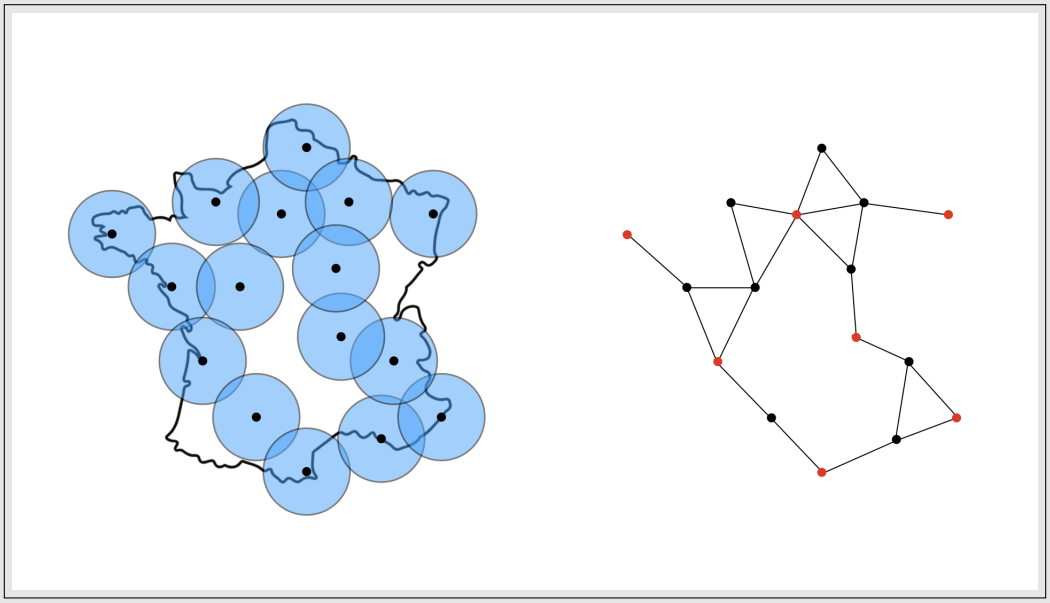
Problem - Solving the MIS problem with the Quantum Adiabatic Algorithm (QAA)
Consider an undirected graph $G(V, E)$ composed of a set of vertices V connected by unweighted edges E. An independent set of this graph is a subset of vertices where any two elements of this subset are not connected by an edge. The Maximum Independent Set (MIS) corresponds to the largest of such subsets, and it is in general an NP-complete problem to determine the MIS of a graph. The MIS problem has several interesting applications, such as portfolio diversification in finance, or broadcast systems (wifi or cellular network) optimization.

For example, assume an ensemble of identical radio transmitters over French cities that each have the same radius of transmission. It was quickly realized that two transmitters with close or equal frequencies could interfere with one another, hence the necessity to assign non-interfering frequencies to overlapping transmitting towers. Because of the limited amount of bandwidth space, some towers have to be assigned the same or close frequencies. The MIS of a graph of towers indicate the maximum number of towers that can have close or equal given frequency (red points).



When looking for the MIS of a graph, we separate the vertices into two distinct classes: an independence one and the others. We can attribute a status z to each vertex, where $z_i = 1$ if vertex i is attributed to the independent set, and $z_i = 0$ otherwise. The Maximum Independent Set corresponds to the minima of the following cost function:

$$C(z_1, \ldots, z_N) = -\sum_{i=1}^{N} z_i + U \sum_{(i,j)\in E} z_i z_j,$$

In this cost function, we want to promote a maximal number of vertices to the 1 state, but the fact that $U \gg 1$ strongly penalizes two adjacent vertices in state. The minimum of $C(z_1, \ldots, z_N)$ therefore corresponds to the maximum independent set of the graph.

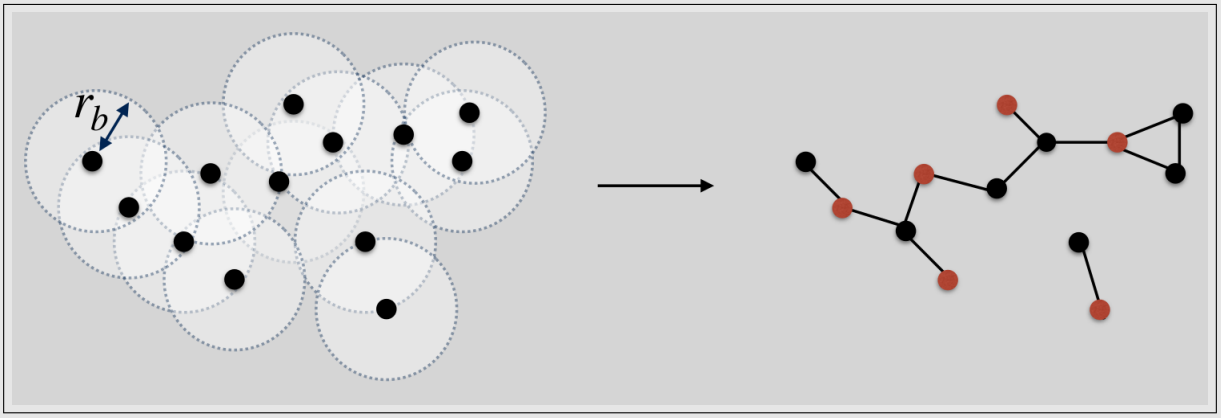
The MIS problem on unit disks graphs can be tackled by using an ensemble of interacting cold neutral atoms as a quantum resource, where each atom represents a vertex of the graph under study.

