Counting qubits: Zephyr vs Pegasus

Notebook that maps a Hamiltonian defined through truncated and penalization embeddings on both the Pegasus and the Zephyr topology to comparate the number of needed physical qubits.

Legenda:
TO DO (Th): notes to myself, additional theory to add for the thesis presentation.
TO DO (IF): notes to myself, a non-necessary idea to implement.
TO DO (Crit): problem or doubt to discuss and fix.

Setup

In [ ]:
```
# import
import my_lib as lib
import article_lib as article

import examples_anf as anf

import numpy as np
import pandas as pd
from sympy import symbols
from dwave.inspector import show
```

In [ ]:
```
## bit list

bit_max = 8                                        # max number of bits to run
bit_list = [i for i in range(4, bit_max+1, 2)]      # even numbers from 4 to bit_max

p_list = []
x_list = []

for bits in bit_list:
```

```
    x = symbols(" ".join((f"x{i}" for i in range(1, bits+1))))

    if bits == 4:
        p, sol = anf.example_4_anf(*x)
    elif bits == 6:
        p, sol = anf.example_6_anf(*x)
    elif bits == 8:
        p, sol = anf.example_8_anf(*x)
    elif bits == 10:
        p, sol = anf.example_10_anf(*x)
    elif bits == 12:
        p, sol = anf.example_12_anf(*x)
    elif bits == 14:
        p, sol = anf.example_14_anf(*x)
    elif bits == 16:
        p, sol = anf.example_16_anf(*x)
    elif bits == 18:
        p, sol = anf.example_18_anf(*x)

    p_list.append(p)
    x_list.append(x)
```

Truncated Embedding

Building the Hamiltonian

Since we will run the truncated Hamiltonians both via Pegasus and Zephyr topology, it's best to define them a priori and store them away.

They will be added to a designated list already in QUBO format whilist their offsets will be stored in another list, in the same order.

In [ ]:

```
## building truncated H

trunc_Hs = []
trunc_offsets = []

for i in range(len(bit_list)):
```

```
    # symbolic def
    truncated = article.truncated_embedding(bit_list[i], p_list[i], x_list[i])
    trunc_H = truncated.create_hamiltonian()
    trunc_sym = truncated.get_symbols()

    # QUBO model
    d_art = article.dwave_annealing(trunc_H, bit_list[i], trunc_sym)
    trunc_H_qubo, trunc_qubo_offset = d_art.symbolic_to_dwave(trunc_H, d_art.get_symbol_num(trunc_sym))

    # store
    trunc_Hs.append(trunc_H_qubo)
    trunc_offsets.append(trunc_qubo_offset)
```

Pegasus

In [ ]:
```
p_trunc_qubits = {}

for i in range(len(bit_list)):

    trunc_runner = lib.dwave_runners(trunc_Hs[i], trunc_offsets[i],
                                     bit_list[i], topology='Pegasus',
                                     chosen_chainstrength='Article')

    logical_qubits, physical_qubits = trunc_runner.counting_qubits(average=4)

    p_trunc_qubits[f'{bit_list[i]}_bits'] = [logical_qubits, physical_qubits]
```

```
Building the BQM model.
Running on Pegasus Topology.
You chose the Article chainstrength: 27.530612244897956.
Finished running the experiment #1!
Finished running the experiment #2!
Finished running the experiment #3!
Finished running the experiment #4!

--------------------------------------------------------------------------

Building the BQM model.
Running on Pegasus Topology.
You chose the Article chainstrength: 28.797385620915033.
Finished running the experiment #1!
Finished running the experiment #2!
Finished running the experiment #3!
Finished running the experiment #4!

--------------------------------------------------------------------------

Building the BQM model.
Running on Pegasus Topology.
You chose the Article chainstrength: 30.021599999999996.
Finished running the experiment #1!
Finished running the experiment #2!
Finished running the experiment #3!
Finished running the experiment #4!

--------------------------------------------------------------------------
```

