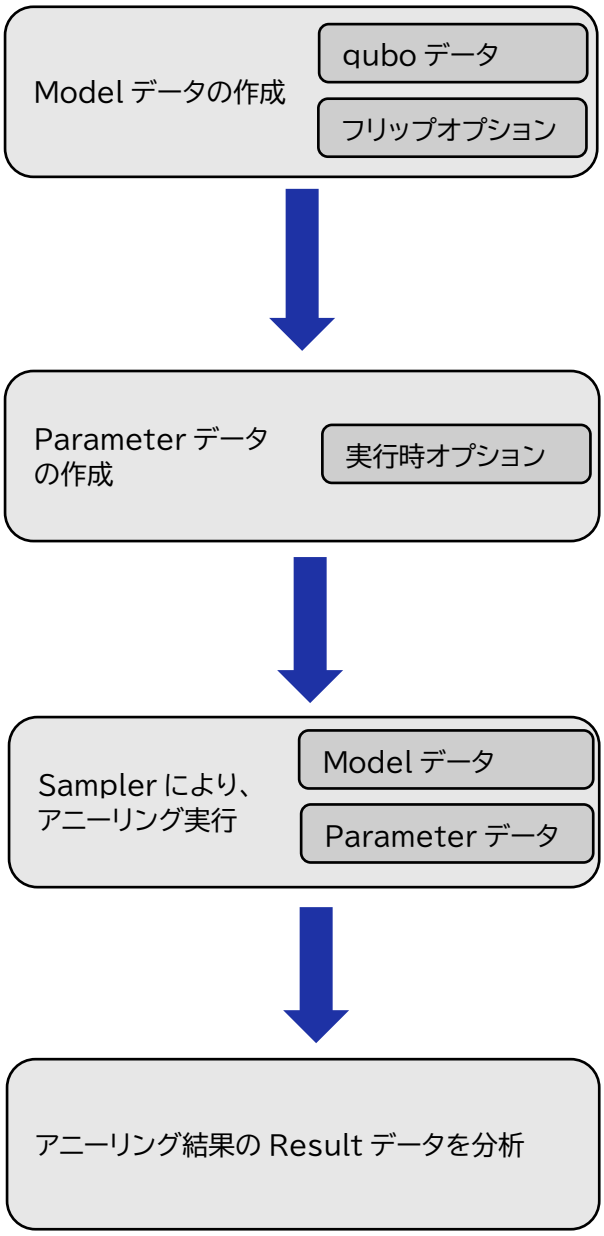


実行動作に影響を与える環境変数は Python API を使用した時と同じものが利用できます。「Vector Annealing PythonAPI リファレンスガイド」を参照ください。

3 API の使用方法

3.1 API の使用の流れ

C++ API を使用する際の大まかな流れを説明します。コード部分は説明の補助的なもので、そのまま動作するものではありません。クラスや API の詳細は「4 API」を参照してください。



問題を qubo で表現したモデルを作成します。
Model オブジェクトを作成し、qubo の要素と Offset を追加していきます。

```
Model model(100);
model.AddWeight(0, 0, -1.0);
model.AddWeight(0, 1, 2.0);
model.AddOffset(10.0);
```

必要ならフリップオプションのデータ(例 onehot)を作成し、Model オブジェクトに設定します。

```
model.SetOneHot(onehot);
```

アニーリングを実行する準備をします。
Parameter オブジェクトを作成し、VA の実行時オプションを設定します。

```
Parameter param;
param.NumSweeps.Set(1000);
```

作成したモデルを、実行時オプションを用いてアニーリングを実行します。その際、Sampler オブジェクトを用います。

```
Sampler sa;
std::vector<std::shared_ptr<Result>>
results;

results = sa.Sample(model, param);
```

アニーリングの実行結果を取り出します。

```
for (int i=0; i < results.size(); i++)
    std::cout << results[i]->Energy;
```

3.2 ソースファイル

以降では、VA のインストール PATH を VA_PATH=/opt/va/V4.0.0X とします。API を使用するソースファイルには以下の include 文を加えて下さい。

```
#include “VectorAnnealing/Model.hpp”
#include “VectorAnnealing/Sampler.hpp”
```

3.3 コンパイル時の指定

g++でコードをコンパイルする時には、以下のコンパイルオプションを指定してください。

```
-std=c++11 -fopenmp -I${VA_PATH}/include
```

また、プログラムをリンクする時には${VA_PATH}/lib64/ にある以下のライブラリをリンクしてください。

```
libVectorAnnealing.a
libminisat.so
libpblib.so
```

