QHyper: an integration library for hybrid quantum-classical optimization

Tomasz Lamża[1,2], Justyna Zawalska[1,2], Kacper Jurek[1,2], Mariusz Sterzel[2], Katarzyna Rycerz[1,2]

[1]AGH University of Krakow, Faculty of Computer Science, Poland [2]Academic Computer Center CYFRONET AGH, Poland

QHyper is a Python library that provides a unified interface for experimenting with gate-based, quantum annealing, and classical optimization solvers. It allows users to specify combinatorial optimization problems, select solvers, manage problem hyperparameters, and standardize output for ease of use.

1. MOTIVATION

- Lack of a common programming interface for defining and solving a combinatorial problem using various optimization algorithms.
- Every time a user wants to check optimization results from a new type of solver, they are forced to program it independently.

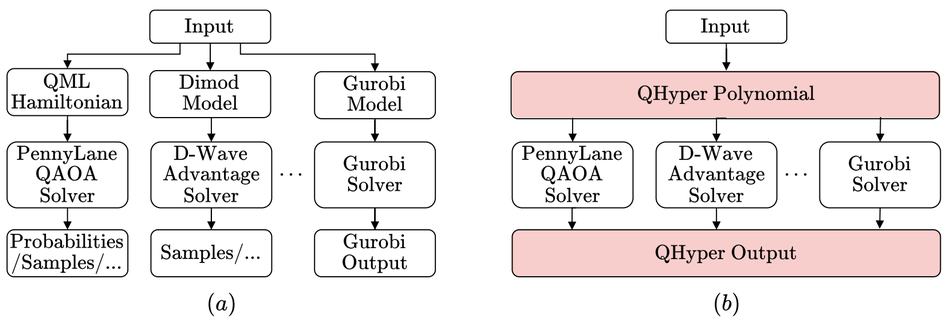

Figure 1. User effort when forced to program each optimization software (a) independently (b) and with common QHyper interface .

2. FUNCTIONAL INSIGHTS

- **Unification**: Provides a single platform for hybrid quantum-classical optimization.
- **Flexibility**: Supports diverse solvers and problem formulations.
- **Efficiency**: Simplifies experimentation and comparison of optimization techniques.
- **Extensibility**: Enables easy customization for varied research needs.
- **Accessibility**: Lowers barriers for researchers by standardizing workflows.

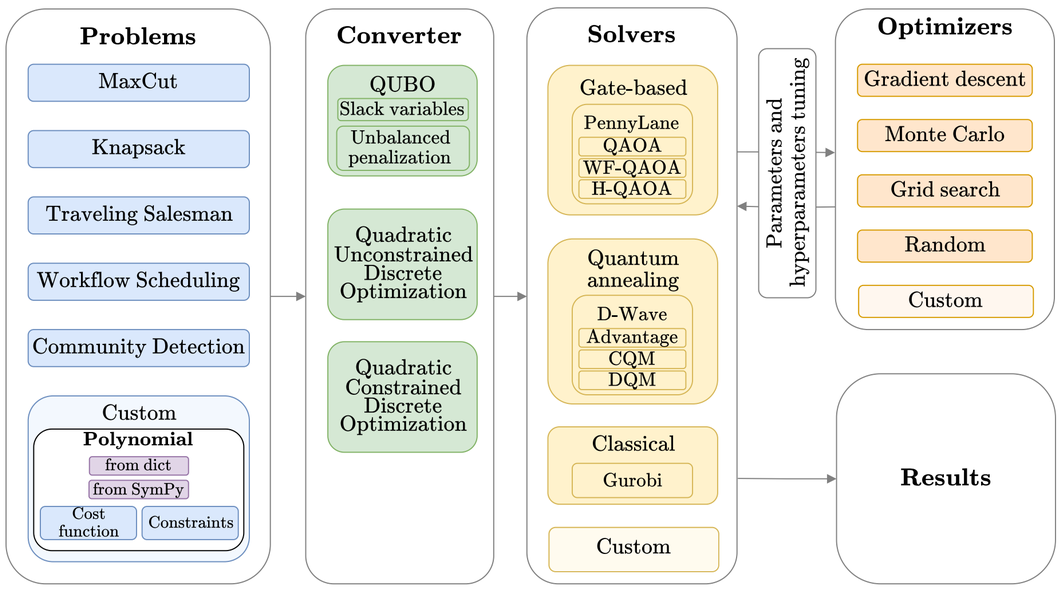

Figure 2. The main components of QHyper's architecture.

3. EVALUATION

Table 1. A comparison of the total execution times for the direct PennyLane QAOA implementation and QHyper QAOA implementation using PennyLane for the MaxCut problem.

Graph $G(N, E)$	PennyLane [s]	PennyLane +QHyper [s]
$N = 5, E = 5$	1.29 ± 0.02	1.42 ± 0.06
$N = 10, E = 25$	5.66 ± 0.07	5.8 ± 0.03
$N = 10, E = 30$	6.8 ± 0.3	6.8 ± 0.1
$N = 12, E = 66$	19.3 ± 0.4	20.0 ± 0.3
$N = 15, E = 50$	30.2 ± 0.6	30.0 ± 0.4
$N = 15, E = 105$	63 ± 1	65 ± 2

Table 2. A comparison of execution times for the direct implementation on D-Wave Advantage/Gurobi and the QHyper implementation using D-Wave Advantage/Gurobi Optimizer for the MaxCut problem.

