HAL open science

Informatique quantique - travaux pratiques

Benoît Prieur

▶ To cite this version:

Benoît Prieur. Informatique quantique - travaux pratiques. Master. Travaux pratiques d'informatique quantique, ECE Paris, France. 2019, pp.49. hal-02374075v2

HAL Id: hal-02374075
https://hal.science/hal-02374075v2

Submitted on 20 Mar 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution - ShareAlike 4.0 International License

Informatique quantique - travaux pratiques

Majeure objets connectés, réseaux et services (M2)

ECE Paris, 30 octobre, 8 et 20 novembre 2019

Benoît Prieur - SOARTHEC - CC-By-SA

Contexte

- Les éditions E.N.I, spécialiste de l'informatique et du développement logiciel.
- Offre cohérente et exhaustive entre autres en ce qui concerne Microsoft.
- Après WPF et C#, écriture d'un livre au sujet de la programmation Q#.
- Développement Microsoft .Net, Python (Flask, Django), à propos de Wikidata, OpenStreetMap etc.

Objectifs des sessions

- Manipuler avec deux environnements distincts et améliorer ses connaissances de certains langages :
  - **QISKit** d'IBM qui permet par ailleurs d'améliorer sa connaissance de la programmation Python notamment sur le version scientifique (*numpy*, *matplotlib*, *Jupyter*).
  - **Quantum computing** de Microsoft qui permet de découvrir le langage **Q#** et de programmer en **C#**.

Rappels des principales notions (1)

- La notion de bit quantique (qubit).
- Rappel de l'écriture *bracket* :

$$\alpha \cdot |0\rangle + \beta \cdot |1\rangle \qquad |\alpha|^2 + |\beta|^2 = 1$$

$$\alpha \cdot |00\rangle + \beta \cdot |01\rangle + \gamma \cdot |10\rangle + \delta \cdot |11\rangle \qquad |\alpha|^2 + |\beta|^2 + |\gamma|^2 + |\delta|^2 = 1$$

- La mesure quantique : de la physique quantique à la physique classique.

Rappels des principales notions (2)

- La superposition quantique.
- L'intrication quantique.
- Les **états de Bell** correspondent à l'intrication maximale. De manière classique, on implémente ces **états de Bell** comme première illustration quelque soit la plate-forme étudiée.

Rappels des principales notions (3)

- Circuit quantique composée de portes quantiques et se “termine” par une ou plusieurs mesures dont le résultat est transmis au programme classique.
- Dans le cas des **états de Bell** : le circuit quantique est composé d’une porte de Hadamard (H) et d’une porte CNOT.

Qiskit, framework open-source d'informatique quantique

- Une initiative d'IBM Research.
- Permet d'exécuter des programmes quantiques sur des prototypes/simulateurs en ligne.
- Disponible sur les trois OS ; forte utilisation de Python.

Notion d’environnement virtuel en Python

- Création d’un environnement comprenant une certaine version de Python et les versions des librairies.
- Exemples de systèmes d’environnements virtuels : *venv*, *virtualenv* etc.
- Ici environnement virtuel *conda*.

Qiskit, installation

- Le lien d'installation : https://qiskit.org/documentation/install.html
- Installer préalablement *Anaconda* pour bénéficier d'un environnement virtuel de type *conda*.
- Des fichiers de distribution *C++* à installer avec Windows.
- Avec des IDE type *Pycharm*, bien sélectionner l'environnement virtuel *conda*

Création d'un compte IBM Quantum Computing

- Se rendre à cette url : https://quantum-computing.ibm.com/login
- Permet de générer un jeton (*token*) indispensable à l'exécution de la simulation en local.
- Possibilité de créer un notebook Jupyter (Data science) https://quantum-computing.ibm.com/jupyter
    - Exemple d'un notebook Jupyter (dans un autre contexte)

Récapitulatif succinct

QISKit c'est :

- Environnement de programmation quantique sur simulateur ou sur plate-forme réelle (plusieurs essais disponibles).
- Langage **Python**, mais possibilité d'utiliser **Swift** et **JavaScript**.
- Forte connexion à Jupyter, les exemples sont d'ailleurs souvent présentés ainsi.

