

# 第11期 成果発表会

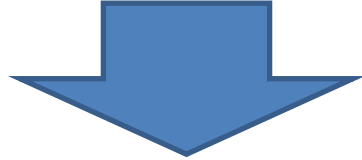
## 『朝のスケジューリングを 量子コンピュータで挑戦』

2021年10月08日

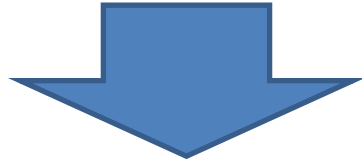
先端IT活用推進コンソーシアム  
クラウド・テクノロジー活用部会  
アドソル日進株式会社 荒本道隆

# やりたいこと

- 朝の出勤までの行動スケジュールを最適化したい



- 「スケジュールの最適化」に、量子アニーリングを適用



- まだ、各家庭に量子コンピュータはない
- Amazon Braket を試してみる



- 今回は、実用性は二の次
- 「量子コンピュータを触ってみる」が一番の目的



# 参考: AWS Braket

- 個人でも簡単に量子コンピュータを使用できる
  - 3種類の量子コンピュータが選択できる



The screenshot shows the AWS Braket website interface. At the top, there's a navigation bar with the AWS logo, links for 'お問い合わせ', 'サポート', '日本語', 'アカウント', and a 'コンソールにサインイン' button. Below this is a secondary navigation bar with links for '製品', 'ソリューション', '料金', 'ドキュメント', '学ぶ', 'パートナーネットワーク', 'AWS Marketplace', 'カスタマーサポート', 'イベント', and a search icon. The main content area is titled 'Amazon Braket' and has a sub-header '量子ハードウェア技術'. There are three main sections: '量子アニーラ' (Quantum Annealing) with a red border, 'ゲート方式、イオントラッププロセッサ' (Gate-based, Ion Trap Processors), and 'ゲート方式の超伝導プロセッサ' (Gate-based Superconducting Processors). Each section includes a brief description, a logo (D-Wave, IonQ, and Rigetti respectively), and a link for more details.

## 量子ハードウェア技術

### 量子アニーラ

量子アニーリングでは、低エネルギー構成を見つけるための物理プロセスを使用して、最適化問題の解が符号化されます。Amazon Braket では、D-Wave の超伝導量子ビットに基づく量子アニーリング技術を利用できます。

**D-Wave**  
The Quantum Computing Company

[量子アニーラの詳細はこちら »](#)

### ゲート方式、イオントラッププロセッサ

イオントラップ量子コンピュータでは、イオンと呼ばれる電荷を帯びた原子の電子状態を利用する量子ビットが導入されています。このイオンは、電磁界によって自由空間に閉じ込められて浮遊しています。Amazon Braket では、IonQ のイオントラップ量子コンピュータを利用できます。

**IONQ**

[ゲート方式、イオントラッププロセッサの詳細はこちら »](#)

### ゲート方式の超伝導プロセッサ

超伝導量子ビットは、極低温で動作する超伝導電気回路を使用して構築されます。Amazon Braket では、超伝導量子ビットを使用する Rigetti の量子ハードウェアを利用できます。

**rigetti**

[ゲート方式の超伝導プロセッサの詳細はこちら »](#)

# 参考資料 & 引用元

- 古典プログラマ向け量子プログラミング入門 [フル版]

<https://www.slideshare.net/OsSAL-org/ss-198802364>

全299ページ

C言語に慣れ親しんでいる人に最適

LangEdge, Inc.

1

## 古典プログラマ向け 量子プログラミング入門 【フル版】

- ショアのアルゴリズムから巡回セールスマン問題まで -



OsSAL.org サル量子部  
<https://www.ossal.org/qc/>

LangEdge, Inc.  
有限会社 ラング・エッジ

宮地直人 (miyachi@langedge.jp)

Ver1.1 2019年11月22日

LangEdge, Inc.

