Finding multiple optimal solutions using the sampling functionality of the D-Wave system

REFERENCE EXAMPLE

2019-10-06

Overview

D-Wave systems use quantum annealing to solve problems by seeking minimums of an energy function. Because multiple anneals may return different solutions, we can “sample” the problem and examine the lowest-energy solutions of the returned results.

This document presents examples of this process for maximum independent set problems.

CONTACT

Corporate Headquarters
3033 Beta Ave
Burnaby, BC V5G 4M9
Canada
Tel. 604-630-1428

US Office
2650 E Bayshore Rd
Palo Alto, CA 94303

Email: info@dwavesys.com

www.dwavesys.com

14-1038A-A

Notice and disclaimer

Contents

1 Introduction

D-Wave systems use quantum annealing to solve problems by seeking minimums of an energy function. Because multiple anneals may return different solutions, we can "sample" the problem and examine the lowest-energy solutions of the returned results.

1.1 MIS problems

In this document, we introduce the sampling functionality of the D-Wave system using maximum independent set (MIS) problems as examples. A MIS is a set of a graph's vertices with no edge connecting any of its member pairs. The examples use the D-Wave system to find the combination that maximizes the number of vertices.

1.2 Ocean SDK

We use D-Wave's Ocean SDK to create programs to solve the example problems and run them on the D-Wave system. The Ocean SDK is a set of open-source Python tools for programming D-Wave quantum computers.

Links:

- Ocean SDK on GitHub: https://github.com/dwavesystems/dwave-ocean-sdk
- Documentation: https://docs.ocean.dwavesys.com/en/latest/

Note: Program code is provided in the Appendix.

1.3 Probabilistic nature of solutions

In the results shown here, you will notice that some of the solutions are suboptimal. Because the D-Wave system is probabilistic, you usually see a distribution of answers among the returned results, some of which may be suboptimal. For this reason, you should almost always request multiple anneals with your problem submission.

With quantum annealing, samples are generated extremely fast even for large problems. And as you will see, additional anneals have a small effect on the amount of time required to solve the problem.

2 Example 1

Find the MIS for the following graph with 7 vertices.

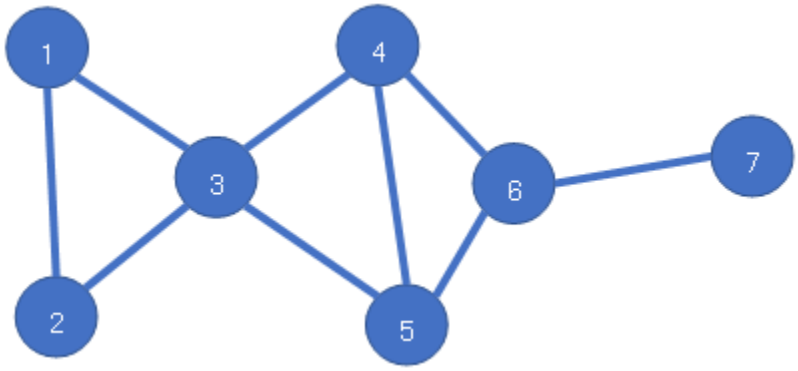

Figure 1. In this graph, if vertex 1 is selected, vertices 2 and 3 cannot be selected. When vertex 3 is selected, vertices 1, 2, 4, and 5 cannot be selected.

The following sets all have the maximum number of vertices (3) for the graph; i.e., they are optimal solutions to the problem:

- 1, 4, 7
- 1, 5, 7
- 2, 4, 7
- 2, 5, 7

We can use the D-Wave system to seek all the optimal solutions. Because we can request multiple anneals in a single problem submission, the system can return multiple solutions at once.

Use the Ocean SDK to create and execute the program to solve this problem; use the code provided in the appendix. Data is transferred from the PC to the D-Wave system at the line:

```
result = sampler.sample_qubo (Q, num_reads = 1000)
```

Quantum annealing is performed the specified number of times (1000 times in this example), and result is returned to the PC.

Contents of array Q:

```
{(1, 1): -1.0, (2, 2): -1.0, (3, 3): -1.0, (4, 4): -1.0, (5, 5): -1.0, (6, 6): -1.0,
(7, 7): -1.0, (1, 2): 2.0, (1, 3): 2.0, (2, 3): 2.0, (3, 4): 2.0, (3, 5): 2.0, (4, 5):
2.0, (4, 6): 2.0, (5, 6): 2.0, (6, 7): 2.0}
```

The returned result contains the following fields: sample (vertex combination pattern), energy value, and number of occurrences:

```
{1: 0, 2: 1, 3: 0, 4: 0, 5: 1, 6: 0, 7: 1} -3.0 189
```

In the vertex combination pattern, `1: 0` means “Vertex 1: Not selected,” `2: 1` means "Vertex 2: selected", `3: 0` means “Vertex 3: Not selected,” and so on. The energy value of the combination of vertices 2, 5, and 7 is `-3.0`, and it appears 189 out of 1000 times.