Qiskit (local), affichage de la version

```
import qiskit

from qiskit import IBMQ

IBMQ.save_account('TOKEN')

print(qiskit.__version__)
```

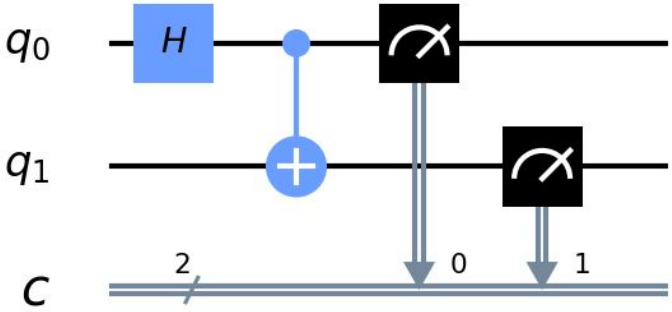
```
>> 0.10.0
```

```
qiskit.__qiskit_version__
```

```
{'qiskit-terra': '0.10.0',
'qiskit-aer': '0.3.2',
'qiskit-ignis': '0.2.0',
'qiskit-ibmq-provider':
'0.3.3', 'qiskit-aqua': '0.6.1',
'qiskit': '0.13.0'}
```

QISKit (local) : états de Bell (1)

- Circuit à deux bits quantiques impliquant une porte de Hadamard (H) et une porte CNOT ; on s'intéresse au résultat de la mesure.



QISKit (local) : états de Bell (2)

L'approche **QISKit** :

- Construction du circuit quantique.
- Exécution du programme quantique.
- Analyse des données issues de la mesure.

Le code de l'exemple est en ligne ici.

QISKit (local) : états de Bell (3)

On commence par l'**import des packages** ; on importe numpy et on déclare le simulateur qui va chercher une solution (Aer).

```
import numpy as np
from qiskit import(
  QuantumCircuit,
  execute,
  Aer)
from qiskit.visualization import plot_histogram
```

QISKit (local) : états de Bell (4)

On poursuit par l'**initialisation des variables** ; on veut pour rappel un circuit à deux bits quantiques avec deux lignes de mesure quantique (relatives à la physique classique donc).

```
circuit = QuantumCircuit(2, 2)
```

QISKit (local) : exemple de circuit quantique (5)

On passe à la création du circuit quantique :

```
circuit.h(0)

circuit.cx(0, 1)

circuit.measure([0,1], [0,1])
```

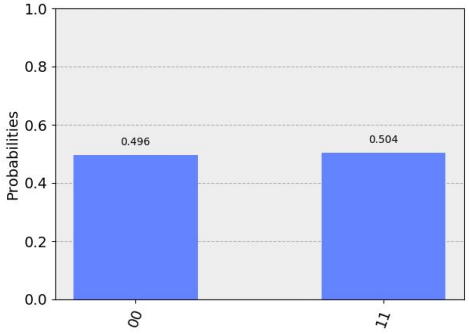
QISKit (local) : exemple de circuit quantique (6)

On exécute notre circuit quantique :

```
simulator = Aer.get_backend('qasm_simulator')
job = execute(circuit, simulator, shots=1000)
result = job.result()
counts = result.get_counts(circuit)
print("\nTotal count for 00 and 11 are:",counts)
```

QISKit (local) : états de Bell (7)

- Exploitation des résultats ; *Numpy* et *Matplotlib* permettent une exploitation visuelle immédiate (gros point fort / Microsoft).



Organisation de QisKit (1)

- **Terra** : ce qu'on a utilisé ici. Permet de définir des circuits quantiques et de les simuler.
- **Aqua** : fournit des outils qui peuvent être utilisés sans aucune programmation quantique explicite. Des sujets comme la finance, l'IA etc.
- **Aer** : contient un set de petits simulateurs quantiques.
- **Ignis** : contient des ressources pour caractériser le bruit et évaluer les erreurs relatives à une simulation donnée.

