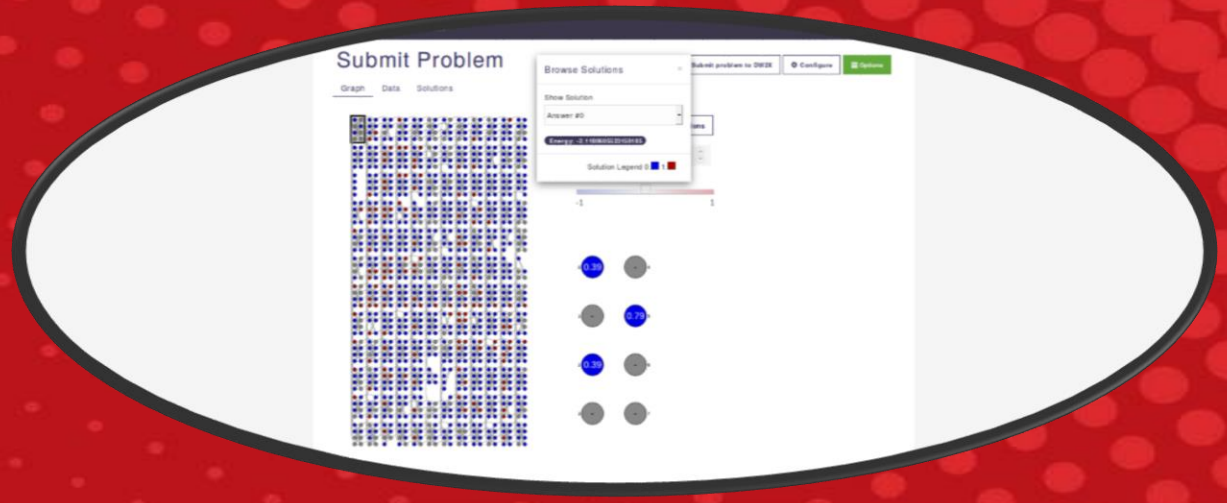
JOURNEE QUANTIQUE organisée par le GT RO Quantique du GDR RO

12 mai 2022 au vendredi 13 mai 2022



DE LA RECHERCHE À L'INDUSTRIE

TP : Solving the maximum matching problem with quantum annealing

VERT Daniel

Déroulement

Introduction

- D-Wave
- Limitation de l'architecture / Les instances QUBO
- Famille des graphes G_n

Partie 1 : Résolution du problème de maximum matching

- De G_n à QUBO
- De QUBO à D-Wave

Partie 2 : Résultats

- Interpretation
- Conclusion ?

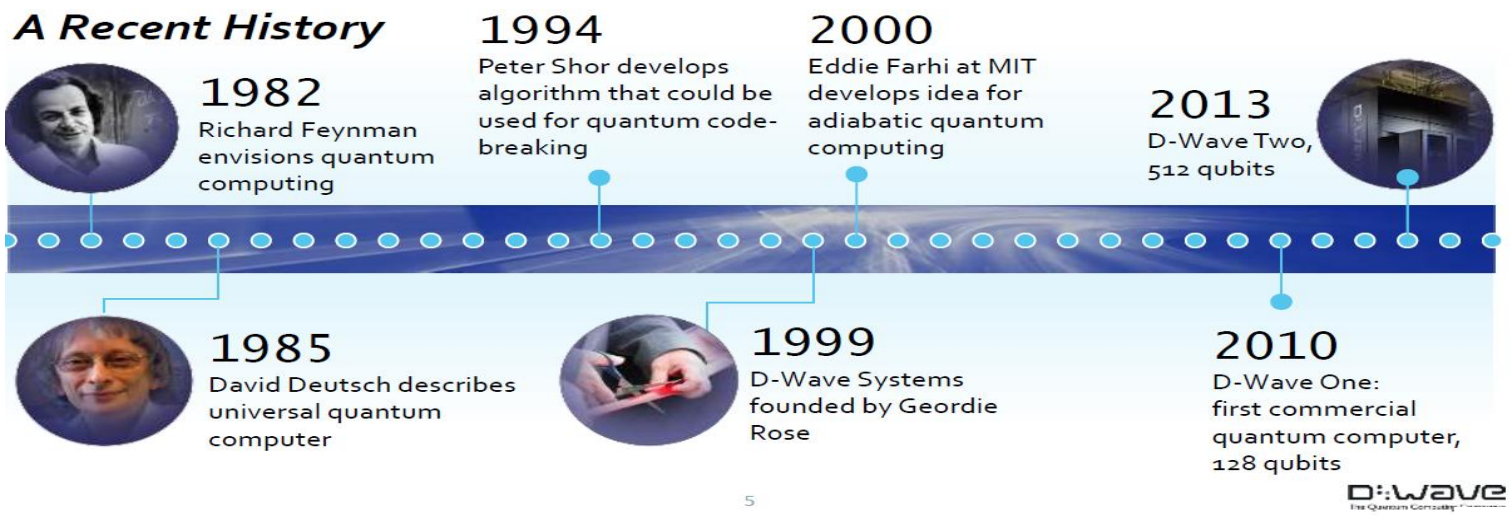
Partie 3 : Pour aller plus loin

- Question 1
- Question 2

"In my opinion, the crucial experiment (which has not yet been done) would be to compare the adiabatic algorithm head-on against simulated annealing and other classical heuristics. *The evidence for the adiabatic algorithm's performance would be much more convincing if the known classical algorithms took exponential time on the same random instances.*"

Scott Aaronson

L'ORDINATEUR QUANTIQUE ANALOGIQUE




TOUTE L'ACTUALITÉ / DATACENTER / MAINFRAME ET SUPERCALCULATEUR

15 millions de dollars pour l'ordinateur quantique D-Wave 2000Q

Sharon Gaudin / IDG News Service (adapté par Jean Elyan), publié le 25 Janvier 2017

D-Wave Systems a annoncé hier la disponibilité générale du 2000Q, un ordinateur présenté comme quantique à 2000 qubits, soit deux fois plus que son système précédent. Une entreprise de cybersécurité a déjà passé commande. Prix du nouveau système : 15 millions de dollars.

Le 2000Q de D-Wave se présente comme un ordinateur quantique d'une puissance de 2000 qubits. Son prix : 15 millions de dollars.

Après un premier ordinateur quantique D-Wave 2X à 1000 qubits, actuellement testé par Google, la

- **Machine quantique analogique**
  - Machine spécialisée dans la résolution d'un difficile problème NP-difficiles.
  - Utilise un algorithme similaire au recuit simulé → (méta)heuristique pour résoudre rapidement des classes de problèmes complexes (problèmes d'optimisation, d'apprentissage machine ou de recherche opérationnelle) inspirés de la physique statistique.
  - D-Wave est donc une sorte d'oracle d'optimisation pour le problème typique de la machine (verre de spin) utilisant les phénomènes quantiques.

- **L'idée du TP :**
  - Comportement/Performance sur un problème connu pour être difficile (existe-t-il un facteur d'accélération entre QA et SA)

LE RECUIT QUANTIQUE

- Principe [FARHI 2000] : Interpolation entre un Hamiltonien difficile à déterminer (HP) et un Hamiltonien facile à décrire (H0)
  - Configuration de l'Hamiltonien initial: $\mathcal{H}(t) = A(t)\mathcal{H}_0 + B(t)\mathcal{H}_P$
  - Avec un état initial facile : $\mathcal{H}_0 = \sum_i \sigma_i^x$
  - Evolution "lente" vers l'Hamiltonien décrit comme un modèle d'Ising : $\mathcal{H}_P = \sum_i h_i \sigma_i^z + \sum_{(ij)} J_{i,j} \sigma_i^z \sigma_j^z$
  - Initialement $B(t=0) = 0$ et $A(t=\tau) = 0$ → $\mathcal{H}(t=0) = A(t=0)\mathcal{H}_0$
  - Evolution de l'état t = 0 à t = τ → $\mathcal{H}(t=\tau) = B(t=\tau)\mathcal{H}_P$
    H(t) est " recuit " sous une forme purement classique (l'état fondamental H(0) =H0 évolue vers un état H(τ) = HP)

Théorème adiabatique [MESSIAH 1979]: Si l'évolution temporelle est assez lente (τ est assez grand) → La solution optimale ε(σ) est obtenue avec une forte probabilité
(le système reste dans l'état fondamental et son état final décrit une solution au problème d'Ising).

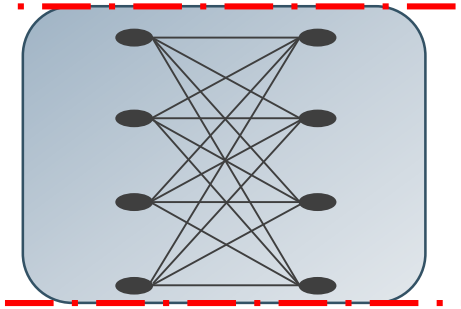
PROBLEMES ADRESSABLES ?

Problèmes combinatoires : Problèmes discrets avec un nombre extrêmement grand de solutions mais … fini !

Voyageur de commerce (NP-hard) : Chercher un chemin entre N villes → N! possibilités
Aucun algorithme (classique et/ou quantique) ne peut trouver une solution exacte rapidement !

The traveling salesperson problem is an optimization problem that can be solved using exactly-one-true constraints. © 2009), Google.

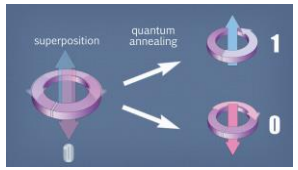
Bipartie (polynomial) : problem de matching: Problème de couplage couvrant autant de sommets que possible → N! solutions possibles sur un graphe complet biparti



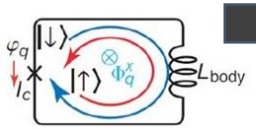
Optimisation des flux de trafic : Minimiser le temps de parcours d'un ensemble de voitures → Algorithmes classiques inefficaces pour ~ milliers de véhicules

ARCHITECTURE D-Wave QPU : TOPOLOGIES

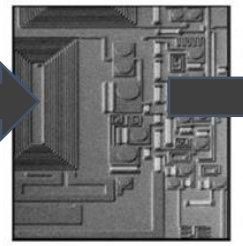
QPU :



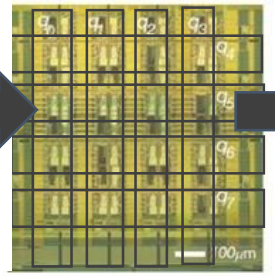
Qbit implementation
-Rf SQUID* Flux Qbit
-Compound-Compound Josephson Junction



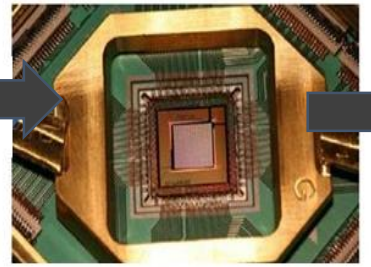
Niobium on silicon



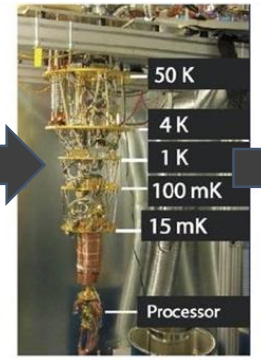
8-qbit unit cell



1152-qbit « Washington » chip



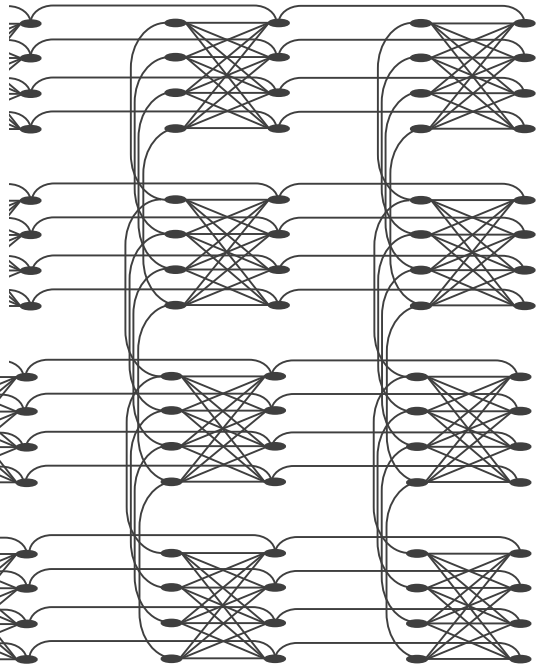
Pulse tube dilution refrigerator

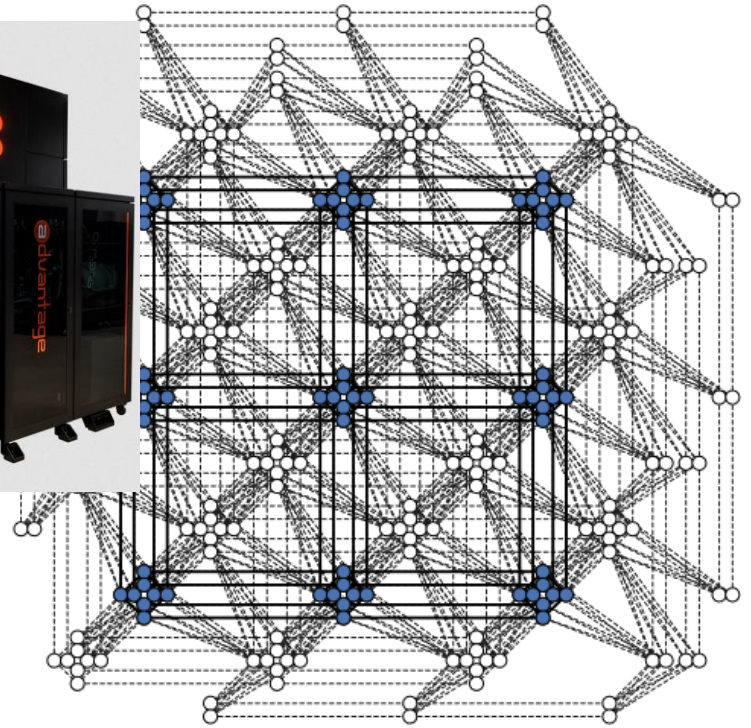


D:wave
The Quantum Computing Company™

D-Wave 2000Q/Advantage → plus de 2000/5600 qubits

D-Wave, Burnaby, Canada

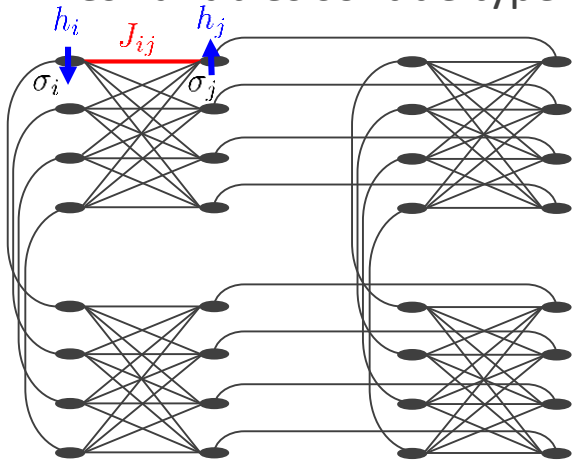




- **Pour le QPU : Ising ou QUBO**

- Modèle d'Ising : Traditionnellement utilisé en mécanique statistique → Les variables sont de type "spin up" (↑: +1) et "spin down" (↓ : -1).
  - Fonction objectif :

$$E_{ising}(\mathbf{s}) = \sum_{i}^{N} h_i \sigma_i + \sum_{i<j}^{N} J_{ij} \sigma_i \sigma_j$$

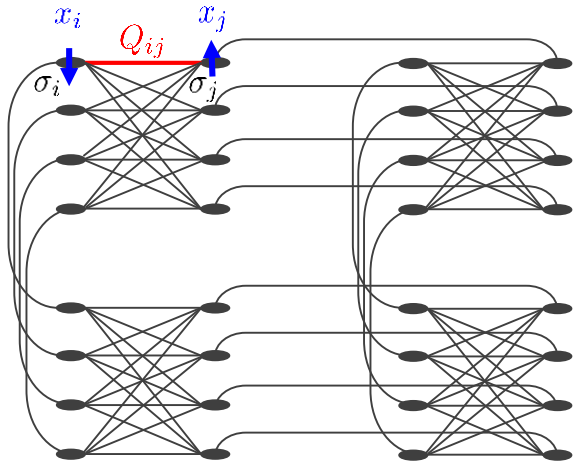


hi : coefficients linéaires correspondant aux biais des qubits

Ji,j les coefficients quadratiques correspondant aux forces de couplage

- Modèle QUBO : Traditionnellement utilisé en informatique (variable binaire) → matrice triangulaire supérieure Q de taille N x N
  - Fonction objectif :

$$O(\mathbf{Q}, \mathbf{x}) = \sum_{i} Q_{ii} x_i + \sum_{i<j} Q_{ij} x_i x_j$$



Qij est la force de couplage entre les variables

Qii est le biais pour la variable xi.

- ISING ⟷ QUBO :

$$\forall i, Q_{ii} = h_i, \forall i, j/i \neq j, Q_{ij} = J_{ij}, \forall i, s_i = 2x_i - 1$$

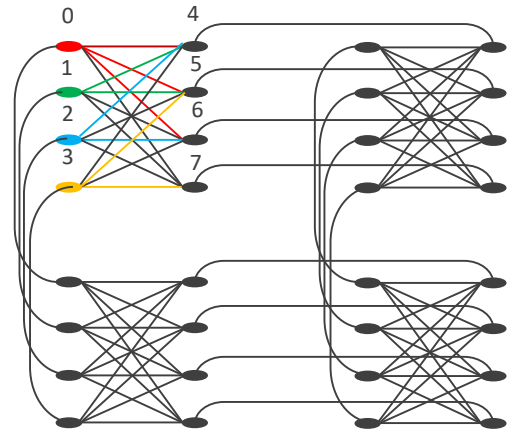
INTEGRATION DU QUBO SUR D-WAVE ET LIMITATION

Intégration de la matrice QUBO dans le graphe interne des qubits physiques

- Approche directement intégrable (exemple fictif) :

	0	1	2	3	4	5	6	7
0	17	0	0	0	−5	2	−21	0
1	0	17	0	0	−16	−1	0	9
2	0	0	17	0	−1	0	4	0
3	0	0	0	17	0	16	0	8
4	0	0	0	0	17	0	0	0
5	0	0	0	0	0	17	0	0
6	0	0	0	0	0	0	17	0
7	0	0	0	0	0	0	0	17

,



Pas de problème pour le D-Wave!

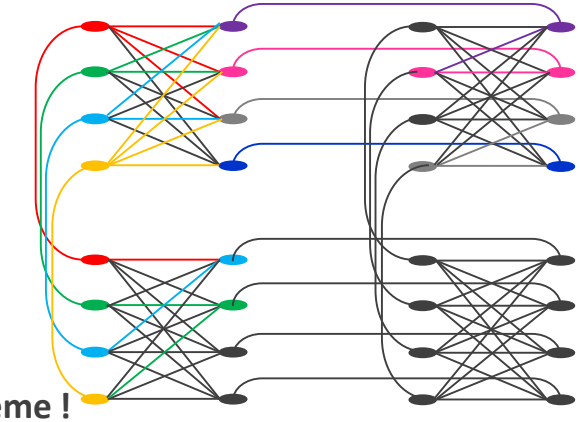
1 variable → N qubits

Pas assez de connexions entre les qubits (hi) → maximum 6!

- Variable 1,2,5,7 dupliqué 3 fois
- Variable 0,3,4,6 dupliqué 2 fois
- Nombre total de couplages: 15
- Nombre total de qubits utilisés: 20

	0	1	2	3	4	5	6	7
0	17	0	4	0	−5	2	−21	0
1	0	17	0	−2	−16	−1	0	9
2	0	0	17	0	−1	0	4	0
3	0	0	0	17	9	16	−1	8
4	0	0	0	0	17	−6	0	0
5	0	0	0	0	0	17	0	0
6	0	0	0	0	0	0	17	−12
7	0	0	0	0	0	0	0	17

,



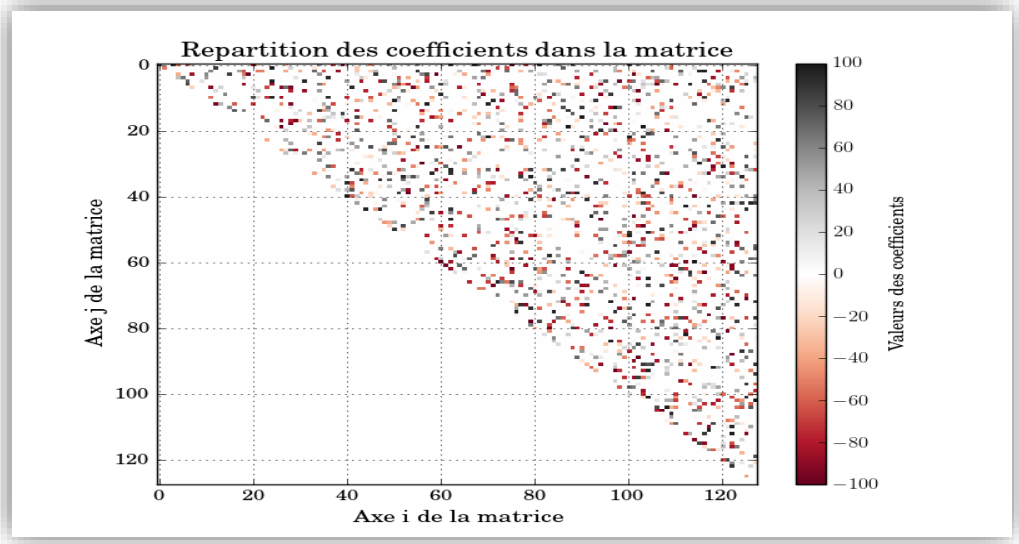
Plusieurs qubits physiques sont UNE variable du problème !

