MRF Compiler

Converting Graphical Models to Quantum Circuits

Shyamal Suhana Chandra

MRF Compiler Project

November 17, 2025

Overview

- **Problem**: Bridge classical probabilistic models and quantum computing
- **Solution**: Automated conversion from graphical models to quantum circuits
- **Support**: 8+ major quantum computing frameworks
- **Implementation**: High-performance C++ compiler
- **Status**: Open source, actively developed

Motivation

Classical Side

- Probabilistic Graphical Models
- Bayesian Networks
- Markov Random Fields
- Inference challenges

Quantum Side

- Quantum algorithms
- Superposition & Entanglement
- Potential speedups
- Framework diversity

Challenge: Manual conversion is error-prone and time-consuming

What is MRF Compiler?

Definition

A comprehensive framework that automatically converts probabilistic graphical models (directed or undirected) into Markov Random Fields and subsequently into quantum circuit representations compatible with multiple quantum computing frameworks.

- **Input**: Text-based graphical model specification
- **Processing**: Automatic moralization, clique finding, Ising encoding
- **Output**: Framework-specific quantum circuit code

Key Features

1. **Graph Support**: Both directed and undirected graphs
2. **Automatic Conversion**: Moralization for directed graphs
3. **Clique Finding**: Efficient maximal clique identification
4. **Ising Encoding**: MRF to quantum Hamiltonian mapping
5. **Multi-Framework**: Export to 8+ quantum frameworks
6. **High Performance**: Optimized C++ implementation
7. **Easy Integration**: Simple command-line interface

Architecture

Conversion Pipeline

1. **Parse**: Read graphical model from input file
2. **Moralize**: Convert directed to undirected (if needed)
3. **Clique Finding**: Identify maximal cliques
4. **MRF Construction**: Build MRF with clique potentials
5. **Ising Encoding**: Map MRF to Ising Hamiltonian
6. **Quantum Gates**: Translate Hamiltonian to quantum gates
7. **Export**: Generate framework-specific code

Graph Moralization

Directed Graph

- $A \to B$
- $A \to C$
- $B \to D$
- $C \to D$

After Moralization

- $A - B$ (undirected)
- $A - C$ (undirected)
- $B - C$ (added edge)
- $B - D$ (undirected)
- $C - D$ (undirected)

Key Step: Connect all parents of each node to preserve conditional independence

Clique Finding

Maximal Cliques

A clique is a subset of nodes where every pair is connected. A maximal clique cannot be extended.

- **Algorithm**: Bron-Kerbosch with pivoting
- **Complexity**: Exponential worst-case, but efficient for sparse graphs
- **Output**: Set of all maximal cliques
- **Use**: Define potential functions in MRF

Ising Hamiltonian Encoding

Mapping MRF to Quantum

For binary variables, the Ising Hamiltonian is:

$$H = \sum_i h_i \sigma_i^z + \sum_{i<j} J_{ij} \sigma_i^z \sigma_j^z$$

- **Local Fields** h_i: From single-node potentials
- **Interactions** J_{ij}: From edge/clique potentials
- **Quantum Gates**: RY for local fields, CNOT+RZ for interactions

Quantum Circuit Construction

1. **Initialization**: Hadamard gates for superposition

$$H|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$$

2. **Single-Qubit**: RY gates for local field terms
3. **Two-Qubit**: CNOT + RZ for interactions
4. **Measurement**: Computational basis measurement

Result: Quantum state whose measurement probabilities match MRF distribution

Supported Frameworks

- **OpenQASM 2.0**
- **Qiskit** (IBM)
- **Cirq** (Google)
- **PennyLane** (Xanadu)
- **Q#** (Microsoft)
- **AWS Braket** (Amazon)
- **Qulacs** (Simulator)
- **TensorFlow Quantum**

Benefit: Write once, run on any framework

Usage Example

Input File (example.txt)

```
TYPE undirected
NODE 0 A 2
NODE 1 B 2
NODE 2 C 2
EDGE 0 1
EDGE 1 2
```