The same applies below, which is a returned result of 1000 anneals.

```
{1: 1, 2: 0, 3: 0, 4: 1, 5: 0, 6: 0, 7: 1} -3.0 252
{1: 0, 2: 1, 3: 0, 4: 1, 5: 0, 6: 0, 7: 1} -3.0 316
{1: 1, 2: 0, 3: 0, 4: 0, 5: 1, 6: 0, 7: 1} -3.0 236
{1: 0, 2: 0, 3: 1, 4: 0, 5: 0, 6: 1, 7: 0} -2.0 2
{1: 1, 2: 0, 3: 0, 4: 0, 5: 0, 6: 1, 7: 0} -2.0 1
{1: 0, 2: 0, 3: 1, 4: 0, 5: 0, 6: 0, 7: 1} -2.0 3
{1: 0, 2: 1, 3: 0, 4: 1, 5: 1, 6: 0, 7: 1} -2.0 1
```

This result shows that the following combinations of vertices have the lowest energy value, `-3.0`, so the combination with the maximum number of vertices (3) is these four patterns:

- 2, 5, 7
- 1, 4, 7
- 2, 4, 7
- 1, 5, 7

The total time spent on the quantum processing unit (QPU) for this run of 1000 anneals for this problem was 238,606 μs (approximately 0.2 s).

3 Example 2

Next, find the MIS for the following graph.

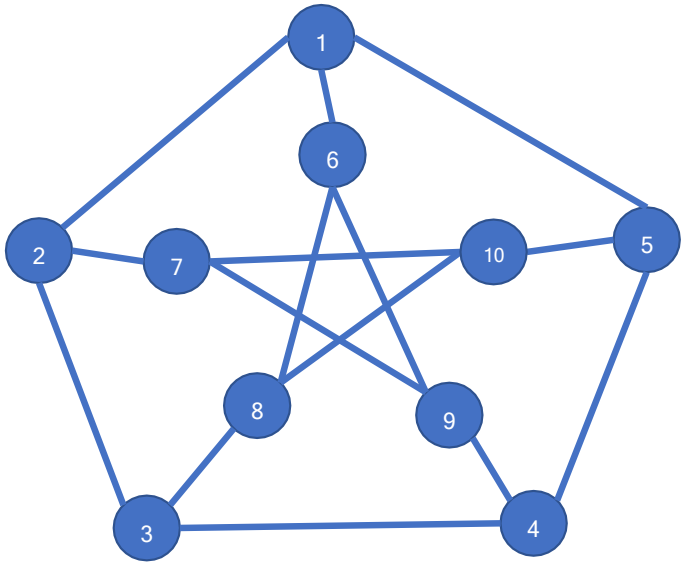

Figure 2.