Si le graphe induit par les couplages non nuls de la matrice n'est pas isomorphe au graphe de Chimera
→ il est nécessaire de dupliquer la ou les variables sur plusieurs qubits physiques !

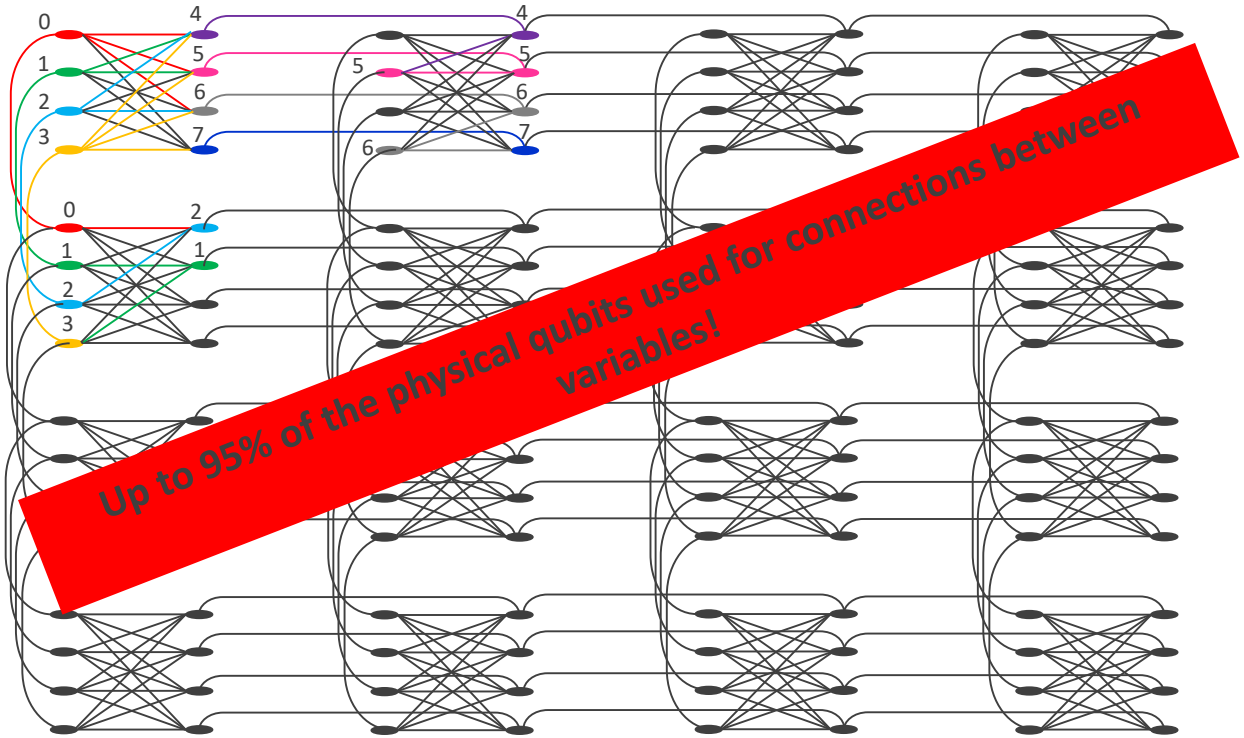
Intégration de la matrice QUBO dans le graphe interne des qubits physiques

1 variable → 1 qubit

Nombre total de couplages : 384 arêtes!



128 h_i (qubits) + 8128 J_{ij} (termes de couplages)



Graphe Chimera 4x4 → 128 qubits physiques

La topologie ne représente que 4% du nombre total de coupleurs nécessaires pour intégrer la matrice dans le pire des cas.

Définition du problème de couplage

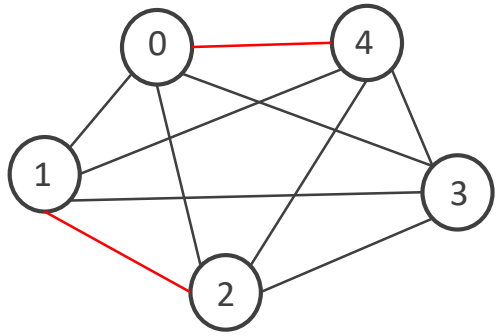
Benchmarking Quantum Annealing Against “Hard” Instances of the Bipartite Matching Problem

DEFINITION DU PROBLEME

- **Problème de Matching dans le cas général :**

  But : sélectionner un ensemble d'arêtes qui donne un **couplage** et qui maximise le nombre de sommets couverts.

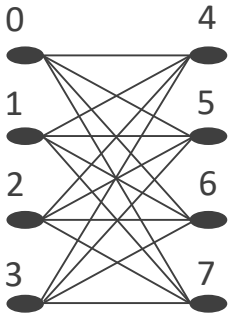
  Problème Polynomial → Algorithme polynomial (Edmonds) permet de le résoudre !



Exemple sur un graphe complet

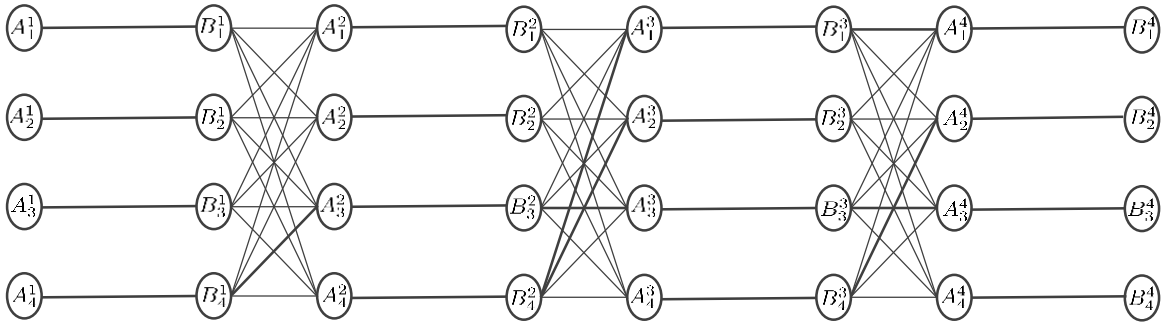
- **Problème de Matching sur un graphe bipartie :**

  Problème Polynomial
  Polonais / algorithme de flow le résout



- **Problème de Maximum matching :**

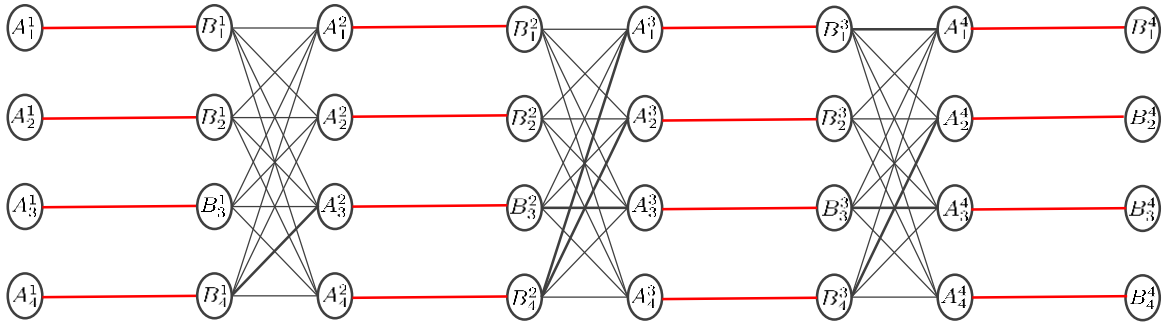
  [sasaki-1988] → Cas particuliers nécessitant un nombre **exponentiel** d'itérations pour obtenir la solution optimale.
  → (trivial à résoudre algorithmiquement)
  → Et même l'oeil



Sommets A non connectés entre eux et idem pour les sommets B

DEFINITION DU PROBLEME DE MATCHING

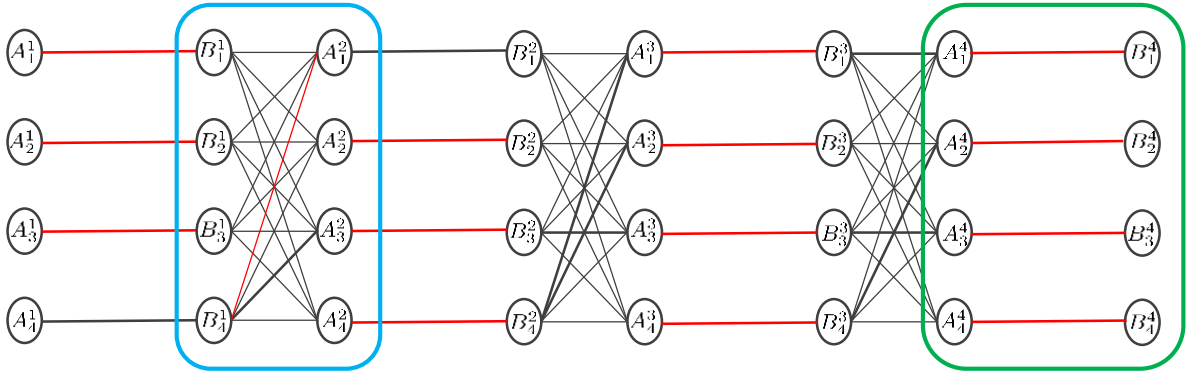
Sélection des arêtes au bords :
→ Valide **et** solution optimale



Example all the red edges have a value of 1

Sélection d'une arêtes dans un bipartie :
→ suppression de deux sommets

Nombre de sommets : $O(n^3)$
Nombre d'arêtes : $O(n+1)^3$
Nombre d'arêtes pour la solution optimal : $O(n^2)$
→ Diffcile pour le recuit simulé



16 edges / 8 vertices

4 edges / 8 vertices

[sasaki-1988] : Le temps de calcul croît **exponentiellement** avec le nombre de sommets du graphe (la plupart des variables se trouvent dans des sous-graphes denses).

Objectif: Comparer QA avec SA

Résolution du problème de matching avec l'ordinateur quantique analogique

Benchmarking Quantum Annealing Against "Hard" Instances of the Bipartite Matching Problem

PROBLEME DE COUPLAGE VERS QUBO

- Problème de couplage s'écrit naturellement sous la forme d'un programme linéaire en nombres entiers (des contraintes) vs QUBO (pas de contraintes)

  → 1 variable pour chaque arrête du graphe $x_e \in \{0,1\}$ 1 si e ϵ C (couplage) 0 sinon

- Fonction économique : Maximiser le nombre d'arrêtes dans le couplage $\sum_{e \in E} x_e$
- Contraintes :

  Pour chaque sommets v : $\sum_{e \in \Gamma(v)} x_e \leq 1$ Chaque sommet est couverts au plus une fois

  → Parmi toutes les arrêtes incidentes en un sommet : Une seule fait parti du couplage (contrainte linéaire)

Transformation du couplage en QUBO

→ QUBO sans contraintes : Rajouter une pénalité sur la violation des contraintes de couplages !

- Technique de pénalité → Mettre ces contraintes dans la fonction économique :

Maximiser $\sum_{e \in E} x_e - \lambda \sum_{v \in V} \left(1 - \sum_{e \in \Gamma(v)} x_e\right)^2$ (QUBO) Avec λ grand de manière à pénaliser les solutions qui ne satisfont pas les contraintes

Contraintes « molles » → Solution qui accepte les non couplages (toutes les solutions même celles qui violent les contraintes)

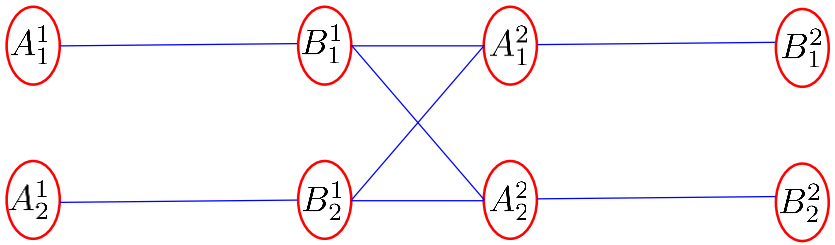
STEP 1 : FORMULATION DU PROBLEME

Pénalité sur la violation des contraintes de couplage → Maximise $$\sum_{e\in E} x_e - \lambda \sum_{v\in V}\left(1-\sum_{e\in\Gamma(v)} x_e\right)^2$$

Construction de la matrice QUBO :

Termes diagonaux: $$Q_{ee} = -(1+2\lambda)$$

Termes Non-diagonaux: $$Q_{ee'} = \begin{cases} 2\lambda & \text{if } e\cap e' \neq \emptyset, \\ 0 & \text{otherwise.} \end{cases}$$



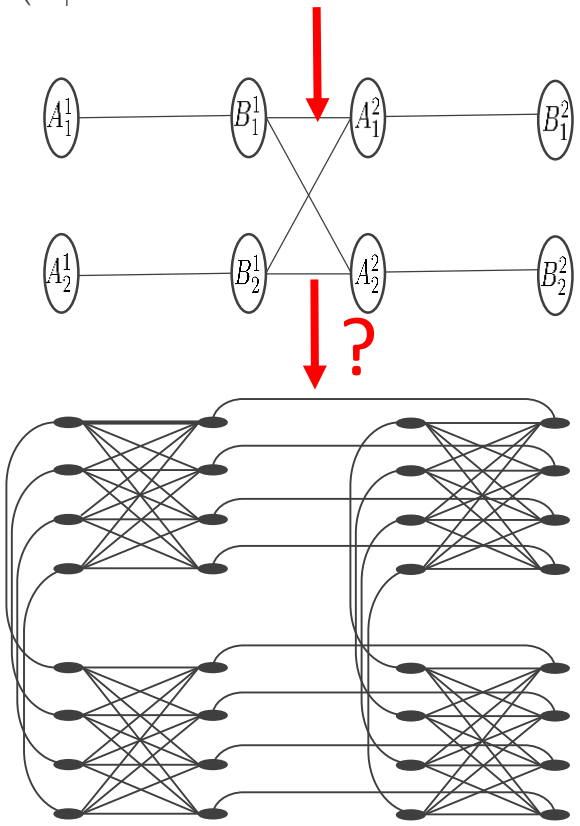
$$\begin{pmatrix} & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 0 & -17 & 0 & 16 & 16 & 0 & 0 & 0 & 0 \\ 1 & 0 & -17 & 0 & 0 & 16 & 16 & 0 & 0 \\ 2 & 0 & 0 & -17 & 16 & 16 & 0 & 16 & 0 \\ 3 & 0 & 0 & 0 & -17 & 0 & 16 & 0 & 16 \\ 4 & 0 & 0 & 0 & 0 & -17 & 16 & 16 & 0 \\ 5 & 0 & 0 & 0 & 0 & 0 & -17 & 0 & 16 \\ 6 & 0 & 0 & 0 & 0 & 0 & 0 & -17 & 0 \\ 7 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -17 \end{pmatrix},$$

```
danielvm@danielvm-VirtualBox: ~/Desktop/tp_dwave
danielvm@danielvm-VirtualBox:~/Desktop/tp_dwave$ python Gn2QUBO.py
('graphe G', 1)
('u(', 0, ',', 0, ')=>', 0)
('u(', 0, ',', 1, ')=>', 1)
('u(', 1, ',', 0, ')=>', 2)
('u(', 1, ',', 1, ')=>', 3)
()
('v(', 0, ',', 0, ')=>', 4)
('v(', 0, ',', 1, ')=>', 5)
('v(', 1, ',', 0, ')=>', 6)
('v(', 1, ',', 1, ')=>', 7)
(8, 'aretes')
coef normalisation: 17.0
Matrice QUBO :
0 0 -17
0 4 16
0 5 16
1 1 -17
1 6 16
1 7 16
2 2 -17
2 4 16
2 6 16
3 3 -17
3 5 16
3 7 16
4 4 -17
4 5 16
4 6 16
5 5 -17
5 7 16
6 6 -17
6 7 16
7 7 -17
danielvm@danielvm-VirtualBox:~/Desktop/tp_dwave$
```

STEP 2 : INTEGRATION DES VARIABLES

	0	1	2	3	4	5	6	7
0	−17	0	16	16	0	0	0	0
1	0	−17	0	0	16	16	0	0
2	0	0	−17	16	16	0	16	0
3	0	0	0	−17	0	16	0	16
4	0	0	0	0	−17	16	16	0
5	0	0	0	0	0	−17	0	16
6	0	0	0	0	0	0	−17	0
7	0	0	0	0	0	0	0	−17

,




```
################################################################################
################################ parti 1 : G_n to QUBO #########################
################################################################################


################################ construction du qubo et du graph ############
n=1

nb_qbits = 31

g_node = read_qbits(nb_qbits, "noeuds_2.txt")
#print("nb_qubits :", g_node)
g_dw2x = read_coupleurs(nb_qbits, "coupleurs_dwaves.txt")
#print("nb_coupleurs :", g_dw2x)
#print()

g_nodep = read_qbitsp(nb_qbits, "noeuds_p.txt")
#print("nb_qubits_p :", g_nodep)
g_dw2xp = read_coupleursp(nb_qbits, "coupleurs_dwavep.txt")
#print("nb_coupleurs_p :", g_dw2xp)


Q1,norma=gntoqubo(n)
print Q1


H = nx.Graph()
H.add_nodes_from(g_nodep)
H.add_edges_from(g_dw2xp)



############################ recherche du mappage de G_n sur le graphe ###########################
embedding = find_embedding(Q1, g_dw2xp, random_seed=1000000000)
print("transformation : ", embedding),"\n"
```

General Embedding

General embedding refers to embedding that may be useful for any type of graph.

The primary utility function, `find_embedding()`, is an implementation of the heuristic algorithm described in [1]. It accepts various optional parameters used to tune the algorithm's execution or constrain the given problem.

This implementation performs on par with tuned, non-configurable implementations while providing users with hooks to easily use the code as a basic building block in research.

[1] https://arxiv.org/abs/1406.2741

find_embedding() [source]

Heuristically attempt to find a minor-embedding of source graph S into a target graph T.

Parameters

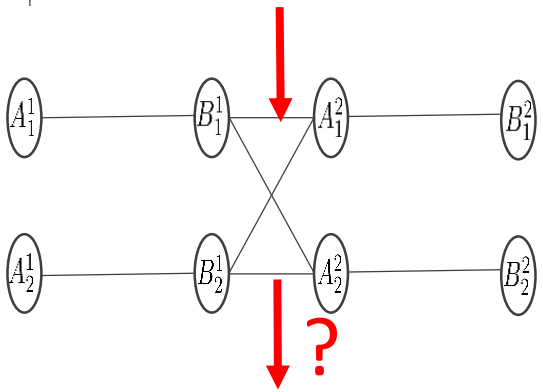
- **S** (*iterable/NetworkX Graph*) – The source graph as an iterable of label pairs representing the edges, or a NetworkX Graph.
- **T** (*iterable/NetworkX Graph*) – The target graph as an iterable of label pairs representing the edges, or a NetworkX Graph.
- ****params** (*optional*) – See below.

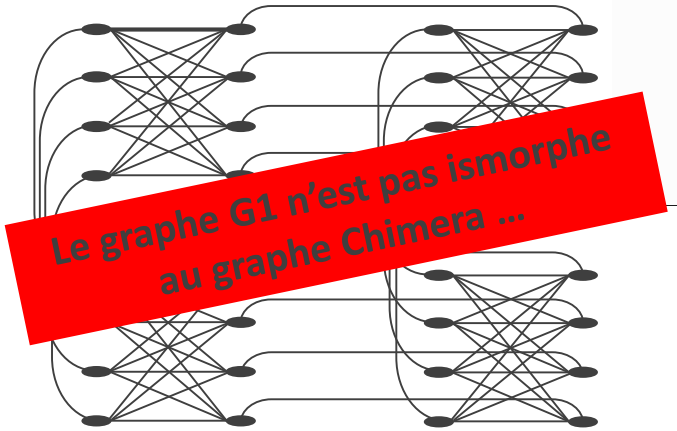
Returns

When the optional parameter `return_overlap` is False (the default), the function returns a

STEP 2 : INTEGRATION DES VARIABLES

	0	1	2	3	4	5	6	7
0	−17	0	16	16	0	0	0	0
1	0	−17	0	0	16	16	0	0
2	0	0	−17	16	16	0	16	0
3	0	0	0	−17	0	16	0	16
4	0	0	0	0	−17	16	16	0
5	0	0	0	0	0	−17	0	16
6	0	0	0	0	0	0	−17	0
7	0	0	0	0	0	0	0	−17





Le graphe G1 n'est pas ismorphe au graphe Chimera ...

General Embedding

General embedding refers to embedding that may be useful for any type of graph.

The primary utility function, `find_embedding()`, is an implementation of the heuristic algorithm described in [1]. It accepts various optional parameters used to tune the algorithm's execution or constrain the given problem.

This implementation performs on par with tuned, non-configurable implementations while providing users with hooks to easily use the code as a basic building block in research.

[1] https://arxiv.org/abs/1406.2741

find_embedding() [source]

Heuristically attempt to find a minor-embedding of source graph S into a target graph T.

