qSym, qMobius, qMargi, qMean and qJennings, 5 Code Generators for Generating Quantum Circuits that Perform Some Artificial Intelligence Related Tasks

Robert R. Tucci
P.O. Box 226
Bedford, MA 01730
tucci@ar-tiste.com

April 2, 2014

Abstract

This paper introduces v1.6 of qSym, qMobius, qMargi, qMean and qJennings, which are 5 Java applications available for free. (Source code included in the distribution.) The algorithms implemented by these 5 applications have been discussed in previous papers. The purpose of qSym is to calculate the symmetrized version of a given function. The purpose of qMobius is to calculate the Mobius Transform of a given function. The purpose of qMargi is to calculate the marginal probability distributions of a given probability distribution. The purpose of qMean is to calculate the mean value of a given function. The purpose of qJennings is to discover from data the structure of a classical Bayesian network.

1 Introduction

We say a unitary operator acting an array of qubits has been compiled if it has been expressed as a Sequence of Elementary Operations (SEO), where by elementary operations we mean operations that act on a few (usually 1,2 or 3) qubits at a time, such as qubit rotations and CNOTS. SEO's are often represented as quantum circuits.

This paper introduces[1] v1.6 of qSym, qMobius, qMargi, qMean and qJennings, which are 5 Java applications available for free. (Source code included in the distribution.) Henceforth in this paper, we will refer to these 5 applications collectively as the qApps. The qApps are "code generators" for generating quantum circuits.

qSym implements an algorithm discussed in Ref. [7]. Its purpose is to calculate the symmetrized version of a given function.

qMobius implements an algorithm discussed in Ref. [8]. Its purpose is to calculate the Mobius Transform of a given function.

qMargi implements an algorithm discussed in Ref. [8]. Its purpose is to calculate the marginal probability distributions of a given probability distribution.

qMean implements an algorithm discussed in Ref. [9]. Its purpose is to calculate the mean value of a given function.

qJennings implements an algorithm discussed in Ref. [10]. Its purpose is to discover from data the structure of a classical Bayesian network.

The qApps all calculate very long sums via a Grover-like algorithm. All 5 applications implement very similar algorithms. They all use a Grover-like algorithm in conjunction with 2 techniques that we first proposed in Ref.[7] and that we call "targeting 2 hypotheses" and "blind targeting".

The algorithms of the qApps could be adapted to work with the original Grover's algorithm or some variant thereof, as long as it drives a starting state $|s\rangle$ to a target state $|t\rangle$. However, the software for v1.6 of the qApps uses my preferred version of Grover's algorithm, called AFGA, discussed in Ref.[11].

Apart from their usefulness as code generators, the qApps are interesting in that they required few lines of code to write, because they rely on classes that form part of a large class library that had been written previously. This class library has been used previously to construct many other applications (for example, QuanSuite, QuSAnn, Multiplexor Expander, Quibbs and QOperAv).

In the notation of Refs.[7, 11],

$$|\langle s|t\rangle|^2 = p = |z_1|^2 + |z_0|^2 \,, \tag{1}$$

[1]The reason for releasing the first public version of qSym, qMobius, qMargi, qMean and qJennings with such an odd version number is that these 5 applications share many Java classes with other previous Java applications of mine (QuanSuite discussed in Refs.[1, 2, 3], QuSAnn discussed in Ref.[4], Multiplexor Expander discussed in Ref.[4], Quibbs discussed in Ref.[5] and QOperAv discussed in Ref.[6]), so I have made the decision to give all these applications, and the class library on which they are based, a single unified version number.

and

$$\gamma = 2 \text{ acos}(|\langle s|t\rangle|) . \tag{2}$$

Ref.[11] on AFGA gives formulas that allow one to calculate sequences $\{\gamma_j\}_{j=0}^{N_{Gro}-1}$ and $\{\hat{s}_j\}_{j=0}^{N_{Gro}-1}$ from the initial conditions $\gamma_0 = \gamma$, and $\hat{s}_0 = (\sin\gamma_0, 0, \cos\gamma_0)$. Here N_{Gro} is the number of Grover iterations (aka Grover steps). In general, p will not be known a priori. The qApps all ask that the user estimate the initial ratio $|z_1|^2/|z_0|^2$. The $|z_0|^2$ is assumed known a priori, theoretically calculated from the input parameters. The qApps calculate an estimate of p via Eq.(1), the input estimate of $|z_1|^2/|z_0|^2$ and the a priori known $|z_0|^2$. The qApps output the quantum circuit that should be used to do one "tail run" of the algorithm with a given p and N_{Gro} steps. Using the technique of blind targeting requires multiple trials to be performed, where each trial consist of a sequence of tail runs.