Rewrite only the edge definition of the program in Example 1 (see the appendix for the code). When this problem is run on the D-Wave system, the result is as follows:

```
{1: 0, 2: 1, 3: 0, 4: 1, 5: 0, 6: 1, 7: 0, 8: 0, 9: 0, 10: 1} -4.0 287
{1: 1, 2: 0, 3: 1, 4: 0, 5: 0, 6: 0, 7: 0, 8: 0, 9: 1, 10: 1} -4.0 144
{1: 0, 2: 1, 3: 0, 4: 0, 5: 1, 6: 0, 7: 0, 8: 1, 9: 1, 10: 0} -4.0 107
{1: 1, 2: 0, 3: 0, 4: 1, 5: 0, 6: 0, 7: 1, 8: 1, 9: 0, 10: 0} -4.0 197
{1: 0, 2: 0, 3: 1, 4: 0, 5: 1, 6: 1, 7: 1, 8: 0, 9: 0, 10: 0} -4.0 243
{1: 0, 2: 0, 3: 1, 4: 0, 5: 1, 6: 0, 7: 1, 8: 0, 9: 0, 10: 0} -3.0 1
{1: 0, 2: 1, 3: 0, 4: 0, 5: 1, 6: 1, 7: 0, 8: 0, 9: 0, 10: 0} -3.0 2
{1: 0, 2: 1, 3: 0, 4: 0, 5: 1, 6: 0, 7: 0, 8: 1, 9: 0, 10: 0} -3.0 1
{1: 0, 2: 0, 3: 1, 4: 0, 5: 1, 6: 1, 7: 0, 8: 0, 9: 0, 10: 0} -3.0 1
{1: 0, 2: 1, 3: 0, 4: 1, 5: 0, 6: 1, 7: 0, 8: 0, 9: 0, 10: 0} -3.0 2
{1: 0, 2: 0, 3: 0, 4: 1, 5: 0, 6: 1, 7: 1, 8: 0, 9: 0, 10: 0} -3.0 4
{1: 1, 2: 0, 3: 0, 4: 1, 5: 0, 6: 0, 7: 0, 8: 1, 9: 0, 10: 0} -3.0 2
{1: 0, 2: 0, 3: 1, 4: 0, 5: 1, 6: 0, 7: 0, 8: 0, 9: 1, 10: 0} -3.0 1
{1: 0, 2: 1, 3: 0, 4: 0, 5: 1, 6: 0, 7: 0, 8: 0, 9: 1, 10: 0} -3.0 1
{1: 0, 2: 0, 3: 1, 4: 0, 5: 0, 6: 1, 7: 0, 8: 0, 9: 0, 10: 1} -3.0 2
{1: 1, 2: 0, 3: 1, 4: 0, 5: 0, 6: 0, 7: 0, 8: 0, 9: 0, 10: 1} -3.0 1
{1: 0, 2: 1, 3: 0, 4: 1, 5: 0, 6: 0, 7: 0, 8: 0, 9: 0, 10: 1} -3.0 1
{1: 1, 2: 0, 3: 1, 4: 0, 5: 0, 6: 0, 7: 0, 8: 0, 9: 1, 10: 0} -3.0 2
{1: 1, 2: 0, 3: 0, 4: 1, 5: 0, 6: 0, 7: 1, 8: 0, 9: 0, 10: 1} -2.0 1
```

In this result, five vertex combinations have the lowest returned energy value, `-4.0`, so the combination with the maximum number of vertices (4) is the five patterns:

- 2, 4, 6, 10
- 1, 3, 9, 10
- 2, 5, 8, 9
- 1, 4, 7, 8
- 3, 5, 6, 7

For this problem, the anneal time spent was 238,622 μs (approximately 0.2 s). Notice that although the number of vertices and the number of edges is larger than the graph in Example 1, the QPU time is almost the same.

4 Example 3

Next, find the MIS for the graph shown in Example 2 with additional vertices and edges as below.

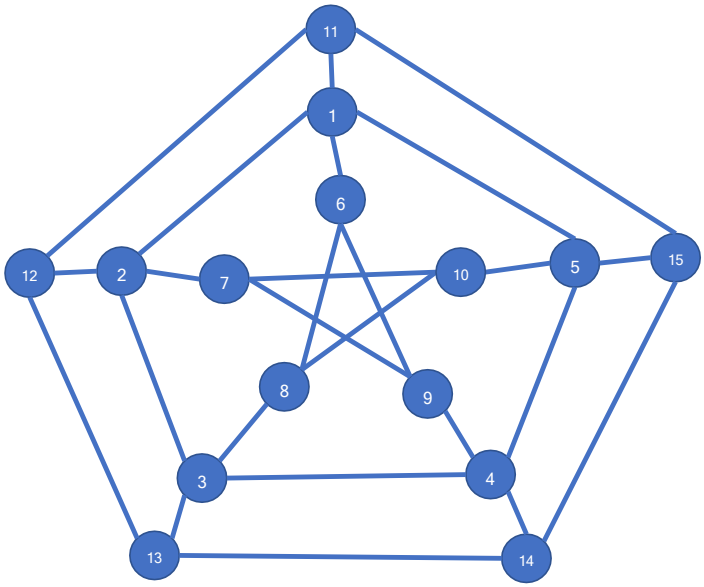

Figure 3.

