


# quark: QUantum Application Reformulation Kernel

Elisabeth Lobe 

Institute for Software Technology, German Aerospace Center (DLR),  
Lilienthalplatz 7, 38108 Braunschweig, Germany

19.07.2023

**Abstract:** Quantum annealers solve Ising problems heuristically. Several standard methods have been established to transform more complex problems into the Ising problem format, which are commonly still applied by hand. In this work, we present our software package `quark`, automating the full transformation process from an arbitrary discrete optimization problem to the corresponding Ising problem. Based on a parameterized formulation of the original problem, a series of easily reproducible experiments can thus be set up. This allows users to evaluate the suitability of the annealing machines in solving their specific problem without a deeper knowledge about the Ising problem specifics.

**Keywords:** Ising Problem, QUBO, Discrete Optimization, Quantum Annealing, Quantum Computing

## 1 Introduction

Quantum annealing is a method aiming at optimizing certain objective functions by taking advantage of a quantum mechanical process, the adiabatic evolution towards a quantum system representing the optimal solution in its ground state [6]. Hereby, several physical effects might disturb the annealing process leading to sub-optimal solutions, which is why the corresponding devices, the quantum annealers, are considered as heuristic samplers [11].

The implemented problems are known as Ising problems, which search for the minimum of a quadratic function over spin  $(+1/-1)$  variables without any further constraints. In the mathematical community, the notation of quadratic unconstrained binary optimization (QUBO) problems is more common, which refers to the same type of problem but acting on binary  $(0/1)$  variables. These problems are known to be NP-hard in general [1] and

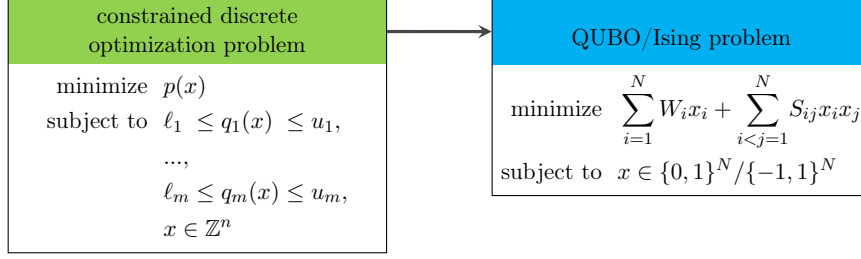


Figure 1: Main transformation step automatically performed with **quark** for providing the problem in a format the quantum annealers can process

therefore also relate to a multitude of other problems for which the best known classical algorithm takes exponential runtime, and which are thus intractable by classical devices above a certain size of the problem instance.

For a lot of, in general rather academic, combinatorial optimization problems the corresponding Ising problem formulations are known, such as for maximum cut or graph coloring. See [9] for an extensive list of examples. Several more advanced applications have also already been reformulated, e.g., the flight-gate assignment (FGA) problem as part of the airport planning [12]. A lot of such industrial use cases refer back to some academic example, like the FGA to the quadratic assignment problem. However, the experience from these applications shows that their actual formulations differ in the details, hence also require individually derived Ising/QUBO formulations.

In Figure 1, we have depicted the general main transformation step with the corresponding mathematical formulations of the problems. Several standard methods have been established to perform this step and to reformulate given arbitrary discrete optimization problems [7]. They are usually applied manually, requiring a deeper understanding of the properties of Ising problems rather than only the domain-specific knowledge of the original optimization problem. This demands for an automation of the full process to shift the complexity away from the end-users, such that they are able to easily produce a large test set and evaluate the capabilities of the solver for their specific problem.

## 2 Package Description

We have developed and constantly extend a software library at <https://gitlab.com/quantum-computing-software/>, whose core package is **quark** [5]. In this section, we briefly summarize the main goals of **quark** and explain its general usage along an example.

## 2.1 Goals and Main Features

Our package **quark** is designed to provide a user-friendly, low-level entry point for domain experts to deal with quantum annealers. Based on a parameterized formulation of the problem, several problem instances can be generated with the same structure. The corresponding optimization problems can then be reformulated to Ising problems by automated transformation steps, which enables the user to perform easily reproducible experiments over a large test set. This allows an analysis of the machine behaviour and experimental results more suited to the domain demands, providing hints whether the problems are suitable to be solved with the annealers. This is supported by an interface to a classical optimizer.

D-Wave already provides an extensive software library surrounding their quantum annealers, also already implementing several transformation steps [3]. We do not want to compete with but rather accompany D-Wave’s API with our instance-centric approach focusing on the original problem. A particular example where we simplify the provided functionality is the handling of polynomials with degree larger than two, which is available through the D-Wave API only via detours, cf. [3], which means that the user needs to know about the structural differences. In **quark** no additional but just the base classes are necessary. With the step from a constrained to an unconstrained problem, reduction variables with corresponding penalty terms are introduced and the degree of the polynomials is thus reduced automatically.

From our own experience in experimenting with the annealing machines, we have also seen the necessity of being able to store or load intermediate data, such that scripts can be interrupted and restarted at any stage of the transformation process. Therefore, all of the base classes are accompanied with input-output functionality. As we use the Hierarchical Data Format (HDF), stored data can quickly be retrieved and analysed by the user.

## 2.2 Usage

In this section, we briefly summarize the main steps and concepts necessary to perform the transformation with **quark**, where the highlighted terms indicate classes available in **quark**. In Figure 2 an example of a Jupyter Notebook for the usage of **quark** is shown correspondingly, to demonstrate the easy accessibility for users with a bit of experience in programming with Python. It is based on a quadratic formulation of the maximum colorable subgraph problem.