実際にコンパイルする場合の記述は以下のようになります。

例:

```
g++ -O3 -std=c++11 -fopenmp -o プログラム名 オブジェクト -L${VA_PATH}/lib64 -
 lVectorAnnealing -lpblib -lminisat -Wl,-rpath=${VA_PATH}/lib64
```

3.4 サンプルコード

VA がインストールされているマシンの/opt/va/V4.0.0X/samples/TSP/C に以下のサンプルコード一式を置いています。

```
Makefile
tsp.cpp
README.md
```

巡回セールスマン問題のサンプルコードであり、README.md のように実行できます。作成する Makefile やコードの参考にしてください。

4 API
4.1 Model クラス

Qubo を用いてモデルを作成するためのクラスです。
Model クラスは、Qubo の要素や制約条件を記述するクラスです。Model クラスの内部では、Qubo の要素の係数を以下の形式で保持します。どちらの形式を使用しても、ユーザプログラムからからみた動作は変わりません。

- コンストラクタの flags に WEIGHT_UPPER_TRIANGLE_MATRIX を指定した場合
  QUBO の要素の係数を上三角行列で保持します。

- コンストラクタの flags に WEIGHT_UPPER_TRIANGLE_MATRIX を指定しない場合
  アニーリング処理を高速化するために、QUBO の要素の係数を上三角行列で保持し、下三角行列部分には、上三角行列に指定した対象成分を格納します。

4.1.1 コンストラクタ

- **Model (WeightMode_t *mode*, SpinNo_t *nspins* = 0, int *flags* = 0)**
- **Model (SpinNo_t *nspins*, int *flags* = 0)**

Model クラスのコンストラクタです。

引数

引数	型	説明
mode	WeightMode_t	qubo のデータ格納方式を指定します。 WEIGHT_MODE_DENSE: 密行列形式 WEIGHT_MODE_SPARSE: 疎行列形式 省略したときは、WEIGHT_MODE_DENSE の指定となります。
nspins	SpinNo_t (uint32_t)	モデル作成に必要なスピン数を指定します。
flags	int32_t	有効にする機能のビットマスク値の論理和を指定します。 V3.0 以降では次のビットマスク値が利用可能です。 WEIGHT_UPPER_TRIANGLE_MATRIX: 上三角行列形式で係数を保持することで内部処理のメモリ使用量を削減します。

例外
- bad_alloc
- runtime_error

4.1.2 qubo の作成

- **void SetWeight(SpinNo_t *spin1*, SpinNo_t *spin2*, float *v*)**

qubo の要素に値を設定します。Qubo の要素は上三角行列の形式で設定してください。コンストラクタの flags のパラメータ WEIGHT_UPPER_TRIANGELE_MATRIX は、Model クラス内部のデータ形式を指定するパラメータであるため、このパラメータの設定の有無は、SetWeight(), AddWeight(), GetWeight() の動作に影響しません。

- **void AddWeight(SpinNo_t *spin1*, SpinNo_t *spin2*, float *v*)**

qubo の要素に値を加算します。
qubo データは 0 で初期化されるので、初回の AddWeight()は SetWeight()と同じです。

引数

引数	型	説明
spin1	SpinNo_t (uint32_t)	qubo の行を指定します。
spin2	SpinNo_t (uint32_t)	qubo の列を指定します。
v	float	qubo の要素の値を指定します。

例外
- runtime_error

- **float GetWeight(SpinNo_t *spin1*, SpinNo_t *spin2*)**

設定されている qubo の(spin1, spin2)の係数を返します。

例外
- runtime_error

- **void SetOffset(float *v*)**

エネルギーの offset 値を設定します。

- **void AddOffset(float *v*)**

エネルギーの offset 値を加算します。

引数

引数	型	説明
v	float	offset の値を設定します。

- **float GetOffset()**

設定されているエネルギーの offset 値を取得します。

例

$H = -s_0s_0 + 2.0 * s_0s_1 + 2.0 * s_0s_2 + 10.0$ の qubo を作成する例です。

```
Model model(3);                 // スピン数 3 のモデル
model.AddWeight(0, 0, -1.0);    // (0, 0) に-1.0 を設定
model.AddWeight(0, 1, 2.0);     // (0, 1) と (1, 0)に 2.0 を設定
model.AddWeight(0, 2, 1.0);     // (0, 2) と (2, 0)に 1.0 を設定
model.AddWeight(2, 0, 1.0);     // (2, 0) と (0, 2)に 1.0 を加算
                                        →(0, 2) と (2, 0)が 2.0 になる
model.AddOffset(10.0);          // Offset に 10.0 を設定
```