Organisation de QisKit (2)

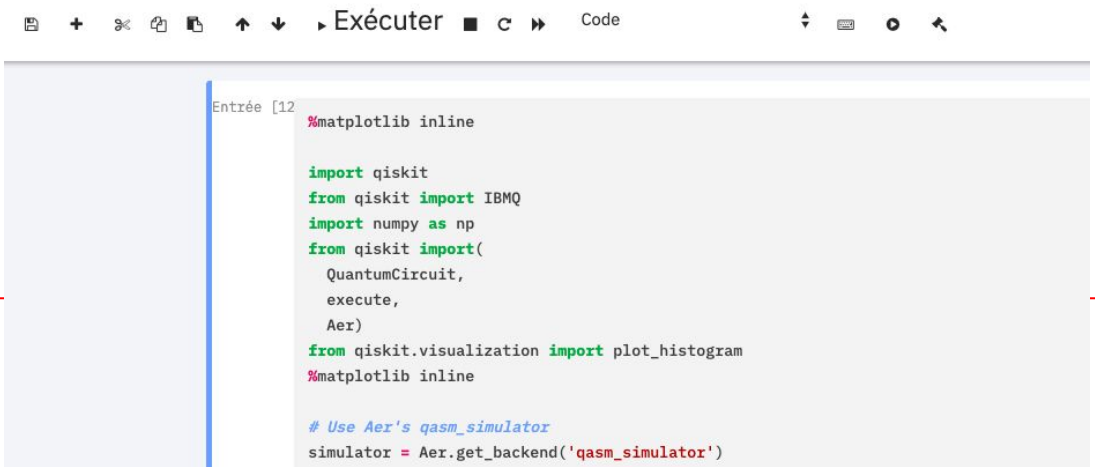
- Notebooks Jupyter organisé selon Terra, Aer, etc. https://nbviewer.jupyter.org/github/Qiskit/qiskit-tutorials/blob/master/qiskit/1_start_here.ipynb
- Le Github de QisKit : https://github.com/Qiskit/

QISKit (online) : Circuit Composer

- Version graphique qui propose la construction du circuit quantique en faisant glisser les différentes portes quantiques sur le circuit.

QISKit (online) : notebook Jupyter

- Utiliser la directive **%matplotlib inline**
- Rappels sur Jupyter



QISKit (online) : notebook Jupyter - code états de Bell

- Le code est ici : https://github.com/benprieur/QISKit/blob/master/Bell%20State%20for%20Jupyter.py
- Le notebook Jupyter sur Github : https://github.com/benprieur/QISKit/blob/master/BELL.ipynb
- Vous créer un Github pour stocker vos notebooks.

QISKit (Jupyter) : téléportation quantique

- Alice veut transmettre $\psi\rangle = \alpha \mid 0\rangle + \beta \mid 1\rangle$ à Bob.
- On veut transmettre alpha et Bêta et non pas le résultat d'une mesure.
- La copie d'un état quantique est impossible.

Qiskit (Jupyter) : téléportation (1)

- On commence par créer deux qubits intriqués selon les états de Bell.
- Alice possède q1 et Bob q2. Elle possède également $|\psi\rangle$ le qubit qu'elle tente de transmettre à Bob.
- Alice applique une porte CNOT sur $q1$.
- Alice applique une porte de Hadamard à $|\psi\rangle$. Elle mesure $q1$ et $|\psi\rangle$.

Qiskit (Jupyter) : téléportation (2)