In [ ]:

```
p_trunc_qubits_df = pd.DataFrame(p_trunc_qubits, index=['logical_qubits', 'physical_qubits'])
print('\033[1m' + '-----------Used Qubits on Pegasus-----------' + '\033[0m')
print(p_trunc_qubits_df)
```

```
-----------Used Qubits on Pegasus-----------
                 4_bits  6_bits  8_bits
logical_qubits    30.00   90.00   231.0
physical_qubits   53.75  226.25   748.0
```

Zephyr

In [ ]:
```
z_trunc_qubits = {}

for i in range(len(bit_list)):

    trunc_runner = lib.dwave_runners(trunc_Hs[i], trunc_offsets[i],
                                     bit_list[i], topology='Zephyr',
                                     chosen_chainstrength='Article')

    logical_qubits, physical_qubits = trunc_runner.counting_qubits(average=4)

    z_trunc_qubits[f'{bit_list[i]}_bits'] = [logical_qubits, physical_qubits]
    max_i = i
```

```
Building the BQM model.
Running on Zephyr Topology.
You chose the Article chainstrength: 27.530612244897956.
Finished running the experiment #1!
Finished running the experiment #2!
Finished running the experiment #3!
Finished running the experiment #4!

-------------------------------------------------------------------------

Building the BQM model.
Running on Zephyr Topology.
You chose the Article chainstrength: 28.797385620915033.
Finished running the experiment #1!
Finished running the experiment #2!
Finished running the experiment #3!
Finished running the experiment #4!

-------------------------------------------------------------------------

Building the BQM model.
Running on Zephyr Topology.
You chose the Article chainstrength: 30.0215999999999996.
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In [6], line 9
      3 for i in range(len(bit_list)):
      5     trunc_runner = lib.dwave_runners(trunc_Hs[i], trunc_offsets[i],
      6                                      bit_list[i], topology='Zephyr',
      7                                      chosen_chainstrength='Article')
----> 9     logical_qubits, physical_qubits = trunc_runner.counting_qubits(average=4)
     11     z_trunc_qubits[f'{bit_list[i]}_bits'] = [logical_qubits, physical_qubits]
     12     max_i = i

File c:\Users\sgala\OneDrive\Desktop\VSCode\Codici_VsCode_Git\Tesi\QA_Zephyr\my_lib.py:399, in dwave_runners.counting_qubits
(self, average)
    395 self.define_chainstrength()
    397 for t in range(average):
--> 399     response = self.counting_run()
    400     print(f'Finished running the experiment #{t+1}!')
    402     used_embedding = response.info['embedding_context']['embedding']

File c:\Users\sgala\OneDrive\Desktop\VSCode\Codici_VsCode_Git\Tesi\QA_Zephyr\my_lib.py:369, in dwave_runners.counting_run(sel
f)
    359 def counting_run(self):
    361     '''
    362         Single run for counting qubits. It's defined to not calculate multiple
    363         times the chainstrength and to avoid useless prints.
   (...)
    366             response = SampleSet object with the details of the run.
    367     '''
--> 369     response = self.sampler.sample(self.H, chain_strength=self.chain_strength,
    370                                 num_reads=self.numruns, annealing_time=self.T,
    371                                 answer_mode='histogram',
    372                                 label = f'mq_on_{self.topology}')
    374     return(response)

File ~\AppData\Roaming\Python\Python310\site-packages\dwave\system\composites\embedding.py:239, in EmbeddingComposite.sample
(self, bqm, chain_strength, chain_break_method, chain_break_fraction, embedding_parameters, return_embedding, warnings, **par
ameters)
    235 embedding = self.find_embedding(source_edgelist, target_edgelist,
    236                                 **embedding_parameters)
    238 if bqm and not embedding:
```

```
--> 239     raise ValueError("no embedding found")
    241 if not hasattr(embedding, 'embed_bqm'):
    242     embedding = EmbeddedStructure(target_edgelist, embedding)

ValueError: no embedding found
```

Here we face our greatest limit: at the moment (December 2023), the Zephyr topology is only available as a prototype which counts roughly 500 qubits. Hence, our examples with $n \geq 8$ bits will not have a valid embedding and there will be no run.