2 qSym

The quantum circuit generated by qSym is described in detail in Ref.[7]. The circuit allows one to calculate

$$|\langle c^n|\pi_{Sym_n}|\psi\rangle|^2 , \tag{3}$$

where $c^n \in \{0, 1, \ldots, d-1\}^n$.

2.1 Input Parameters

Using the notation of Ref.[7], the circuit depends on the following inputs:

***n*:** The number of qudits in $|\psi\rangle$.

***n*$_{sub}$:** The number of sub-qubits in each qudit. Hence, $d = 2^{n_{sub}}$.

the matrix c^n: In Ref.[7], we take $c^n = (c_0, c_1, \ldots, c_{n-1})$, where each c_j is a qudit composed of n_{sub} qubits, so $d = 2^{n_{sub}}$ and $c_j = (c_{j,0}, c_{j,1}, \ldots, c_{j,n_{sub}-1}) \in Bool^{n_{sub}}$.

a circuit that generates state $|\psi\rangle$: The state $|\psi\rangle_{\alpha^n}$ acts on n qudits, each of which has n_{sub} sub-qubits. The demonstration version of qSym uses a trivial, inconsequential circuit for $|\psi\rangle$, but this can be changed easily by subclassing the class of qSym that defines $|\psi\rangle$.

an estimate of $p = |\langle t|s\rangle|^2$

2.2 Output Files

qSym outputs 3 types of files: a Log File, an English File and a Picture File.

A Log File records all the input and output parameters displayed in the **Control Window** (see Sec.2.3), so the user won't forget them.

An English File gives an "in English" description of a quantum circuit. It completely specifies the output SEO. Each line in it represents one elementary operation, and time increases as we move downwards in the file.

A Picture File partially specifies the output SEO. It gives an ASCII picture of the quantum circuit. Each line in it represents one elementary operation, and time increases as we move downwards in the file. There is a one-to-one onto correspondence between the rows of corresponding English and Picture Files.

English and Picture Files are used in many of my previous computer programs. I've explained those files in detail in previous papers so I won't do so again here. See, for example, Ref.[5] for a detailed description of the content of those files and how to interpret that content.

2.3 Control Window

Fig.1 shows the **Control Window** for qSym. This is the main and only window of qSym (except for the occasional error or advice message window). This window is open if and only if qSym is running.

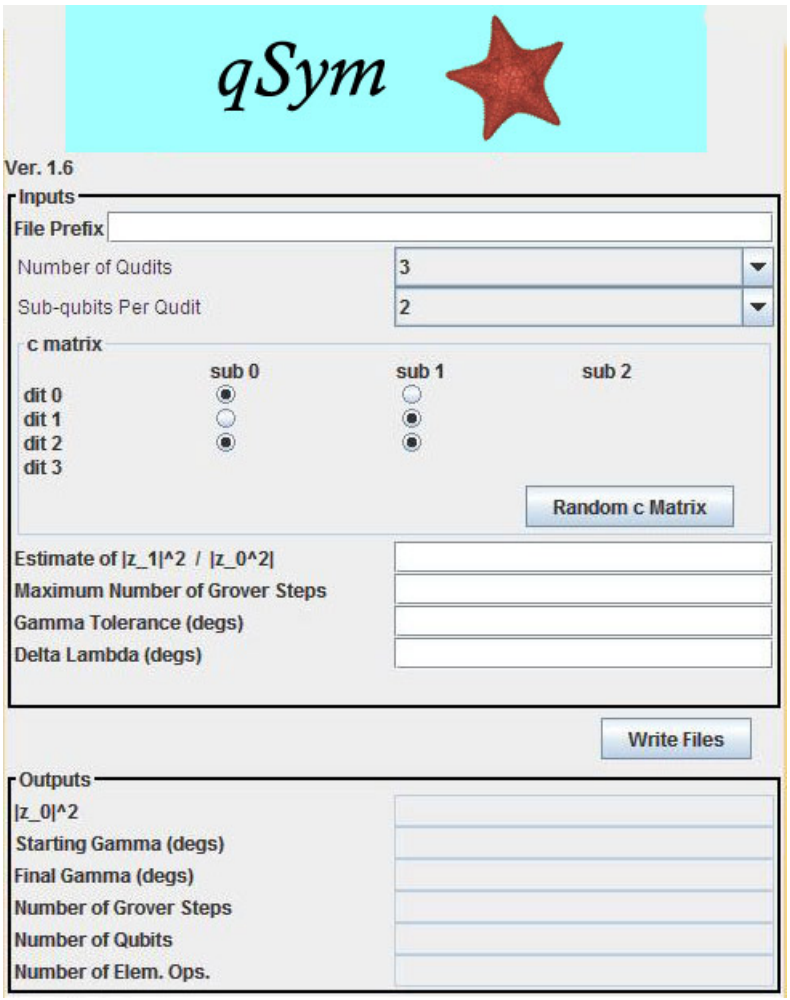

Figure 1: **Control Window** of qSym

The **Control Window** allows the user to enter the following inputs:

File Prefix: Prefix to the 3 output files that are written when the user presses the **Write Files** button. For example, if the user inserts `test` in this text field, the following 3 files will be written:

- `test_qSym_log.txt` This is a Log File.
- `test_qSym_eng.txt` This is an English File
- `test_qSym_pic.txt` This is a Picture File.