17

## 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。

# 「朝の行動スケジュールの最適化」前提条件

- 今回は、朝の行動の3つだけ

- 朝食を食べる: ○○分
- テレビを見る: ○○分
- トイレに入る: ○○分
- 出発

- その他の条件

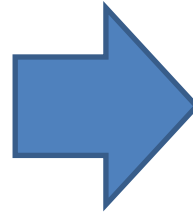
- 現在時刻
- 交通機関の運行状況 → 出発時刻
- 「出発時刻 – 現在時刻」に収まるよう各○○分を求める
- トイレには必ず入る



# 前提条件を变形

## • 必要最低限の前提条件

- 朝食を食べる: 〇〇分
- テレビを見る: 〇〇分
- トイレに入る: 〇〇分
- トイレには必ず入る
- 残り時間は60分



## • 变形後

- 朝食を食べる: 0分 (食べない)
- 朝食を食べる: 15分
- 朝食を食べる: 30分
- テレビを見る: 0分 (見ない)
- テレビを見る: 30分
- テレビを見る: 60分
- トイレに入る: 3分
- トイレに入る: 10分
- できるだけ最大値を取る
- 60分を超えないこと



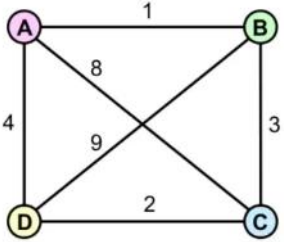
# 巡回セールスマン問題 (p230~) を参考に


LangEdge, Inc. 233


## 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ABCD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。

LangEdge, Inc. 235

## $ij$ 間の距離QUBO行列 ( $H_d$ )

$i \backslash j$	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																

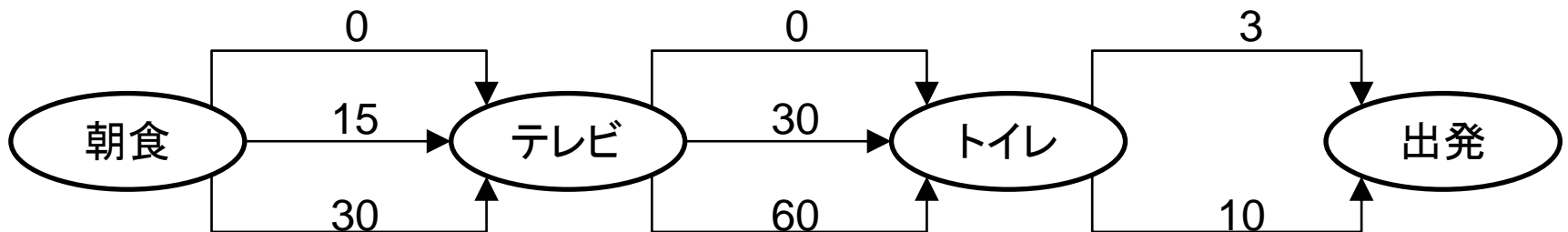
距離

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	$x_{13}$	$x_{14}$	$x_{15}$	$x_{16}$

使われない値は何でも良いが0とする

## 朝の行動スケジュール問題



めっちゃめっちゃ難しそう


今後、項目を増やすのが絶対大変 → 別の方法で






# 改めて、ナップサック問題 (p266~) を参考に

LangEdge, Inc. 267

## ナップサック問題の例

最大荷重10Kgのナップサックに以下のオヤツの合計コストが最高となる組み合わせを求めよ。  
なお組み合わせた重さの合計が10Kgとする。



<b>【オヤツ1】</b> 重さ: 6Kg コスト: 500円 	<b>【オヤツ2】</b> 重さ: 2Kg コスト: 300円 	<b>【オヤツ3】</b> 重さ: 3Kg コスト: 200円 
<b>【オヤツ4】</b> 重さ: 4Kg コスト: 300円 	<b>【オヤツ5】</b> 重さ: 7Kg コスト: 900円 	<b>【オヤツ6】</b> 重さ: 5Kg コスト: 400円 

