Unifying Classical and Quantum Solvers for Combinatorial Optimization:

Open Framework for Benchmarking and Algorithm Development

Jij Inc. COO Hiroshi Nakata @ MQSF

Experienced +50 Use Cases of Quantum & Optimization

Company Overview

Name	Jij Inc.
Foundation	November 2018
Staff	44
Office	Tokyo, Japan / London, UK / Hamburg, Germany (Early 2026~) + UAE

Company Overview

Consortium Member

Business Description

1. Middleware platform “JijZept” dev & sales
2. Use case development

World Class Integrated Team with
Quantum, Optimization, ML, Scientific Computation, OSS & Commercial Experts

Yu Yamashiro

- Quantum
- Optimization
- OSS

Chief Executive Officer & Founder

Kohji Nishimura

- Quantum
- Optimization
- OSS

Chief Technical Officer & Founder

Hiro Nakata

- Commercial

Chief Operating Officer & Board of Director

Ross Grassie

- Quantum
- ML

Global Technical Presales Lead

Louis Chen

- Quantum
- ML

Quantum Computing Researcher

@termoshtt

- Scientific Computation
- OSS

Tech Lead

Hiromi Ishii

- Scientific Computation
- OSS

Senior Software Engineer

Ryuji Takahashi

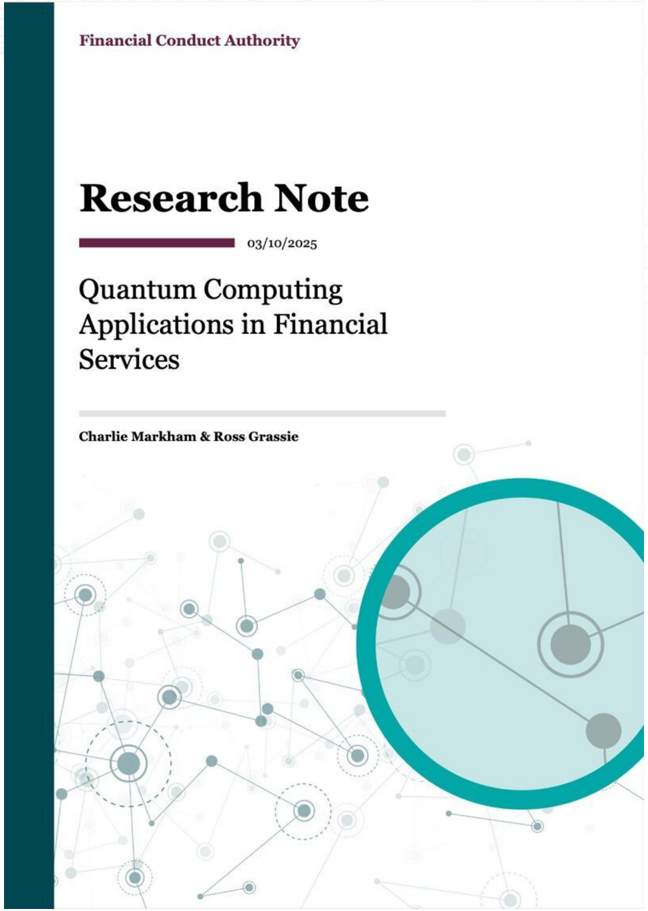
- Commercial

Head of Finance

IEEE Quantum TC Best Paper Award at QCE 2025

IEEE

Research Note for Financial Services



Financial Conduct Authority

Research Note

03/10/2025

Quantum Computing Applications in Financial Services

Charlie Markham & Ross Grassie

FCA FINANCIAL CONDUCT AUTHORITY

Optimization Workflow

Jij

Optimization Workflow

Modeling
Implementing mathematical model

Conversion / Encoding
Convert and encode model for solver

Preprocess
Variable reduction based on models

Run Optimization Algorithm
Run main algorithm performance

Post Process
Decode the solutions

Classical Optimization Software stack

Optimization Workflow	Classical Software
Modeling Implementing mathematical model	**Modeler** Pulp, JuMP, AMPL, ...
Conversion / Encoding Convert and encode model for solver	**Solver** CBC, SCIP, Gurobi, ...
Preprocess Variable reduction based on models	
Run Optimization Algorithm Run main algorithm performance	
Post Process Decode the solutions	