Number of Qudits: This equals n.

Sub-qubits per Qudit: This equals n_{sub}.

c matrix radio buttons: These radio buttons allow the user to specify the matrix $c^n = (c_{j,k})_{\forall j,k}$, where $j \in \{0, 1, \ldots, n-1\}$, $k \in \{0, 1, \ldots, n_{sub} - 1\}$ and $c_{j,k} \in Bool$. The j index (qudit index) of the radio buttons grows downward and is indicated by **dit 0, dit 1, dit 2, dit 3**. The k index (sub-qubit index) of the radio buttons grows towards the right and is indicated by **sub 0, sub 1, sub 2**. For demonstration purposes, this Java applet only allows a maximum number of 4 qudits and a maximum number of 3 sub-qubits. However, the applet is based on a class called `SymMain` which does not have these limitations.

The number of rows (resp., columns) of radio buttons that are visible equals the number chosen in the **Number of Qudits** menu (resp., **Sub-qubits per Qudit** menu). Pressing the **Random c Matrix** button assigns randomly chosen values to the c matrix buttons.

Estimate of |z_1|^2/|z_0|^2: This equals the user's initial estimate of $\frac{|z_1|^2}{|z_0|^2}$.

Maximum Number of Grover Steps: qSym will stop iterating the AFGA if it reaches this number of iterations.

Gamma Tolerance (degs): This is an angle given in degrees. qSym will stop iterating the AFGA if the absolute value of γ_j becomes smaller than this tolerance. (γ_j is an angle in AFGA that tends to zero as the iteration index j tends to infinity. γ_j quantifies how close the AFGA is to reaching the target state).

Delta Lambda (degs): This is the angle $\Delta\lambda$ of AFGA, given in degrees.

The **Control Window** displays the following output text boxes.

|z_0|^2: This equals $|z_0|^2$, the probability of the "null" hypothesis of the two hypotheses being targeted.

Starting Gamma (degs): This is γ_0, the first γ_j, the γ_j for the first Grover iteration, given in degrees.

Final Gamma (degs): This is the γ_j for the final Grover iteration, given in degrees.

Number of Grover Steps: This is N_{Gro}, the total number of Grover iterations that were performed. It must be smaller or equal to the **Maximum Number of Grover Steps**. It will be smaller if the **Final Gamma (degs)** reached the **Gamma Tolerance (degs)** before the **Maximum Number of Grover Steps** was reached.

Number of Qubits: This is the total number of qubits for the output quantum circuit.

Number of Elementary Operations: This is the number of elementary operations in the output quantum circuit. Since there are no LOOPs in qSym v1.6, this is the number of lines in the English File, which equals the number of lines in the Picture File.

3 qMobius

The quantum circuit generated by qMobius is described in detail in Ref.[8]. The circuit allows one to calculate

$$f(x^n) = \sum_{x^{-n} \leq x^n} f^-(x^{-n}) \; , \tag{4}$$

where $x^n, x^{-n} \in Bool^n$ and $f^-(x^{-n}) = |\langle x^{-n}|\psi^-\rangle|^2$.

3.1 Input Parameters

Using the notation of Ref.[8], the circuit depends on the following inputs:

***n*:** The number of $|\psi^-\rangle$ qubits. $x^n, x^{-n} \in Bool^n$.

the vector c^n: We desire to calculate $f(c^n)$, the Mobius transform $f(x^n)$ of $f^-(x^{-n})$ evaluated at $x^n = c^n \in Bool^n$.

a circuit that generates state $|\psi^-\rangle$: The state $|\psi^-\rangle_{\alpha^{-n}}$ acts on n qubits. The demonstration version of qMobius uses a trivial, inconsequential circuit for $|\psi^-\rangle$, but this can be changed easily by subclassing the class of qMobius that defines $|\psi^-\rangle$.

an estimate of $p = |\langle t|s\rangle|^2$

3.2 Output Files

Everything we said in Sec.2.2 also applies to the output files of qMobius.

3.3 Control Window

Fig.2 shows the **Control Window** for qMobius. This is the main and only window of qMobius (except for the occasional error or advice message window). This window is open if and only if qMobius is running.

