Comparing the iterative method on Pegasus vs Zephyr Topology

Title TBD

(Intro TBD better)

Notebook to run the iterative method for the direct embedding fo MQ problems with 5 and 9 bits. The aim of this notebook is to analyze how the code runs on the new Zephyr topology compared to the now state-of-art Pegasus topology.

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_nnf as nnf

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

MQ with 5 bits

In [ ]:
```
## problem parameters
```

```
bits = 5             # number of binary variables
iterations = 5       # number of iterations in iterative method
treshold = 10        # number of low energy solutions to check for the i.m.
```

In [ ]:
```
## problem def

x5_sym = symbols(" ".join((f"x{i}" for i in range(1, bits+1))))
p5_sym, sol5 = nnf.example_5(*x5_sym)
```

In [ ]:
```
## building symbolic H

direct = article.direct_embedding(bits, p5_sym, x5_sym)
H_sym = direct.create_hamiltonian()
sym = direct.get_symbols()
#print(H_sym)
#print(sym)
```

In [ ]:
```
## building QUBO model

d_art = article.dwave_annealing(H_sym, bits, sym)
H_qubo, qubo_offset = d_art.symbolic_to_dwave(H_sym, d_art.get_symbol_num(sym))
#print(H_qubo)
#print(qubo_offset)
```

Pegasus

Since with 5 bits there is no need to run multiple iterations, we call the function `single_run`.

In [ ]:
```
runner = lib.dwave_runs(H_qubo, qubo_offset, bits,
                        topology='Pegasus',
                        chosen_chainstrength='Article' )
p_response = runner.single_run()
```

```
Building the BQM model.
Running on Pegasus Topology.
You chose the Article chainstrength: 37.10526315789473.
Finished running the experiment!
```

In [ ]:
```
solution = runner.decoding_response(p_response)
print(f'The solution found is one of the known solutions: {solution in sol5}')
```

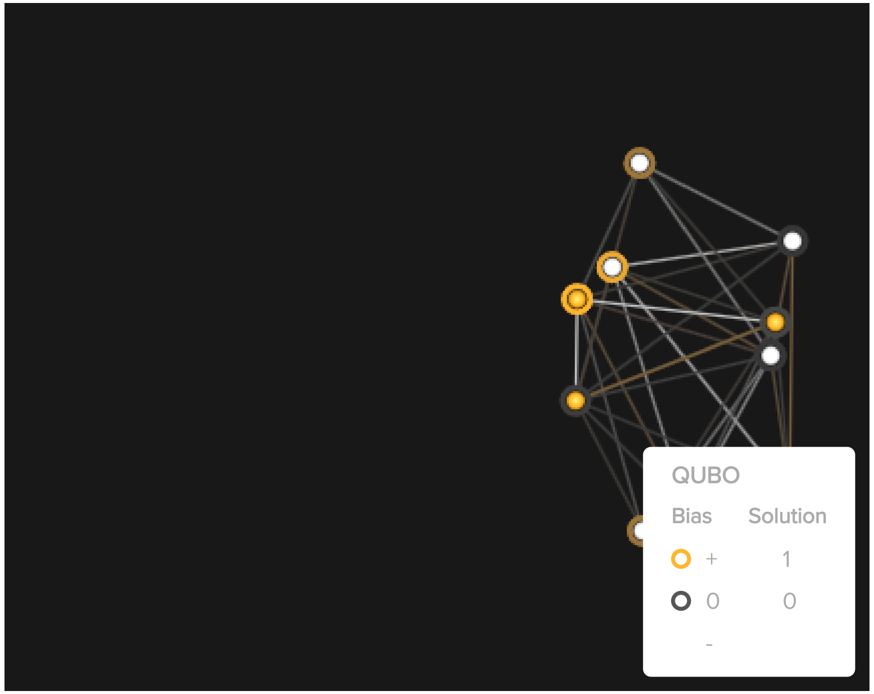
```
Solution found: [1, 1, 0, 0, 1] with energy: 0.0
The solution found is one of the known solutions: True
```

Using the D-Wave `inspector` to visualize the results and the info:

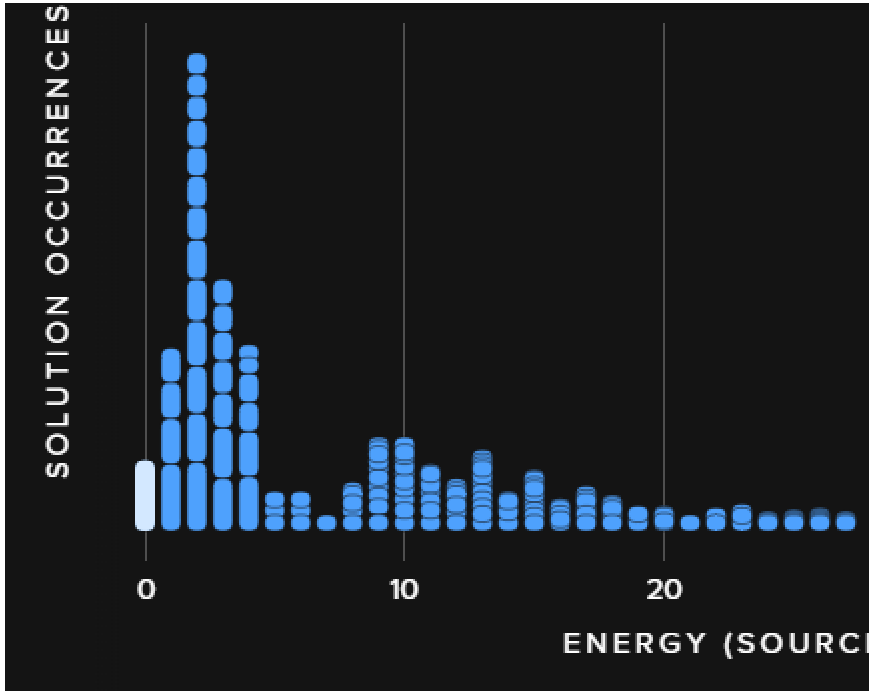
In [ ]:
```
show(p_response)
```

Problem Details Solver Details 1 2

Source - Force Directed



Source - Samples - Histogram



Console All Warning Types Show warnings for selected sample only

No Warnings

Out[ ]: 'http://127.0.0.1:18000/?problemId=b56885ec-1fe7-41a3-beb3-d96d5f305087'

TO DO (T): add more info on embedding

Zephyr

In [ ]:
```
runner = lib.dwave_runs(H_qubo, qubo_offset, bits,
                        topology='Zephyr',
                        chosen_chainstrength='Article' )
z_response = runner.single_run()
```

```
Building the BQM model.
Running on Zephyr Topology.
You chose the Article chainstrength: 37.10526315789473.
Finished running the experiment!
```

In [ ]:
```
solution = runner.decoding_response(z_response)
print(f'The solution found is one of the known solutions: {solution in sol5}')
```

```
Solution found: [1, 1, 0, 0, 1] with energy: 0.0
The solution found is one of the known solutions: True
```