As with any quantum system, the dynamics of the atoms are governed by the Schrödinger equation, involving a Hamiltonian depending on the atomic positions, the electronic energy levels and their interactions. As seen in the course, placing N atoms at positions $\mathbf{r}_j$ in a plane, and coupling the ground state $|0\rangle$ to the Rydberg state $|1\rangle$ with a laser enables the realization of the Ising Hamiltonian:

$$H = \sum_{i=1}^{N} \frac{\hbar\Omega}{2}\sigma_i^x - \sum_{i=1}^{N} \frac{\hbar\delta}{2}\sigma_i^z + \sum_{j<i} \frac{C_6}{|\mathbf{r}_i - \mathbf{r}_j|^6} n_i n_j.$$

where $n_i = \frac{1}{2}(\sigma_i^z + 1)$ counts the number of Rydberg excitations at position i.

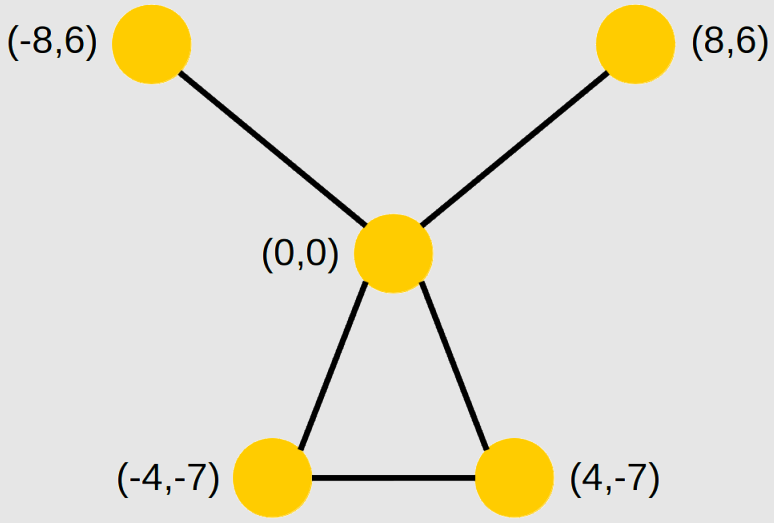
Interestingly, the physical interactions encoded in this Hamiltonian constrain the dynamics to only explore independent sets of the graph under study, then leading to an efficient search in the set of possible solutions. More precisely, the levels of two Rydberg atoms strongly interact if the distance between the atoms is smaller than the Rydberg blockade radius r_b, see left part of the figure below, resulting in the impossibility for the two atoms to be both in the same state at the same time. This naturally corresponds to the independent set constraint in the graph defined by the atoms, with edges linking atoms that sit at a distance closer than r_b, see right part of the figure where red vertices are forming one independent set.



From a spatial configuration of Rydberg atoms to a unit-disk MIS graph problem.

In the context of QAA, keeping the last two terms of H, i.e. by taking $\Omega = 0$ rad/µs, thus corresponds to the quantum cost Hamiltonian H_C . A mixer Hamiltonian H_M can then be realized by keeping the first term and canceling the second term of H, i.e. by taking $\delta = 0$ rad/µs.