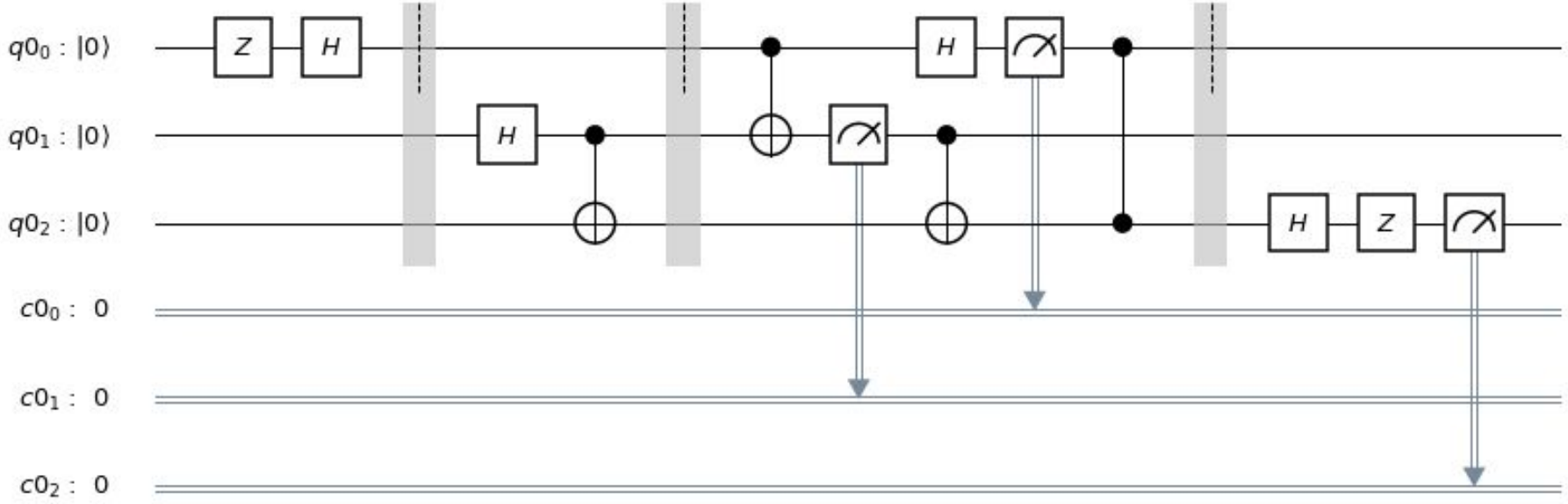
- Alice appelle Bob (de manière classique) :
  - Si elle lui dit 00 $\rightarrow$ ne rien faire
  - Si elle lui dit 01 $\rightarrow$ X
  - Si elle lui dit 10 $\rightarrow$ Z
  - Si elle lui dit 11 $\rightarrow$ ZX

$|\psi\rangle$ s'est téléporté de A à B.

Qiskit (Jupyter) : téléportation (3) ; simulateur

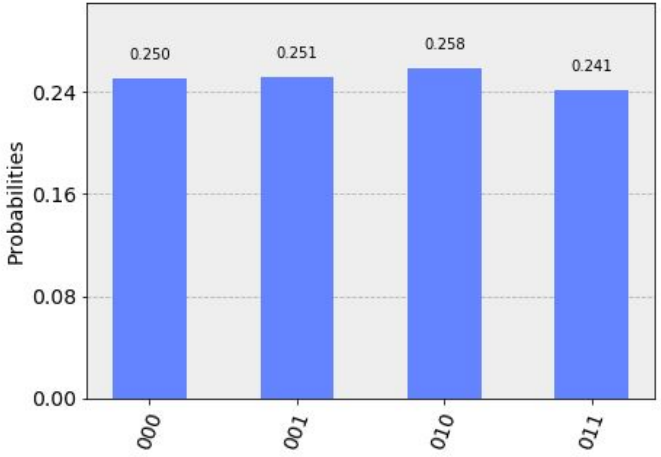
- Le premier code de téléportation est ici : https://github.com/benprieur/QISKit/blob/master/teleportation_1.py
- Le notebook associé est là : https://github.com/benprieur/QISKit/blob/master/Te%CC%81le%CC%81portation%20(1).ipynb (problème d'affichage sur Github)

Qiskit (Jupyter) : téléportation (4) ; schéma



Qiskit (Jupyter) : téléportation (5) ; simulateur

```
{'001': 257, '011': 247, '010': 264, '000': 256}
```