We will manually add this information in order to create the Pandas dataframe. Notice that we can reuse the number of logical qubits from the Pegasus runs, considering that this number only depends on the formulation of the Hamiltonian (which is independent from the topology).

TO DO (Th): add math calculus (especially in the Thesis) for physical qubits on this topology, if possible. In the meantime, it will be `None`.

In [ ]:
```
## completing info

for j in range(max_i+1, len(bit_list)):
    z_trunc_qubits[f'{bit_list[i]}_bits'] = [p_trunc_qubits[f'{bit_list[i]}_bits'][0], None]
```

In [ ]:
```
z_trunc_qubits_df = pd.DataFrame(z_trunc_qubits, index=['logical_qubits', 'physical_qubits'])
print('\033[1m' + '-----------Used Qubits on Zephyr-----------' + '\033[0m')
print(z_trunc_qubits_df)
```

```
-----------Used Qubits on Zephyr-----------
                 4_bits  6_bits  8_bits
logical_qubits    30.00    90.0   231.0
physical_qubits   50.25   213.5     NaN
```

Penalization Embedding

Building the Hamiltonian

In [ ]:
```
## building penalization H a priori

pen_Hs = []
pen_offsets = []
```

```python
for i in range(len(bit_list)):

    # symbolic def
    penalization = article.penalization_embedding(bit_list[i], p_list[i], x_list[i])
    pen_H = penalization.create_hamiltonian()
    pen_sym = penalization.get_symbols()

    # QUBO model
    d_art = article.dwave_annealing(pen_H, bit_list[i], pen_sym)
    pen_H_qubo, pen_qubo_offset = d_art.symbolic_to_dwave(pen_H, d_art.get_symbol_num(pen_sym))

    # store
    pen_Hs.append(pen_H_qubo)
    pen_offsets.append(pen_qubo_offset)
```

```
Total output qubits used: 29

Total output qubits used: 71

Total output qubits used: 161
```

Pegasus

In [ ]:

```python
p_pen_qubits = {}

for i in range(len(bit_list)):

    pen_runner = lib.dwave_runners(pen_Hs[i], pen_offsets[i],
                                   bit_list[i], topology='Pegasus',
                                   chosen_chainstrength='Article')

    logical_qubits, physical_qubits = pen_runner.counting_qubits(average=4)

    p_pen_qubits[f'{bit_list[i]}_bits'] = [logical_qubits, physical_qubits]
```

```
Building the BQM model.
Running on Pegasus Topology.
You chose the Article chainstrength: 28.935483870967744.
Finished running the experiment #1!
Finished running the experiment #2!
Finished running the experiment #3!
Finished running the experiment #4!

--------------------------------------------------------------------------

Building the BQM model.
Running on Pegasus Topology.
You chose the Article chainstrength: 29.934782608695652.
Finished running the experiment #1!
Finished running the experiment #2!
Finished running the experiment #3!
Finished running the experiment #4!

--------------------------------------------------------------------------

Building the BQM model.
Running on Pegasus Topology.
You chose the Article chainstrength: 31.689342403628117.
Finished running the experiment #1!
Finished running the experiment #2!
Finished running the experiment #3!
Finished running the experiment #4!

--------------------------------------------------------------------------
```

In [ ]:

```
p_pen_qubits_df = pd.DataFrame(p_pen_qubits, index=['logical_qubits', 'physical_qubits'])
print('\033[1m' + '-----------Used Qubits on Pegasus-----------' + '\033[0m')
print(p_pen_qubits_df)
```

-----------Used Qubits on Pegasus-----------

	4_bits	6_bits	8_bits
logical_qubits	61.00	150.00	345.0
physical_qubits	92.75	301.75	855.5