**【ナップサック】**  
最大荷重: 10Kgまで

最適解: オヤツ3 + オヤツ5 = 重さ10Kg, コスト1,100円  
局所解1: オヤツ2 + オヤツ3 + オヤツ6 = 重さ10Kg, コスト900円  
局所解2: オヤツ1 + オヤツ4 = 重さ10Kg, コスト800円

## • 朝の行動スケジュール問題

各行動の合計時間が最大、かつ、出発時刻(60分)を超えない組み合わせを求める。

ただし、トイレの項目は必ず含むこと。

想定解: 朝食1(15分) + テレビ1(30分) + トイレ2(10分): 合計55分

朝食1 15分	朝食2 30分
テレビ1 30分	テレビ2 60分
トイレ1 3分	トイレ2 10分



**【出発時刻】**  
残り時間: 60分



# ナップサック問題(P268)を参考に定式化

- 各行動の所要時間を  $t_i$  とする
- 出発までの残り時間(60分)を  $L$  とする
- 選択されたら1になるバイナリ変数  $x_i$  で定式化

朝食1 15分	朝食2 30分
テレビ1 30分	テレビ2 60分
トイレ1 3分	トイレ2 10分

↓

【出発時刻】  
残り時間: 60分

A. コスト項  $H_c$ : コストの合計が最大(最小)になる

$$H_c = - \sum t_i x_i$$

$$H_c = - (15x_1 + 30x_2 + 30x_3 + 60x_4 + 3x_5 + 10x_6)$$

B. 制限項  $H_{w1}$ : 出発までの残り時間が合計時間とイコールになる

$$H_{w1} = (L - \sum t_i x_i)^2$$

$$H_{w1} = (60 - (15x_1 + 30x_2 + 30x_3 + 60x_4 + 3x_5 + 10x_6))^2$$

C. 制限項  $H_{w2}$ : トイレには必ず行く

$$H_{w2} = - (x_5 \text{ OR } x_6)$$

$$H_{w2} = - (x_5 + x_6 - x_5 x_6)$$

式全体 ( $k_1, k_2$  は補正係数)

$$H = H_c + k_1 * H_{w1} + k_2 * H_{w2}$$

# PyQUBOから解く

- 参考資料P270の式を変更(赤字が変更部分)
- CPUを使ってシミュレーション

```

from dimod import *
from pyqubo import *
# 変数の用意
x1, x2, x3 = Binary("x1"), Binary("x2"), Binary("x3")
x4, x5, x6 = Binary("x4"), Binary("x5"), Binary("x6")
# ハミルトニアン式
Hc = -(15*x1 + 30*x2 + 30*x3 + 60*x4 + 3*x5 + 10*x6)
Hw1 = (60 - (15*x1 + 30*x2 + 30*x3 + 60*x4 + 3*x5 + 10*x6))**2
Hw2 = -(x5 + x6 - x5*x6)
# 補正係数と全体のハミルトニアン
k1 = 10
k2 = 100
H = Hc + k1 * Hw1 + k2 * Hw2
# PyQUBOによるコンパイルとQUBO取得
model = H.compile()
qubo, offset = model.to_qubo()
#print (qubo, 'offset=', offset)
# 実行
b = BinaryQuadraticModel.from_qubo(qubo, offset)
r = SimulatedAnnealingSampler().sample(b, num_reads=100)
# 結果出力
print (r)

```

# 計算結果

```

x1 x2 x3 x4 x5 x6 energy num_oc.
3  1  1  0  0  1  1 -118.0      1
6  1  1  0  0  1  1 -118.0      1
8  1  0  1  0  1  1 -118.0      1
10 1  1  0  0  1  1 -118.0      1
11 1  1  0  0  1  1 -118.0      1
15 1  0  1  0  1  1 -118.0      1
23 1  1  0  0  1  1 -118.0      1
31 1  1  0  0  1  1 -118.0      1
:
:
:
95 0  1  1  0  1  0 -73.0       1
96 0  1  1  0  1  0 -73.0       1
97 0  1  1  0  1  0 -73.0       1
98 0  0  0  1  1  0 -73.0       1
99 0  0  0  1  1  0 -73.0       1
['BINARY', 100 rows, 100 samples, 6 variables]