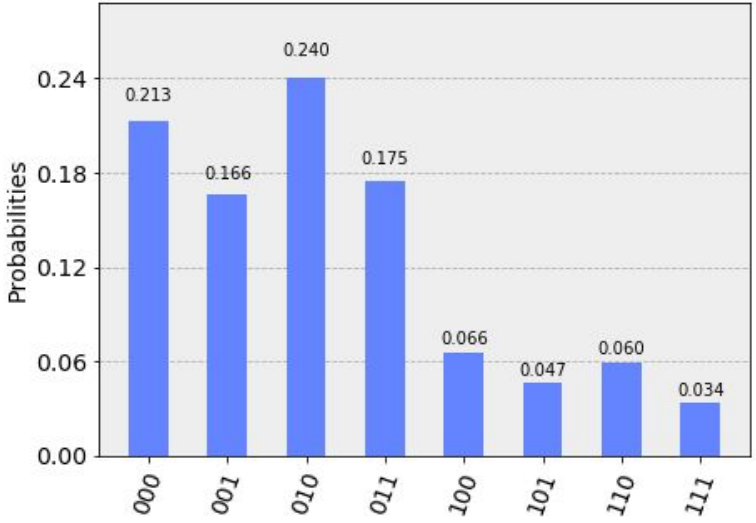


Qiskit (Jupyter) : téléportation (4) ; réel

- Le code de la simulation réelle est ici : https://github.com/benprieur/QISKit/blob/master/teleportation_2.py
- Notebook : https://github.com/benprieur/QISKit/blob/master/Te%CC%81le%CC%81portation%202%20(re%CC%81el).ipynb

Qiskit (Jupyter) : téléportation (5) ; réel

```
{'010': 1966, '110': 489, '100': 541, '101': 384, '000': 1741, '001': 1359, '111': 278, '011': 1434}
```



Microsoft Quantum Development Kit (1)

- Décembre 2017.
- Bibliothèque standard Q# qui inclut tous les aspects du langage, en particulier un compilateur.
- Simulateur de machine quantique exécutable sur la machine locale et pilotable grâce au langage Q#.
- “Pilotage” depuis le code C#.

Microsoft Quantum Development Kit (2) - installation

- Procédure avec VSCode : lien
- Installer *Microsoft Quantum Development Kit for Visual Studio Code* depuis le Market Place (lien) ; installation automatique d'*OmniSharp*

Microsoft Quantum Development Kit (3) - récupérer les exemples et la documentation

- https://github.com/microsoft/Quantum
- Installer git et obtenir les projets de tests :

```
git clone https://github.com/microsoft/Quantum
```

Microsoft Quantum Development Kit (4) - principes

- Host.cs (pilotage C Sharp)
- Programme quantique lui-même extension .qs (Q Sharp)
- Test de la version locale avec l'algorithme de téléportation : https://github.com/microsoft/Quantum/tree/master/samples/getting-started/teleportation

Microsoft Quantum Development Kit (5) - commentaire des états de Bell en Q#

- Code à commenter : lien
- Comparer les résultats obtenus au sujet des états de Bell entre QisKit et Microsoft Quantum

Typologie des problèmes (1)

- Comparaison théorique des algorithmes.
- Quantifier la performance d’un algorithme en fonction des données d’entrée : la **complexité**

Complexité des algorithmes (1)

- Notation de Landau “O” (grand O de quelque chose)
- Exemple simple d’un parcours d’un tableau à une dimension : complexité en O(n)
- Tableau à deux dimensions : O(n*n) ;
- Complexité polynomiale (*vs* exponentielle)

Complexité des algorithmes (2)

- Déterminer une complexité : exemple de la dichotomie

On recherche 73 entre 1 et 100 :

On teste 50, puis on étudie l'intervalle [51, 100]
On teste 75, puis on étudie l'intervalle (50, 74]
On teste 62, puis on étudie l'intervalle [63, 74]
On teste 62, puis on étudie l'intervalle [63, 74]
On teste 68, puis on étudie l'intervalle [69, 74]
On teste 72, puis on étudie l'intervalle [73, 74]
On teste 73 et on obtient le résultat en sept itérations.