Graph $G(N, E)$	Advantage [s]	Advantage +QHyper [s]	Gurobi [s]	Gurobi +QHyper [s]
$(5, 5)$	0.73 ± 0.07	0.8 ± 0.1	$(4 \pm 2)e-4$	$(2 \pm 6)e-2$
$(10, 25)$	0.8 ± 0.2	0.8 ± 0.2	$(3.1 \pm 0.3)e-3$	$(4.57 \pm 0.03)e-3$
$(15, 50)$	0.76 ± 0.06	0.8 ± 0.2	$(8.7 \pm 0.4)e-3$	$(8.8 \pm 0.1)e-3$
$(20, 100)$	1.0 ± 0.2	0.89 ± 0.07	$(1.845 \pm 0.006)e-2$	$(2.19 \pm 0.003)e-2$
$(25, 200)$	1.4 ± 0.5	1.4 ± 0.3	$(2.11 \pm 0.01)e-1$	$(1.9 \pm 0.2)e-1$
$(30, 400)$	2.4 ± 0.6	2.3 ± 0.4	$(1.515 \pm 0.006)e1$	$(1.521 \pm 0.008)e1$
$(35, 595)$	4 ± 1	5 ± 2	$(3.33 \pm 0.04)e-1$	$(3.96 \pm 0.02)e-1$

4. USAGE

1. Install the QHyper library

```
pip install qhyper
```

2. Import an optimization problem

```
from QHyper.problems.knapsack import KnapsackProblem

problem = KnapsackProblem(max_weight=2,
                          item_weights=[1, 1, 1],
                          item_values=[2, 2, 1])
```

3. Create a solver

3.1 Python syntax

(a)
```
from QHyper.solvers.gate_based.pennylane import QAOA
solver = QAOA(problem=CustomProblem(),
              layers=5,
              gamma=OptimizationParameter(init=[0.25]*5),
              beta=OptimizationParameter(init=[-0.5]*5),
              optimizer=QmlGradientDescent(name="adam"),
              penalty_weights=[1, 2.5, 2.5])
```

(b)
```
from QHyper.solvers.quantum_annealing.dwave import Advantage
solver = Advantage(problem=CustomProblem(),
                   penalty_weights=[1, 2.5, 2.5])
```

(c)
```
from QHyper.solvers.classical.gurobi import Gurobi
solver = Gurobi(problem=CustomProblem())
```

3.2 YAML syntax

(a)
```
problem:
  type: CustomProblem
solver:
  category: gate_based
  platform: pennylane
  name: QAOA
  layers: 5
  gamma:
    init: [0.25, 0.25, 0.25, 0.25, 0.25]
  beta:
    init: [-0.5, -0.5, -0.5, -0.5, -0.5]
  penalty_weights: [1, 2.5, 2.5]
  optimizer:
    type: QmlGradientDescent
    name: adam
```

(b)
```
problem:
  type: CustomProblem
solver:
  category: quantum_annealing
  platform: dwave
  name: Advantage
  penalty_weights: [1.0, 2,5,
                    2.5]
```

(c)
```
problem:
  type: CustomProblem
solver:
  category: classical
  platform: gurobi
  name: Gurobi
```

4. Run experiments

```
solver_results = solver.solve()
```

5. Show the results

```
from QHyper.util import sort_solver_results

sorted_results = sort_solver_results(
    solver_results.probabilities, limit_results=5)
print(sorted_results.dtype.names)
for result in sorted_results:
    print(result)
# ('x0', 'x1', 'x2', 'x3', 'x4', 'probability')
# (1, 1, 0, 0, 1, 0.24827694)
# (0, 1, 1, 0, 1, 0.18271937)
# (1, 0, 1, 0, 1, 0.18271937)
# (1, 1, 1, 0, 1, 0.15528488)
# (1, 1, 1, 1, 1, 0.03339847)
```

5. IMPORTANT LINKS

- Documentation & tutorials: `https://qhyper.readthedocs.io`
- Code: `https://github.com/qc-lab/QHyper`
- Demo: `https://codeocean.com/capsule/1274124/tree/v1`

References

[1] T. Lamża, J. Zawalska, K. Jurek, M. Sterzel, and K. Rycerz. (2024). "QHyper: An Integration Library for Hybrid Quantum-Classical Optimization." arXiv preprint. https://arxiv.org/abs/2409.15926.
[2] T. Lamża, J. Zawalska, M. Sterzel, and K. Rycerz. (2023). Software Aided Approach for Constrained Optimization Based on QAOA Modifications. In: Computational Science – ICCS 2023. https://doi.org/10.1007/978-3-031-36030-5_10.

The research presented in this paper received support from the funds assigned by Polish Ministry of Science and Technology to AGH University. We gratefully acknowledge Polish high-performance computing infrastructure PLGrid (HPC Center: ACK Cyfronet AGH) for providing computer facilities and support within computational grant no. PLG/2024/017208.