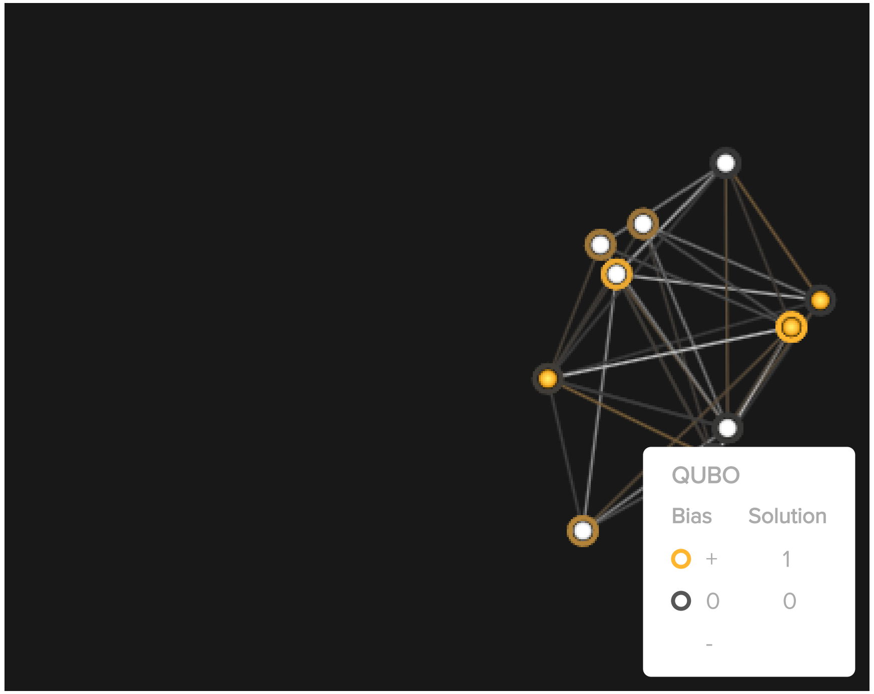
In [ ]:
```
show(z_response)
```

Problem Details

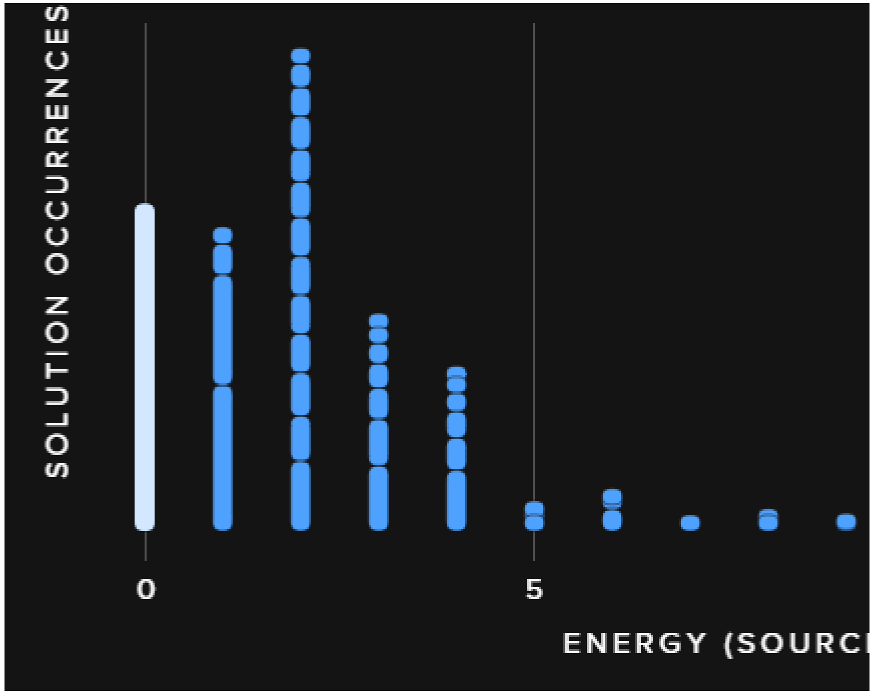
Solver Details

1 2

Source - Force Directed



Source - Samples - Histogram



Console

All Warning Types

Show warnings for selected sample only

No Warnings

Out[ ]: 'http://127.0.0.1:18000/?problemId=8fd22536-33bd-41ff-bb7d-b24452a5bcaf'

Analysis

Let's compare the informations on the runs on the two topologies:

In [ ]:
```
## pegasus info
pegasus_timing = p_response.info['timing']
pegasus_embedding = p_response.info['embedding_context']['embedding']

## zephyr info
zephyr_timing = z_response.info['timing']
zephyr_embedding = z_response.info['embedding_context']['embedding']
```

In [ ]:
```
## comparing timing data from the runs

diff = []
for key in zephyr_timing:
    diff.append(zephyr_timing[key] - pegasus_timing[key])

data = {'Zephyr': zephyr_timing.values(),
        'Pegasus': pegasus_timing.values(),
        'Differences': diff}

timings_df = pd.DataFrame(data, index = zephyr_timing.keys())
print('\033[1m' + '-----------Difference in timings (Zephyr vs Pegasus)-----------' + '\033[0m')
print(timings_df)
```

```
-----------Difference in timings (Zephyr vs Pegasus)-----------
                                Zephyr     Pegasus  Differences
qpu_sampling_time             74140.00    92860.00    -18720.00
qpu_anneal_time_per_sample       20.00       20.00         0.00
qpu_readout_time_per_sample      33.12       52.32       -19.20
qpu_access_time               80986.01   108619.17    -27633.16
qpu_access_overhead_time        484.99     3460.83     -2975.84
qpu_programming_time           6846.01    15759.17     -8913.16
qpu_delay_time_per_sample        21.02       20.54         0.48
post_processing_overhead_time  1678.00     1430.00       248.00
total_post_processing_time     1678.00     1430.00       248.00
```

TO DO (T): add comparison to press release of Zephyr, to see if the timing improvement obtained is in the same order of the one presented (?)