```

# 前提条件にあてはめてみると

```

3  朝食:45分+テレビ: 0分+トイレ:13分、合計:58分
6  朝食:45分+テレビ: 0分+トイレ:13分、合計:58分
8  朝食:15分+テレビ:30分+トイレ:13分、合計:58分
10 朝食:45分+テレビ: 0分+トイレ:13分、合計:58分
11 朝食:45分+テレビ: 0分+トイレ:13分、合計:58分
15 朝食:15分+テレビ:30分+トイレ:13分、合計:58分
23 朝食:45分+テレビ: 0分+トイレ:13分、合計:58分
31 朝食:45分+テレビ: 0分+トイレ:13分、合計:58分
:
:
:
95 朝食: 0分+テレビ:60分+トイレ: 3分、合計:63分
96 朝食:30分+テレビ:30分+トイレ: 3分、合計:63分
97 朝食:30分+テレビ:30分+トイレ: 3分、合計:63分
98 朝食:30分+テレビ:30分+トイレ: 3分、合計:63分
99 朝食: 0分+テレビ:60分+トイレ: 3分、合計:63分

```

最適解:朝食45分+テレビ 0分+トイレ13分、合計:58分

最適解:朝食15分+テレビ30分+トイレ13分、合計:58分

想定解以上のものが2つも出てきた

# 量子コンピュータで解く

# 量子コンピュータを使用するように変更(赤字が変更部分)

```
from braket.ocean_plugin import BraketSampler, BraketDWaveSampler
from dwave.system.composites import EmbeddingComposite
s3_folder = ("amazon-braket-xxxxxxxxxxxxx", "Your-Folder-Name") #自分のS3のBucket名を指定
from dimod import *
from pyqubo import *
# 変数の用意
x1, x2, x3 = Binary("x1"), Binary("x2"), Binary("x3")
x4, x5, x6 = Binary("x4"), Binary("x5"), Binary("x6")
# ハミルトニアン式
Hc = -(15*x1 + 30*x2 + 30*x3 + 60*x4 + 3*x5 + 10*x6)
Hw1 = (60 - (15*x1 + 30*x2 + 30*x3 + 60*x4 + 3*x5 + 10*x6))**2
Hw2 = -(x5 + x6 - x5*x6)
# 補正係数と全体のハミルトニアン
k1 = 10
k2 = 100
H = Hc + k1 * Hw1 + k2 * Hw2
# PyQUBOによるコンパイルとQUBO取得
model = H.compile()
qubo, offset = model.to_qubo()
#print (qubo, 'offset=', offset)
# 実行
b = BinaryQuadraticModel.from_qubo(qubo, offset)
#r = SimulatedAnnealingSampler().sample(b, num_reads=100)
# run BQM: solve with the D-Wave 2000Q device
sampler = BraketDWaveSampler(s3_folder, 'arn:aws:braket:::device/qpu/d-wave/DW_2000Q_6')
sampler = EmbeddingComposite(sampler)
r = sampler.sample(b, num_reads=100)
# 結果出力
print (r)
```

# 量子コンピュータでの計算結果

	x1	x2	x3	x4	x5	x6	energy	num_oc.	chain_.
0	1	1	0	0	1	1	-118.0	3	0.0
1	1	0	1	0	1	1	-118.0	3	0.0
2	0	1	1	0	1	0	-73.0	17	0.0
3	0	0	0	1	1	0	-73.0	3	0.0
4	0	1	1	0	0	0	-60.0	22	0.0
5	0	0	0	1	0	0	-60.0	4	0.0
6	1	0	1	0	0	1	95.0	5	0.0
7	1	1	0	0	0	1	95.0	3	0.0
8	0	0	0	1	0	1	830.0	5	0.0
9	0	1	1	0	0	1	830.0	10	0.0
10	1	0	1	0	1	0	1292.0	4	0.0
11	0	0	0	1	1	1	1517.0	2	0.0
12	0	1	1	0	1	1	1517.0	1	0.0
13	1	1	1	0	0	0	2175.0	5	0.0
14	1	0	0	1	0	0	2175.0	2	0.0
15	1	1	0	0	0	0	2205.0	1	0.0
16	1	0	1	0	0	0	2205.0	6	0.0
17	0	0	1	0	1	1	2747.0	1	0.0
18	1	1	1	0	1	0	3062.0	2	0.0
19	0	1	0	1	1	0	10697.0	1	0.0

