GI Quantum Computing Workshop 2023

# quark: QUantum Application Reformulation Kernel

Dr. Elisabeth Lobe

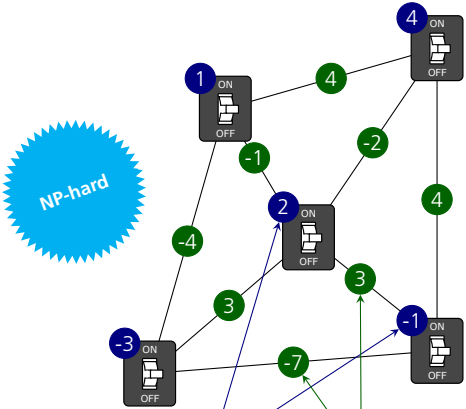
High-Performance Computing
Institute for Software Technology
DLR German Aerospace Center

29th September 2023


DLR

# Optimizing QUBO Problems

- **Q**uadratic **U**nconstrained **B**inary **O**ptimization
  - minimization
  - over binary variables
  - of a quadratic objective function
  - without further constraints




$$\text{minimize} \sum_{i=1}^{n} W_i \, x_i + \sum_{i<j=1}^{n} S_{ij} \, x_i x_j$$

subject to $x \in \{0,1\}^n$

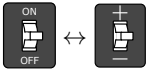
# Optimizing QUBO Problems

- **Q**uadratic **U**nconstrained **B**inary **O**ptimization
  - minimization
  - over binary variables
  - of a quadratic objective function
  - without further constraints

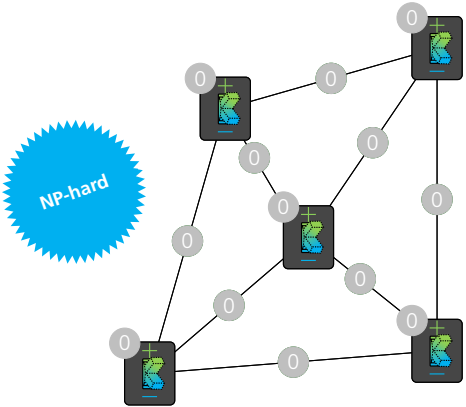
- Equivalent to Ising problem

$$x_v = \frac{1}{2}(s_v - 1)$$

$$s_v = 2x_v - 1 \in \{-1, 1\}$$



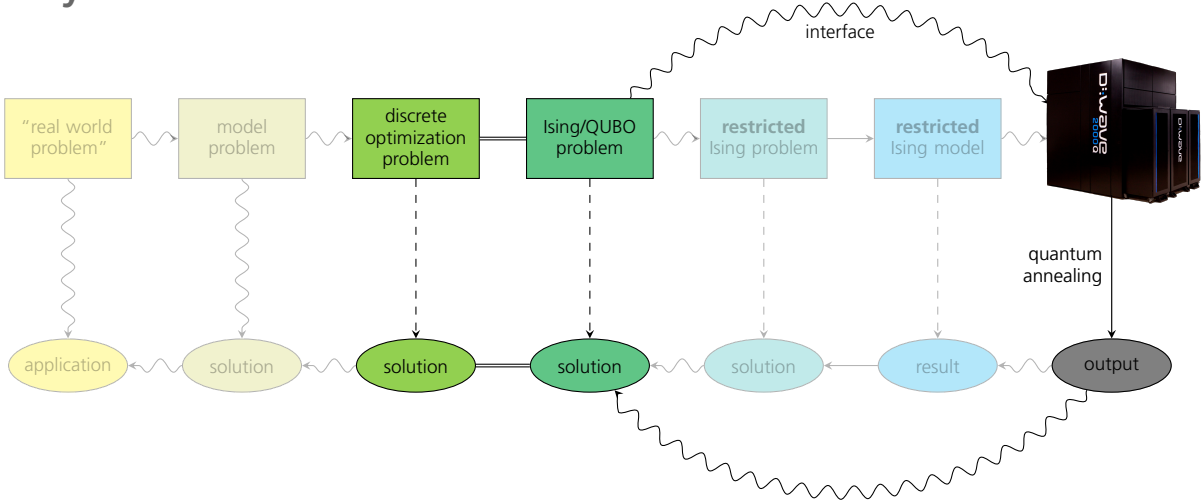

- "Programming" the Quantum Annealer
  - providing the weights $W \in \mathbb{R}^n$ and strengths $S \in \mathbb{R}^{n \times n}$




$$\text{minimize} \sum_{i=1}^{n} \frac{1}{4} H_i \, s_i + \sum_{i<j=1}^{n} \frac{1}{4} J_{ij} \, s_i s_j + c$$

subject to $s \in \{-1, +1\}^n$

# Layers of Abstraction



# Goals

- Easily reproducible experiments
  - reformulate arbitrary optimization problems to Ising problems
  - by automated transformation steps
  - based on a parameterized formulation of the problem instances
  - generate several problem instances with the same structure
  - store or load intermediate data at every stage of the transformation process
- Allow analysis of the machine behaviour and experimental results
  - provide hints whether problems are suitable to be solved with the annealers
  - provide solutions of classical solver for comparison
- Modularity for flexible usage
  - adapt to different hardware architectures and thus restrictions
  - base for further algorithmic implementations

# quark Objects for Main Transformation Step

**constrained discrete optimization problem**

$$\begin{aligned} \text{minimize} \quad & p(x) \\ \text{subject to} \quad & \ell_1 \leq q_1(x) \leq u_1, \\ & \ldots, \\ & \ell_m \leq q_m(x) \leq u_m, \\ & x \in \mathbb{Z}^n \end{aligned}$$