In [ ]:
```
## comparing embedding data from the runs

z_count, z_chains = runner.get_info_on_embedding(zephyr_embedding)
p_count, p_chains = runner.get_info_on_embedding(pegasus_embedding)
```

In [ ]:
```
print(f'The run on the Zephyr topology required {z_count} physical qubits.')
print(f'The run on the Pegasus topology required {p_count} physical qubits.\n')

data = {'Zephyr': z_chains,
        'Pegasus': p_chains}
chains_df = pd.DataFrame(data, index = z_chains.keys()).transpose()
print('\033[1m' + '------------Difference in embeddings (Zephyr vs Pegasus)------------' + '\033[0m')
print(chains_df)
```

```
The run on the Zephyr topology required 13 physical qubits.
The run on the Pegasus topology required 15 physical qubits.

------------Difference in embeddings (Zephyr vs Pegasus)------------
         lq_0  lq_1  lq_2  lq_3  lq_4  lq_5  lq_6  lq_7  lq_8  lq_9
Zephyr      2     2     2     1     1     1     1     1     1     1
Pegasus     2     1     2     2     1     2     2     1     1     1
```

MQ with 9 bits

In [ ]:
```
## problem parameters

bits = 9             # number of binary variables
iterations = 5       # number of iterations in iterative method
treshold = 10        # number of low energy solutions to check for the i.m.
```

In [ ]:
```
## problem def

x9_sym = symbols(" ".join((f"x{i}" for i in range(1, bits+1))))
p9_sym, sol9 = nnf.example_9(*x9_sym)
```

In [ ]:
```
## building symbolic H

direct = article.direct_embedding(bits, p9_sym, x9_sym)
H_sym = direct.create_hamiltonian()
sym = direct.get_symbols()
#print(H_sym)
#print(sym)
```

In [ ]:
```
## building QUBO model

d_art = article.dwave_annealing(H_sym, bits, sym)
H_qubo, qubo_offset = d_art.symbolic_to_dwave(H_sym, d_art.get_symbol_num(sym))
#print(H_qubo)
#print(qubo_offset)
```

Pegasus

Here, we need the iterative method.

In [ ]:
```
runner = lib.dwave_runs(H_qubo, qubo_offset, bits, 'Pegasus', 'Article')
print('\n----------------------------------------------------------------------\n')
p_solution, p_timing_info, p_physical_qubits, p_final_it = runner.iterative()
```

```
Building the BQM model.
Running on Pegasus Topology.

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

Number of variables: 46
You chose the Article chainstrength: 217.7682481751825.
Finished running the experiment!
Energy of best sample at iteration 0: 39.0
Fixing ancillae...
Fixed 12 qubits:
12 : 0
14 : 0
16 : 0
23 : 0
27 : 0
28 : 0
37 : 0
38 : 0
39 : 0
41 : 0
44 : 0
45 : 0

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

Number of variables: 34
You chose the Article chainstrength: 296.72903225806454.
Finished running the experiment!
Energy of best sample at iteration 1: 3.0
Fixing ancillae...
Fixed 16 qubits:
12 : 0
14 : 0
16 : 0
23 : 0
27 : 0
28 : 0
37 : 0
38 : 0
39 : 0
```

```
41 : 0
44 : 0
45 : 0
11 : 0
18 : 0
20 : 0
21 : 0

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

Number of variables: 30
You chose the Article chainstrength: 309.54545454545456.
Finished running the experiment!
Energy of best sample at iteration 2: 2.0
Fixing ancillae...
Fixed 21 qubits:
12 : 0
14 : 0
16 : 0
23 : 0
27 : 0
28 : 0
37 : 0
38 : 0
39 : 0
41 : 0
44 : 0
45 : 0
11 : 0
18 : 0
20 : 0
21 : 0
13 : 0
17 : 0
19 : 0
22 : 0
24 : 0

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

Number of variables: 25
```

```
You chose the Article chainstrength: 335.8857142857143.
Finished running the experiment!
Energy of best sample at iteration 3: 0.0
Solution found with final iteration 3.
Reconstructing the final state...
Solution found: [1, 0, 0, 1, 0, 1, 0, 1, 1]
```