Complexité des algorithmes (3)

- 1 *(échantillon final)* <= n *(échantillon initial)* / 2 exp k, *avec k nombre d'itérations*
- On applique la fonction log à gauche et à droite
- On obtient k <= log (n) / log (2)

On en déduit que la complexité est en O (log(n))

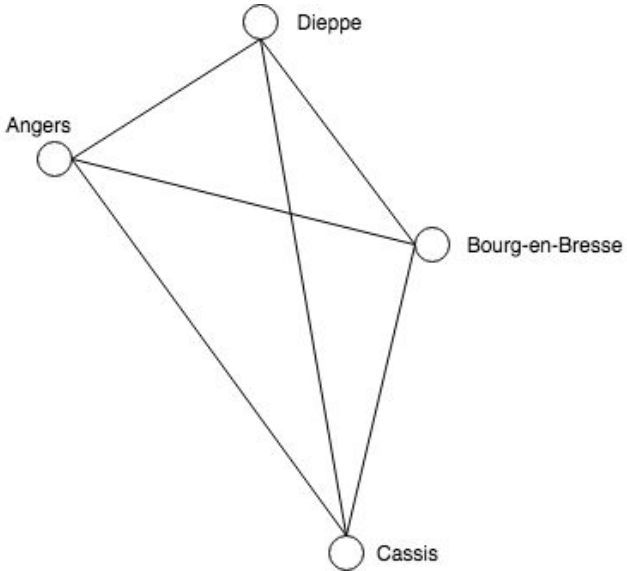
Complexité des algorithmes (4)

- Exemple du problème du voyageur de commerce
- (factorielle de (n - 1)) / 2 *chemins différents*

  Angers – Dieppe – Bourg-en-Bresse – Cassis

  Angers – Bourg-en-Bresse – Cassis - Dieppe

  Angers – Cassis – Dieppe – Bourg-en-Bresse



Complexité des algorithmes (5)

Défi à l'informatique classique et intérêt futur de l'informatique quantique

- 20 villes => 6.082255 . 10^16
  - 6.082255 . 10^16 ⇔ 2 exp k => k = **55 qubits**
  - 55 qubits à mettre en perspective avec l'expérience de Google

Complexité des algorithmes (6)

La **classe P** représente la classe des problèmes qui peuvent être résolus (ou « décidés ») par une **machine de Turing** (c'est-à-dire une sorte d'ordinateur déterministe) en un temps polynomial par rapport à la taille des données en entrée. De tels problèmes voient leurs solutions obtenues en un **temps polynomial**.

La **classe NP** représente de son côté un problème décidable en un temps polynomial par une machine de Turing non-déterministe. Plus prosaïquement cela signifie que l'on peut vérifier en un temps polynomial, c'est-à-dire « rapidement » qu'une solution proposée est bien solution du problème considéré.

Complexité des algorithmes (7)

Si un problème A est de classe P il peut être résolu en un temps polynomial. Cela signifie intuitivement que l'on n'aura aucun mal si on fournit une solution éventuelle à la vérifier en un temps polynomial. Il est donc également de classe NP. Plus généralement on a donc **P ⊂ NP** qui se lit comme « la classe P est incluse dans la classe NP ».

Un problème moderne qui se pose est la question de l'inclusion de NP dans P qui si cela est vrai (ce n'est très probablement pas le cas) => P = NP ?

Le fait de démontrer que P est égal ou non à NP constitue un des **problèmes du millénaire** qui est nommé « **conjecture du siècle** ». C'est un des problèmes majeurs et non résolus de l'informatique.

Complexité des algorithmes (8) - NP-Complet

Deux conditions :

- Il appartient à la classe NP (on vérifie en un temps polynomial une de ses solutions éventuelles) ;
- Tous les problèmes NP sont réductibles en ce problème par une réduction polynomiale. C'est-à-dire que si on résout ce problème on résout tous les autres.

Deux exemples de problèmes NP-Complets : le **problème du voyageur de commerce** et le **problème du sac à dos**.

Suprématie quantique (1)

- La **suprématie quantique** correspond au nombre de qubits au-delà duquel aucun superordinateur classique n'est capable de simuler son équivalent quantique (en un temps raisonnable), pour un problème donné.
- Annonce de **Google** en 2019.
- Vidéo officielle => https://www.youtube.com/watch?v=-ZNEzzDcllU

Suprématie quantique (2)

Vidéo de la chaîne “Sciences étonnantes” :
https://www.youtube.com/watch?v=KaRd_eB2qOA

Me contacter

- Formulaire de contact sur le site de Soarthec
- Twitter : benprieur