Parameters

- **S** (*iterable/NetworkX Graph*) – The source graph as an iterable of label pairs representing the edges, or a NetworkX Graph.
- **T** (*iterable/NetworkX Graph*) – The target graph as an iterable of label pairs representing the edges, or a NetworkX Graph.
- ****params** (*optional*) – See below.

Returns

When the optional parameter `return_overlap` is False (the default), the function returns a

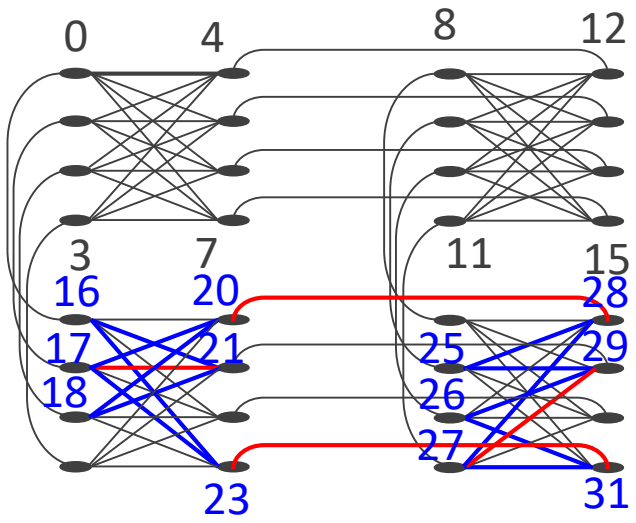
```
danielvm@danielvm-VirtualBox:~/Desktop/tp_dwave$ python umbedQ.py
('QUBO init :', {(6, 7): 16, (4, 6): 16, (0, 0): -17, (3, 7): 16, (3, 3): -17, (
6, 6): -17, (4, 5): 16, (5, 5): -17, (1, 7): 16, (2, 2): -17, (2, 4): 16, (2, 6)
: 16, (0, 5): 16, (0, 4): 16, (7, 7): -17, (1, 6): 16, (5, 7): 16, (1, 1): -17,
(4, 4): -17, (3, 5): 16})

('coupleur :', [(0, 4), (1, 4), (2, 4), (3, 4), (0, 5), (1, 5), (2, 5), (3, 5),
(0, 6), (1, 6), (2, 6), (3, 6), (0, 7), (1, 7), (2, 7), (3, 7), (4, 12), (8, 12)
, (9, 12), (10, 12), (11, 12), (5, 13), (8, 13), (9, 13), (10, 13), (11, 13), (6
, 14), (8, 14), (9, 14), (10, 14), (11, 14), (7, 15), (8, 15), (9, 15), (10, 15)
, (11, 15), (12, 20), (16, 20), (17, 20), (18, 20), (19, 20), (13, 21), (16, 21)
, (17, 21), (18, 21), (19, 21), (16, 22), (17, 22), (18, 22), (19, 22), (15, 23)
, (16, 23), (17, 23), (18, 23), (19, 23), (20, 28), (24, 28), (25, 28), (26, 28)
, (27, 28), (21, 29), (24, 29), (25, 29), (26, 29), (27, 29), (22, 30), (24, 30)
, (25, 30), (26, 30), (27, 30), (23, 31), (24, 31), (25, 31), (26, 31), (27, 31)
])

('transformation : ', {0: [26], 1: [18], 2: [25], 3: [16], 4: [29, 27], 5: [23,
31], 6: [20, 28], 7: [21, 17]})

danielvm@danielvm-VirtualBox:~/Desktop/tp_dwave$
```

Solution : dupliquer les qubits physiques !

STEP 2 : INTEGRATION DES VARIABLES

```
danielvm@danielvm-VirtualBox:~/Desktop/tp_dwave$ python umbedQ.py
('QUBO init :', {(6, 7): 16, (4, 6): 16, (0, 0): -17, (3, 7): 16, (3, 3): -17, (
6, 6): -17, (4, 5): 16, (5, 5): -17, (1, 7): 16, (2, 2): -17, (2, 4): 16, (2, 6)
: 16, (0, 5): 16, (0, 4): 16, (7, 7): -17, (1, 6): 16, (5, 7): 16, (1, 1): -17,
(4, 4): -17, (3, 5): 16})

('coupleur :', [(0, 4), (1, 4), (2, 4), (3, 4), (0, 5), (1, 5), (2, 5), (3, 5),
(0, 6), (1, 6), (2, 6), (3, 6), (0, 7), (1, 7), (2, 7), (3, 7), (4, 12), (8, 12)
, (9, 12), (10, 12), (11, 12), (5, 13), (8, 13), (9, 13), (10, 13), (11, 13), (6
, 14), (8, 14), (9, 14), (10, 14), (11, 14), (7, 15), (8, 15), (9, 15), (10, 15)
, (11, 15), (12, 20), (16, 20), (17, 20), (18, 20), (19, 20), (13, 21), (16, 21)
, (17, 21), (18, 21), (19, 21), (16, 22), (17, 22), (18, 22), (19, 22), (15, 23)
, (16, 23), (17, 23), (18, 23), (19, 23), (20, 28), (24, 28), (25, 28), (26, 28)
, (27, 28), (21, 29), (24, 29), (25, 29), (26, 29), (27, 29), (22, 30), (24, 30)
, (25, 30), (26, 30), (27, 30), (23, 31), (24, 31), (25, 31), (26, 31), (27, 31)
])

('transformation : ', {0: [26], 1: [18], 2: [25], 3: [16], 4: [29, 27], 5: [23,
31], 6: [20, 28], 7: [21, 17]})

danielvm@danielvm-VirtualBox:~/Desktop/tp_dwave$
```

Solution :
V0 → qubits 26
V1 → qubits 18
…
V4 → qubits 29 et 27
…
Total :
8 Variables pour 12 qubits
(1.5 fois plus de qubits)



Duplication de Qubit → Plus de variables et de coefficients non nuls dans le QUBO
→ Contraintes supplémentaires (contrainte forte sur les qubits dupliqués qui doivent être à la même valeur)
→ Augmentation de l'amplitude des coefficients pour représenter mon problème

STEP 2 : INTEGRATION DES VARIABLES

```
################################################################################################
############################### parti 1 : G_n to QUBO ##########################################
################################################################################################


############################### construction du qubo et du graph ###############################
n=1

nb_qbits = 31

g_node = read_qbits(nb_qbits, "noeuds_2.txt")
#print("nb_qubits :", g_node)
g_dw2x = read_coupleurs(nb_qbits, "coupleurs_dwaves.txt")
#print("nb_coupleurs :", g_dw2x)
#print()

g_nodep = read_qbitsp(nb_qbits, "noeuds_p.txt")
#print("nb_qubits_p :", g_nodep)
g_dw2xp = read_coupleursp(nb_qbits, "coupleurs_dwavep.txt")
#print("nb_coupleurs_p :", g_dw2xp)


Q1,norma=gntoqubo(n)
print Q1


H = nx.Graph()
H.add_nodes_from(g_nodep)
H.add_edges_from(g_dw2xp)



############################ recherche du mappage de G_n sur le graphe ##########################
embedding = find_embedding(Q1, g_dw2xp, random_seed=1000000000)
print("transformation : ", embedding),"\n"
```

Optional Parameters:

max_no_improvement (int, optional, default=10):

Maximum number of failed iterations to improve the current solution, where each iteration attempts to find an embedding for each variable of S such that it is adjacent to all its neighbours.

random_seed (int, optional, default=None):

Seed for the random number generator. If None, seed is set by `os.urandom()`.

timeout (int, optional, default=1000):

Algorithm gives up after timeout seconds.

max_beta (double, optional, max_beta=None):

Qubits are assigned weight according to a formula (beta^n) where n is the number of chains containing that qubit. This value should never be less than or equal to 1. If None, `max_beta` is effectively infinite.

tries (int, optional, default=10):

Number of restart attempts before the algorithm stops. On D-WAVE 2000Q, a typical restart takes between 1 and 60 seconds.

inner_rounds (int, optional, default=None):

The algorithm takes at most this many iterations between restart attempts; restart attempts are typically terminated due to `max_no_improvement`. If None, `inner_rounds` is effectively infinite.

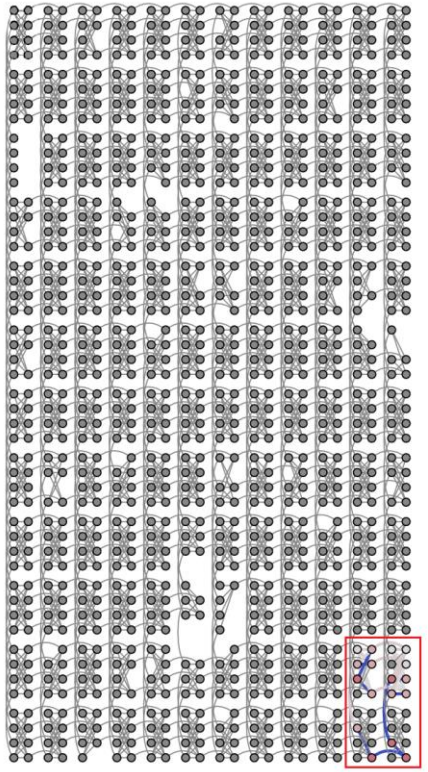
```
('transformation : ', {0: [112], 1: [122], 2: [121], 3: [114], 4: [116, 124], 5:
 [119, 115], 6: [125, 123], 7: [126, 118]})
```

```
('transformation : ', {0: [201], 1: [205], 2: [210, 212], 3: [331], 4: [215, 207
], 5: [206, 203], 6: [202, 204], 7: [200, 328, 332]})
```

Changer la seed (donc changer l'endroit d'intégration du problème sur la puce) change la qualité de la solution ?

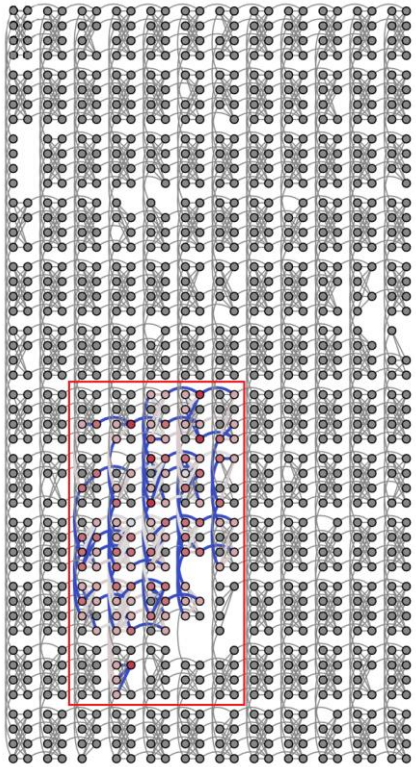
INSTANCES SUR DWAVE (DW2X → 1095 qubits)

- G_1 -



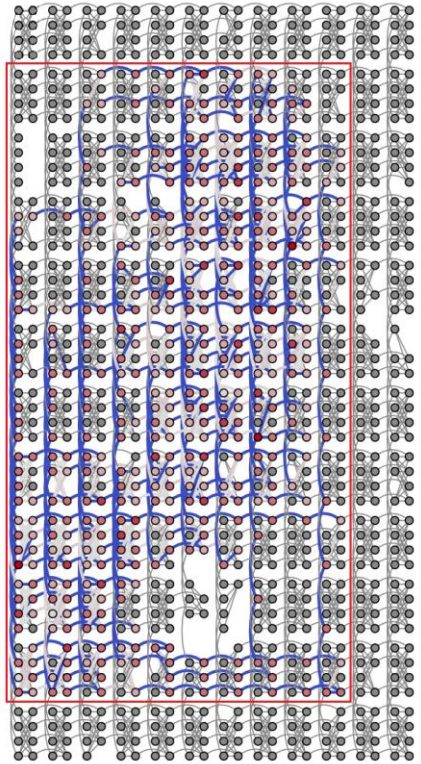
8 variables
16 qubits physiques → ~2%
Moy. dup. : 2.0
12 coefficients non 0

- G_2 -



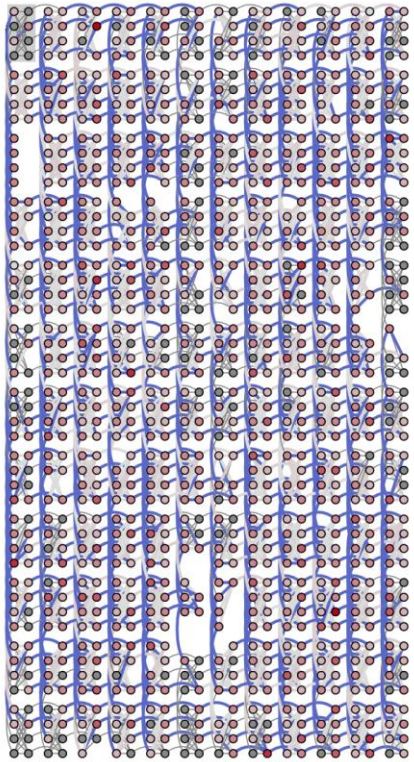
27 variables
100 qubits physiques → ~10%
Moy. dup. : 3.7
72 coefficients non 0

- G_3 -



64 variables
431 qubits physiques → ~40%
Moy. dup. : 6.7
240 coefficients non 0

- G_4 -



125 variables
951 qubits physiques → ~87%
Moy. dup. : 7.6
600 coefficients non 0

Daniel Vert. Étude des performances des machines à recuit quantique pour la résolution de problèmes combinatoires. Université Paris-Saclay, 2021. [(tel-03208838v2)]

STEP 3 : QUBO SUR DWAVE

	0	1	2	3	4	5	6	7
0	−17	0	16	16	0	0	0	0
1	0	−17	0	0	16	16	0	0
2	0	0	−17	16	16	0	16	0
3	0	0	0	−17	0	16	0	16
4	0	0	0	0	−17	16	16	0
5	0	0	0	0	0	−17	0	16
6	0	0	0	0	0	0	−17	0
7	0	0	0	0	0	0	0	−17

QUBO init

	0	1	2	3	4	5	6	7
0	−1	0	0.94	0.94	0	0	0	0
1	0	−1	0	0	0.94	0.94	0	0
2	0	0	−1	0.94	0.94	0	0.94	0
3	0	0	0	−1	0	0.94	0	0.94
4	0	0	0	0	−1	0.94	0.94	0
5	0	0	0	0	0	−1	0	0.94
6	0	0	0	0	0	0	−1	0
7	0	0	0	0	0	0	0	−1

QUBO norm

```
danielvm@danielvm-VirtualBox:~/Desktop/tp_dwave$ python umbedQ.py
('QUBO init :', {(6, 7): 16, (4, 6): 16, (0, 0): -17, (3, 7): 16, (3, 3): -17, (
6, 6): -1
, 5): 16, (5, 5): -17, (1, 7): 16, (2, 2): -17, (2, 4): 16, (2, 6)
: 16,
): 16, (0, 4): 16, (7, 7): -17, (1, 6): 16, (5, 7): 16, (1, 1): -17,
(4, 4
-17, (3, 5): 16})

('c
upleur :', [(0, 4), (1, 4), (2, 4), (3, 4), (0, 5), (1, 5), (2, 5), (3, 5),
(0
6), (1, 6), (2, 6), (3, 6), (0, 7), (1, 7), (2, 7), (3, 7), (4, 12), (8, 12)
, (9, 12), (10, 12), (11, 12), (5, 13), (8, 13), (9, 13), (10, 13), (11, 13), (6
, 14), (8, 14), (9, 14), (10, 14), (11, 14), (7, 15), (8, 15), (9, 15), (10, 15)
, (11, 15), (12, 20), (16, 20), (17, 20), (18, 20), (19, 20), (13, 21), (16, 21)
, (17, 21), (18, 21), (19, 21), (16, 22), (17, 22), (18, 22), (19, 22), (15, 23)
, (16, 23), (17, 23), (18, 23), (19, 23), (20, 28), (24, 28), (25, 28), (26, 28)
, (27, 28), (21, 29), (24, 29), (25, 29), (26, 29), (27, 29), (22, 30), (24, 30)
, (25, 30), (26, 30), (27, 30), (23, 31), (24, 31), (25, 31), (26, 31), (27, 31)
]

('
ransformation : ', {0: [26], 1: [18], 2: [25], 3: [16], 4: [29, 27], 5: [23,
31
6: [20, 28], 7: [21, 17]})

('QU
norm :', {(6, 7): 0.9411764705882353, (4, 6): 0.9411764705882353, (0, 0):
-1.0,
7): 0.9411764705882353, (3, 3): -1.0, (6, 6): -1.0, (4, 5): 0.9411764
7058823
5): -1.0, (1, 7): 0.9411764705882353, (2, 2): -1.0, (2, 4): 0.941
1764705882
, (2, 6): 0.9411764705882353, (0, 5): 0.9411764705882353, (0, 4): 0
.9411764705882353, (7, 7): -1.0, (1, 6): 0.9411764705882353, (5, 7): 0.941176470
5882353, (1, 1): -1.0, (4, 4): -1.0, (3, 5): 0.9411764705882353})
```

h_range

Supported Solvers:	Leap's Hybrid	QPU	Other
	✗	✓	✓ VFYC

Range of values possible for the qubit biases (linear coefficients), h, for this solver.

The auto_scale parameter, which rescales h and J values in the problem to use as much of the range of h (h_range) and the range of J (j_range) as possible, enables you to submit problems with values outside these ranges and have the system automatically scale them to fit.

Example

For the solver configured in the example setup above,

```
>>> qpu_advantage.properties["h_range"]
[-2.0, 2.0]
```

j_range

Supported Solvers:	Leap's Hybrid	QPU	Other
	✗	✓	✓ VFYC

Range of values possible for the coupling strengths (quadratic coefficients), J, for this solver.

The auto_scale parameter, which rescales h and J values in the problem to use as much of the range of h (h_range) and the range of J (j_range) as possible, enables you to submit problems with values outside these ranges and have the system automatically scale them to fit.

See also extended_j_range.

Example

For the solver configured in the example setup above,

```
>>> qpu_advantage.properties["j_range"]
[-1.0, 1.0]
```

Normalisation dans les paramètres de DWave

STEP 3 : QUBO SUR DWAVE

	0	1	2	3	4	5	6	7
0	−17	0	16	16	0	0	0	0
1	0	−17	0	0	16	16	0	0
2	0	0	−17	16	16	0	16	0
3	0	0	0	−17	0	16	0	16
4	0	0	0	0	−17	16	16	0
5	0	0	0	0	0	−17	0	16
6	0	0	0	0	0	0	−17	0
7	0	0	0	0	0	0	0	−17

QUBO init

	0	1	2	3	4	5	6	7
0	−1	0	0.94	0.94	0	0	0	0
1	0	−1	0	0	0.94	0.94	0	0
2	0	0	−1	0.94	0.94	0	0.94	0
3	0	0	0	−1	0	0.94	0	0.94
4	0	0	0	0	−1	0.94	0.94	0
5	0	0	0	0	0	−1	0	0.94
6	0	0	0	0	0	0	−1	0
7	0	0	0	0	0	0	0	−1

QUBO norm

```
danielvm@danielvm-VirtualBox:~/Desktop/tp_dwave$ python umbedQ.py
('QUBO init :', {(6, 7): 16, (4, 6): 16, (0, 0): -17, (3, 7): 16, (3, 3): -17, (
6, 6): -17, (4, 5): 16, (5, 5): -17, (1, 7): 16, (2, 2): -17, (2, 4): 16, (2, 6)
: 16, (0, 5): 16, (0, 4): 16, (7, 7): -17, (1, 6): 16, (5, 7): 16, (1, 1): -17,
(4, 4): -17, (3, 5): 16})

('coupleur :', [(0, 4), (1, 4), (2, 4), (3, 4), (0, 5), (1, 5), (2, 5), (3, 5),
(0, 6), (1, 6), (2, 6), (3, 6), (0, 7), (1, 7), (2, 7), (3, 7), (4, 12), (8, 12)
, (9, 12), (10, 12), (11, 12), (5, 13), (8, 13), (9, 13), (10, 13), (11, 13), (6
, 14), (8, 14), (9, 14), (10, 14), (11, 14), (7, 15), (8, 15), (9, 15), (10, 15)
, (11, 15), (12, 20), (16, 20), (17, 20), (18, 20), (19, 20), (13, 21), (16, 21)
, (17, 21), (18, 21), (19, 21), (16, 22), (17, 22), (18, 22), (19, 22), (15, 23)
, (16, 23), (17, 23), (18, 23), (19, 23), (20, 28), (24, 28), (25, 28), (26, 28)
, (27, 28), (21, 29), (24, 29), (25, 29), (26, 29), (27, 29), (22, 30), (24, 30)
, (25, 30), (26, 30), (27, 30), (23, 31), (24, 31), (25, 31), (26, 31), (27, 31)
])

('transformation : ', {0: [26], 1: [18], 2: [25], 3: [16], 4: [29, 27], 5: [23,
31], 6: [20, 28], 7: [21, 17]})

('QUBO norm :', {(6, 7): 0.9411764705882353, (4, 6): 0.9411764705882353, (0, 0):
-1.0, (3, 7): 0.9411764705882353, (3, 3): -1.0, (6, 6): -1.0, (4, 5): 0.9411764
705882353, (5, 5): -1.0, (1, 7): 0.9411764705882353, (2, 2): -1.0, (2, 4): 0.941
1764705882353, (2, 6): 0.9411764705882353, (0, 5): 0.9411764705882353, (0, 4): 0
.9411764705882353, (7, 7): -1.0, (1, 6): 0.9411764705882353, (5, 7): 0.941176470
5882353, (1, 1): -1.0, (4, 4): -1.0, (3, 5): 0.9411764705882353})

('QUBO dwave:', {(16, 16): -0.25, (23, 31): -1.0, (20, 20): 0.375, (16, 21): 0.2
3529411764705882, (20, 17): 0.23529411764705882, (29, 27): -1.0, (23, 17): 0.235
29411764705882, (26, 29): 0.23529411764705882, (27, 28): 0.23529411764705882, (1
8, 20): 0.23529411764705882, (25, 29): 0.23529411764705882, (26, 26): -0.25, (25
, 25): -0.25, (16, 23): 0.23529411764705882, (21, 21): 0.375, (27, 31): 0.235294
11764705882, (18, 21): 0.23529411764705882, (25, 28): 0.23529411764705882, (27,
27): 0.375, (29, 29): 0.375, (28, 28): 0.375, (23, 23): 0.375, (31, 31): 0.375,
(26, 31): 0.23529411764705882, (20, 28): -1.0, (18, 18): -0.25, (21, 17): -1.0,
(17, 17): 0.375})
danielvm@danielvm-VirtualBox:~/Desktop/tp_dwave$
```

```
# Embed the QUBO
target_Q = dimod.embed_qubo(Q1, embedding, H)
#print np.asarray(target_Q.items())

Ud= np.asarray(target_Q.items())[:,1]/4 # normalisation de dwave pour la duplication
QUBO=dict(zip(list(target_Q.keys()),Ud))
print("QUBO dwave:", QUBO), "\n"
```