Quantum Optimization Software stack

Jij

Optimization Workflow	Classical Software	Quantum Software
Modeling Implementing mathematical model	**Modeler** Pulp, JuMP, AMPL, …	Write by your hand 🖊
Conversion / Encoding Convert and encode model for solver		Map your model to the Hamiltonian by your hand 🖊
Preprocess Variable reduction based on models	**Solver** CBC, SCIP, Gurobi, …	Do by your hand 🖊
Run Optimization Algorithm Run main algorithm performance		Implement algorithm with Quantum SDK by your hand **Run on Quantum Hardware** Superconducting, Ion trap, Neutral Atoms, …
Post Process Decode the solutions		Decode bitstrings by your hand 🖊

Jij Optimization Software stacks

Optimization Workflow	Jij's Optimization Software
Modeling Implementing mathematical model	JijModeling
Conversion / Encoding Convert and encode model for solver	OMMX
Preprocess Variable reduction based on models	Classical Solver (CBC, SCIP, Gurobi, …) / Qamomile / JijPresolve
Run Optimization Algorithm Run main algorithm performance	Classical Solver (CBC, SCIP, Gurobi, …) / Your favorite Quantum SDK / Run on Your favorite Quantum Hardware
Post Process Decode the solutions	Classical Solver (CBC, SCIP, Gurobi, …) / Qamomile

Jij Optimization Workflow

Jij

Jij's Optimization Software

- JijModeling
- OMMX
- Qamomile
- JijPresolve
- Your favorite Quantum SDK
- Run on Your favorite Quantum Hardware
- Qamomile

Implement your model

```python
def graph_coloring_problem() -> jm.Problem:
    # define variables
    V = jm.Placeholder("V")
    E = jm.Placeholder("E", ndim=2)
    N = jm.Placeholder("N")
    x = jm.BinaryVar("x", shape=(V, N))
    n = jm.Element("i", belong_to=(0, N))
    v = jm.Element("v", belong_to=(0, V))
    e = jm.Element("e", belong_to=E)
    # set problem
    problem = jm.Problem("Graph Coloring")
    # set one-hot constraint that each vertex has only one color

    problem += jm.Constraint("one-color", x[v, :].sum() == 1, forall=v)
    # set objective function: minimize edges whose vertices connected by edges are the same
    problem += jm.sum([n, e], x[e[0], n] * x[e[1], n])
    return problem
```

```python
interpreter = jm.Interpreter(instance_data)
instance: ommx.v1.Instance = interpreter.eval_problem(problem)
```

Problem: Graph Coloring

$$\min \quad \sum_{i=0}^{N-1} \sum_{e \in E} x_{e_0,i} \cdot x_{e_1,i}$$

{s.t.}

one-color $\quad \sum_{*_1=0}^{N-1} x_{v,*_1} = 1 \quad \forall v \in \{0, \ldots, V-1\}$

{where}

x 2-dim binary variable

Convert your model to Quantum Algorithm

```python
qaoa_converter = qm.qaoa.QAOAConverter(instance)
qaoa_converter.ising_encode(multipliers={"one-color": 5})
qaoa_circuit = qaoa_converter.get_qaoa_ansatz(p=1)
qaoa_cost = qaoa_converter.get_cost_hamiltonian()
```

Currently support
QAOA, QRAO, FQAOA

Transpile Algorithm to Quantum SDK

```python
qk_transpiler = QiskitTranspiler()
qk_circuit = qk_transpiler.transpile_circuit(qaoa_circuit)
qk_cost = qk_transpiler.transpile_hamiltonian(qaoa_cost)
```

Qiskit, CUDA-Q, QuTiP, PENNYLANE, Bloqade Analog, QURI PARTS

Analyze your result

```python
sampler = qk_pr.StatevectorSampler()
qk_circuit.measure_all()
plt.show()
job = sampler.run([(qk_circuit, result.x)], shots=10000)
job_result = job.result()
```

```python
sampleset = qaoa_converter.decode(qk_transpiler, job_result[0].data['meas'])
```

Automatically, check objective values, constraint violation

What’s next?

Algorithms

We plan to enhance available algorithms!

Write your favorite Quantum Optimization algorithm in the discord at **Qamomile channel**!