4.1.3 フリップオプションの設定

フリップオプションを指定することにより、シミュレーションで効率よく解を求めることができます。
フリップオプションを指定してもハミルトニアンの定式には、指定したフリップオプションに相当するペナルティ項を含める必要があります。ただし、実行時オプション VectorMode が VECTOR_MODE_CONSTRAINT または VECTOR_MODE_CONSTRAINT_ONLY の場合は、そのペ

ナルティ項を含める必要はありません。

① One Hot

スピンのグループを定義し、指定したグループ内の 1 個のスピンのみが状態=1 となる制約条件に対して使用する制約フリップオプションです。

- **void SetOneHot(std::vector<std::vector<SpinNo_t>> &*OneHot*)**

One hot 制約フリップオプションを設定します。

引数

引数	型	説明
OneHot	std::vector< std::vector<SpinNo_t> >	OneHot 制約を構成するスピン番号を1グループとして vector 配列で表し、複数グループを vector 配列に格納したものを指定します。グループが一つの場合でも vector 配列にする必要があります。

例外
- bad_alloc
- runtime_error

例

$s_0 + s_1 + s_2 + s_3 + s_4 == 1$, $s_{14} + s_{15} + s_{16} == 1$ の制約条件を設定する例です。

```
Model model(100);
std::vector<std::vector<SpinNo_t>> onehot{{0, 1, 2, 3, 4}, {14, 15, 16}};
model.SetOneHot(onehot);
```

- **std::vector<std::vector<SpinNo_t>> GetOneHot()**

One hot 制約フリップオプションの設定状態を取得します。

例外
- bad_alloc
- runtime_error

- **void ClearOneHot()**

One hot 制約フリップオプションをクリアします。

② Fixed Spin

スピンの状態を指定した値(0/1)に固定する制約条件に対して使用する制約フリップオプションです。
このオプションはハミルトニアンの定式にペナルティ項を含める必要はありません。

- **void SetFixedSpin(std::vector<spin_t> &*FixedSpin*)**

Fixed spin 制約フリップオプションを設定します。

引数

引数	型	説明
FixedSpin	std::vector<spin_t>	状態を固定するスピン番号と固定値の組の構造体の

		vector 配列を指定します。
		
		typedef struct {
		  SpinNo_t spin;
		  SpinState_t state;
		} spin_t;
		
		typedef uint32_t SpinNo_t;
		typedef uint32_t SpinState_t;

例外
- bad_alloc
- runtime_error

例

$s_0 == 0$, $s_1 == 0$, $s_2 == 1$, $s_3 == 1$ と固定する制約条件を設定する例です。

```
Model model(100);
std::vector<spin_t> fixedspin{{0, 0} , {1, 0} , {2, 1}, {3, 1}};
model.SetFixedSpin(fixedspin);
```

- **std::vector<spin_t> GetFixedSpin()**

Fixed spin 制約フリップオプションの設定状態を取得します。

例外
- bad_alloc
- runtime_error

- **void ClearFixedSpin()**

Fixed spin 制約フリップオプションをクリアします。

③ Min Max One

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 となるスピン数の範囲を指定する制約条件に対して使用する制約フリップオプションです。
範囲を指定するパラメータとして、{ minone, maxone, condition_spin, condition_state, { spin0, spin1, spin2, ... } } の順とします。minone、maxone の大小関係により、以下の2通りの意味の制約条件となります。

minone 値 ≦ maxone 値 の場合
minone 値 ≦ 状態が 1 のスピン数 ≦ maxone 値

minone 値 > maxone 値 の場合
状態が 1 のスピン数 ≦ maxone 値　もしくは　minone 値 ≦ 状態が 1 のスピン数

condition_spin, condition_state は、実行時オプション VectorMode に VECTOR_MODE_CONSTRAINT または VECTOR_MODE_CONSTRAINT_ONLY を設定した場合のみ使用することができるパラメータです。これらのパラメータを指定することで、スピンの状態により制約条件の有効・無効を設定することができます。condition_spin に指定したスピンの状態が condition_state である場合、制約条件が有効となり、スピンの状態が condition_state でない場合、制約条件が無効となります。
また、condition_spin, condition_state の記述を省略し、{ minone, maxone, { $spin_0$, $spin_1$, $spin_2$, … } } と記載した場合は、常に制約条件が有効な制約となります。この場合は、VectorMode の設

定と関係なく、常に使用可能です。

- **void SetMinMaxOne(std::vector<MinMaxOne_t> &*MinMaxOne*)**

Min Max One 制約フリップオプションを設定します。

引数

引数	型	説明

MinMaxOne	std::vector<MinMaxOne_t>	Min Max One 制約を構成するスピン番号を1グループとして vector 配列で表し、上下限値と合わせて構造体 MinMaxOne_t(以下を参照)にし、複数グループを vector 配列に格納したものを指定します。グループが一つの場合でも vector 配列にする必要があります。
		