We will now illustrate how one can use Pulser to tackle the MIS problem through an adiabatic evolution. The graph $G(V, E)$ we propose you to study is the following:



First of all, copy/past the following imports in a new notebook:

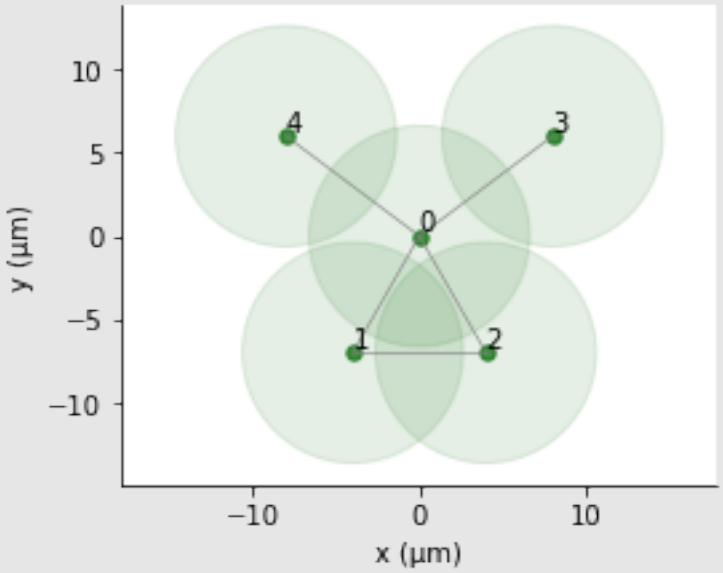
```
!pip install pulser

from pulser import Pulse, Sequence, Register
from pulser_simulation import Simulation
from pulser.devices import Chadoq2
from pulser.waveforms import InterpolatedWaveform
import matplotlib.pyplot as plt
import numpy as np
```

a. (Graph architecture) Considering a value of 1 rad/µs for Ω in the definition of the Rydberg blockade radius and using the `Chadoq2` device available in Pulser, insert the missing information in the following lines of code in order to build a register that will reproduce the graph to study:

```
qubits = ...
reg = Register(...)
reg.draw(
    blockade_radius=...,
    draw_graph=True,
    draw_half_radius=True,
)
```

When executing, your code should output the following figure:



b. (Maximum Independent Set) How many maximal independent sets do you identify on this graph? Give the corresponding configuration(s) in binary.

c. (Estimating sequence parameters) Now, you must build the sequence to implement an adiabatic evolution of your neutral atoms array. The idea behind the adiabatic algorithm is to slowly evolve the system from an easy-to-prepare ground-state to the ground-state of the cost Hamiltonian H_C. If done slowly enough, the system of atoms stays in the instantaneous ground-state.

For the problem under study, you will continuously vary the parameters $\Omega(t)$, $\delta(t)$ in time, starting with $\Omega(0) = 0$ rad/µs, $\delta(0) < 0$ rad/µs and ending with $\Omega(0) = 0$ rad/µs, $\delta(0) > 0$ rad/µs. The ground-state of $H(0)$ then corresponds to the initial state $|00000\rangle$ and the ground-state of $H(t_f)$ corresponds to the ground-state of H_C.

To ensure that we are not exciting the system to states that do not form independent sets, we have to estimate the minimal distance between atoms which are not connected in the graph (this yields Ω_{min}), and estimate the furthest distance between two connected atoms (this yields Ω_{max}). Keeping $\Omega \in [\Omega_{min}, \Omega_{max}]$ insures that only independent sets appear in the dynamics.

From the graph coordinates, estimate `no_link_min`, the min distance between not connected atoms in micrometers, and `link_max`, the furthest distance between two connected atoms. Then complete the following lines of code:

```
no_link_min = ...
link_max = ...
Omega_min = Chadoq2.interaction_coeff / no_link_min**6
Omega_max = Chadoq2.interaction_coeff / link_max**6
Omega = (Omega_max - Omega_min) / 2
```

We now need to define start and stop values for $\delta(t)$. So for that, copy/past the following lines of code:

```
delta_0 = -5  # just has to be negative
delta_f = -delta_0  # just has to be positive
```

d. (Building the sequence) From these parameters we can now build the sequence:

```
T = 4500  # time in ns, we choose a time long enough to ensure
          # the propagation of information in the system

adiabatic_pulse = Pulse(
    InterpolatedWaveform(T, [1e-9, Omega, 1e-9]),
    InterpolatedWaveform(T, [delta_0, 0, delta_f]),
    0,
)
seq = Sequence(reg, Chadoq2)
seq.declare_channel("ising", "rydberg_global")
seq.add(adiabatic_pulse, "ising")
seq.draw()
```

e. (Executing the sequence) We can finally execute the sequence:

```
simul = Simulation(seq)
results = simul.run()
final = results.get_final_state()
count_dict = results.sample_final_state()
```

The following function will allow you to display your results in the form of a bar chart, with red bars corresponding to MIS configurations:

```
def plot_distribution(C):
    C = dict(sorted(C.items(), key=lambda item: item[1], reverse=True))
    indexes = ["01011", "00111"]  # MIS indexes
    color_dict = {key: "r" if key in indexes else "g" for key in C}
    plt.figure(figsize=(12, 6))
    plt.xlabel("bitstrings")
    plt.ylabel("counts")
    plt.bar(C.keys(), C.values(), width=0.5, color=color_dict.values())
    plt.xticks(rotation="vertical")
    plt.show()
```