Matrice QUBO intégrable sur le graphe de D-Wave

D-Wave Leap

Leap In

EMAIL ADDRESS

jane@gmail.com

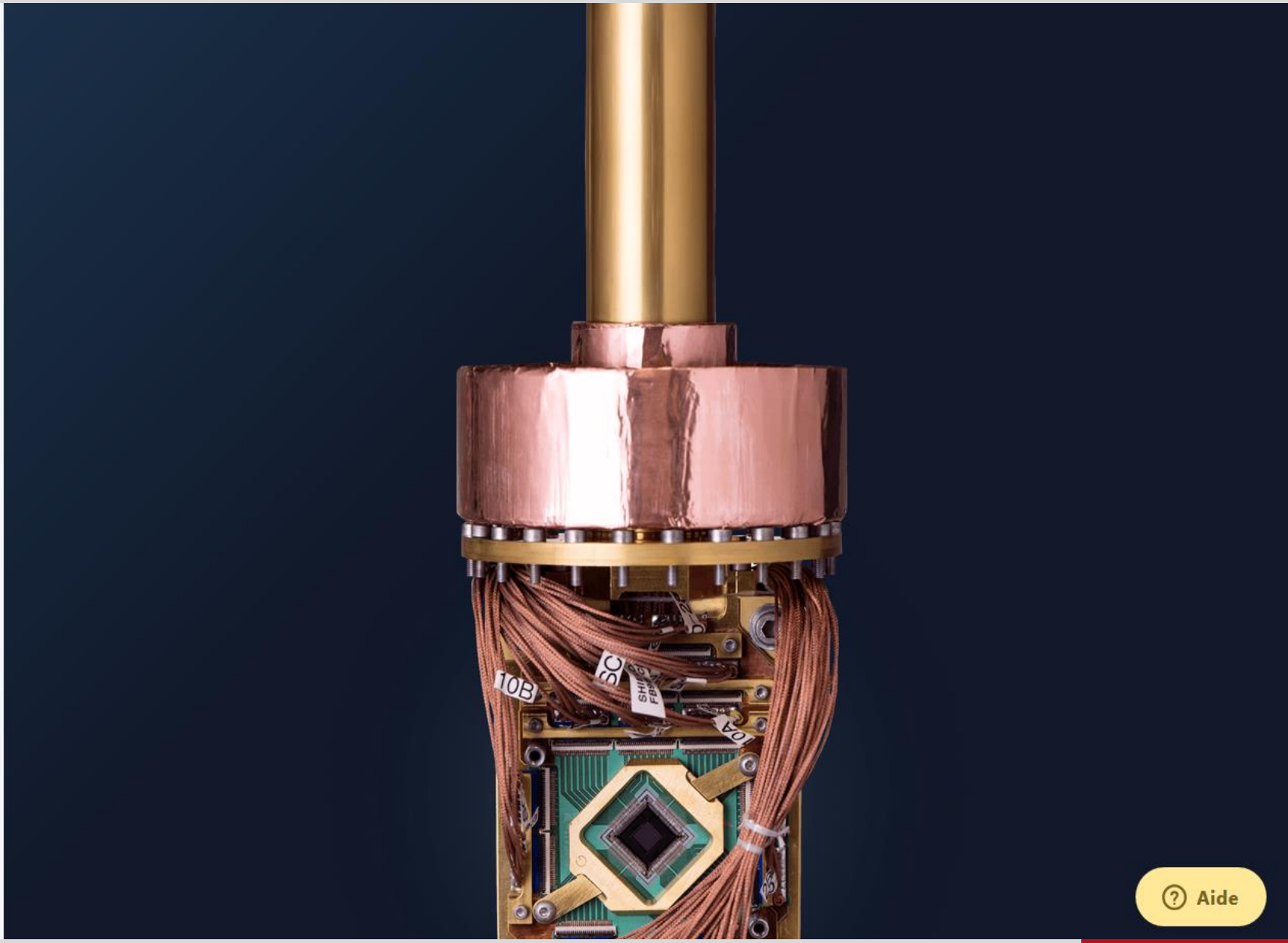
PASSWORD

Forgot password?
Having trouble logging in?

LOG IN

Don't have an account? Sign up

Lost activation link? Resend link



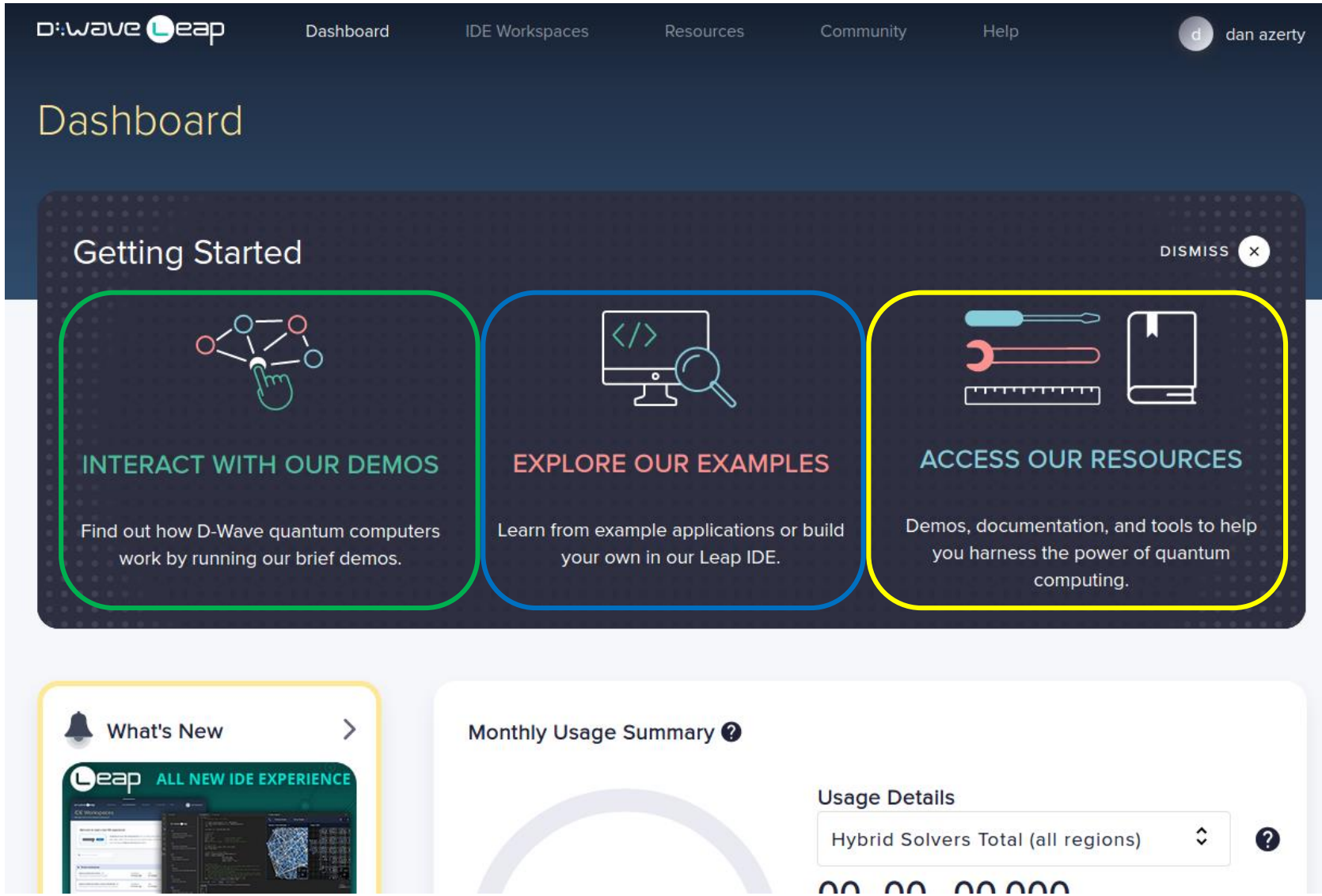
Aide

STEP 4 : Paramètres DWave

Interact with our demos

Explore our examples

Access our resources



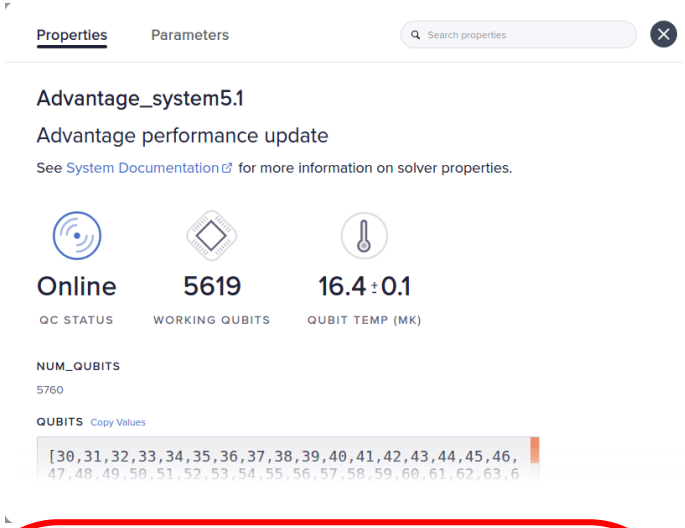
STEP 4 : PARAMETRES DE D-WAVE

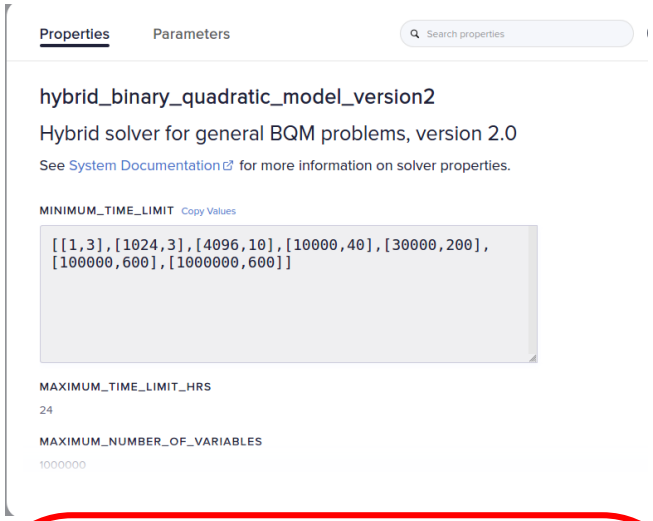
Europe (eu-central-1)

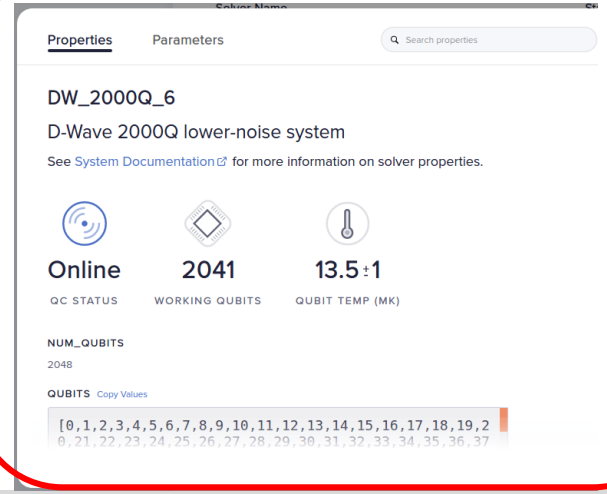
Solver Name	Status
QPU	
Advantage_system5.1 Advantage performance update	Online

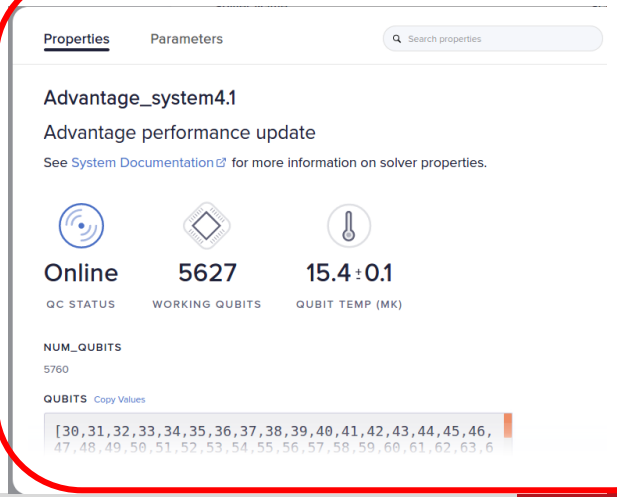
North America (na-west-1)

Solver Name	Status
Hybrid	
hybrid_binary_quadratic_model_version2 Hybrid solver for general BQM problems, version 2.0	Online
hybrid_constrained_quadratic_model_version1 Hybrid solver for general CQM problems, version 1.0	Online
hybrid_discrete_quadratic_model_version1 Hybrid solver for general DQM problems, version 1.0	Online
QPU	
Advantage_system4.1 Advantage performance update	Online
DW_2000Q_6 D-Wave 2000Q lower-noise system	Online



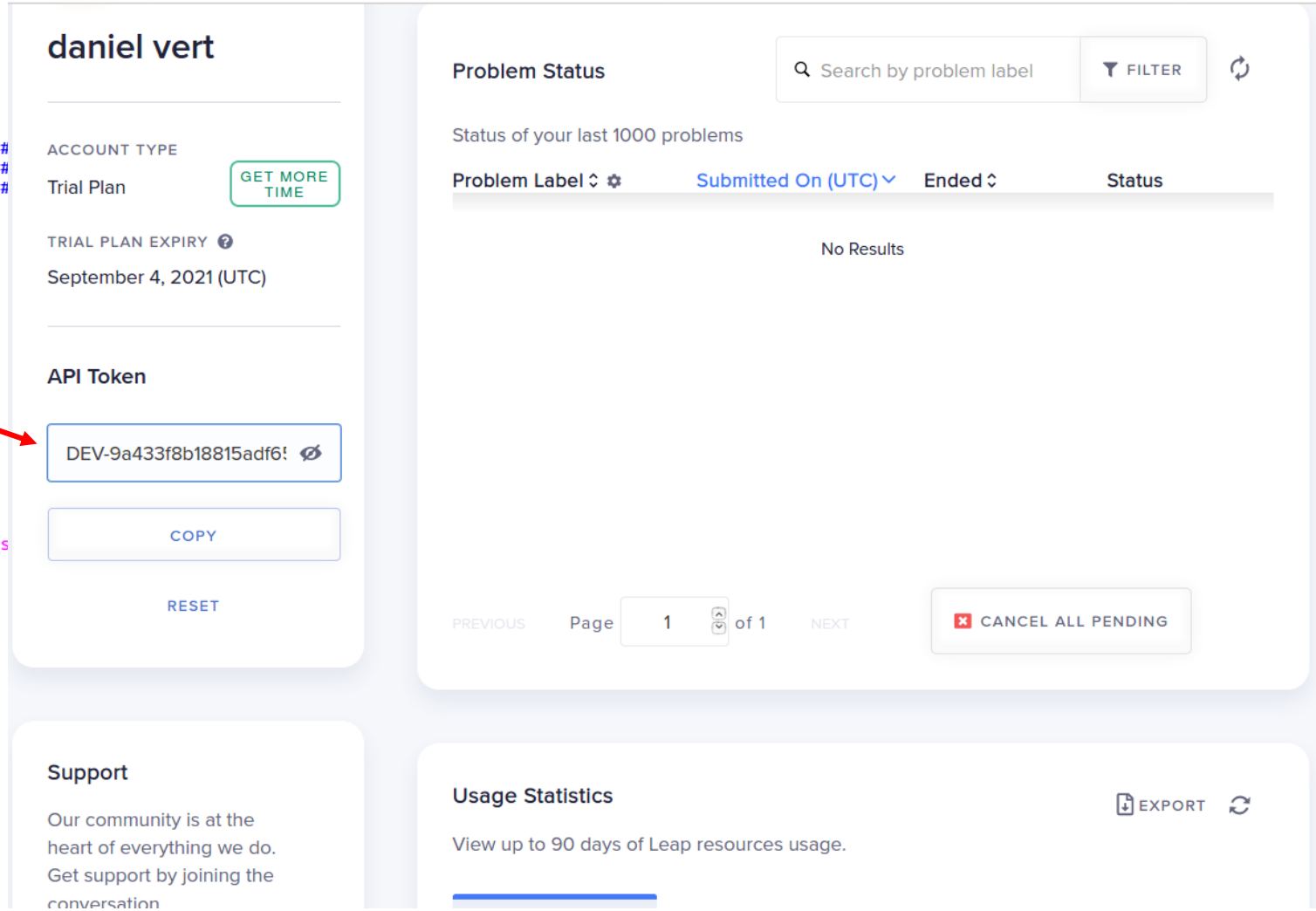






STEP 4 : PARAMETRES DE D-WAVE

```
##########################################################################################
################################### partie 2 : dwave_leap ################################
##########################################################################################

import dwave_sapi2
from dwave_sapi2.remote import RemoteConnection
from dwave_sapi2.core import solve_ising, solve_qubo

url = 'https://cloud.dwavesys.com/sapi/'
token = 'DEV-32b473f753f4641ca3e87d4318f325295665204c'

# create a remote connection
conn = RemoteConnection(url, token)

# get the solver
solver = conn.get_solver('DW_2000Q_6')
#solver = conn.get_solver('Advantage_system4.1')

params = { "num_reads": 1000, "max_answers": 1000, "answer_mode": "histogram", "auto_s

# solve QUBO problem with parameters
answer = solve_qubo(solver, QUBO, **params)
print"\n"
print("Energie:", answer["energies"])
print("Nb occurences:", answer["num_occurrences"])

i = 0
count = 0
while (answer["energies"][i]*(norma) <= -68) :

        count = count + answer["num_occurrences"][i]
        i = i + 1


print([energie*(norma) for energie in answer["energies"]])


print("Taux de réussite : " + str(float(count)/sum(answer["num_occurrences"])))

print("Energie moyenne : " + str(sum([answer["energies"][i] * answer["num_occurrences"][i] for i in range (len(answer["energies"]))]) / sum(answer["num_occurrences"])*(norma)))
```



STEP 4 : PARAMETRES DE D-WAVE

```
##########################################################################################################
################################### partie 2 : dwave_leap ################################################
##########################################################################################################

import dwave_sapi2
from dwave_sapi2.remote import RemoteConnection
from dwave_sapi2.core import solve_ising, solve_qubo

url = 'https://cloud.dwavesys.com/sapi/'
token = 'DEV-32b473f753f4641ca3e87d4318f325295665204c'

# create a remote connection
conn = RemoteConnection(url, token)

# get the solver
solver = conn.get_solver('DW_2000Q_6')
#solver = conn.get_solver('Advantage_system4.1')

params = { "num_reads": 1000, "max_answers": 1000, "answer_mode": "histogram", "auto_scale": True, "annealing_time": 20}, "num_spin_reversal_trans

# solve QUBO problem with parameters
answer = solve_qubo(solver, QUBO, **params)
print"\n"
print("Energie:", answer["energies"])
print("Nb occurences:", answer["num_occurrences"])

i = 0
count = 0
while (answer["energies"][i]*(norma) <= -68) :

        count = count + answer["num_occurrences"][i]
        i = i + 1


print([energie*(norma) for energie in answer["energies"]])


print("Taux de réussite : " + str(float(count)/sum(answer["num_occurrences"])))

print("Energie moyenne : " + str(sum([answer["energies"][i] * answer["num_occurrences"][i] for i in range (len(answer["energies"]))]) / sum(answer["num_occurrences"])*(norma)))
```

The examples below use the following setup:

```
>>> from dwave.system import DWaveSampler, LeapHybridSampler, LeapHybridCQMSampler
...
>>> qpu_advantage = DWaveSampler(solver={'topology__type': 'pegasus'})
>>> qpu_2000q = DWaveSampler(solver={'topology__type': 'chimera'})
>>> hybrid_bqm_sampler = LeapHybridSampler()
>>> hybrid_cqm_sampler = LeapHybridCQMSampler()
```

- anneal_offset_ranges
- anneal_offset_step
- anneal_offset_step_phi0
- annealing_time_range
- beta_range
- category
- chip_id
- couplers
- default_annealing_time
- default_beta
- default_programming_thermalization
- default_readout_thermalization
- extended_j_range
- h_gain_schedule_range
- h_range
- j_range
- max_anneal_schedule_points
- max_h_gain_schedule_points
- maximum_number_of_biases
- maximum_number_of_constraints
- maximum_number_of_quadratic_variables
- maximum_number_of_variables
- maximum_time_limit_hrs
- minimum_time_limit
- minimum_time_limit_s
- num_biases_multiplier
- num_constraints_multiplier
- num_qubits
- num_reads_range
- num_variables_multiplier
- parameters
- per_qubit_coupling_range
- problem_run_duration_range
- programming_thermalization_range

RESULTATS DE G1 SUR DWAVE

- **STEP 4 :** RESULTATS
  - Sortie 2000Q

```
('Energie:', [-1.0, -0.7794117647058824, -0.7794117647058824, -0.7794117647058824, -0.7794117647058824, -0.7647058823529412, -0.7647058823529412, -0.7647058823529412, -0.7647058823529412, -0.7647058823529412, -0.7647058823529412, -0.764705882352941, -0.7500000000000001, -0.7500000000000001, -0.7500000000000001, -0.7500000000000001, -0.7500000000000001, -0.7499999999999999])
('Nb occurences:', [930, 17, 9, 3, 7, 2, 2, 2, 3, 5, 6, 2, 3, 3, 2, 1, 1, 2, 1, 1])
[-68.0, -53.0, -53.0, -53.0, -53.0, -52.00000000000001, -52.00000000000001, -52.00000000000001, -52.00000000000001, -52.00000000000001, -52.00000000000001, -51.999999999999986, -51.00000000000001, -51.00000000000001, -51.00000000000001, -51.00000000000001, -51.00000000000001, -50.99999999999999]
Taux de réussite : 0.93
Energie moyenne : -66.908
danielvm@danielvm-VirtualBox:~/Desktop/tp_dwave$
```

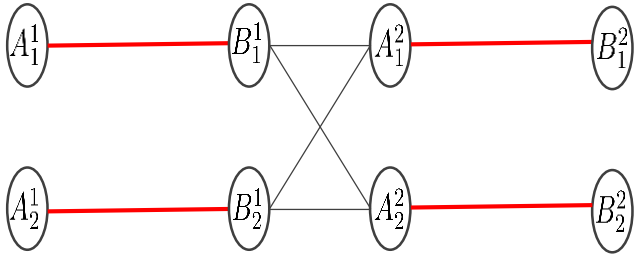
← Energies normalisées

← Energies totale pour G1

#q_p (va)	26(0)	18(1)	25(2)	16(3)	29(4)	27(4)	23(5)	31(5)	20(6)	28(6)	21(7)	17(7)
Ising	1	1	1	1	1	1	-1	-1	-1	-1	-1	-1
QUBO	1	1	1	1	1	1	0	0	0	0	0	0

Solution :11110000 Energie: -1x17x4 = -68 !