		Typedef struct ConditionSpin {
		SpinNo_t Spin;
		SpinState_t State;
		} ConditionSpin_t;
		
		typedef struct {
		uint32_t MinOne;
		uint32_t MaxOne;
		ConditionSpin_t Condition;
		std::vector<SpinNo_t> spin;
		} MinMaxOne_t;

例外
- bad_alloc
- runtime_error

例

$1 <= s_0 + s_1 + s_2 + s_3 + s_4 <= 2$, $s_{14} + s_{15} + s_{16} == 2$ の制約条件を設定する例です。

```
Model model(100);
std::vector<MinMaxOne_t> minmaxone{{1, 2, {0, 1, 2, 3, 4}}, {2, 2, {14, 15, 16}}};
model.SetMinMaxOne (minmaxone);
```

- **std::vector<MinMaxOne_t> GetMinMaxOne()**

Min Max One 制約フリップオプションの設定状態を取得します。

例外
- bad_alloc
- runtime_error

- **void ClearMinMaxOne()**

Min Max One 制約フリップオプションをクリアします。

④ Cubic supplement

スピン x_1, x_2, y が式 $y=x_1x_2$ を満たす状態をもつ制約条件です。次数下げに伴い発生する制約条件に対し

て使用する制約フリップオプションです。
高次項の指定(HighOrder)とは同時に使用できません。

- **void SetCubicSupplement(std::vector<CubicSupplement_t> &*CubicSupplement*)**

Cubic supplement 制約フリップオプションを設定します。

引数

引数	型	説明
CubicSupplement	std::vector CubicSupplement_t>	Cubic supplement 制約を構成するスピン番号からなる構造体の vector 配列を指定します。構造体が一つの場合でも vector 配列にする必要があります。 Typedef struct { SpinNo_t y; SpinNo_t x1; SpinNo_t x2; } CubicSupplement_t; y = x1 * x2 の関係を満たします。

例外
- bad_alloc
- runtime_error

例

$s_{101}=s_1s_2$, $s_{102}=s_0s_3$ の制約条件を設定する例です。

```
Model model(110);
std::vector<CubicSupplement_t> cubicsup{{101, 1, 2}, {102, 0, 3}};
model.SetCubicSupplement(cubicsup);
```

- **std::vector<CubicSupplement_t> GetCubicSupplement()**

Cubic supplement 制約フリップオプションの設定状態を取得します。

例外
- bad_alloc
- runtime_error

- **void ClearCubicSupplement()**

Cubic supplement 制約フリップオプションをクリアします。

⑤ And zero

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 個のスピンが状態=0 となる制約条件に対して使用する制約フリップオプションです。

- **void SetAndZero(std::vector<std::vector<SpinNo_t>> &*AndZero*)**

And zero 制約フリップオプションを設定します。

引数

引数	型	説明
AndZero	std::vector< std::vector<SpinNo_t> >	And zero 制約を構成するスピン番号を1グループとして vector 配列で指定し、複数グループを vector 配列に格納したものを指定します。グループが一つの場合でも vector 配列にする必要があります。

例外
- bad_alloc
- runtime_error

例

s_0 & s_1 & s_2 & s_3 & s_4 == 0, s_{14} & s_{15} & s_{16} == 0 の制約条件を設定する例です。

```
Model model(100);
std::vector<std::vector<SpinNo_t>> andzero{{0, 1, 2, 3, 4}, {14, 15, 16}};
model.SetAndZero(andzero);
```

- **std::vector<std::vector<SpinNo_t>> GetAndZero()**

And zero 制約フリップオプションの設定状態を取得します。

例外
- bad_alloc
- runtime_error

- **void ClearAndZero()**

And zero 制約フリップオプションをクリアします。

⑥ Or one

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 個のスピンが状態=1 となる制約条件に対して使用する制約フリップオプションです。

- **void SetOrOne(std::vector<std::vector<SpinNo_t> > &*OrOne*)**

Or one 制約フリップオプションを設定します。

引数

引数	型	説明
OrOne	std::vector< std::vector<SpinNo_t> >	Or one 制約を構成するスピン番号を1グループとして vector 配列で指定し、複数グループを vector 配列に格納したものを指定します。グループが一つの場合でも vector 配列にする必要があります。

例外
- bad_alloc
- runtime_error

例

$s_0 | s_1 | s_2 | s_3 | s_4 == 1$, $s_{14} | s_{15} | s_{16} == 1$ の制約条件を設定する例です。

```
Model model(100);
std::vector<std::vector<SpinNo_t>> orone{{0, 1, 2, 3, 4}, {14, 15, 16}};
model.SetOrOne(orone);
```