In only a few micro-seconds, you have found an excellent solution !

f. (How does the time evolution affect the quality of the results?) You may wondering how does the time evolution affect the quality of the results ? Let's execute our sequence for different time evolution and see how the success probability of the algorithm evolves.

```
success_probability = []
for T in 1000 * np.linspace(1, 10, 10):
    seq = Sequence(reg, Chadoq2)
    seq.declare_channel("ising", "rydberg_global")
    adiabatic_pulse = Pulse(
        InterpolatedWaveform(T, [1e-9, Omega, 1e-9]),
        InterpolatedWaveform(T, [delta_0, 0, delta_f]),
        0,
    )
    seq.add(adiabatic_pulse, "ising")
    simul = Simulation(seq)
    results = simul.run()
    final = results.get_final_state()
    count_dict = results.sample_final_state()
    success_probability.append((count_dict["01011"]+count_dict["00111"])/1000)

plt.figure(figsize=(12, 6))
plt.plot(range(1, 11), np.array(success_probability), "--o")
plt.xlabel("total time evolution (µs)", fontsize=14)
plt.ylabel("approximation ratio", fontsize=14)
plt.show()
```

As you can see, no more than 4-5 µs are necessary to reach high quality results.

Solution: This graph has two maximal independent sets: $(1, 3, 4)$ and $(2, 3, 4)$, respectively 01011 and 00111 in binary.

```
!pip install pulser

from pulser import Pulse, Sequence, Register
from pulser_simulation import Simulation
from pulser.devices import Chadoq2
from pulser.waveforms import InterpolatedWaveform
import matplotlib.pyplot as plt
import numpy as np

# Reproducing the graph architecture
```

```
qubits = {'0':(0, 0), '1':(-4, -7), '2':(4, -7), '3':(8, 6), '4':(-8, 6)}
reg = Register(qubits)
reg.draw(
    blockade_radius=Chadoq2.rydberg_blockade_radius(1.0),
    draw_graph=True,
    draw_half_radius=True,
)

# Estimating sequence parameters
link_max = 10  # Distance in micrometers between qubit 0 and 3
no_link_min = 13.6  # Distance in micrometers between qubit 2 and 3

Omega_min = Chadoq2.interaction_coeff / no_link_min**6
Omega_max = Chadoq2.interaction_coeff / link_max**6

Omega = (
    Omega_max - Omega_min
) / 2  # we choose a random value between the min and the max

delta_0 = -5  # just has to be negative
delta_f = -delta_0  # just has to be positive
T = 4500  # time in ns, we choose a time long enough to ensure            # the propagation of in

# Building the sequence
adiabatic_pulse = Pulse(
    InterpolatedWaveform(T, [1e-9, Omega, 1e-9]),
    InterpolatedWaveform(T, [delta_0, 0, delta_f]),
    0,
)
seq = Sequence(reg, Chadoq2)
seq.declare_channel("ising", "rydberg_global")
seq.add(adiabatic_pulse, "ising")
seq.draw()

# Executing the sequence
simul = Simulation(seq)
results = simul.run()
final = results.get_final_state()
count_dict = results.sample_final_state()

# Displaying results
def plot_distribution(C):
    C = dict(sorted(C.items(), key=lambda item: item[1], reverse=True))
    indexes = ["01011", "00111"]  # MIS indexes
    color_dict = {key: "r" if key in indexes else "g" for key in C}
    plt.figure(figsize=(12, 6))
    plt.xlabel("bitstrings")
    plt.ylabel("counts")
    plt.bar(C.keys(), C.values(), width=0.5, color=color_dict.values())
    plt.xticks(rotation="vertical")
    plt.show()

plot_distribution(count_dict)
```

```
# Success probability vs Time evolution
success_probability = []
for T in 1000 * np.linspace(1, 10, 10):
    seq = Sequence(reg, Chadoq2)
    seq.declare_channel("ising", "rydberg_global")
    adiabatic_pulse = Pulse(
        InterpolatedWaveform(T, [1e-9, Omega, 1e-9]),
        InterpolatedWaveform(T, [delta_0, 0, delta_f]),
        0,
    )
    seq.add(adiabatic_pulse, "ising")
    simul = Simulation(seq)
    results = simul.run()
    final = results.get_final_state()
    count_dict = results.sample_final_state()
    success_probability.append((count_dict["01011"]+count_dict["00111"])/1000)

plt.figure(figsize=(12, 6))
plt.plot(range(1, 11), np.array(success_probability), "--o")
plt.xlabel("total time evolution (µs)", fontsize=14)
plt.ylabel("approximation ratio", fontsize=14)
plt.show()
```