['BINARY', 20 rows, 100 samples, 6 variables]

# 前提条件にあてはめてみると

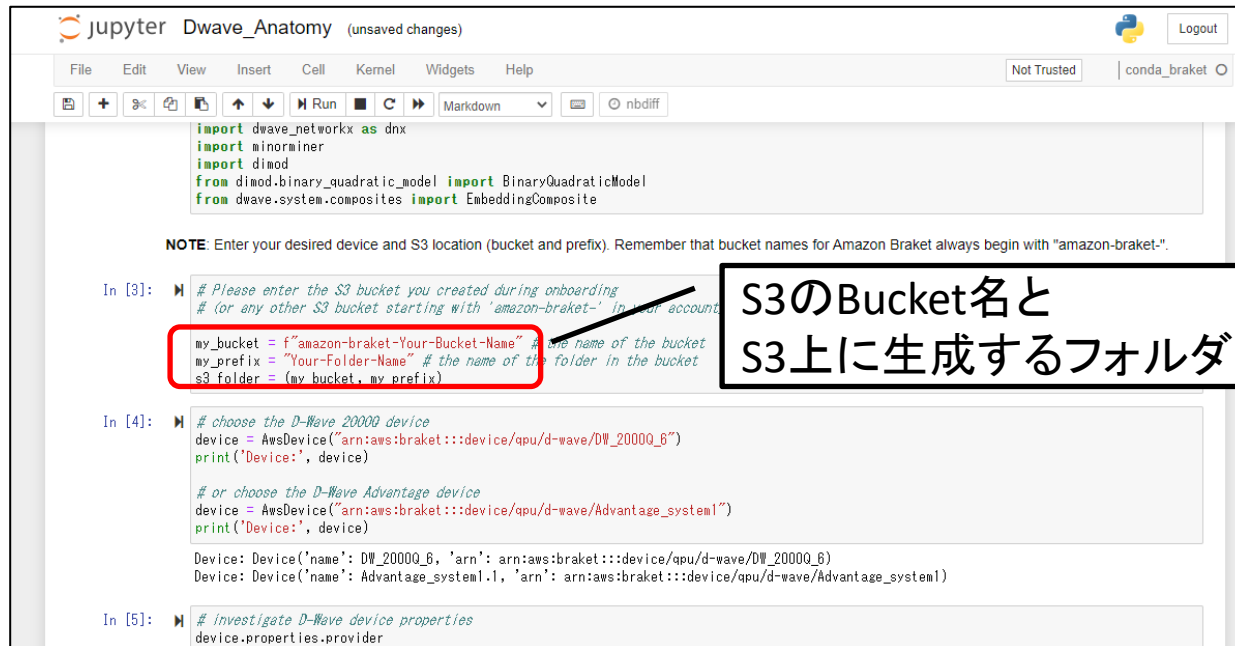
0	朝食:45分+テレビ: 0分+トイレ:13分、合計:58分
1	朝食:15分+テレビ:30分+トイレ:13分、合計:58分
2	朝食:30分+テレビ:30分+トイレ: 3分、合計:63分
3	朝食: 0分+テレビ:60分+トイレ: 3分、合計:63分
4	朝食:30分+テレビ:30分+トイレ: 0分、合計:60分
5	朝食: 0分+テレビ:60分+トイレ: 0分、合計:60分
6	朝食:15分+テレビ:30分+トイレ:10分、合計:55分
7	朝食:45分+テレビ: 0分+トイレ:10分、合計:55分
8	朝食: 0分+テレビ:60分+トイレ:10分、合計:70分
9	朝食:30分+テレビ:30分+トイレ:10分、合計:70分
10	朝食:15分+テレビ:30分+トイレ: 3分、合計:48分
11	朝食: 0分+テレビ:60分+トイレ:13分、合計:73分
12	朝食:30分+テレビ:30分+トイレ:13分、合計:73分
13	朝食:45分+テレビ:30分+トイレ: 0分、合計:75分
14	朝食:15分+テレビ:60分+トイレ: 0分、合計:75分
15	朝食:45分+テレビ: 0分+トイレ: 0分、合計:43分
16	朝食:30分+テレビ:30分+トイレ: 0分、合計:60分
17	朝食: 0分+テレビ:30分+トイレ:13分、合計:43分
18	朝食:45分+テレビ:30分+トイレ: 3分、合計:78分
19	朝食:30分+テレビ:60分+トイレ: 3分、合計:93分