- **std::vector<std::vector<SpinNo_t>> GetOrOne()**

Or one 制約フリップオプションの設定状態を取得します。

例外

- bad_alloc
- runtime_error

- **void ClearOrOne()**

Or one 制約フリップオプションをクリアします。

⑦ Pattern

スピンの状態を指定する制約フリップオプションです。
Pattern を指定するパラメータとして{ condition_spin, condition_state, equi, prohibit, { $spin_0$, $spin_1$, $spin_2$, …}, { $state_0$, $state_1$, $state_2$, …} }の順で指定します。Pattern 制約フリップオプションは、指定するパラメータによって動作が異なります。
condition_spin, condition_state はスピンの状態により制約条件の有効・無効を設定するパラメータです。condition_spin に指定したスピンの状態が condition_state である場合、制約条件が有効となり、スピンの状態が condition_state でない場合、制約条件が無効となります。また、condition_spin, condition_state の記述を省略し、{ equi, prohibit, { $spin_0$, $spin_1$, $spin_2$, …}, { $state_0$, $state_1$, $state_2$, …}} と記載した場合は、常に制約条件が有効な制約となります。
equi は、制約条件が制約を満たした場合、他の制約条件を有効化するために使用するパラメータです。
equi を指定し、スピンが指定したパターンを満たした場合、equi に指定したスピンに 1 が設定されます。パターンを満たさない場合は 0 が設定されます。この equi に指定したスピンを他の制約条件の condition_spin に設定することで、pattern 制約の結果を別の制約条件の有効・無効の条件とすることができます。また equi の記述を省略し、{ condition_spin, condition_state, prohibit, { $spin_0$, $spin_1$, $spin_2$, …}, { $state_0$, $state_1$, $state_2$, …}} と記載した場合は、equi は無効となり、制約条件の状態はスピンに設定されません。
prohibit, { $spin_0$, $spin_1$, … }, { $state_0$, $state_1$, … }はスピンのパターンを指定するパラメータです。{ $spin_0$, $spin_1$, … } にスピン番号のベクトルを指定し、{ $state_0$, $state_1$, … } にスピンの状態ベクトルを指定します。Prohibit に true を指定した場合、スピンが指定した状態を満たさない制約条件(スピンが指定した状態以外となる制約条件)となり、false を指定した場合は指定した状態となる制約条件となります。Prohibit を省略した場合は、false を指定した場合と同様、指定した状態を満たす制約条件となります。スピンのベクトルとスピンの状態のベクトルを省略して記述することはできません。
condition_spin/condition_state, equi, prohibit の各パラメータを任意の組み合わせで指定することはできません。

condition_spin, condition_state	Equi	Prohibit	動作
指定	省略	False or 省略	Condition_spin が condition_state である場合、指定したパターンを満たす制約
省略	省略	True	指定したパターン以外となる制約
指定	省略	True	Condition_spin が condition_state である場合、指定したパターン以外となる制約
省略	指定	False or 省略	指定したパターンを満たした場合に equi スピンは 1 となり、満たさない場合は 0 となる制約

上記以外	エラーとなる

実行時オプション VectorMode が VECTOR_MODE_CONSTRAINT または
VECTOR_MODE_CONSTRAINT_ONLY のときのみ使用可能です。

- **void SetPattern (std::vector<Pattern_t> & *PatternSet*)**

Pattern 制約フリップオプションを設定します。

引数

引数	型	説明
PatternSet	std::vector<Pattern_t >	Pattern 制約を構成するスピン番号(spin)とそのスピンの状態(SpinPattern)から成る構造体を vector 配列に格納したものを指定します。構造体が一つの場合でも vector 配列にする必要があります。 typedef struct { ConditionSpin_t Condition; SpinNo_t EquiSpin; bool Prohibit; std::vector<SpinNo_t> Spin; std::vector<SpinState_t> SpinPattern; } Pattern_t;

例外
- bad_alloc
- runtime_error

例
{s_0, s_1, s_2, s_3, s_4}が{1, 0, 0, 0, 1}と{1, 1, 1, 1, 1}の状態を禁止する制約条件を設定する例です。

```
Model model(100);
std::vector<Pattern_t> prohibited{
    {  True, {0, 1, 2, 3, 4}, {1, 0, 0, 0, 1}},
    {  True, {0, 1, 2, 3, 4}, {1, 1, 1, 1, 1}}};
model.SetPattern (prohibited);
```

s0 が 1 の場合に{s0, s1, s2, s3, s4}が{1, 0, 1, 0, 1}状態を禁止する制約条件を設定する例です。

```
Model model(100);
std::vector<Pattern_t> prohibited{
    {  0, 1, True, {0, 1, 2, 3, 4}, {1, 0, 1, 0, 1}},

};
model.SetPattern (prohibited);
```