Zephyr

In [ ]:
```
runner = lib.dwave_runs(H_qubo, qubo_offset, bits, 'Zephyr', 'Article')
print('\n------------------------------------------------------------------------\n')
z_solution, z_timing_info, z_physical_qubits, z_final_it = runner.iterative()
```

```
Building the BQM model.
Running on Zephyr Topology.

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

Number of variables: 46
You chose the Article chainstrength: 217.7682481751825.
Finished running the experiment!
Energy of best sample at iteration 0: 1.0
Fixing ancillae...
Fixed 18 qubits:
11 : 0
12 : 0
14 : 0
16 : 0
18 : 0
19 : 0
20 : 0
21 : 0
22 : 0
23 : 0
24 : 0
25 : 0
27 : 0
28 : 0
29 : 0
30 : 0
31 : 0
32 : 0

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

Number of variables: 28
You chose the Article chainstrength: 327.22813688212926.
Finished running the experiment!
Energy of best sample at iteration 1: 2.0
Fixing ancillae...
Fixed 20 qubits:
11 : 0
12 : 0
14 : 0
```

```
16 : 0
18 : 0
19 : 0
20 : 0
21 : 0
22 : 0
23 : 0
24 : 0
25 : 0
27 : 0
28 : 0
29 : 0
30 : 0
31 : 0
32 : 0
13 : 0
15 : 0

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

Number of variables: 26
You chose the Article chainstrength: 337.4605263157895.
Finished running the experiment!
Energy of best sample at iteration 2: 1.0
Fixing ancillae...
Fixed 20 qubits:
11 : 0
12 : 0
14 : 0
16 : 0
18 : 0
19 : 0
20 : 0
21 : 0
22 : 0
23 : 0
24 : 0
25 : 0
27 : 0
28 : 0
29 : 0
```

```
30 : 0
31 : 0
32 : 0
13 : 0
15 : 0

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

Number of variables: 26
You chose the Article chainstrength: 337.4605263157895.
Finished running the experiment!
Energy of best sample at iteration 3: 1.0
Fixing ancillae...
Fixed 22 qubits:
11 : 0
12 : 0
14 : 0
16 : 0
18 : 0
19 : 0
20 : 0
21 : 0
22 : 0
23 : 0
24 : 0
25 : 0
27 : 0
28 : 0
29 : 0
30 : 0
31 : 0
32 : 0
13 : 0
15 : 0
10 : 0
17 : 0

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

Number of variables: 24
You chose the Article chainstrength: 333.01554404145077.
```

```
Finished running the experiment!
Energy of best sample at iteration 4: 2.0
Fixing ancillae...
Fixed 23 qubits:
11 : 0
12 : 0
14 : 0
16 : 0
18 : 0
19 : 0
20 : 0
21 : 0
22 : 0
23 : 0
24 : 0
25 : 0
27 : 0
28 : 0
29 : 0
30 : 0
31 : 0
32 : 0
13 : 0
15 : 0
10 : 0
17 : 0
9 : 0

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

Reconstructing the final state...
Solution found: [0, 0, 0, 1, 0, 1, 0, 1, 1]
```

Analysis

In [ ]:

```
print(f'The iterative method on the Zephyr topology required a total of {z_physical_qubits} physical qubits for {z_final_it+1}
print(f'It found a solution: {z_solution in sol9}.\n')
```

```
print(f'The iterative method on the Pegasus topology required a total of {p_physical_qubits} physical qubits for {p_final_it+1
print(f'It found a solution: {p_solution in sol9}.')
```

```
The iterative method on the Zephyr topology required a total of 390 physical qubits for 5 iterations.
It found a solution: False.

The iterative method on the Pegasus topology required a total of 383 physical qubits for 4 iterations.
It found a solution: True.
```

In [ ]:

```
## comparing timing data

diff_iterative = []
for key in zephyr_timing:
    diff_iterative.append(z_timing_info[key] - p_timing_info[key])

data = {'Zephyr': z_timing_info.values(),
        'Pegasus': p_timing_info.values(),
        'Differences': diff_iterative}

iterative_timings_df = pd.DataFrame(data, index = z_timing_info.keys())
print('\033[1m' + '-------------Difference in timings (Zephyr vs Pegasus)-------------' + '\033[0m')
print(iterative_timings_df)
```

-------------Difference in timings (Zephyr vs Pegasus)-------------

	Zephyr	Pegasus	Differences
qpu_sampling_time	354840.00	414640.00	-59800.00
qpu_anneal_time_per_sample	100.00	60.00	40.00
qpu_readout_time_per_sample	149.74	293.02	-143.28
qpu_access_time	389069.25	461918.31	-72849.06
qpu_access_overhead_time	6600.75	8986.69	-2385.94
qpu_programming_time	34229.25	47278.31	-13049.06
qpu_delay_time_per_sample	105.10	61.62	43.48
post_processing_overhead_time	9135.00	4636.00	4499.00
total_post_processing_time	9135.00	4636.00	4499.00

TO DO (Crit): the timings values should be normalized somehow, given that more iterations bring a bigger time if it's working better.

If no normalization can be found, maybe we can restrict the analysis to some of these values and comparate those for each single_run?