Rewrite only the edge definition of the program in Example 2 as follows and execute it.

```
G.add_edges_from([(1,2),(2,3),(3,4),(4,5),(5,1),(6,8),(8,10),(10,7),(7,9),(9,6),(1,6),
(2,7),(3,8),(4,9),(5,10),(1,11),2,12),(3,13),(4,14),(5,15),(11,12),(12,13),(13,14),
(14,15),(15,11)])
```

The following 10 patterns are obtained as the combination (best found solutions) with minimum energy (`-6.0`). The maximum number of vertices is 6.

```
{1: 0, 2: 1, 3: 0, 4: 1, 5: 0, 6: 1, 7: 0, 8: 0, 9: 0, 10: 1, 11: 1, 12: 0, 13: 1, 14:
0, 15: 0} -6.0 96

{1: 0, 2: 1, 3: 0, 4: 1, 5: 0, 6: 1, 7: 0, 8: 0, 9: 0, 10: 1, 11: 0, 12: 0, 13: 1, 14:
0, 15: 1} -6.0 68

{1: 1, 2: 0, 3: 0, 4: 1, 5: 0, 6: 0, 7: 1, 8: 1, 9: 0, 10: 0, 11: 0, 12: 1, 13: 0, 14:
0, 15: 1} -6.0 20

{1: 1, 2: 0, 3: 1, 4: 0, 5: 0, 6: 0, 7: 0, 8: 0, 9: 1, 10: 1, 11: 0, 12: 1, 13: 0, 14:
1, 15: 0} -6.0 284

{1: 1, 2: 0, 3: 0, 4: 1, 5: 0, 6: 0, 7: 1, 8: 1, 9: 0, 10: 0, 11: 0, 12: 0, 13: 1, 14:
0, 15: 1} -6.0 25

{1: 0, 2: 0, 3: 1, 4: 0, 5: 1, 6: 1, 7: 1, 8: 0, 9: 0, 10: 0, 11: 0, 12: 1, 13: 0, 14:
1, 15: 0} -6.0 88

{1: 0, 2: 0, 3: 1, 4: 0, 5: 1, 6: 1, 7: 1, 8: 0, 9: 0, 10: 0, 11: 1, 12: 0, 13: 0, 14:
1, 15: 0} -6.0 100
```

```
{1: 0, 2: 1, 3: 0, 4: 0, 5: 1, 6: 0, 7: 0, 8: 1, 9: 1, 10: 0, 11: 1, 12: 0, 13: 0, 14:
1, 15: 0} -6.0 82

{1: 1, 2: 0, 3: 1, 4: 0, 5: 0, 6: 0, 7: 0, 8: 0, 9: 1, 10: 1, 11: 0, 12: 1, 13: 0, 14:
0, 15: 1} -6.0 146

{1: 0, 2: 1, 3: 0, 4: 0, 5: 1, 6: 0, 7: 0, 8: 1, 9: 1, 10: 0, 11: 1, 12: 0, 13: 1, 14:
0, 15: 0} -6.0 65
```

The anneal time for this run of the problem was 238,640 μs (approximately 0.2 s). Again, although the number of vertices and edges is larger than the previous graphs, the QPU time is almost the same.

Appendix A: Program code

The functions used below are provided in the Ocean SDK.

A.1 Example 1 code

```
# Import necessary packages

import networkx as nx

from dwave_networkx.algorithms.independent_set import
maximum_weighted_independent_set_qubo

from dwave.system.samplers import DWaveSampler

from dwave.system.composites import EmbeddingComposite

# Create empty graph(G)

G = nx.Graph()

# Configure graph (G) by defining each edge

G.add_edges_from([(1,2), (1,3), (2,3), (3,4), (3,5), (4,5), (4,6), (5,6), (6,7)])

# Create QUBO (Q) for finding the maximum independent set from graph (G)

Q = maximum_weighted_independent_set_qubo(G)

# Print the contents of Q

print(Q)

# Set sampler (map to D-Wave structure)

sampler = EmbeddingComposite(DWaveSampler())

# Map Q and perform annealing 1000 times with D-Wave

result = sampler.sample_qubo(Q, num_reads=1000)

# Print the result of the 1000 annealings

for s, e, n in result.data(['sample', 'energy', 'num_occurrences']):

    print(s, e, n)
```

A.2 Example 2 code

```
import networkx as nx

from dwave_networkx.algorithms.independent_set import
maximum_weighted_independent_set_qubo

from dwave.system.samplers import DWaveSampler

from dwave.system.composites import EmbeddingComposite

G = nx.Graph()

G.add_edges_from([(1,2), (2,3), (3,4), (4,5), (5,1), (6,8), (8,10), (10,7), (7,9),
(9,6), (1,6), (2,7), (3,8), (4,9), (5,10)])

Q = maximum_weighted_independent_set_qubo(G)

print(Q)

sampler = EmbeddingComposite(DWaveSampler())

result = sampler.sample_qubo(Q, num_reads=1000)

for s, e, n in result.data(['sample', 'energy', 'num_occurrences']):

    print(s, e, n)
```