- **std::vector<Pattern_t> GetPattern()**

Pattern 制約フリップオプションの設定状態を取得します。

例外
- bad_alloc
- runtime_error

- **void ClearPattern ()**

  Pattern 制約フリップオプションをクリアします。

⑧ Weighted sum

重み付き和($w_0s_0 + w_1s_1 + w_2s_2 + w_3s_3 + w_4s_4$)を用いた等号/不等号の制約条件に対して使用する制約フリップオプションです。w_iが整数値の重み係数、s_iがスピン変数です。
Weighted sum のパラメータは{{w_0, w_1, w_2, …}, {s_0, s_1, s_2, …}, type ,val, condition_spin, consition_state} の順で指定します。
type には、LE, EQ, GE のいずれかを指定します。それぞれ、≦, =, ≧ を意味しています。Val には比較対象値を設定します。Type に LE を指定した場合、Weighted sum 制約は以下の式を満たす制約となります。

$$\Sigma(w_i \times s_i) \leqq val$$

condition_spin, condition_state はスピンの状態により制約条件の有効・無効を設定するパラメータです。condition_spin に指定したスピンの状態が condition_state である場合、制約条件が有効となり、スピンの状態が condition_state でない場合、制約条件が無効となります。また、condition_spin, condition_state の記述を省略し、{ {w_0, w_1, w_2, …}, {s_0, s_1, s_2, …}, type, val } と記載した場合は、常に制約条件が有効な制約となります。
実行時オプション VectorMode が VECTOR_MODE_CONSTRAINT または VECTOR_MODE_CONSTRAINT_ONLY のときのみ使用可能です。

- **void SetWeightedSum(std::vector<WeightedSum_t > & *WeightedSumSet*)**

  Weighted sum 制約フリップオプションを設定します。

引数

引数	型	説明

WeightedSumSet	std::vector<WeightedSum_t>	Weighted sum 制約を構成するスピン番号(spin)、重み(weight)、比較演算子(type)、比較対象値(val)から成る構造体を vector 配列に格納したものを指定します。構造体が一つの場合でも vector 配列にする必要があります。
		
		重みと比較対象値は整数値のみ指定可能です(負の値も可能)。
		
		typedef struct ConditionSpin {
		SpinNo_t Spin;
		SpinState_t State;
		} ConditionSpin_t;
		
		typedef struct {
		std::vector<int64_t> weight; [1]
		std::vector<SpinNo_t> spin;
		WeightedSumType type;
		int64_t val; [2]

[1] weight は 64bits 整数型の変数のベクトルですが、32bits 整数値しか設定できません。
[2] val は 64bits 整数型の変数ですが、32bits 整数値しか設定できません。

		Condition Spin_tCondition; } WeightedSum_t; enum WeightedSumType LE: <= GE: >= EQ: ==

例外
- bad_alloc
- runtime_error

例

$10s_0 + 20s_1 + 30s_2 + 50s_3 + 100s_4 <= 160$, $s_{14} + 2s_{15} + 3s_{16} == 3$ の制約条件を設定する例です。

```
Model model(100);
std::vector<WeightedSum_t> weighted{
      {{10, 20, 30, 50, 100}, {0, 1, 2, 3, 4}, LE, 160},
      {{1, 2, 3}, {14, 15, 16}, EQ, 3}};
model.SetWeightedSum (weighted);
```

- **std::vector<WeightedSum_t> GetWeightedSum()**

Weighted sum 制約フリップオプションの設定状態を取得します。

例外
- bad_alloc
- runtime_error

- **void ClearWeightedSum()**

Weighted sum 制約フリップオプションをクリアします。

4.1.4 高次項の設定(HighOrder オプション)

3 次以上の高次項を持つエネルギー計算を、2 次式への次数下げをすることなく演算することができます。
マルチ VE、CubicSupplement とは同時に使用できません。

- **void Model::SetHighOrderDimensions (std::vector< uint32_t > & *dimensions*)**

高次項に使用する次数を設定します。SetHighOrder()より前に使用してください。

引数

引数	型	説明
dimensions	std::vector<uint32_t>	使用する高次項の次数を指定します。

例外
- runtime_error

- **void Model::SetHighOrder (int32_t *order*, std::vector< HighOrderCoupler > & *CouplerSet*)**

高次項の項を設定します。同じスピン変数の積は $s_i s_i == s_i$ の関係があるため低い次数で表現して設定してください。(例： $s_1 s_2 s_2 s_2 s_5$ の場合、$s_1 s_2 s_5$ と扱う)。