最適解:朝食45分+テレビ 0分+トイレ13分、合計:58分

最適解:朝食15分+テレビ30分+トイレ13分、合計:58分

# 参考: AWS Braketの動かし方

- 「Amazon Braket」を開く
- 「Notebooks」→「Create notebook instance」
  - 作成されるまで数分かかる
- サンプルを選択
- 自分のS3のBucket名を指定する
- メニューの「Cell」→「Run All」で実行する
  - サンプルは結構複雑な処理 → **結構課金される**



```

import dwave_networkx as dnx
import minorminer
import dimod
from dimod.binary_quadratic_model import BinaryQuadraticModel
from dwave.system.composites import EmbeddingComposite

NOTE: Enter your desired device and S3 location (bucket and prefix). Remember that bucket names for Amazon Braket always begin with "amazon-braket-".

In [3]: # Please enter the S3 bucket you created during onboarding
# (or any other S3 bucket starting with 'amazon-braket-' in your account)

my_bucket = f"amazon-braket-Your-Bucket-Name" # the name of the bucket
my_prefix = "Your-Folder-Name" # the name of the folder in the bucket
s3_folder = (my_bucket, my_prefix)

In [4]: # choose the D-Wave 2000Q device
device = AwsDevice("arn:aws:braket::device:qpu/d-wave/DW_2000Q_6")
print('Device:', device)

# or choose the D-Wave Advantage device
device = AwsDevice("arn:aws:braket::device:qpu/d-wave/Advantage_system1")
print('Device:', device)

Device: Device('name': 'DW_2000Q_6', 'arn': 'arn:aws:braket::device:qpu/d-wave/DW_2000Q_6')
Device: Device('name': 'Advantage_system1.1', 'arn': 'arn:aws:braket::device:qpu/d-wave/Advantage_system1')

In [5]: # investigate D-Wave device properties
device.properties.provider
    
```

S3のBucket名と  
S3上に生成するフォルダ名を指定



# 参考：AWS Braketの利用料金

- D-wave 2000Qの場合
  - $\$0.3 + \$0.00019 \times \text{shot数}$
- 量子アニーリングでは、何度も試行して分布を調べる
  - 例：100回試行する場合
    - $\$0.3 + \$0.00019 \times 100 = \$0.319 = \text{約35円}$
  - 毎分再評価すると、1時間あたり
    - $\text{約35円} \times 60\text{回} = \text{約2,100円／時}$

Pricing Tables

Quantum Computers

Hardware Provider	QPU family	Per-task price	Per-shot price
D-Wave	2000Q	\$0.30000	\$0.00019
D-Wave	Advantage	\$0.30000	\$0.00019
IonQ	IonQ device	\$0.30000	\$0.01000
Rigetti	Aspen-9	\$0.30000	\$0.00035