- Solution optimale : 4 x (-17) = -68 (11110000)



- **STEP 4 :** RESULTATS
  - Sortie 2000Q

```
('Energie:', [-1.0, -0.7794117647058824, -0.7794117647058824, -0.779411764705882
4, -0.7794117647058824, -0.7647058823529412, -0.7647058823529412, -0.76470588235
29412, -0.7647058823529412, -0.7647058823529412, -0.7647058823529412, -0.7647058
823529412, -0.764705882352941, -0.7500000000000001, -0.7500000000000001, -0.7500
000000000001, -0.7500000000000001, -0.7500000000000001, -0.7499999999999999])
('Nb occurences:', [930, 17, 9, 3, 7, 2, 2, 3, 5, 6, 2, 3, 3, 2, 1, 1, 2, 1, 1])
[-68.0, -53.0, -53.0, -53.0, -53.0, -52.00000000000001, -52.00000000000001, -52.
00000000000001, -52.00000000000001, -52.00000000000001, -52.00000000000001, -52.
00000000000001, -51.999999999999986, -51.00000000000001, -51.00000000000001, -51
.00000000000001, -51.00000000000001, -51.00000000000001, -50.99999999999999]
Taux de réussite : 0.93
Energie moyenne : -66.908
danielvm@danielvm-VirtualBox:~/Desktop/tp_dwave$
```

Energies normalisées

Energies totale pour G1

  - Sortie Advantage4.1

```
('transformation : ', {0: [600], 1: [571], 2: [3123], 3: [3107], 4: [3138], 5: [615], 6
: [586], 7: [3092]})

('QUBO norm :', {(6, 7): 0.9411764705882353, (4, 6): 0.9411764705882353, (0, 0): -1.0,
(3, 7): 0.9411764705882353, (3, 3): -1.0, (6, 6): -1.0, (4, 5): 0.9411764705882353, (5,
 5): -1.0, (1, 7): 0.9411764705882353, (2, 2): -1.0, (2, 4): 0.9411764705882353, (2, 6)
: 0.9411764705882353, (0, 5): 0.9411764705882353, (0, 4): 0.9411764705882353, (7, 7): -
1.0, (1, 6): 0.9411764705882353, (5, 7): 0.9411764705882353, (1, 1): -1.0, (4, 4): -1.0
, (3, 5): 0.9411764705882353})

('QUBO dwave:', {(3138, 615): 0.23529411764705882, (615, 3092): 0.23529411764705882, (5
71, 3092): 0.23529411764705882, (3123, 3123): -0.25, (600, 615): 0.23529411764705882, (
586, 586): -0.25, (3092, 3092): -0.25, (3107, 3092): 0.23529411764705882, (615, 615): -
0.25, (3138, 586): 0.23529411764705882, (571, 586): 0.23529411764705882, (600, 600): -0
.25, (600, 3138): 0.23529411764705882, (3138, 3138): -0.25, (3107, 3107): -0.25, (3107,
 615): 0.23529411764705882, (3123, 586): 0.23529411764705882, (3123, 3138): 0.235294117
64705882, (571, 571): -0.25, (586, 3092): 0.23529411764705882})



('Energie:', [-0.9999999999999999])
('Nb occurences:', [1000])
[-67.99999999999999]
Taux de réussite :  100.
Energie moyenne : -68.0
danielvm@danielvm-VirtualBox:~/Desktop/tp_dwave$
```

Pas de duplication

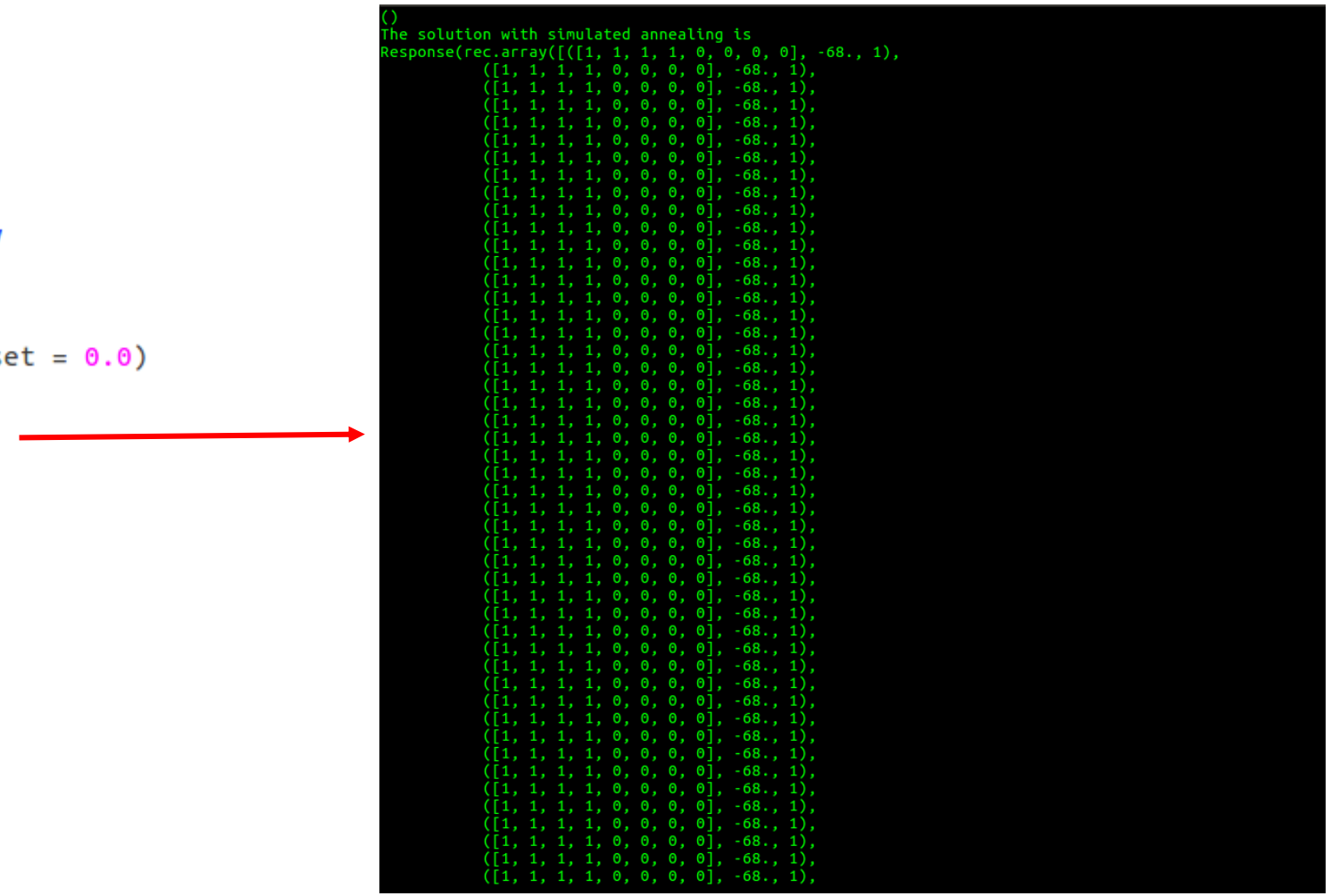
Wahou 100% de réussite ☺

STEP 4 : RESULTATS (suite)

Solver exact

```
# State the problem that we want to solve is very easy
# We will start with a simple case


model =  dimod.BinaryQuadraticModel.from_qubo(Q1, offset = 0.0)
model.linear
print("The model that we are going to solve is")
print(model)
print()

# We can solve it exactly


from dimod.reference.samplers import ExactSolver
sampleset = dimod.ExactSolver().sample_qubo(Q1)
#solution = sampler.sample(model)
print("The exact solution is")
print(sampleset)
print()
```

```
The model that we are going to solve is
BinaryQuadraticModel({0: -17, 1: -17, 2: -17, 3: -17, 4: -17, 5: -17, 6: -17, 7: -17}, {(2, 6): 16, (
6, 7): 16, (4, 6): 16, (4, 5): 16, (0, 4): 16, (3, 7): 16, (0, 5): 16, (1, 7): 16, (1, 6): 16, (5, 7)
: 16, (2, 4): 16, (3, 5): 16}, 0.0, Vartype.BINARY)
()
The exact solution is
Response(rec.array([([0, 0, 0, 0, 0, 0, 0, 0],   0., 1),
           ([1, 0, 0, 0, 0, 0, 0, 0], -17., 1),
           ([1, 1, 0, 0, 0, 0, 0, 0], -34., 1),
           ([0, 1, 0, 0, 0, 0, 0, 0], -17., 1),
           ([0, 1, 1, 0, 0, 0, 0, 0], -34., 1),
           ([1, 1, 1, 0, 0, 0, 0, 0], -51., 1),
           ([1, 0, 1, 0, 0, 0, 0, 0], -34., 1),
           ([0, 0, 1, 0, 0, 0, 0, 0], -17., 1),
           ([0, 0, 1, 1, 0, 0, 0, 0], -34., 1),
           ([1, 0, 1, 1, 0, 0, 0, 0], -51., 1),
           ([1, 1, 1, 1, 0, 0, 0, 0], -68., 1),
           ([0, 1, 1, 1, 0, 0, 0, 0], -51., 1),
           ([0, 1, 0, 1, 0, 0, 0, 0], -34., 1),
           ([1, 1, 0, 1, 0, 0, 0, 0], -51., 1),
           ([1, 0, 0, 1, 0, 0, 0, 0], -34., 1),
           ([0, 0, 0, 1, 0, 0, 0, 0], -17., 1),
           ([0, 0, 0, 1, 1, 0, 0, 0], -34., 1),
           ([1, 0, 0, 1, 1, 0, 0, 0], -35., 1),
           ([1, 1, 0, 1, 1, 0, 0, 0], -52., 1),
           ([0, 1, 0, 1, 1, 0, 0, 0], -51., 1),
           ([0, 1, 1, 1, 1, 0, 0, 0], -52., 1),
           ([1, 1, 1, 1, 1, 0, 0, 0], -53., 1),
           ([1, 0, 1, 1, 1, 0, 0, 0], -36., 1),
           ([0, 0, 1, 1, 1, 0, 0, 0], -35., 1),
           ([0, 0, 1, 0, 1, 0, 0, 0], -18., 1),
           ([1, 0, 1, 0, 1, 0, 0, 0], -19., 1),
           ([1, 1, 1, 0, 1, 0, 0, 0], -36., 1),
           ([0, 1, 1, 0, 1, 0, 0, 0], -35., 1),
           ([0, 1, 0, 0, 1, 0, 0, 0], -34., 1),
           ([1, 1, 0, 0, 1, 0, 0, 0], -35., 1),
           ([1, 0, 0, 0, 1, 0, 0, 0], -18., 1),
           ([0, 0, 0, 0, 1, 0, 0, 0], -17., 1),
           ([0, 0, 0, 0, 1, 1, 0, 0], -18., 1),
           ([1, 0, 0, 0, 1, 1, 0, 0],  -3., 1),
           ([1, 1, 0, 0, 1, 1, 0, 0], -20., 1),
           ([0, 1, 0, 0, 1, 1, 0, 0], -35., 1),
           ([0, 1, 1, 0, 1, 1, 0, 0], -36., 1),
           ([1, 1, 1, 0, 1, 1, 0, 0], -21., 1),
           ([1, 0, 1, 0, 1, 1, 0, 0],  -4., 1),
           ([0, 0, 1, 0, 1, 1, 0, 0], -19., 1),
           ([0, 0, 1, 1, 1, 1, 0, 0], -20., 1),
           ([1, 0, 1, 1, 1, 1, 0, 0],  -5., 1),
           ([1, 1, 1, 1, 1, 1, 0, 0], -22., 1),
```



RESULTATS DE G1 SUR SOLVEUR CLASSIQUE

- **STEP 4 :** RESULTATS (suite)
  - Recuit simulé (~100% de réussites)

```
# State the problem that we want to solve is very easy
# We will start with a simple case


model =  dimod.BinaryQuadraticModel.from_qubo(Q1, offset = 0.0)
model.linear
print("The model that we are going to solve is")
print(model)
print()

# We can solve it exactly


from dimod.reference.samplers import ExactSolver
sampleset = dimod.ExactSolver().sample_qubo(Q1)
#solution = sampler.sample(model)
print("The exact solution is")
print(sampleset)
print()
```

```
()
The solution with simulated annealing is
Response(rec.array([([1, 1, 1, 1, 0, 0, 0, 0], -68., 1),
           ([1, 1, 1, 1, 0, 0, 0, 0], -68., 1),
           ([1, 1, 1, 1, 0, 0, 0, 0], -68., 1),
           ([1, 1, 1, 1, 0, 0, 0, 0], -68., 1),
           ([1, 1, 1, 1, 0, 0, 0, 0], -68., 1),
           ([1, 1, 1, 1, 0, 0, 0, 0], -68., 1),
           ([1, 1, 1, 1, 0, 0, 0, 0], -68., 1),
           ([1, 1, 1, 1, 0, 0, 0, 0], -68., 1),
           ([1, 1, 1, 1, 0, 0, 0, 0], -68., 1),
           ([1, 1, 1, 1, 0, 0, 0, 0], -68., 1),
           ([1, 1, 1, 1, 0, 0, 0, 0], -68., 1),
           ([1, 1, 1, 1, 0, 0, 0, 0], -68., 1),
           ([1, 1, 1, 1, 0, 0, 0, 0], -68., 1),
           ([1, 1, 1, 1, 0, 0, 0, 0], -68., 1),
           ([1, 1, 1, 1, 0, 0, 0, 0], -68., 1),
           ([1, 1, 1, 1, 0, 0, 0, 0], -68., 1),
           ([1, 1, 1, 1, 0, 0, 0, 0], -68., 1),
           ([1, 1, 1, 1, 0, 0, 0, 0], -68., 1),
           ([1, 1, 1, 1, 0, 0, 0, 0], -68., 1),
           ([1, 1, 1, 1, 0, 0, 0, 0], -68., 1),
           ([1, 1, 1, 1, 0, 0, 0, 0], -68., 1),
           ([1, 1, 1, 1, 0, 0, 0, 0], -68., 1),
           ([1, 1, 1, 1, 0, 0, 0, 0], -68., 1),
           ([1, 1, 1, 1, 0, 0, 0, 0], -68., 1),
           ([1, 1, 1, 1, 0, 0, 0, 0], -68., 1),
           ([1, 1, 1, 1, 0, 0, 0, 0], -68., 1),
           ([1, 1, 1, 1, 0, 0, 0, 0], -68., 1),
           ([1, 1, 1, 1, 0, 0, 0, 0], -68., 1),
           ([1, 1, 1, 1, 0, 0, 0, 0], -68., 1),
           ([1, 1, 1, 1, 0, 0, 0, 0], -68., 1),
           ([1, 1, 1, 1, 0, 0, 0, 0], -68., 1),
           ([1, 1, 1, 1, 0, 0, 0, 0], -68., 1),
           ([1, 1, 1, 1, 0, 0, 0, 0], -68., 1),
           ([1, 1, 1, 1, 0, 0, 0, 0], -68., 1),
           ([1, 1, 1, 1, 0, 0, 0, 0], -68., 1),
           ([1, 1, 1, 1, 0, 0, 0, 0], -68., 1),
           ([1, 1, 1, 1, 0, 0, 0, 0], -68., 1),
           ([1, 1, 1, 1, 0, 0, 0, 0], -68., 1),
           ([1, 1, 1, 1, 0, 0, 0, 0], -68., 1),
           ([1, 1, 1, 1, 0, 0, 0, 0], -68., 1),
           ([1, 1, 1, 1, 0, 0, 0, 0], -68., 1),
           ([1, 1, 1, 1, 0, 0, 0, 0], -68., 1),
           ([1, 1, 1, 1, 0, 0, 0, 0], -68., 1),
           ([1, 1, 1, 1, 0, 0, 0, 0], -68., 1),
           ([1, 1, 1, 1, 0, 0, 0, 0], -68., 1),
           ([1, 1, 1, 1, 0, 0, 0, 0], -68., 1),
           ([1, 1, 1, 1, 0, 0, 0, 0], -68., 1),
```

daniel.vert2@cea.fr

STEP 4 : G_max?

```
('transformation : ', {0: [631, 623], 1: [620, 628], 2: [495, 489], 3: [235, 480
, 608, 224, 352, 229, 237], 4: [381, 373, 365], 5: [505, 377, 249], 6: [114], 7:
 [108], 8: [372], 9: [614, 630, 617, 622], 10: [360, 616, 492, 488], 11: [508, 6
26, 498, 500], 12: [613, 625, 629, 621], 13: [618, 362, 490], 14: [382, 366, 374
, 624, 368, 496], 15: [363, 619, 491], 16: [497, 369, 502, 494], 17: [509, 499,
493, 501], 18: [245, 117, 240, 112], 19: [107, 104, 109], 20: [232, 238, 370, 24
6, 242], 21: [378, 250, 122, 127, 119], 22: [367, 111, 361, 233, 105], 23: [380,
 376, 383, 375], 24: [118, 121, 126, 115], 25: [255, 239, 234, 247, 110, 106], 2
6: [252, 244, 371, 243]})



('Nb occurences:', [23, 2, 1, 3, 1, 4, 3, 2, 3, 1, 6, 1, 2, 3, 3, 3, 2, 3, 2, 5,
 7, 3, 5, 2, 3, 8, 1, 1, 1, 2, 7, 1, 1, 2, 1, 4, 2, 1, 1, 2, 1, 2, 1, 1, 2, 1, 1
, 1, 4, 1, 1, 2, 2, 1, 3, 2, 2, 1, 1, 2, 2, 1, 1, 2, 1, 2, 3, 5, 1, 1, 2, 7, 5,
3, 1, 2, 1, 4, 1, 2, 1, 2, 1, 3, 1, 8, 2, 2, 1, 3, 1, 1, 1, 2, 2, 1, 1, 4, 1, 1]
)
[-495.0, -442.0000000000021, -442.0000000000013, -442.0000000000013, -442.000000
0000013, -442.0000000000005, -442.0000000000005, -442.0000000000005, -442.000000
0000005, -442.0000000000005, -442.0000000000005, -442.0000000000005, -442.000000
0000005, -441.9999999999998, -441.9999999999998, -441.9999999999998, -441.999999
9999998, -441.9999999999998, -441.9999999999998, -441.9999999999998, -441.999999
9999998, -441.9999999999998, -441.9999999999998, -441.9999999999998, -441.999999
9999998, -441.9999999999998, -441.999999999999, -441.999999999999, -441.99999999
9999, -441.999999999999, -441.999999999999, -441.999999999999, -441.999999999999
, -441.999999999999, -441.999999999999, -441.0000000000022, -441.0000000000022,
-441.0000000000014, -441.0000000000014, -441.0000000000014, -441.0000000000014,
-441.0000000000007, -441.0000000000007, -441.0000000000007, -441.0000000000007,
-441.0000000000007, -441.0000000000007, -441.0000000000007, -441.0000000000007,
-441.0000000000007, -441.0000000000007, -441.0000000000007, -441.0000000000007,
-441.0000000000007, -441.0000000000007, -441.0000000000007, -441.0000000000007,
-440.9999999999999, -440.9999999999999, -440.9999999999999, -440.9999999999999,
-440.9999999999999, -440.9999999999999, -440.9999999999999, -440.9999999999999,
-440.9999999999999, -440.9999999999999, -440.9999999999999, -440.9999999999999,
-440.9999999999999, -440.9999999999999, -440.9999999999999, -440.9999999999999,
-440.9999999999999, -440.9999999999999, -440.9999999999999, -440.9999999999999,
-440.9999999999991, -440.9999999999991, -440.9999999999991, -440.9999999999991,
-440.9999999999991, -440.9999999999991, -440.9999999999991, -440.9999999999983,
-440.0000000000016, -440.0000000000016, -440.0000000000016, -440.0000000000008,
-440.0000000000008, -440.0000000000008, -440.0000000000008, -440.0, -440.0, -440
.0, -440.0]
Taux de réussite : 0.0934959349593
Energie moyenne : -446.361788618
```

G2

~100 qubits pour 27 varaibles …

Trouve la solution optimale mais → 23 fois sur 1000 …

- **STEP 4 :** G_max?