引数

引数	型	説明
order	int32_t	項の次数を指定します。
CouplerSet	std::vector< HighOrderCoupler >	高次項を構成(スピン番号と係数)するオブジェクト(HighOrderCoupler)を vector 配列に格納したものを指定します。

例外
- runtime_error

- **HighOrderCoupler クラス**

高次項を指定する際に使用します。

コンストラクタ

- **HighOrderCoupler ()**
- **HighOrderCoupler(std::vector<SpinNo_t>_*spin*, float _*weight*)**

メンバ名	型	説明
spin	std::vector<SpinNo_t>	項を構成するスピン番号をベクトルで表します。
weight	float	項の係数の値を表します。

メンバは public です。

例

3 次項として $4.0 s_0 s_1 s_2$、$2.0 s_1 s_3 s_5$ を、5 次項として $1.0 s_1 s_2 s_3 s_4 s_5$ を設定する例です。

```
Model model(100);
std::vector<uint32_t> highdim{3, 5};
model.SetHighOrderDimensions(highdim);    // 使用する次数の設定

std::vector<HighOrderCoupler> couplerset3{{{0, 1, 2}, 4.0}, {{1, 3, 5}, 2.0}};
model.SetHighOrder(3, couplerset3);    // 3 次項の設定

std::vector<HighOrderCoupler> couplerset5{{{1, 2, 3, 4, 5}, 1.0}};
model.SetHighOrder(5, couplerset5);    // 5 次項の設定
```

4.2 Parameter クラス

VA の実行時オプションを設定するためのクラスです。

4.2.1 メンバ

VA の実行時オプションに対応します。

メンバ名	型	説明
NumSweeps	int32_t	アニーリングのsweep数を指定します。既定値は500です。
NumReads	int32_t	サンプリング回数を指定します。既定値は1です。
NumResults	int32_t	取得するサンプリング結果の数を指定します。指定できる値は、1 または NumReads で指定した値です。既定値は1です。
BetaRange	Range	{start,end,nsteps}で温度の逆数であるβパラメータの開始(start)と終了(end)の値を指定します。各要素の型は(double, double, int)です。 nsteps は start から end までの分割数で省略可能です。 既定値は、{10, 100, 200}です。 BetaRange と BetaList の両方を設定した場合、BetaList が優先します。
BetaList	std::vector< double >	温度の逆数であるβパラメータのアニーリングの sweep ごとのスケジュールを指定します。
VectorMode	VectorMode_t (enum)	アニーリングの方式の Vector mode を指定します VECTOR_MODE_SPEED: 速度優先 VECTOR_MODE_CONSTRAINT: 制約優先 VECTOR_MODE_CONSTRAINT_ONLY: 制約限定(エネルギーの探索を行わない) 制約優先/制約限定は、フリップオプションで指定した制約条件を確実に満たす解を探索します。制約条件間に矛盾があり、制約条件を満たすことが出来なければ探索を打ち切ります。 既定値は、VECTOR_MODE_SPEED です。環境変数 VA_QUBO_VECTOR_MODE でも指定することができます。
DenseMode	Bool	VA エンジン内の qubo データ格納方式を指定します true: 密行列モード false: 疎行列モード 指定しないときは 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。
NumThreads	int32_t	VA エンジンを並列実行するスレッド数を指定します。ただし、0 以下の値(既定値)を指定した場合やライセンスにより使用可能なスレッド数より大きな値を指定した場合は、ライセンスにより使用可能なスレッドを使用します。 VECTOR_MODE_CONSTRAINT_ONLY の場合は、

		1 スレッド限定です。
Precision	Precision_t (enum)	アニーリング時のエネルギー計算の演算精度を指定します。 PRECISION_COMPUTE_SINGLE: 単精度 PRECISION_COMPUTE_DOUBLE: 倍精度 既定値は、PRECISION_COMPUTE_SINGLE です。
Seed	long	使用する疑似乱数の Seed を指定します。指定しないときは、実行したときの時間情報等を用いて動的に決定します。
InitialSpin	std::vector<spin_t>	初期スピン状態を指定します。スピンの一部のみの指定も可能で、その場合は残りのスピンはランダムに決まります。 typedef struct { SpinNo_t spin; SpinState_t state; } spin_t; typedef uint32_t SpinNo_t; typedef uint32_t SpinState_t;

4.2.2 メンバへの値の設定/取得

メンバへのアクセスはメンバが持つメンバ関数を使います。

- **設定**
  BetaRange 以外
  Set(値)
  BetaRange
  Set(start, end, nsteps)
  Set(start, end)

- **設定状態取得**
  メンバに値が設定されている場合 true、設定されていない場合 false が返却されます。
  BetaRange 以外
  IsValid()
  BetaRange
  IsValid() : start, end に値が設定されているか否か
  IsValidSteps() : nsteps に値が設定されているか否か