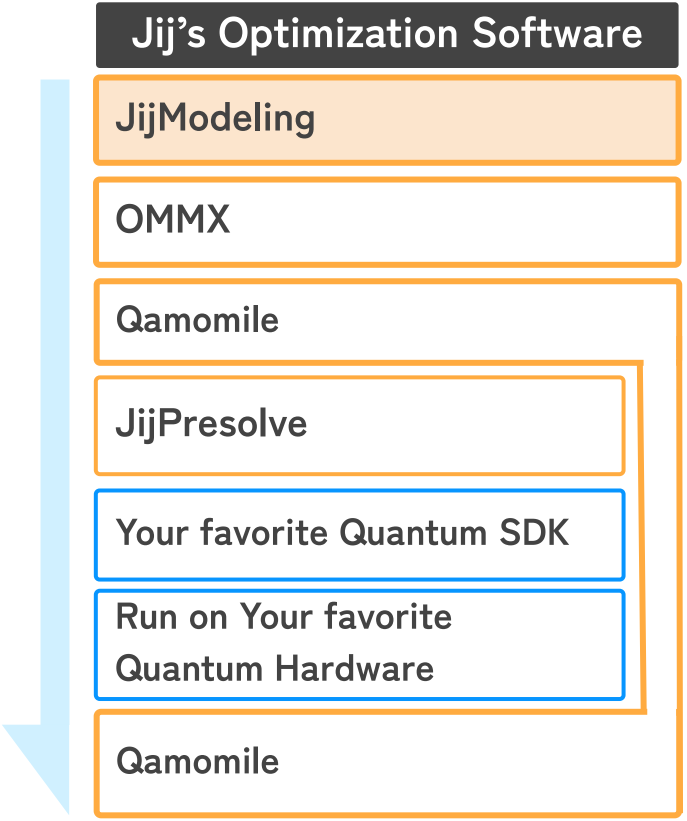
Benchmark

We are developing OMMXQuantumBenchmark.
Please check our poster!

Jij Open Community

Want to know more? Feel free to ask on Discord or stop by our poster!

Demo: QAOA for graph coloring problem



Implement your model

```python
def graph_coloring_problem() -> jm.Problem:
    # define variables
    V = jm.Placeholder("V")
    E = jm.Placeholder("E", ndim=2)
    N = jm.Placeholder("N")
    x = jm.BinaryVar("x", shape=(V, N))
    n = jm.Element("i", belong_to=(0, N))
    v = jm.Element("v", belong_to=(0, V))
    e = jm.Element("e", belong_to=E)
    # set problem
    problem = jm.Problem("Graph Coloring")
    # set one-hot constraint that each vertex has only one color

    problem += jm.Constraint("one-color", x[v, :].sum() == 1, forall=v)
    # set objective function: minimize edges whose vertices connected by edges are the same
    problem += jm.sum([n, e], x[e[0], n] * x[e[1], n])
    return problem

problem = graph_coloring_problem()
problem
```

Problem: Graph Coloring

$$\min \sum_{i=0}^{N-1} \sum_{e \in E} x_{e_0,i} \cdot x_{e_1,i}$$

{s.t.}

one-color $\quad \sum_{*_1=0}^{N-1} x_{v,*_1} = 1 \quad \forall v \in \{0, \ldots, V-1\}$

{where}

x 2-dim binary variable

Demo: QAOA for graph coloring problem

Jij

Jij’s Optimization Software

- JijModeling
- OMMX
- Qamomile
- JijPresolve
- Your favorite Quantum SDK
- Run on Your favorite Quantum Hardware
- Qamomile

Transpile your model to Ising Hamiltonian and QAOA circuit

```
interpreter = jm.Interpreter(instance_data)
instance: ommx.v1.Instance = interpreter.eval_problem(problem)
```

```
qaoa_converter = qm.qaoa.QAOAConverter(instance)
qaoa_converter.ising_encode(multipliers={"one-color": 5})
qaoa_circuit = qaoa_converter.get_qaoa_ansatz(p=1)
qaoa_cost = qaoa_converter.get_cost_hamiltonian()
```

You can transpile your model to

- QAOA
- QRAO
- FQAOA

Demo: QAOA for graph coloring problem

Jij

Jij’s Optimization Software

- JijModeling
- OMMX
- Qamomile
- JijPresolve
- Your favorite Quantum SDK
- Run on Your favorite Quantum Hardware
- Qamomile

Convert Algorithm to your favorite Quantum SDK

```
qk_transpiler = QiskitTranspiler()
qk_circuit = qk_transpiler.transpile_circuit(qaoa_circuit)
qk_cost = qk_transpiler.transpile_hamiltonian(qaoa_cost)
```

Qamomile currently supports

Qiskit CUDA-Q PENNYLANE

QURI PARTS QuTiP

Demo: QAOA for graph coloring problem

Jij

Jij's Optimization Software

- JijModeling
- OMMX
- Qamomile
- JijPresolve
- Your favorite Quantum SDK
- Run on Your favorite Quantum Hardware
- Qamomile

Decode bitstring from device to meaningful optimization results

```
sampler = qk_pr.StatevectorSampler()
qk_circuit.measure_all()
plt.show()
job = sampler.run([(qk_circuit, result.x)], shots=10000)
job_result = job.result()
```

```
sampleset = qaoa_converter.decode(qk_transpiler, job_result[0].data['meas'])
```