G2

```
^[^[^[The solution with simulated annealing is
Response(rec.array([([1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], -495., 1),
           ([1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], -495., 1),
           ([1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], -495., 1),
           ([1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], -495., 1),
           ([1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], -495., 1),
           ([1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], -495., 1),
           ([1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], -495., 1),
           ([1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], -495., 1),
           ([1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], -495., 1),
           ([1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], -495., 1),
           ([1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], -495., 1),
           ([1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], -495., 1),
           ([1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], -495., 1),
           ([1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], -495., 1),
           ([1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], -495., 1),
           ([1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], -495., 1),
           ([1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], -495., 1),
           ([1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], -495., 1),
           ([1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], -495., 1),
           ([1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], -495., 1),
           ([1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], -495., 1),
           ([1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], -495., 1),
           ([1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], -495., 1),
           ([1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], -495., 1),
           ([1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], -495., 1),
           ([1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], -495., 1),
           ([1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], -495., 1),
           ([1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], -495., 1),
           ([1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], -495., 1),
           ([1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], -495., 1),
           ([1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], -495., 1),
           ([1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], -495., 1),
           ([1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], -495., 1),
           ([1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], -495., 1),
           ([1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], -495., 1),
           ([1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], -495., 1),
           ([1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], -495., 1),
           ([1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], -495., 1),
           ([1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], -495., 1),
           ([1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], -495., 1),
           ([1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], -495., 1),
           ([1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], -495., 1),
           ([1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], -495., 1),
           ([1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], -495., 1),
           ([1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], -495., 1),
           ([1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], -495., 1),
           ([1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], -495., 1),
           ([1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], -495., 1),
```

Trouve la solution optimale ~100% sur 1000

RESULTATS (suite)

- **STEP 4 :** G_max?

G3

```
('transformation : ', {0: [441, 447], 1: [314, 186], 2: [75, 77], 3: [456, 328], 4: [216, 254, 246, 238, 230, 88, 603, 475, 3
47, 219, 222], 5: [476, 468, 484, 594, 466, 338, 210, 207, 199, 215], 6: [327, 343, 212, 340, 335, 211, 339], 7: [221, 218, 4
55, 463, 495, 346, 474, 487, 479, 471], 8: [482, 610, 354, 738, 866], 9: [363, 491, 1131, 748, 1003, 875, 747, 619], 10: [752
, 636, 625, 627, 626, 624, 628], 11: [984, 856, 344, 472, 735, 600, 736, 741, 733, 728], 12: [1000, 1128, 1127, 1135], 13: [9
89, 985, 857, 729, 734], 14: [745, 749, 757, 765, 761, 1017, 895, 889], 15: [997], 16: [606, 598, 582, 576, 448, 48, 176, 590
, 304, 432, 437, 453, 445], 17: [205, 197, 191, 440, 312, 189, 184], 18: [187, 319, 443, 315], 19: [442, 446, 450, 454], 20:
[58, 55, 67, 64, 95, 63, 87, 79, 71], 21: [190, 192, 206, 198], 22: [320, 326, 57, 318, 185, 313], 23: [195, 317, 325, 451, 3
23], 24: [92, 84, 52, 60, 68, 201, 73, 76], 25: [78, 72, 200], 26: [188, 196, 203, 204], 27: [69, 70, 449, 321, 65, 193], 28:
 [214, 461, 80, 208, 464, 336, 333, 329, 457, 470, 462], 29: [458, 334, 74, 330, 202], 30: [316, 324, 331, 332], 31: [452, 45
9, 460], 32: [253, 377, 505, 633, 617, 638, 630, 249, 614, 622], 33: [241, 365, 369, 381, 373], 34: [632, 504, 248, 376, 508,
 500], 35: [225, 497, 502, 609, 353, 481, 494, 478, 486], 36: [621, 605, 597, 613], 37: [620, 618, 490, 362, 492], 38: [593,
465, 469, 485, 499, 477, 501, 493], 39: [473, 601, 602, 345, 596, 612, 604], 40: [350, 342, 611, 483, 227, 358, 355], 41: [34
8, 372, 356, 364], 42: [220, 224, 228, 236, 244, 240, 496, 368], 43: [341, 349, 357, 608, 480, 352], 44: [242, 233, 616, 226,
 360, 488, 232, 229, 245, 237], 45: [374, 489, 366, 361], 46: [375, 370, 503, 498], 47: [351, 367, 359], 48: [872, 870, 878],
 49: [884, 860, 876, 868], 50: [869, 877, 1001, 873], 51: [871, 993, 1121, 865], 52: [1116, 1002, 1140, 1124, 1130, 874, 1132
], 53: [737, 1117, 740, 732, 730, 858, 1125, 1114, 986], 54: [888, 1016, 1144, 1149, 1141, 1129, 1133], 55: [994, 1122, 1118,
 1126, 1134], 56: [1006, 1143, 754, 1138, 1014, 1010, 886, 882], 57: [883, 742, 750, 755, 758], 58: [893, 880, 885], 59: [100
5, 1137, 879, 887, 1009, 1013, 753, 1142, 881], 60: [1120, 998, 864, 992], 61: [1011, 1012, 1004, 731, 996, 859, 987, 988], 6
2: [1008, 1023, 1015, 1007, 991, 999], 63: [1123, 759, 751, 743, 739, 867, 995]})


('Nb occurences:', [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1])
[-1935.9999999999777, -1934.9999999999486, -1810.0000000000211, -1810.0000000000064, -1809.9999999999845, -1809.9999999999625
, -1809.0000000000432, -1809.0000000000214, -1809.0000000000214, -1809.0000000000066, -1809.0000000000066, -1808.999999999992
, -1808.999999999992, -1808.9999999999773, -1808.9999999999627, -1808.9999999999627, -1808.9999999999627, -1808.999999999948,
 -1808.999999999948, -1808.0000000000362, -1808.000000000014, -1807.9999999999995, -1807.9999999999923, -1807.9999999999923,
-1807.9999999999923, -1807.9999999999923, -1807.9999999999775, -1807.9999999999775, -1807.9999999999702, -1807.9999999999702,
 -1807.9999999999702, -1807.9999999999554, -1807.9999999999554, -1807.9999999999482, -1807.9999999999336, -1807.9999999999336
, -1806.9999999999704, -1806.9999999999632, -1805.9999999999927, -1683.00000000005, -1683.0000000000207, -1683.0000000000132,
 -1683.000000000006, -1683.000000000006, -1683.000000000006, -1682.9999999999914, -1682.9999999999914, -1682.9999999999914, -
1682.9999999999693, -1682.9999999999545, -1682.0000000000575, -1682.0000000000355, -1682.0000000000282, -1682.0000000000282,
-1682.000000000021, -1682.0000000000136, -1682.0000000000136, -1682.0000000000061, -1682.0000000000061, -1682.0000000000061,
-1682.0000000000061, -1682.0000000000061, -1682.0000000000061, -1681.9999999999989, -1681.9999999999989, -1681.9999999999989,
 -1681.9999999999989, -1681.9999999999989, -1681.9999999999989, -1681.9999999999989, -1681.9999999999989, -1681.9999999999916
, -1681.9999999999916, -1681.9999999999916, -1681.9999999999916, -1681.9999999999916, -1681.9999999999916, -1681.999999999984
, -1681.999999999984, -1681.999999999984, -1681.999999999984, -1681.999999999984, -1681.999999999984, -1681.999999999984, -16
81.9999999999768, -1681.9999999999768, -1681.9999999999768, -1681.9999999999768, -1681.9999999999768, -1681.9999999999768, -1
681.9999999999695, -1681.9999999999695, -1681.9999999999623, -1681.9999999999623, -1681.999999999955, -1681.999999999955, -16
81.9999999999475, -1681.9999999999475, -1681.9999999999402, -1681.9999999999402]
Taux de réussite : 0.0
Energie moyenne : -1733.97
```

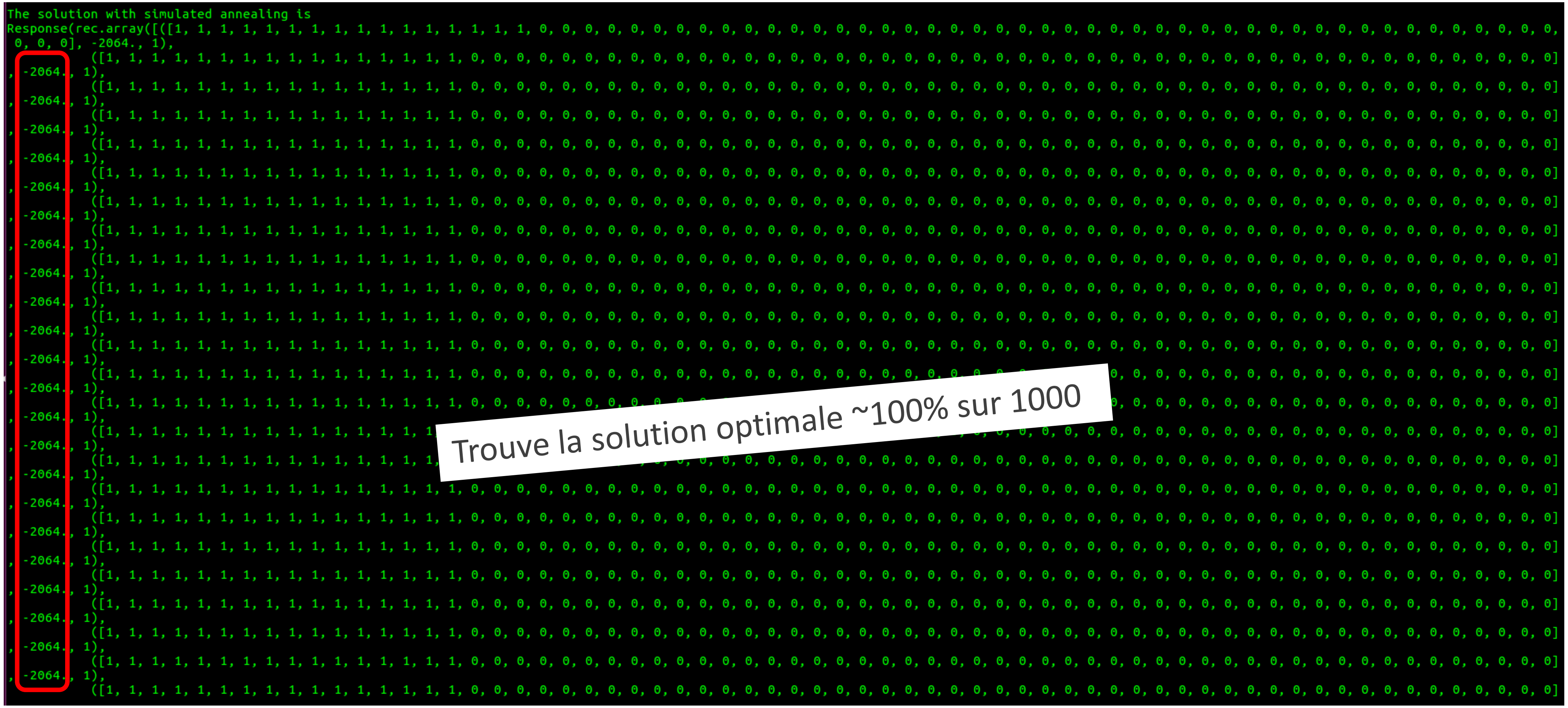
~430 qubits pour 64 varaibles …

Solution optimale 0 …

~ 6% d'écart dans le meilleur des cas
(ici on a eu de la chance)

- **STEP 4 :** G_max?

G3



- **STEP 4 :** G_max?

G4


```
('transformation : ', {0: [1572, 1580], 1: [1717], 2: [1046], 3: [1558, 1566], 4: [1341], 5: [1574, 1440, 1312, 1056, 928, 800, 544, 416, 288, 160, 166, 174, 190, 198, 206, 1184, 238, 182, 230, 1568, 672,
 222, 214], 6: [1045, 1048, 1053, 1061, 1569, 1441, 247, 239, 231, 223, 215, 207, 1313, 1185, 1057, 929, 801, 673, 417, 161, 167, 175, 183, 289, 191, 545, 199], 7: [1049, 1050, 1587, 1203, 1075, 1044, 108
2, 1331, 1107, 1060, 1068, 1108, 1459, 1076, 1084, 1092, 1052, 1116, 1100], 8: [1350, 1346, 1218, 1058, 1186, 1189, 1197, 1205, 1213, 1221, 1229, 1261, 1253, 1245, 1237], 9: [1699, 1443, 1367, 1359, 1351,
 1335, 1327, 1571, 1319, 212, 204, 196, 188, 220, 180, 164, 172, 163, 419, 291, 547, 675, 803, 1343, 931, 1059, 1187, 1315], 10: [2014, 2006, 1998, 1990, 758, 1984, 1856, 1728, 1600, 1472, 1344, 1216, 108
8, 1094, 1102, 1098, 970, 714, 718, 726, 734, 1002, 842, 874, 746, 750, 742], 11: [1890, 1395, 1391, 1241, 1369, 1762, 1634, 1375, 1399, 1383, 1378, 1506], 12: [1376, 1377, 1249, 1250, 1236, 1268, 1260, 1
248, 1754, 1626, 1498, 1370, 1242, 1244, 1252], 13: [1649, 1264, 1136, 1396, 1392, 1393, 1521], 14: [1620, 1612, 1604, 1596, 1588, 1586, 1458, 1202, 1074, 1330, 610, 738, 740, 732, 724, 716, 708, 700, 692
, 690, 818, 946], 15: [1732, 1893, 1897, 959, 967, 1091, 2031, 2023, 2015, 2025, 2007, 1999, 1991, 1987, 963, 1859, 1731, 1901, 1603, 1475, 1347, 1219], 16: [1380, 1388, 1372, 1364, 1348, 1490, 1485, 1477
, 1473, 1356, 1493, 1345, 1095, 1077, 1085, 1093, 1089, 1217], 17: [1532, 1662, 1530, 1786, 1534, 1658, 731, 603, 599, 607, 1661, 615, 623, 631, 639, 1400, 1272, 1144, 1016, 733, 888, 1528, 632, 760], 18:
 [1875, 1885, 1869, 1861, 1853, 1845, 1837, 1829, 1821, 1813, 1805, 1801, 1673, 1545, 1417, 1289, 1161, 1033, 1877, 905, 521, 649, 777], 19: [1743, 1997, 1992, 1736, 1608, 1864, 1480, 453, 461, 469, 467,
595, 723, 851, 979, 980, 972, 958, 966, 974, 968, 1096, 1224, 1352], 20: [427], 21: [685, 693, 819, 691], 22: [450, 578], 23: [197, 192, 320], 24: [706, 834, 837], 25: [1428, 1578, 1436, 1450, 1444, 1452]
, 26: [1301, 1309, 1577, 1449, 1321, 1325, 1317], 27: [1564, 1308, 1437, 1562, 1178, 1306, 1434], 28: [1576, 1316, 1324, 1064, 1192, 1448, 1320], 29: [1314, 1698, 1445, 1570, 1442], 30: [1455, 1709, 1590,
 1451, 1582, 1707, 1579], 31: [1426, 1554, 1559, 1713, 1585, 1567, 1575, 1591, 1583], 32: [1552, 1686, 1690, 1296, 1715, 1680, 1702, 1694, 1710, 1424, 1718], 33: [1079, 1072, 1200, 1332, 1328, 1456, 1712,
 1584], 34: [1553, 1681, 1695, 1687, 1714, 1703, 1711, 1719], 35: [1175, 1043, 1439, 1431, 1447, 1427, 1299, 1171], 36: [1172, 1298, 1042, 1170], 37: [1040, 1181, 1173, 1168], 38: [1054, 1179, 1051, 1047,
 1071, 1063, 1055], 39: [1419, 1422, 1041, 1169, 1297, 1425, 1446, 1438, 1430], 40: [1581, 1555, 1557, 1573, 1565], 41: [1303, 1560, 1432, 1176, 1311, 1304], 42: [1689, 1561, 1433, 1177, 1305], 43: [1563,
 1435, 1307, 1342, 1334, 1318, 1326, 1310], 44: [1550, 1549, 1688, 1691, 1547, 1675, 1676, 1684, 1700, 1692], 45: [1589, 1597, 1595, 1322, 1194, 1323, 1339, 1467, 1211, 1183, 1215, 1207, 1195, 1191, 1199]
, 46: [1457, 1201, 1180, 1188, 1212, 1333, 1204, 1193, 1329, 1196], 47: [1174, 1726, 1722, 1466, 1338, 1210, 1182, 1190, 1594, 1198, 1214, 1206], 48: [1070, 1078, 1086, 1340, 1080, 1336, 1065, 1208], 49:
[1461, 1469, 1708, 1716, 1454, 1462, 1470, 1453, 1727, 1724, 1721, 1465, 1593, 1209, 1337], 50: [233, 489, 361, 617, 745, 866, 873, 876, 860, 868], 51: [344, 472, 600, 1368, 216, 728, 856, 984, 1112, 1240
], 52: [224, 352, 480, 608, 1109, 736, 864, 1117, 1125, 1120, 992], 53: [1141, 1149, 254, 249, 377, 246, 505, 633, 761, 861, 869, 877, 1017, 889, 893, 1145, 885], 54: [227, 355, 483, 735, 739, 751, 867, 6
11, 743], 55: [240, 368, 496, 854, 894, 624, 752, 880, 862, 870, 878, 886], 56: [336, 464, 1360, 1270, 1262, 592, 720, 848, 208, 1254, 976, 1104, 1232, 1246, 1238], 57: [371, 1022, 994, 1267, 1139, 499, 7
55, 883, 1011, 627, 998, 1006, 243, 1014], 58: [1404, 255, 251, 1403, 379, 507, 1278, 763, 891, 1019, 635, 1275, 1147], 59: [232, 360, 488, 598, 606, 638, 630, 614, 616, 622], 60: [730, 1114, 858, 996, 98
8, 986], 61: [1371, 1247, 987, 1115, 1243], 62: [1122, 1110, 1118, 1126], 63: [1111, 1151, 1119, 1143, 1127, 1135], 64: [1124, 859, 863, 1121, 993, 609, 737, 871, 865], 65: [1004, 1008, 1013, 1385, 997, 1
129, 1001, 1257, 1005], 66: [1235, 1363, 1366, 1384, 1379, 1374, 1390, 1382], 67: [983, 1007, 999, 995, 1251, 991, 1123], 68: [1142, 1134, 1279, 1150, 1387, 1003, 1255, 1271, 1131, 1259, 1263], 69: [748,
619, 747, 1256, 1128, 1000, 744, 875, 879, 872], 70: [217, 981, 1106, 473, 978, 989, 985, 601, 345, 729, 857], 71: [1269, 1277, 1273, 1401, 1234, 1365, 1405, 1362, 1397, 1389, 1381, 1373], 72: [468, 465,
1239, 597, 990, 982, 1105, 1233, 593, 721, 1361, 849, 977], 73: [347, 475, 476, 484, 492, 500, 497, 625, 753, 219, 881, 1009, 1148, 1140, 1265, 1137], 74: [346, 1402, 1274, 1146, 634, 637, 474, 1018, 602,
 890, 762, 629, 621, 218, 605, 613], 75: [1900, 2026, 2022, 1896, 2024, 1768, 2030], 76: [1130, 1258, 1899, 1771, 1645, 1386, 1514, 1517, 1515, 1643], 77: [754, 882, 1010, 1138, 1266, 1394, 2038, 2036, 19
10, 2034, 1906, 1522, 1653, 1778, 1650], 78: [2008, 1880, 1902, 1894, 1886], 79: [1988, 2027, 1993, 2009, 1996, 2004, 2028, 2020, 2012], 80: [1766, 1774, 1642, 1898, 1770], 81: [1622, 1618, 1624, 1630, 16
46, 1638], 82: [1782, 1911, 1907, 1654, 1523, 1779, 1651], 83: [1746, 1874, 1871, 1903, 1895, 1879, 1887], 84: [1655, 1647, 1865, 1737, 1609, 1615, 1639, 1631, 1623], 85: [1888, 1757, 2016, 1765, 1760], 8
6: [1632, 1501, 1509, 1504], 87: [1657, 1889, 1785, 1791, 1775, 1505, 1633, 1783, 1761, 1767], 88: [1497, 1625, 1881, 1759, 1753], 89: [1636, 1635, 1891, 1882, 1763, 1868, 1876, 2010, 1892, 1884], 90: [17
77, 1781, 1773, 1513, 1641, 1769], 91: [1527, 1503, 1496, 1519, 1495, 1511], 92: [1512, 1640, 1644, 1660, 1652], 93: [1878, 1507, 1518, 1872, 1526, 1510, 1744, 1616, 1488, 1494, 1502], 94: [1520, 2005, 20
13, 2021, 1525, 2029, 2032, 2037, 1904, 1648, 1776], 95: [1471, 1483, 1611, 1487, 1739, 1772, 1764, 1740, 1463, 1748, 1479, 1756], 96: [1491, 1747, 1637, 1629, 1621, 1619], 97: [1460, 1468, 1476, 1738, 14
82, 1489, 1610, 1524, 1516, 1484, 1492, 1500, 1508], 98: [1750, 1758, 1499, 1627, 1883, 1628, 1755], 99: [2001, 1873, 1751, 1617, 1745], 100: [944, 441, 444, 951, 436, 430, 822, 438, 816, 688, 560, 432],
101: [952, 815, 823, 847, 839, 824, 831], 102: [960, 832, 704, 452, 448, 576], 103: [1223, 1231, 1225, 1097, 969, 585, 205, 201, 841, 713, 447, 455, 463, 329, 457], 104: [846, 835, 830, 838], 105: [1090,
962, 964, 948, 940, 956, 936, 309, 301, 428, 296, 808, 680, 552, 424], 106: [1073, 945, 817, 812, 836, 828, 820], 107: [1069, 1067, 939, 811, 683, 555, 557, 565, 573, 581], 108: [317, 325, 321, 961, 833,
577, 705, 449], 109: [1087, 1081, 965, 957, 953, 813, 821, 825, 829], 110: [304, 376, 380, 299, 372, 364, 356, 504, 348, 459, 331, 332, 324, 340, 300, 316, 308], 111: [556, 563, 844, 636, 620, 612, 604, 3
06, 434, 562, 564, 628, 572, 580, 596, 840, 712, 584, 588], 112: [454, 589, 590, 462, 458, 460, 591, 586], 113: [322, 722, 594, 466, 338, 327, 343, 330, 335], 114: [845, 971, 719, 725, 843, 973, 717, 587,
 715], 115: [660, 668, 676, 684, 652, 298, 682, 554, 431, 426], 116: [654, 662, 670, 678, 297, 686, 558, 809, 425, 553, 681], 117: [527, 535, 543, 551, 569, 559, 567, 583, 575], 118: [184, 393, 189, 271,
287, 295, 279, 303, 311, 568, 265, 440, 312, 319], 119: [707, 655, 663, 671, 679, 696, 711, 687, 697, 695, 703], 120: [566, 561, 305, 310, 429, 433, 445, 437], 121: [955, 315, 694, 443, 827, 571, 702, 699
], 122: [323, 451, 574, 579, 582], 123: [339, 342, 326, 314, 334, 318], 124: [442, 709, 701, 570, 954, 826, 698]})

('Nb occurences:', [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1])
[-5275.000000000502, -5028.0000000005475, -5027.999999999976, -5027.000000000422, -5027.000000000251, -5027.00000000008, -5027.00000000008, -5026.000000000239, -5025.000000000286, -4780.0000000002965, -47
79.999999999725, -4779.000000000227, -4779.000000000056, -4778.999999999943, -4778.999999999885, -4778.999999999657, -4778.00000000073, -4778.000000000558, -4778.000000000387, -4778.0000000002165, -4778.0
000000000455, -4777.999999999988, -4777.00000000049, -4777.00000000049, -4777.0000000003765, -4777.0000000003765, -4777.000000000091, -4777.000000000091, -4776.999999999976, -4776.999999999805, -4776.0000
00000365, -4776.000000000307, -4776.000000000194, -4776.000000000194, -4775.999999999908, -4775.999999999908, -4775.999999999851, -4756.407407407119, -4728.518518519035, -4719.222222222246, -4717.22222222
2281, -4679.875000000399, -4678.461538461128, -4634.55555555553, -4623.400000000422, -4531.000000000205, -4530.000000000593, -4530.000000000536, -4530.000000000478, -4530.000000000422, -4530.000000000307,
 -4530.000000000307, -4530.000000000194, -4530.000000000136, -4530.000000000079, -4530.000000000022, -4529.999999999908, -4529.000000000696, -4529.000000000696, -4529.000000000525, -4529.0000000004675, -4
529.0000000004675, -4529.00000000041, -4529.00000000041, -4529.000000000239, -4529.000000000182, -4529.000000000182, -4529.000000000182, -4529.000000000182, -4529.000000000182, -4529.000000000068, -4529.0
00000000068, -4529.000000000011, -4528.999999999954, -4528.999999999896, -4528.999999999896, -4528.99999999984, -4528.999999999783, -4528.999999999783, -4528.999999999725, -4528.000000000513, -4528.000000
000399, -4528.000000000399, -4528.000000000342, -4528.000000000285, -4528.000000000171, -4528.000000000171, -4528.000000000171, -4528.000000000114, -4528.000000000056, -4528.0, -4528.0, -4527.999999999943
, -4527.999999999943, -4527.999999999943, -4527.000000000331, -4527.000000000331, -4527.000000000331, -4527.000000000331, -4527.000000000274]
Taux de réussite : 0.0
Energie moyenne : -4658.81662464
```