- **設定値取得**
  BetaRange 以外
  Get()
  BetaRange
  GetStart () : BetaRange の start の取得
  GetEnd () : BetaRange の end の取得
  GetNumSteps () : BetaRange の nsteps の取得

- **設定状態クリア**
  メンバの値が設定されていない状態にします。

BetaRange 以外
Clear()
BetaRange
Clear()　　: start, end, nsteps 全てをクリア

4.2.3　コンストラクタ

- **Parameter()**

Parameter クラスのコンストラクタです。引数はありません。

例

Parameter オブジェクトに実行時オプションを設定する例です。

```
Parameter param;                    // Parameter オブジェクト
param.NumSweeps.Set(1000);      // sweep 数を 1000 に設定
param.BetaRange.Set(10, 500, 600);    // βの範囲が 10～500 で 600 分割に設定

std::vector<spin_t> InitSpinList{{0, 1}, {1, 1}, {2, 1}, {3, 0}};
                                // スピン状態が(1, 1, 1, 0, 以降ランダム)のデータ準備
param.InitialSpin.Set(InitSpinList);    // InitSpinList で初期スピン状態を設定
```

4.3 Sampler クラス

VA を実行するためのクラスです。

4.3.1 コンストラクタ

- **Sampler();**

Sampler クラスのコンストラクタです。引数はありません。

例外

- bad_alloc
- runtime_error

4.3.2 アニーリングの実行

- **std::vector<std::shared_ptr<Result>> Sample(Model &*model*, Parameter *param*);**
- **std::vector<std::shared_ptr<Result>> Sample(Model &*model*);**

Model オブジェクトを指定してアニーリングを実行します。Parameter オブジェクトは省略可能ですが、既定値の動作を行い自動で適切な値が選択される訳ではありません。
使用例は Result クラスの所の例を参照してください。

戻り値

1 回のサンプリング結果が Result オブジェクトとなり、Result オブジェクトへのポインタの vector 配列が戻り値となります。Vector 配列のサイズはデフォルトでは1で、Parameter.NumResults を指定した場合はその指定数となります(ただし指定可能な値は、1 または Parameter.NumReads に指定した値のみです)。複数個の場合は、フリップオプションで指定した制約を満たす解を優先してエネルギーの低い順になります。

引数

引数	型	説明
model	Model	qubo を設定した Model オブジェクトを指定します。
param	Parameter	VA の実行時オプションを設定した Parameter オブジェクトを指定します。

例外

- bad_alloc
- runtime_error

4.4 Result クラス

VA の実行結果を格納するためのクラスです。
Sampler クラスの説明にあるように VA からの実行結果の型は
std::vector<std::shared_ptr<Result>> と複数の結果を格納した形式になります。

4.4.1 メンバ

メンバ名	型	説明
Time	double	実行したアニーリング時間(秒)です。
Energy	float	アニーリング結果のエネルギー値です。オフセット値を含みます。
Spin	SpinData	アニーリング結果のスピン状態です。
Constraint	bool	アニーリング結果がフリップオプションで指定した制約を満たしているかどうかを示します。 true: 満たしている false: 満たしていない フリップオプションを指定していない場合は、true となります。

4.4.2 メンバの値の取得

メンバは public です。直接アクセスして値を取得ください。
メンバ Spin に関しては SpinData のメンバ関数を使用できます。

- **SpinNo_t SpinData::GetNumSpins()**
  スピン数を取得します。

- **int & SpinData::operator [](SpinNo_t *index*)**
  スピン番号(index)を指定して、そのスピンの状態を取得します。

例
VA を実行し、その実行結果から各種データを取り出す例です。

```
// Model オブジェクト(model)、Parameter オブジェクト(param)は作成したと想定
std::vector<std::shared_ptr<Result>> results;  // 結果格納用データ準備
Sampler sa;                                    // Sampler オブジェクト
results = sa.Sample(model, param);   // model, param で VA を実行し、結果を変数に格納
float energy = results[0]->Energy;   // 1 番目の結果からエネルギー値を取得
bool is_constraint = results[0]->Constraint;   // 1 番目の結果から制約充足を取得
for (int i=0; i < results[0]->Spin.GetNumSpins(); i++)
    std::cout << results[0]->Spin[i];          // 1 番目の結果からスピン状態を表示
```

A) 発行履歴

発行履歴一覧

2024 年 1月	初版
2024 年 1月	第 2 版

追加・変更点詳細

初版	新規作成
第 2 版	V4.0.0X 向けに内容を修正

NEC Vector Annealing
C++ API(x86 版)
ユーザーズガイド

2024 年　 11 月　 第 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)