Zephyr

In [ ]:
```
z_pen_qubits = {}

for i in range(len(bit_list)):

    pen_runner = lib.dwave_runners(pen_Hs[i], pen_offsets[i],
                                    bit_list[i], topology='Zephyr',
                                    chosen_chainstrength='Article')

    logical_qubits, physical_qubits = pen_runner.counting_qubits(average=4)

    z_pen_qubits[f'{bit_list[i]}_bits'] = [logical_qubits, physical_qubits]
    max_i = i
```

```
Building the BQM model.
Running on Zephyr Topology.
You chose the Article chainstrength: 28.935483870967744.
Finished running the experiment #1!
Finished running the experiment #2!
Finished running the experiment #3!
Finished running the experiment #4!

--------------------------------------------------------------------------

Building the BQM model.
Running on Zephyr Topology.
You chose the Article chainstrength: 29.934782608695652.
Finished running the experiment #1!
Finished running the experiment #2!
Finished running the experiment #3!
Finished running the experiment #4!

--------------------------------------------------------------------------

Building the BQM model.
Running on Zephyr Topology.
You chose the Article chainstrength: 31.689342403628117.
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In [7], line 9
      3 for i in range(len(bit_list)):
      5     pen_runner = lib.dwave_runners(pen_Hs[i], pen_offsets[i],
      6                                    bit_list[i], topology='Zephyr',
      7                                    chosen_chainstrength='Article')
----> 9     logical_qubits, physical_qubits = pen_runner.counting_qubits(average=4)
     11     z_pen_qubits[f'{bit_list[i]}_bits'] = [logical_qubits, physical_qubits]
     12     max_i = i

File c:\Users\sgala\OneDrive\Desktop\VSCode\Codici_VsCode_Git\Tesi\QA_Zephyr\my_lib.py:399, in dwave_runners.counting_qubits
(self, average)
    395 self.define_chainstrength()
    397 for t in range(average):
--> 399     response = self.counting_run()
    400     print(f'Finished running the experiment #{t+1}!')
    402     used_embedding = response.info['embedding_context']['embedding']

File c:\Users\sgala\OneDrive\Desktop\VSCode\Codici_VsCode_Git\Tesi\QA_Zephyr\my_lib.py:369, in dwave_runners.counting_run(sel
f)
    359 def counting_run(self):
    361     '''
    362         Single run for counting qubits. It's defined to not calculate multiple
    363         times the chainstrength and to avoid useless prints.
   (...)
    366             response = SampleSet object with the details of the run.
    367     '''
--> 369     response = self.sampler.sample(self.H, chain_strength=self.chain_strength,
    370                           num_reads=self.numruns, annealing_time=self.T,
    371                           answer_mode='histogram',
    372                           label = f'mq_on_{self.topology}')
    374     return(response)

File ~\AppData\Roaming\Python\Python310\site-packages\dwave\system\composites\embedding.py:239, in EmbeddingComposite.sample
(self, bqm, chain_strength, chain_break_method, chain_break_fraction, embedding_parameters, return_embedding, warnings, **par
ameters)
    235 embedding = self.find_embedding(source_edgelist, target_edgelist,
    236                                 **embedding_parameters)
    238 if bqm and not embedding:
```

```
--> 239     raise ValueError("no embedding found")
    241 if not hasattr(embedding, 'embed_bqm'):
    242     embedding = EmbeddedStructure(target_edgelist, embedding)

ValueError: no embedding found
```

In [ ]:

```
## completing info

for j in range(max_i+1, len(bit_list)):
    z_pen_qubits[f'{bit_list[i]}_bits'] = [p_pen_qubits[f'{bit_list[i]}_bits'][0], None]
```

In [ ]:

```
z_pen_qubits_df = pd.DataFrame(z_pen_qubits, index=['logical_qubits', 'physical_qubits'])
print('\033[1m' + '-----------Used Qubits on Zephyr-----------' + '\033[0m')
print(z_pen_qubits_df)
```

```
-----------Used Qubits on Zephyr-----------
                 4_bits  6_bits  8_bits
logical_qubits     61.0   150.0   345.0
physical_qubits    91.0   265.5     NaN
```