Penalty Model

Apr 26, 2022

Contents

One approach to solve a constraint satisfaction problem (CSP) using an Ising model or a QUBO, is to map each individual constraint in the CSP to a 'small' Ising model or QUBO. This mapping is called a *penalty model*.

Imagine that we want to map an AND clause to a QUBO. In other words, we want the solutions to the QUBO (the solutions that minimize the energy) to be exactly the valid configurations of an AND gate. Let $z = AND(x_1, x_2)$.

Before anything else, let's import that package we will need.

```
import penaltymodel
import dimod
import networkx as nx
```

Next, we need to determine the feasible configurations that we wish to target (by making the energy of these configuration in the binary quadratic low). Below is the truth table representing an AND clause.

Table 1: AND Gate

x_1	x_2	z
0	0	0
0	1	0
1	0	0
1	1	1

The rows of the truth table are exactly the feasible configurations.

```
feasible_configurations = [{'x1': 0, 'x2': 0, 'z': 0},
                            {'x1': 1, 'x2': 0, 'z': 0},
                            {'x1': 0, 'x2': 1, 'z': 0},
                            {'x1': 1, 'x2': 1, 'z': 1}]
```

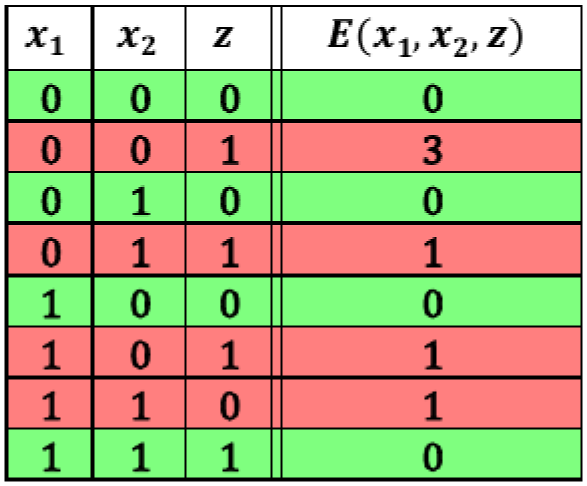
At this point, we can get a penalty model

```
bqm, gap = pm.get_penalty_model(feasible_configurations)
```

However, if we know the QUBO, we can build the penalty model ourselves. We observe that for the equation:

$$E(x_1, x_2, z) = x_1 x_2 - 2(x_1 + x_2)z + 3z + 0$$

We get the following energies for each row in our truth table.



x_1	x_2	z	$E(x_1, x_2, z)$
0	0	0	0
0	0	1	3
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

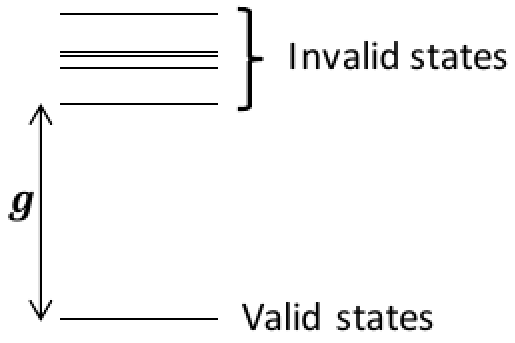
We can see that the energy is minimized on exactly the desired feasible configurations. So we encode this energy function as a QUBO. We make the offset 0.0 because there is no constant energy offset.

```
qubo = dimod.BinaryQuadraticModel({'x1': 0., 'x2': 0., 'z': 3.},
                                  {('x1', 'x2'): 1., ('x1', 'z'): 2., ('x2', 'z'): 2.},
                                  0.0,
                                  dimod.BINARY)
```

We know from the table that our ground energy is 0, but we can calculate it using the qubo to check that this is true for the feasible configuration $(0, 1, 0)$.

```
ground_energy = qubo.energy({'x1': 0, 'x2': 1, 'z': 0})
```

The last value that we need is the classical gap. This is the difference in energy between the lowest infeasible state and the ground state.



With all of the pieces, we can now build the penalty model.

```
classical_gap = 1
p_model = pm.PenaltyModel.from_specification(spec, qubo, classical_gap, ground_energy)
```

CHAPTER 1

Reference Documentation

This package implements the generation and caching of penalty models.

The main function for penalty models is:

In addition to `get_penalty_model()`, there are some more advanced interfaces available.

1.1 Cache

1.1.1 Methods

`PenaltyModelCache.close`
`PenaltyModelCache.insert_binary_quadratic_model`
`PenaltyModelCache.insert_graph`
`PenaltyModelCache.insert_penalty_model`
`PenaltyModelCache.insert_sampleset`
`PenaltyModelCache.iter_binary_quadratic_models`
`PenaltyModelCache.iter_graphs`
`PenaltyModelCache.iter_penalty_models`
`PenaltyModelCache.iter_samplesets`
`PenaltyModelCache.retrieve`

1.2 Exceptions

`ImpossiblePenaltyModel`

Continued on next page

Table 2 – continued from previous page

MissingPenaltyModel

1.3 Utilities

as_graph