~1030 qubits
pour 125 varaibles

Solution optimale 0 …

~ 15% d'écart

PREMIERE CONCLUSION

- **Solutions sans topologie**

		opt.	best	worst	mean	median	stdev
G_1	n	-68	-68	-68	-68	-68	0
	$n^{1.5}$	-68	-68	-68	-68	-68	0
	n^2	-68	-68	-68	-68	-68	0
G_2	n	-495	-495	-495	-495	-495	0
	$n^{1.5}$	-495	-495	-495	-495	-495	0
	n^2	-495	-495	-495	-495	-495	0
G_3	n	-2064	-2064	-1810	-2004.7	-2064	79.9
	$n^{1.5}$	-2064	-2064	-2064	-2064	-2064	0
	n^2	-2064	-2064	-2064	-2064	-2064	0
G_4	n	-6275	-6275	-5528	-5785.3	-5777	178.9
	$n^{1.5}$	-6275	-6275	-6026	-6241.8	-6275	86.1
	n^2	-6275	-6275	-6275	-6275	-6275	0

Paramètres Annealing : - Standard cooling time of the form T_{k+1}=0,95T_k
- Stop annealing at T<10^-3.
- Number of iterations of the Metropolis algorithm
  → For each k to T = cst and n: number of variables in QUBO.

For **n iterations** per temperature plate: Lower quality results → standard solutions.

For **n^2 iterations** per plateau: Results of much better quality, but with a much longer computing time.

- **Better** performance simulated annealing
- **Worst** solutions over 30 cycles: Almost always better than D-Wave over 10,000 cycles.
- Asymptotic regime of the exponential number of iterations of Sasaki & Hajek's theorem not reached with annealing (sol. Opt. reached for G4)

- Instances small enough to remain easy to anneal in the conventional way

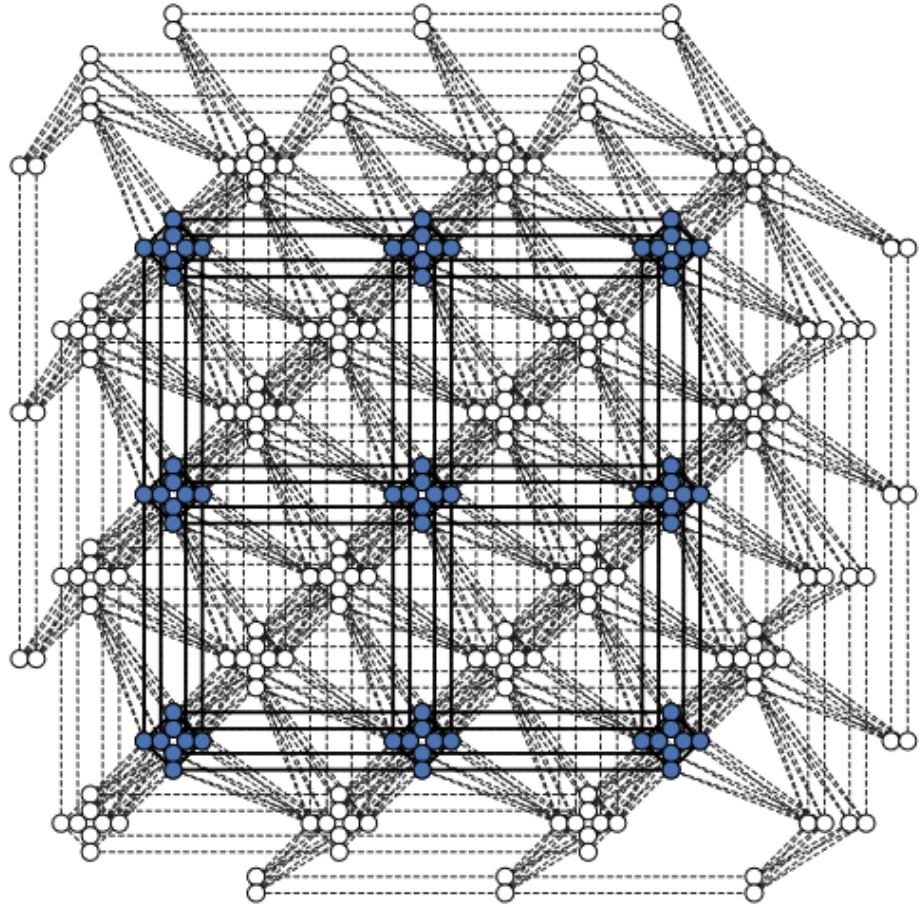
Unduplicated resolved instances for comparison with D-Wave

- **Solutions avec topologie (Chim. & Peg.)**

		opt.	best	worst	mean	median
G_4 (Chim.)	n	-6275	-2213	3662	1453.9	1401.0
	$n^{1.5}$	-6275	-4526	-2654	-3585.6	-3699.8
	n^2	-6275	-5028	-4027	-4473.1	-4527.0
D-Wave		-6275	-5025	-3551	-4447.7	-4525
G_4 (Peg.)	n	-6275	-3930	-785	-2609.3	-2708.5
	$n^{1.5}$	-6275	-5028	-3580	-4305.5	-4281.0
	n^2	-6275	-5278	-4530	-5035.9	-5028.0

- Topologie d'interconnexions entre qubits de 15

	#var.	#qubits (Chim.)	#qubits (Peg.)
G_1	8	16	8
G_2	27	100	46
G_3	64	431	164
G_4	125	958	513



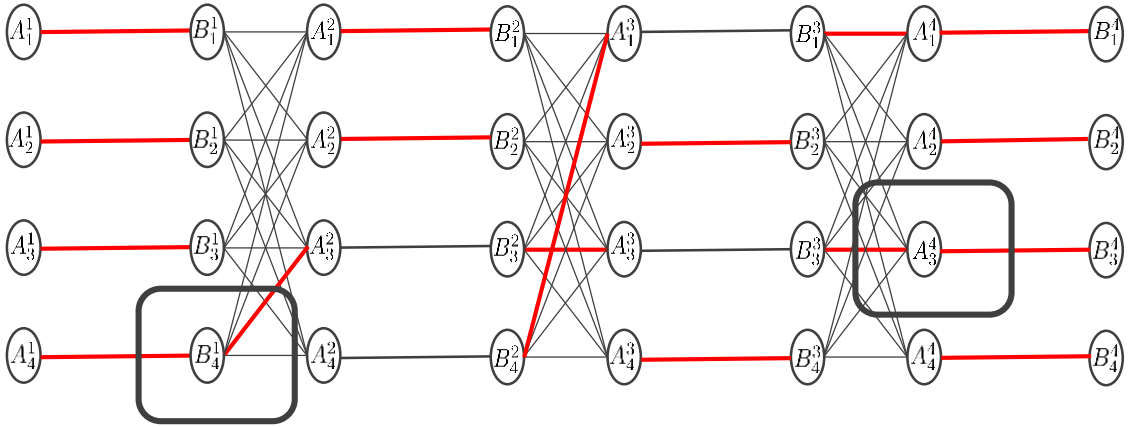
- Résultats sur D-Wave **compétitif** avec ceux du SA.
- Les instances dupliquées **sont beaucoup plus difficiles** à résoudre que les instances non dupliquées sur SA.
- La topologie plus dense de Pegasus conduit à des QUBO dupliqués plus petits et donne de meilleurs résultats (mais pas optimaux).

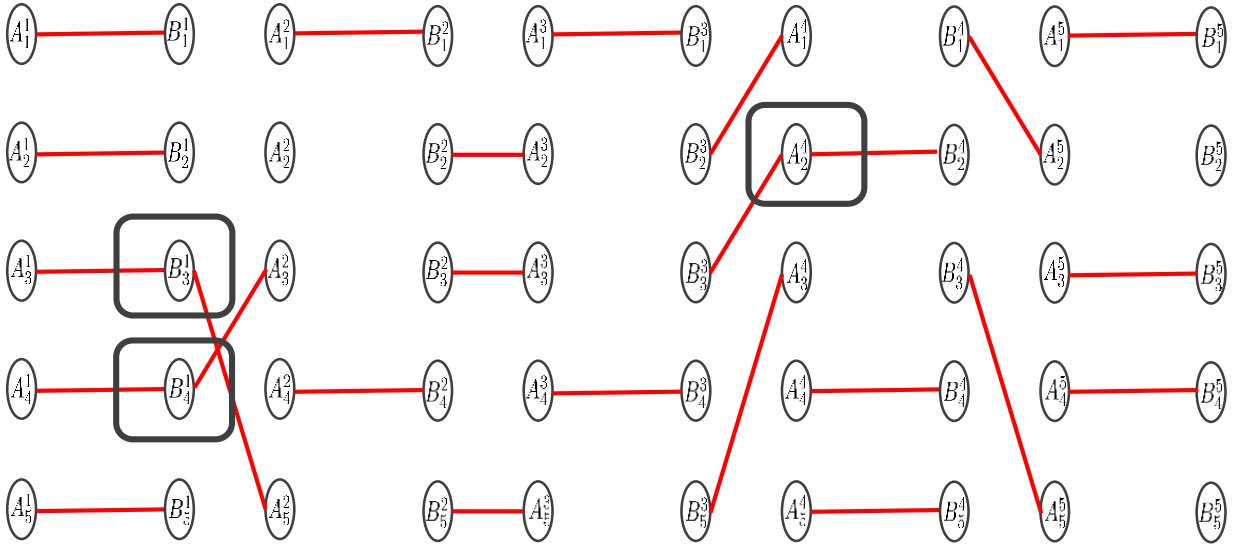
DEUXIEME CONCLUSION

Need for duplication → Limits the size of the instances to be mapped.
Max G4 size: 125 variables for 958/1098 available qubits (~ 87%) sur DW2X
125 variables for 1031/2041 avaibles qubits (~ 50%) sur D2000Q

	#var.	#qubits	average dup.	max. dup.
G_1	8	16	2.0	6
G_2	27	100	3.7	6
G_3	64	431	6.7	18
G_4	125	951	7.6	18

- Observation of results :
  - Best solution found for G3 / G4 not a matching
  - D-Wave does not get the optimal solution





DEUXIEME CONCLUSION AVEC ADVANTAGE

- **Advantage4.1 (Peg.)**
  - **5627 qubits**

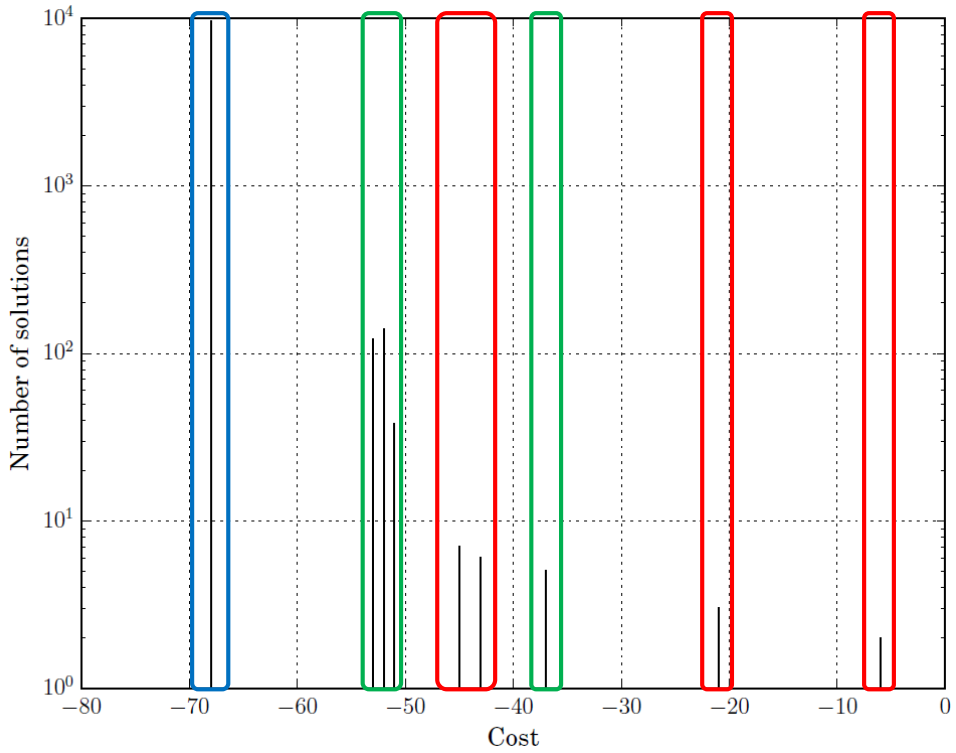
	#var.	#qubits	#Opt.	Opt.	Best	Worst	Mean	Median	Stdev
G1	8	8	1000	-68	-68	-68	-68	-68	0
G2	27	49	80	-495	-495	-225	-389.2	-387	34.5
G3	64	174	0	-2064	-1937 (x3)	-916	-1565.5	-1553	137.8
G4	125	471	0	-6275	-5278	-2782	-4268.9	-4278	370.4
G5	216	1033	0	-15588	-12134	-6527	-9629.3	-9549	912.3
G6	343	2372	0	-33663	-23383	-8301	-16299.9	-16528	2286.4
G7	512	4186	0	-65600	-40019	-11361	-27278.3	-27734	4645.5

G3 → ~6% d'écart avec Opt.
G4 → ~16% d'écart avec Opt.
G5 → ~22%
G6 → ~31%
G7 → ~39%

Résultats sur Advantage …. Identique à D2000Q ?
On passe de 6 connexions à 15 !

Question1 : besoin de post-processing ? Si oui → Constrainte(s): rapide (garde l'avantage quantique) et donne une solution **valide** !
Question 2 : Possible d'améliorer les résultats ?