『Amazon Braket Pricing』

# 今回のコードの実行にかかった金額

- 前日の「今月の請求書」

主にサンプルを3つ実行したせい

▼ Braket		\$17.21
▼ US West (Oregon)		\$17.21
Amazon Braket CompleteTask		\$2.81
\$0.00019 per-shot for D-Wave-2000Q in US West (Oregon) Region	11,780.000 Quantum-Shot	\$2.24
\$0.00019 per-shot for D-Wave-Advantage_system in US West (Oregon) Region	3,010.000 Quantum-Shot	\$0.57
Amazon Braket Task		\$14.40
\$0.30 per-task for D-Wave-2000Q in US West (Oregon) Region	44.000 Quantum-Task	\$13.20
\$0.30 per-task for D-Wave-Advantage_system in US West (Oregon) Region	4.000 Quantum-Task	\$1.20

- 1回(100試行)だけ実行した、翌日の「今月の請求書」

▼ Braket		\$17.53
▼ US West (Oregon)		\$17.53
Amazon Braket CompleteTask	+100 Q-Shot	\$2.83
\$0.00019 per-shot for D-Wave-2000Q in US West (Oregon) Region	11,880.000 Quantum-Shot	\$2.26
\$0.00019 per-shot for D-Wave-Advantage_system in US West (Oregon) Region	3,010.000 Quantum-Shot	\$0.57
Amazon Braket Task	+1 Q-Task	\$14.70
\$0.30 per-task for D-Wave-2000Q in US West (Oregon) Region	45.000 Quantum-Task	\$13.50
\$0.30 per-task for D-Wave-Advantage_system in US West (Oregon) Region	4.000 Quantum-Task	\$1.20

+\$0.32

- 上記に加えて、Notebookの利用料金(0.14\$/GB-Mo)もかかる

# やってみた感想

- AWS Braketのサンプルを適当に3つ動かしてみたら
  - 実行にちょっと時間がかかるなあ、と数分間眺めていたら、
  - 翌日、請求額を確認すると\$11.93(約1,312円)になっていた
- ナップサック問題をベースにすることで
  - 想定していた以上の最適解が出た
  - 項目の追加が容易
  - 複雑な条件が出てきた時、数式にできるのだろうか？
  - 条件をクリアしていない結果も返ってくるので、チェックする実装も必要
- 実用になるのか？
  - 35円／回、毎分再評価すると2,100円／時、はかかりすぎ
    - きっかけがあった時だけ再計算したい
    - 数百円／時、なら許容範囲？
    - 東芝の「シミュレーテッド分岐マシン」(AWS上)なら \$3.06/時間  
<https://aws.amazon.com/marketplace/pp/prodview-f3hbaz4q3y32y>
  - 朝食は、「毎日食べる」か「毎日食べない」が体重維持には良いらしい
  - テレビを見るかどうかは、やっているニュース次第
  - トイレにかかる時間は、自分ではコントロールできない

# まとめ

- 理由をつけて、量子コンピュータを触ってみる：**達成**
  - なんとなくデモを動かしてみるよりも理解が深まった
  - 超高額なハードウェアを利用するので、それなりに費用がかかる
    - 実機をレンタルするよりは、はるかに安価だけど
- 実際に運用できそうか？：**しばらくは無理そう**
  - 量子コンピュータの費用が結構かかる
  - 優先するものが、人ごとに、日によっても異なりそう
  - 外部／内部の状況をもっと取り込んで反映させたい
    - テレビで放送中のニュースは、興味があるか？
    - 便意の強度
- 今回の実証実験は意味があったか？：**もちろん、意味があった**
  - やってみよう → 具体化しなきゃならない
    - 定式化してみるだけでも、色んなものが見えてきた
    - 定式化したものは、量子コンピュータじゃなくても実行できる
  - 補正係数は、どうやって決める？
    - 補正係数 = 優先度 → 人ごとに違いそう

ありがとうございました



<http://aitc.jp>



<https://www.facebook.com/aitc.jp>



ハルミン

AITC非公式イメージキャラクター