Graph Simplicity

Status	Done

How to use GBS to construct a similarity measure between graphs, known as a graph kernel. Kernels can be applied to graph-based data for machine learning tasks such as classification using a support vector machine.

- The graph data used is from a dataset of 188 graphs each corresponding to the structure of a chemical compound. (MUTAG) - There are 20000 samples for 4 graphs. - 1 and 4 are isomorphic (similar) and 2 and 3 are similar.
    - The data module has pre-calculated GBS samples for MUTAG dataset.
    - You get each of these samples by encoding the graph into a GBS device and collecting photon click events.
- We want to create a feature vector to describe each graph. It can be composed in many ways.
    - Our approach is to associate features of with the relative frequencies of certain types of measurement being recorded from a GBS device configured to sample from the graph.
- GBS feature vectors can be composed of probabilities of coarse-grained combinations of elementary samples. We consider two coarse grainings:
    - **Orbits:** Combine all samples that can be made identical under permutation. Orbits are written simply as a sorting of integer photon number samples in non-increasing order with the zeros at the end removed. For example, `[1, 1, 2, 0]` and `[2, 1, 0, 1]` both belong to the `[2, 1, 1]` orbit.
        - Sample to orbit function
        - orbit to sample function
    - **Events:** Combine all k-photon orbits where the maximum photon count in any mode does not exceed a fixed value n_{max} into an event $E_{k,nmax}$. For example, orbits `[2, 1]`, `[1, 1, 1]` are part of the $E_{3,2}$event, while orbit `[3]` is not.
        - orbits(n) function gives all the orbits with max photons n.
        - use sample to event to add max count per node.

The similarity module provides the following tools for dealing with coarse-grained orbits and events.

Creating a feature vector

A feature vector of a graph can be created by choosing a collection of orbits or events and evaluating their probabilities with respect to GBS with the embedded graph. These probabilities are then selected to be elements of the feature vector.

Evaluating the probabilities of orbits or events can be achieved through three approaches:

- **Direct sampling:** infer the probability of orbits or events from a set of sample data.

```
print(similarity.feature_vector_events_sampling(m0, [2, 4], 2))

'''
Output:
[0.19035, 0.2047]
''''

#Use any orbits of our choice instead of events
print(similarity.feature_vector_orbits_sampling(m0, [[1, 1], [2], [1, 1, 1, 1], [2, 1, 1]]))

'''
Output:
```

```
[0.19035, 0.0, 0.1352, 0.05175]
'''
```

- **Monte Carlo estimation:** samples within an orbit or event are selected uniformly at random and their exact probabilities are calculated.
- **Exact calculation:** probabilities are calculated exactly, which involves calculating a large number of hafnians.

Built-in functions `feature_vector_orbits()` and `feature_vector_events()` can be used to get exact feature vectors. These functions use a keyword argument `samples` to signal producing either exact or Monte Carlo estimated probabilities, as shown later. `samples` is set to `None` to get an exact feature vector by default. To use Monte Carlo estimation, `samples` can be set to the number of samples desired to be used in the estimation.

💡 Once we get the feature vectors we can run SVM for classification or similar tasks.

References

1. https://strawberryfields.ai/photonics/apps/run_tutorial_similarity.html
2. https://strawberryfields.readthedocs.io/en/stable/code/api/api/strawberryfields.apps.data.Mutag0.html#strawberry
3. https://strawberryfields.readthedocs.io/en/stable/code/api/strawberryfields.apps.similarity.html
4. https://en.wikipedia.org/wiki/Support_vector_machine