Command

```
./mrf_compiler -f qiskit example.txt circuit.py
```

Output

Python file with Qiskit `QuantumCircuit` ready to use

Generated Qiskit Code

```
from qiskit import QuantumCircuit

def create_circuit():
    qc = QuantumCircuit(3, 3)
    # Superposition
    qc.h(0)
    qc.h(1)
    qc.h(2)
    # Edge (0,1)
    qc.cx(0, 1)
    qc.rz(0.5, 1)
    qc.cx(0, 1)
    # Edge (1,2)
    qc.cx(1, 2)
    qc.rz(0.5, 2)
    qc.cx(1, 2)
    return qc
```

Export to All Frameworks

Single Command

```
./mrf_compiler -a example.txt
```

- Generates 8 output files simultaneously
- One file per framework
- Consistent circuit representation
- Easy framework comparison

Performance

Scalability

- **Small graphs** (2-10 nodes): Instantaneous
- **Medium graphs** (10-50 nodes): Seconds
- **Large graphs** (50-200 nodes): Minutes (depends on structure)

Optimizations

- Efficient adjacency list representation
- Pruned clique finding algorithm
- Template-based code generation

Use Cases

1. **Research**: Explore quantum algorithms for probabilistic inference
2. **Education**: Learn quantum computing through familiar graphical models
3. **Development**: Rapid prototyping of quantum-classical hybrid systems
4. **Comparison**: Test same model across different quantum frameworks
5. **Integration**: Bridge existing ML pipelines with quantum computing

Advantages

Automation

- No manual encoding
- Reduces errors
- Saves time

Flexibility

- Multiple frameworks
- Easy switching
- Framework-agnostic

Key Benefit

Focus on your problem, not on framework-specific details

Technical Details

Implementation

- **Language**: C++17 for performance
- **Graph Representation**: Adjacency lists
- **Clique Algorithm**: Bron-Kerbosch with pivoting
- **Code Generation**: Template-based exporters

Input Format

Simple text-based format, easy to generate programmatically

Example: Chain Graph

Graph Structure

- 3 nodes: A, B, C
- 2 edges: A-B, B-C
- Undirected graph

Result

- 2 maximal cliques
- 3 qubits
- 2 CNOT gates
- 2 RZ rotations

Quantum Circuit: Represents pairwise interactions in superposition

Example: Directed Graph

Original

- $A \to B$
- $A \to C$
- V-structure

After Moralization

- $A - B$
- $A - C$
- $B - C$ (added)
- Single clique

Key Point: Moralization preserves conditional independence

Integration Workflow

1. **Define Model**: Create input file with graph structure
2. **Compile**: Run MRF Compiler
3. **Import**: Use generated code in your project
4. **Execute**: Run on quantum simulator or hardware
5. **Analyze**: Process measurement results

Seamless: Works with existing quantum computing workflows

Future Directions

- **Extended Input**: Support for custom potential functions
- **Optimization**: Circuit optimization and compilation
- **More Frameworks**: Additional quantum computing platforms
- **Visualization**: Graph and circuit visualization tools
- **GUI**: Graphical user interface
- **Learning**: Automatic potential learning from data

Getting Started

Installation

```
git clone https://github.com/shyamalchandra/MRFcompiler.git
cd MRFcompiler
make
```

Quick Start

```
./mrf_compiler -f qiskit example.txt circuit.py
```

Documentation

- Research paper (13 pages)
- Reference manual (complete API)
- Examples and tutorials

Conclusion

Key Takeaways

- Automated conversion from graphical models to quantum circuits
- Support for 8+ major quantum computing frameworks
- High-performance C++ implementation
- Easy to use command-line interface
- Open source and actively developed

Impact: Enables researchers to leverage quantum computing for probabilistic inference without manual circuit design

Questions?

Thank You! **Resources:**

- GitHub: `https://github.com/shyamalchandra/MRFcompiler`
- Documentation: Complete reference manual available
- Paper: 13-page research paper with details