In general, the start is to implement an **Instance**, the container for the problem defining parameters, as shown in cell 2 of Figure 2. From the instance, the **ConstrainedObjective** is constructed, which is a factory processing the instance data to obtain the objective function and a collection of constraints, cf. cell 3. The latter can then be automatically transformed into the corresponding penalty objective terms, which are contained in the

```

[1]: from quark import PolyBinary, ConstraintBinary, ConstrainedObjective, ScipModel

[2]: 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

[3]: 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

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

[5]: instance = MCSInstance(edges, colors)
    constrained_objective = MCSConstrainedObjective(instance=instance)
    objective_terms = constrained_objective.get_objective_terms()
    objective = objective_terms.get_objective()
    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 +...

[6]: model = ScipModel.get_from_objective(objective)
    solution = model.solve()
    print(solution)

X_a_blue = 1    X_a_green = 0    X_a_red = 0
X_b_blue = 0    X_b_green = 1    X_b_red = 0
X_c_blue = 1    X_c_green = 0    X_c_red = 0
X_d_blue = 0    X_d_green = 0    X_d_red = 1

[7]: from dwave.samplers import SimulatedAnnealingSampler

[8]: sampler = SimulatedAnnealingSampler()
    objective_ising = objective.to_ising()
    sample = sampler.sample_ising(objective_ising.polynomial.linear_plain,
                                  objective_ising.polynomial.quadratic,
                                  num_reads=10)
    sample.first.sample

[8]: {('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'): 1}

```

Figure 2: Jupyter Notebook exemplarily showing the usage of `quark` to obtain a corresponding Ising problem from a given maximum colorable subgraph problem instance (where only the output was slightly adjusted for better readability)

`ObjectiveTerms` together with the actual objective function. The weighted sum of the objective terms forms the `Objective`, the final Ising/QUBO problem. The above steps are all executed in cell 5, starting with the instantiation of a concrete instance with the parameters defined in cell 4.

All of the mentioned objective objects as well as the constraints contain `Polynomials` representing the functions. The special polynomials `PolyBinary` and `PolyIsing` take advantage of the restriction to either binary or spin variables and add specific preprocessing or conversion methods. As apparent in the figure, also multi-index variables can be processed.

The `ScipModel`, used in cell 6, is an interface to the classical MILP solver SCIP [2] based on the Python package `PySCIPOpt` [10], which can solve a `ConstrainedObjective` or a (small enough) `Objective` for comparison. The result of an optimization is stored in the `Solution`. This includes not only the found variable assignment but also further information, like the runtime, which are obtained during the solving process. In cell 8, we then show how to solve the constructed objective with a D-Wave sampler, which works analogously to the access of an actual machine but is freely available over the D-Wave API [4].

The package `quark` contains more classes which take further specific hardware restrictions into account. They form an anchor to the other packages `complete_graph_embedding` and `weight_distribution` of our library, supporting embedded Ising problem formulations [8]. In the package `quapps`, we have assembled a library of exemplary applications, such as prime factorization or the FGA problem, with individual parameterized implementations of the `ConstrainedObjective` or the `ObjectiveTerms` based on the `Instance` definition.

## References

- [1] Francisco Barahona. “On the computational complexity of Ising spin glass models”. In: *Journal of Physics A: Mathematical and General* 15.10 (1982), pp. 3241–3253. DOI: 10.1088/0305-4470/15/10/028.
- [2] Ksenia Bestuzheva et al. *The SCIP Optimization Suite 8.0*. Technical Report. Optimization Online, Dec. 2021. URL: [http://www.optimization-online.org/DB\\_HTML/2021/12/8728.html](http://www.optimization-online.org/DB_HTML/2021/12/8728.html).
- [3] D-Wave Systems Inc. *D-Wave System Documentation*. visited 2023-06-07. URL: <https://docs.dwavesys.com/docs/latest/index.html>.
- [4] D-Wave Systems Inc. `dwave-ocean-sdk`. version 6.4.0. 2023. URL: <https://github.com/dwavesystems/dwave-ocean-sdk>.
- [5] DLR-SC. `quark`. version 0.1. 2023. DOI: <https://gitlab.com/quantum-computing-software/quark>.

- [6] Edward Farhi et al. “Quantum computation by adiabatic evolution”. In: *preprint* (2000). arXiv: 0001106v1 [quant-ph].
- [7] Fred Glover et al. “Quantum bridge analytics I: a tutorial on formulating and using QUBO models”. In: *Annals of Operations Research* 314.1 (2022), pp. 141–183. DOI: 10.1007/s10479-022-04634-2.
- [8] Elisabeth Lobe and Volker Kaibel. “Optimal Sufficient Requirements on the Embedded Ising Problem in Polynomial Time”. In: *preprint* (2023). arXiv: 2302.04162 [quant-ph].
- [9] Andrew Lucas. “Ising formulations of many NP problems”. In: *Frontiers in physics* 2.5 (2014). DOI: 10.3389/fphy.2014.00005.
- [10] Stephen Maher et al. “PySCIPOpt: Mathematical Programming in Python with the SCIP Optimization Suite”. In: *Mathematical Software – ICMS 2016*. Springer International Publishing, 2016, pp. 301–307. DOI: 10.1007/978-3-319-42432-3\_37.
- [11] Catherine C McGeoch. “Theory versus practice in annealing-based quantum computing”. In: *Theoretical Computer Science* 816 (2020), pp. 169–183. DOI: 10.1016/j.tcs.2020.01.024.
- [12] Tobias Stollenwerk, Elisabeth Lobe, and Martin Jung. “Flight Gate Assignment with a Quantum Annealer”. In: *Lecture Notes in Computer Science* 11413 (2019), pp. 99–110. DOI: 10.1007/978-3-030-14082-3\_9.