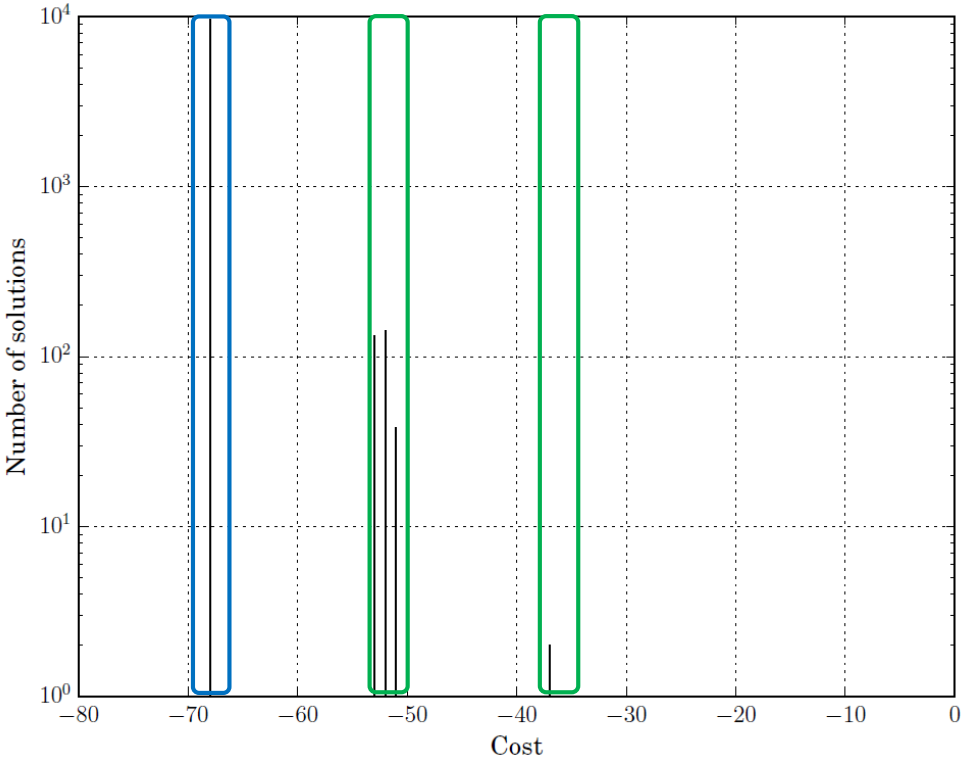
- Résultat sur 10000 runs de G_1 :
  - Solution optimale trouvée (9673 times)
  - Solution non couplage
  - Solution avec mauvaise duplication



Mauvaise duplication sur le qubit « 6 »
→ Solution non valide

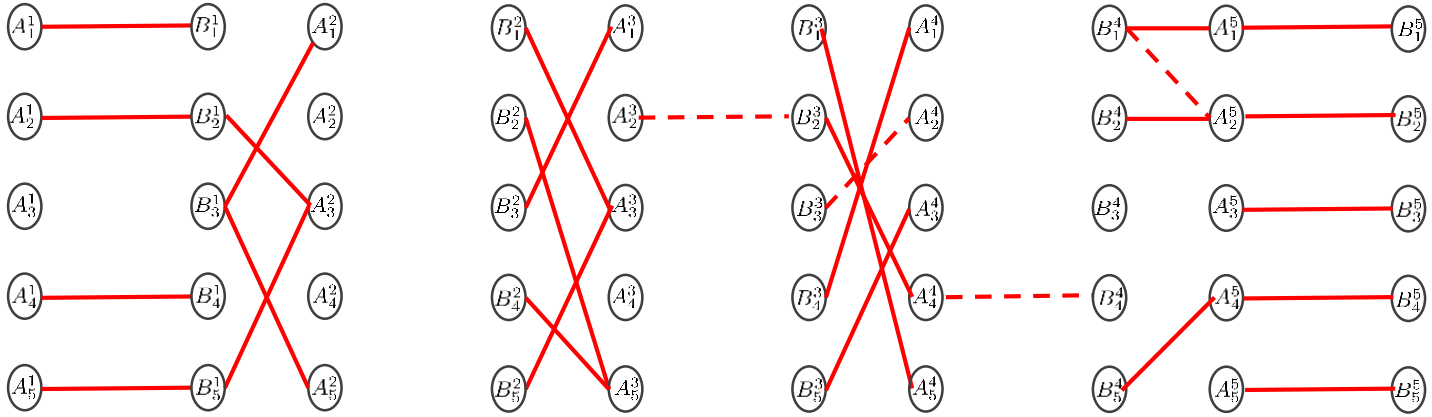
qubits	variable	best	worst
1040	0	1	1
1041	4	0	0
1042	5	0	0
1044	5	0	0
1045	4	0	0
1047	5	0	0
1048	1	1	1
1050	6	0	0
1051	3	1	1
1052	7	0	0
1053	2	1	1
1054	6	0	0
1055	3	1	1
1137	6	0	0
1143	6	0	1
1146	6	0	0
1151	6	0	1

- Post-traitement → vote majoritaire :
  - Solution optimale trouvée (9686 times)
  - Solution non couplage



D Vert, R Sirdey, S Louise - arXiv preprint arXiv:1910.05129, 2019

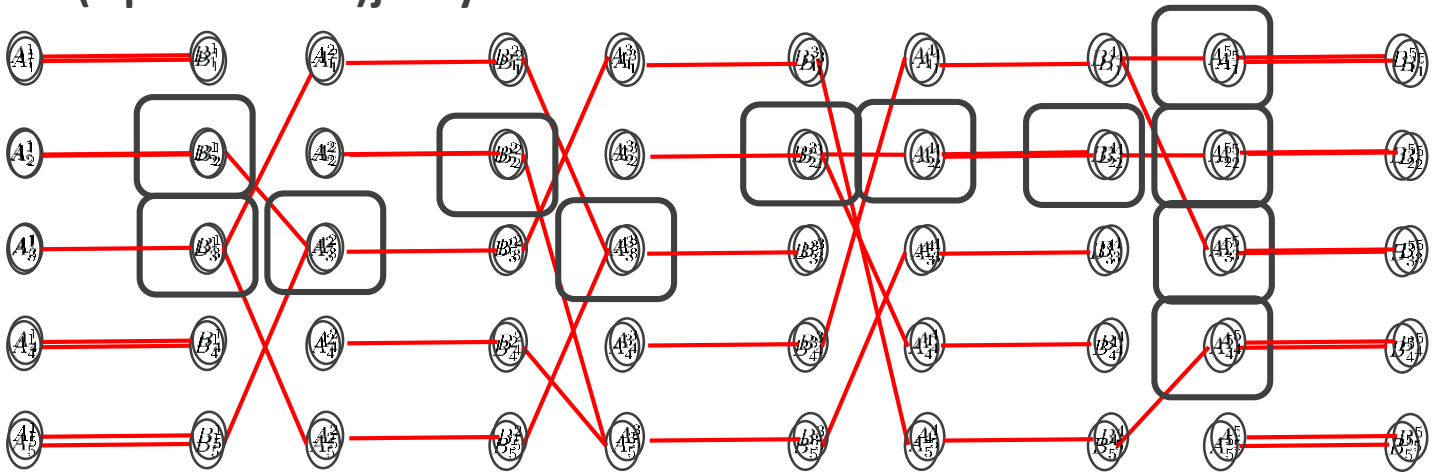
Résult → Inconsistent duplications



Need for post-processing

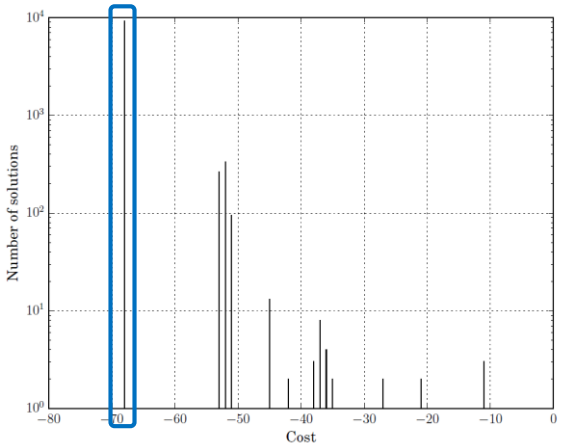
Constraint(s): quick and have a valid solution!

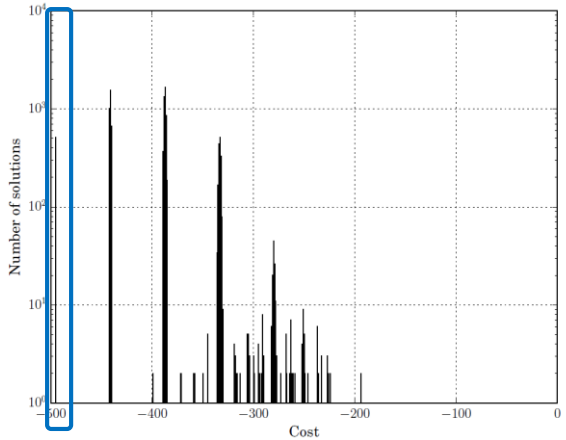
Result Solution: majority vote



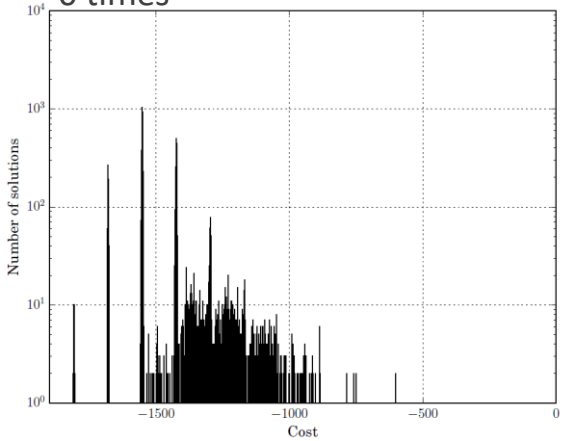
QUESTION 1

	opt.	best	worst	mean	median	stdev	best	worst	mean	median	stdev
G_1	-68	-68	-9	-66.8	-68	4.6	-68	-37	-66.8	-68	4.2
G_2	-495	-495	-29	-398.2	-388	48.1	-495	-277	-400.4	-388	44.6
G_3	-2064	-1810	-505	-1454.8	-1548	157.7	-1810	-911	-1496.5	-1550	111.8
G_4	-6275	-5527	-2507	-4609.9	-4675	346.5	-5527	-3030	-4579.2	-4527	314.1

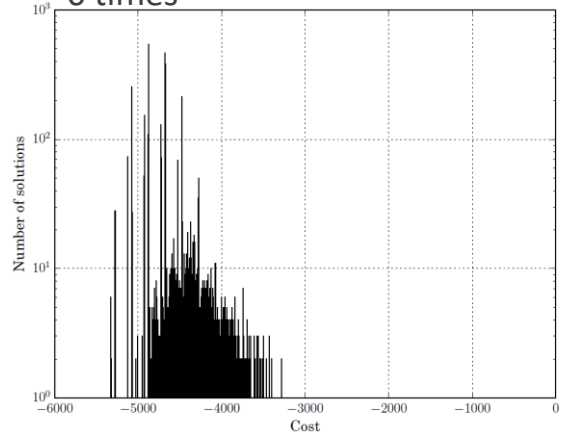


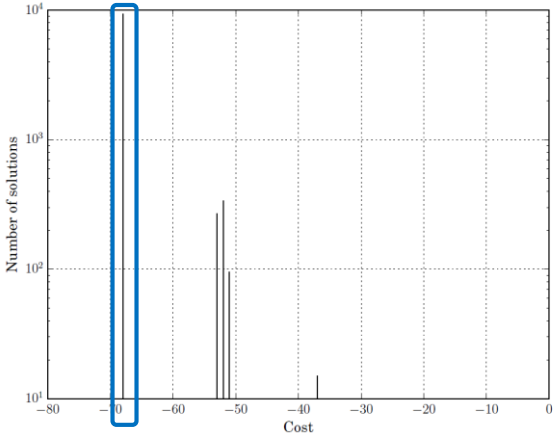


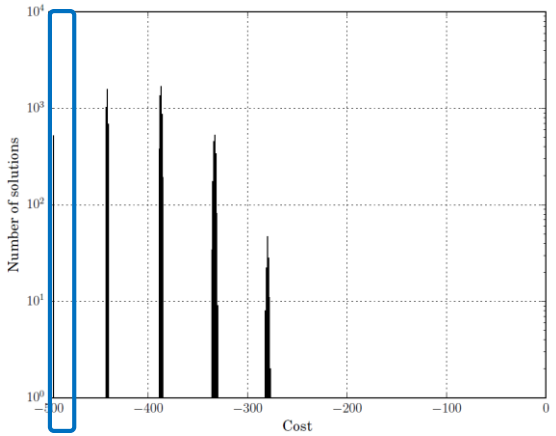
0 times



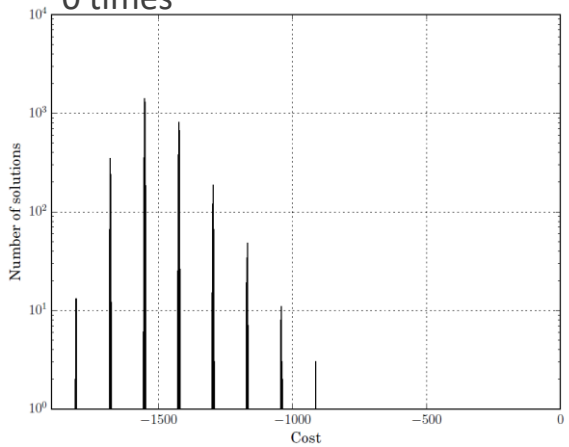
0 times



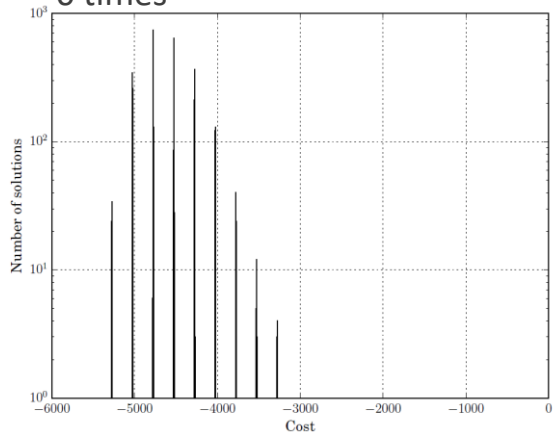




0 times

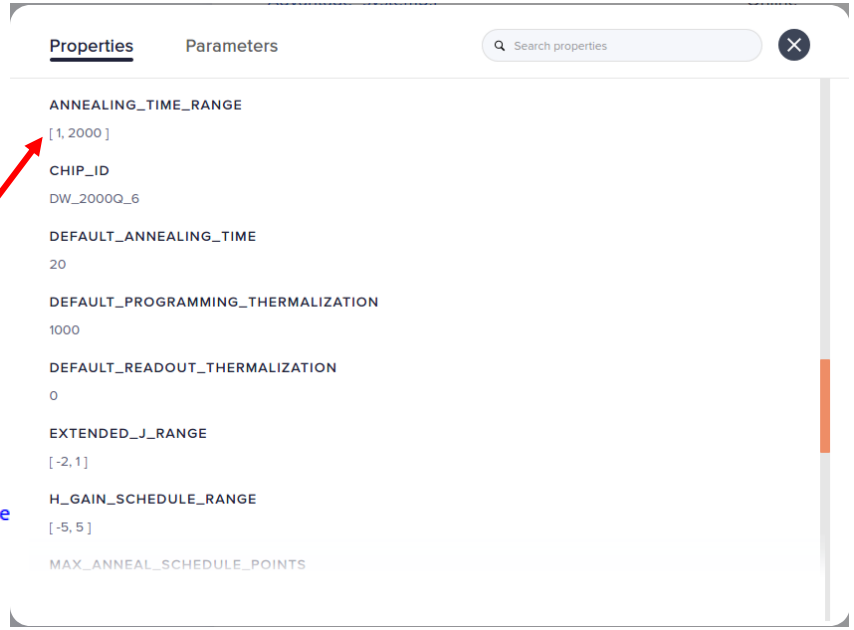


0 times



- **Améliorer les résultats ?**
  - **Théorème adiabatique :** Si l'évolution temporelle est assez lente (τ est assez grand) → La solution optimale ε(σ) est obtenue avec une forte probabilité.

```
##########################################################################################################
################################### partie 2 : dwave_leap ################################################
##########################################################################################################

import dwave_sapi2
from dwave_sapi2.remote import RemoteConnection
from dwave_sapi2.core import solve_ising, solve_qubo

url = 'https://cloud.dwavesys.com/sapi/'
token = 'DEV-32b473f753f4641ca3e87d4318f325295665204d'


# create a remote connection
conn = RemoteConnection(url, token)

# get the solver
solver = conn.get_solver('DW_2000Q_6')
#solver = conn.get_solver('Advantage_system4.1')


params = { "num_reads": 1000, "max_answers": 1000, "answer_mode": "histogram", "auto_scale": True, "annealing_time": 20, "num_spin_reve

# solve QUBO problem with parameters
answer = solve_qubo(solver, QUBO, **params)
print"\n"
print("Energie:", answer["energies"])
print("Nb occurences:", answer["num_occurrences"])

i = 0
count = 0
while (answer["energies"][i]*(norma) <= -68) :

        count = count + answer["num_occurrences"][i]
        i = i + 1


print([energie*(norma) for energie in answer["energies"]])


print("Taux de réussite : " + str(float(count)/sum(answer["num_occurrences"])))

print("Energie moyenne : " + str(sum([answer["energies"][i] * answer["num_occurrences"][i] for i in range (len(answer["energies"]))]) / sum(answer["num_occurrences"])*(norma)))
```



Améliorer les résultats ?

	AT	#var.	#qubits	#Opt.	Opt.	Best	Worst	Mean	Median	Stdev
G3	20	64	360	0	-2064	-1808	-1105	-1482.15	-1486.7	86.8
	100	64	360	0	-2064	-1937	-1680	-1714.3	-1785.9	77.7
	1000	64	360	0	-2064	-1810	-1680	-1700.2	-1713.9	82.5
	2000	64	360	0	-2064	-1810	-1426	-1565.3	-1573.6	89.9
G4	20	125	1031	0	-6275	-5029	-4776	-4808.5	-4996.4	197.3
	100	125	1031	0	-6275	-5275	-4778	-4754.8	-4859.8	178.6
	1000	125	1031	0	-6275	-5276	-4280	-4554.8	-4598.9	221.4
	2000	125	1031	0	-6275	-5277	-4280	-4562.7	-4603.3	219.1

Résultats « de base »

Résultats sensiblement similaire → AT permet « d'écraser » les pires solutions vers les meilleures

Résultats « de base »

Améliore la qualité de la solution optimale

Sur D2000Q → Non …

Et sur Advantadge4.1 ?

```
N = 3, RCS:0.496            (best,    worst,    mean,     median,    stdev)
AT:40.89, NSRT:20, Energy: [(-1937., -1170., -1628.17, -1616.5, 164.46063693)]
AT:404.44, NSRT:0, Energy: [(-1936., -1043., -1659.94, -1680., 153.89514742)]
AT:848.77, NSRT:10, Energy: [(-1936., -1170., -1635.82, -1680., 157.63295214)]
AT:1232.52, NSRT:0, Energy: [(-2064., -1170., -1690.67, -1681., 165.02612248)]
AT:1616.26, NSRT:0, Energy: [(-2064., -1171., -1735.756, -1683., 140.86804628, b'9/1000')]
```

Solution optimale !

```
N = 4, RCS:0.667            (best,    worst,    mean,     median,    stdev)
AT:182.27, NSRT:5, Energy: [(-5275., -3775., -4541.33, -4527., 341.89059815)]
AT:626.61, NSRT:0, Energy: [(-5527., -3526., -4766.153, -4776., 308.55241628, b'1/1000')]
AT:868.97, NSRT:20, Energy: [(-5028., -3280., -4516.4, -4527., 381.59651466)]
AT:1434.48, NSRT:0, Energy: [(-5028., -3773., -4618.57, -4528.5, 288.32891825)]
AT:1697.05, NSRT:18, Energy: [(-5027., -3777., -4551.25, -4527., 283.53131661)]
```

Mieux que DW2000Q !
(11.92% best vs opt.)

```
N = 5, RCS:0.569            (best,    worst,    mean,     median,    stdev)
AT:384.24, NSRT:25, Energy: [(-12134., -8245., -10835.38, -10834., 772.65500425)]
AT:626.61, NSRT:50, Energy: [(-12564., -8246., -10696.91, -10835., 863.21299915)]
AT:990.15, NSRT:0, Energy: [(-12562., -7385., -10666.95, -10834.5, 885.29777335)]
AT:1575.86, NSRT:5, Energy: [(-12562., -8678., -10792.02, -10834., 746.78836333)]
AT:1919.21, NSRT:0, Energy: [(-12994., -8676., -11122.638, -11266., 733.42026898, b'1/1000')]

N = 6, RCS:0.888            (best,    worst,    mean,     median,    stdev)
AT:364.05, NSRT:0, Energy: [(-26805., -16521., -22291.68, -22683., 1796.02446464)]
AT:687.2, NSRT:18, Energy: [(-25433., -17205., -21824.25, -22002., 1571.67414164)]
AT:929.56, NSRT:20, Energy: [(-25430., -15835., -21947.48, -22002., 1851.31472462)]
AT:1414.29, NSRT:5, Energy: [(-25429., -17885., -22468.23, -22685.5, 1548.27271406)]
AT:1656.65, NSRT:5, Energy: [(-27487., -18574., -23302.174, -23374., 1463.85336073, b'1/1000')]

N = 7, RCS:1.157            (best,    worst,    mean,     median,    stdev)
AT:61.09, NSRT:15, Energy: [(-43075., -29769., -36523.03, -36930.5, 3067.31622255)]
AT:444.83, NSRT:10, Energy: [(-44098., -20554., -36778.89, -36931., 3393.44609474)]
AT:1091.14, NSRT:0, Energy: [(-46148., -28746., -39074.07, -38982.5, 3295.13821032)]
AT:1535.47, NSRT:5, Energy: [(-47167., -32837., -40003.79, -40004.5, 2973.82983472)]
AT:1899.02, NSRT:5, Energy: [(-51266., -31819., -41678.632, -42048., 2807.29091556, b'1/1000')]
```

G5 → ~16.64%
G6 → ~18.35%
G7 → ~21.85%

CONCLUSION

- **Difficult problem for simulated and quantum annealing**
  First benchmark on a D-Wave
  With topology → Similar results between D-Wave and SA.

  The constraints imposed by the graph show that SA and QA cannot solve the problem!

- **Low quality results**
  - The need to duplicate qubits strongly limits the size of accessible problems.

  - Necessity of post-treatments
    Duplication errors ☹
    not representing a valid solution

- **Sparse topology**
  - Need more connections between qubits → mapping denser and larger problems!

MERCI

[ACM-CF] « *On the limitations of the Chimera graph topology in using analog quantum computers* » , in « *ACM International Conference on Computing Frontiers* » 2019, Alghero, Sardinia, Italy.

[ICCS] « Revisiting old combinatorial beasts in the quantum age : quantum annealing versus maximal matching », in « INTERNATIONAL CONFERENCE ON COMPUTATIONAL SCIENCE » (ICCS 2020 - Quantum Computing Workshop).

[ISVLSI] « Operational Quantum Annealers are Cursed by Their Qubits InterconnectionTopologies », in « 1st International Workshop on Quantum Computing : Circuits Systems Automation and Applications » (QC-CSAA) (ISVLSI2020 - Quantum Workshop).

[SN] « Benchmarking quantum annealing against ”hard” instances of the bipartite matching problem », in Special issue for Springer journals (SN Computer Science).