→

**QUBO/Ising problem**

$$\begin{aligned} \text{minimize} \quad & \sum_{i=1}^{N} W_i x_i + \sum_{i<j=1}^{N} S_{ij} x_i x_j \\ \text{subject to} \quad & x \in \{0,1\}^N / \{-1,1\}^N \end{aligned}$$

- `Polynomial` { `PolyBinary`, `PolyIsing` }
- `ConstraintBinary`
- `VariableMapping`

- `Instance`
- `ConstrainedObjective`
- `ObjectiveTerms`
- `Objective`
- `Solution`
- `ScipModel`

# Instance Definition

[1]:
```
class MCSInstance():

    def __init__(self, edges, colors):
        self.edges = edges
        self.nodes = set(node for edge in self.edges for node in edge)
        self.colors = colors

edges = [('a', 'b'), ('b', 'c'), ('b', 'd'), ('c', 'd')]
colors = ['red', 'blue', 'green']

instance = MCSInstance(edges, colors)
```

# Constrained Objective Definition

[2]:
```
from quark import PolyBinary, ConstraintBinary, ConstrainedObjective

class MCSConstrainedObjective(ConstrainedObjective):

    @staticmethod
    def _get_objective_poly(instance):
        # sum_[c in Colors] sum_[(n, m) in Edges] (1 * x_n_c * x_m_c)
        return PolyBinary({(('X', node1, color), ('X', node2, color)): 1
                           for node1, node2 in instance.edges
                           for color in instance.colors})
    @staticmethod
    def _get_constraints(instance):
        constraints = {}
        # for all n in Nodes: sum_[c in Colors] x_n_c == 1
        for node in instance.nodes:
            poly = PolyBinary({(('X', node, color),): 1 for color in instance.colors})
            constraints[f'one_color_for_{node}'] = ConstraintBinary(poly, 1, 1)
        return constraints
```

# Objective Construction

[3]:
```
instance               = MCSInstance(edges, colors)
constrained_objective = MCSConstrainedObjective(instance=instance)
objective_terms        = constrained_objective.get_objective_terms()
terms_weights          = objective_terms.get_default_terms_weights()
objective              = objective_terms.get_objective(terms_weights)
print(objective.polynomial)
```

```
+4 -1 X_a_blue -1 X_a_green -1 X_a_red -1 X_b_blue -1 X_b_green -1 X_b_red
-1 X_c_blue -1 X_c_green -1 X_c_red -1 X_d_blue -1 X_d_green -1 X_d_red
+2 X_a_blue X_a_green +2 X_a_blue X_a_red +1 X_a_blue X_b_blue +...
```

# IO Concept

- All objects can be stored and loaded
- at every stage of the transformation process
- completely automated except for the `Instance`

- use library `hdf5` to store data in h5-files
- ‘outsourced’ to not interfere with logic
- needs to be loaded explicitly

# Our Software Library

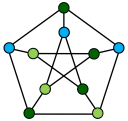
application layer

transformation layer

hardware restrictions layer

problem parameters

**quapps**



$$\min \sum_{c \in C} \sum_{nm \in E} x_{nc} x_{mc}$$

$$\text{s.t.} \sum_{c \in C} x_{nc} = 1 \; \forall n \in N,$$

$$x \in \{0,1\}^{N \times C}$$

problem description

(cost function, constraints)

**quark**

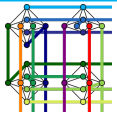
```
class testPolynomial(unittest.TestCase):
    def setUp(self):
        # cubic: 3 x1 + 9 x2 + 3 x3 x5 + x4 + 3 x2 x4 +
        self.Q1 = polynomial.Polynomial({(1,): 3, (2,):
        # quadratic: 3 x1 x2 + 5 x1 x3 + x4
        self.Q2 = polynomial.Polynomial({(1,): 3, (2, 4)
        self.Q3 = polynomial.Polynomial({(1,): 3, (2,):
        self.Q3 = polynomial.Polynomial({(1,): 7, (2,):
        self.Q4 = polynomial.Polynomial({(('a', 1),): 7,

        self.filename_h5 = testdir + "/test_qubo.h5"
        self.filename_txt = testdir + "/test_qubo.txt"

    def testEqual(self):
        Q = polynomial.Polynomial({(1,): 7, (2,): 2, (3,
        self.assertFalse(Q == self.Q1)
```

$$\min \sum_{i=1}^{n} H_i s_i + \sum_{i,j=1}^{n} J_{ij} s_i s_j$$

$$\text{s.t.} \; s \in \{-1,+1\}^V$$

(restricted) Ising problem

**complete graph embedding**

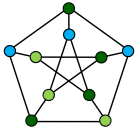


**weight distribution**

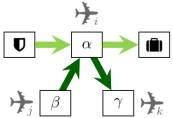
# quapps: QUantum APPlicationS

- Max Cut

- Max Colorable Subgraph



- Prime Factorization $N = p \cdot q$
- Flight Gate Assignment




- ...

individual parameterized implementations of

`ConstrainedObjective`/ `ObjectiveTerms`

based on the corresponding `Instance` definition

# Questions?

Dr. Elisabeth Lobe

High-Performance Computing
Institute for Software Technology
DLR German Aerospace Center

elisabeth.lobe@dlr.de

# Code

Source: `https://gitlab.com/quantum-computing-software/`
Package: `https://anaconda.org/dlr-sc/quark`


DLR