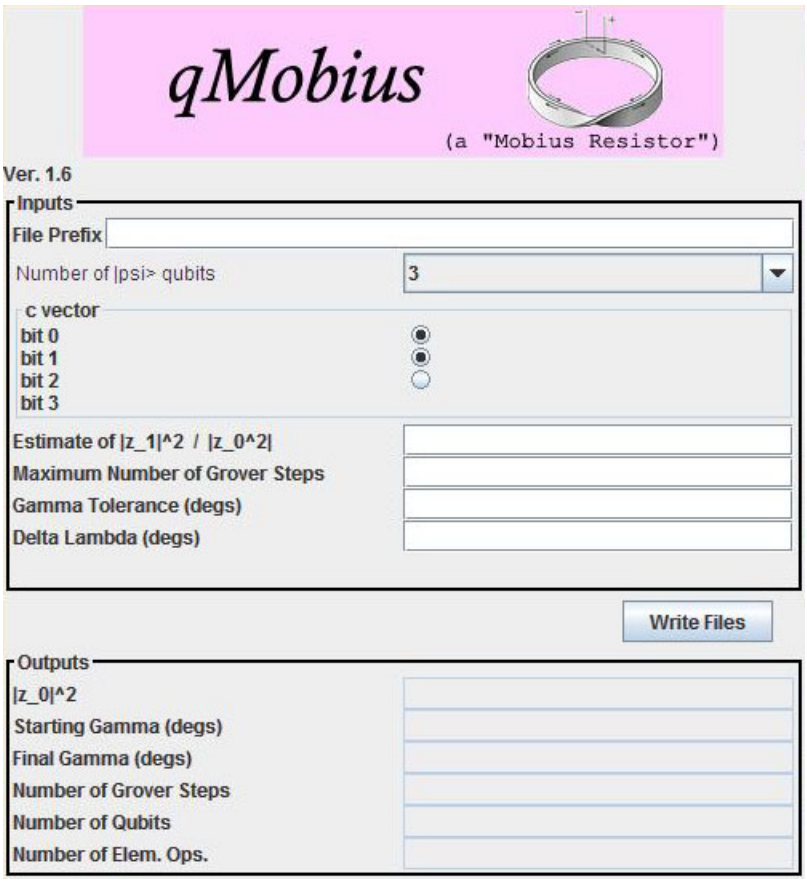

Figure 2: **Control Window** of qMobius

The **Control Window** allows the user to enter the following inputs:

File Prefix: Prefix to the 3 output files that are written when the user presses the **Write Files** button. For example, if the user inserts `test` in this text field, the following 3 files will be written:

- `test_qMob_log.txt` This is a Log File.
- `test_qMob_eng.txt` This is an English File
- `test_qMob_pic.txt` This is a Picture File.

Number of |psi > Qubits: This equals n.

c vector radio buttons: These radio buttons allow the user to specify the vector $c^n = (c_j)_{\forall j} \in Bool^n$. The j index of the radio buttons grows downward and is indicated by **bit 0, bit 1, bit 2, bit 3**. For demonstration purposes, this Java applet only allows a maximum number of 4 $|\psi\rangle$ qubits. However, the applet is based on a class called `MobMain` which does not have these limitations.

The number of rows of radio buttons that are visible equals the number chosen in the **Number of Qubits** menu.

Estimate of |z_1|^2/|z_0|^2:
Maximum Number of Grover Steps:
Gamma Tolerance (degs):
Delta Lambda (degs):
} Same as in Sec.2.3.

The **Control Window** displays the following output text boxes.

|z_0|^2:
Starting Gamma (degs):
Final Gamma (degs):
Number of Grover Steps:
Number of Qubits:
Number of Elementary Operations:
} Same as in Sec.2.3. (5)

4 qMargi

The quantum circuit generated by qMargi is described in detail in Ref.[8]. The circuit allows one to calculate

$$P(x^{n_0}) = \sum_{x^{-n}} \theta(x^{n_0} = x^{-n_0}) P(x^{-n}) \ , \tag{6}$$

where $n > n_0 > 0$, $x^{-n} = (x^{-(n-n_0)}, x^{-n_0}) \in Bool^n$, $P^-(x^{-n}) = |A^-(x^{-n})|^2$, and $A^-(x^{-n}) = \langle x^{-n}|\psi^-\rangle$.

4.1 Input Parameters

Using the notation of Ref.[8], the circuit depends on the following inputs:

***n*:** The number of $|\psi^-\rangle$ qubits. $x^{-n} \in Bool^n$.

***n*$_0$:** The number of qubits in the marginalized probability distribution. Must have $n > n_0 > 0$.

the vector c^{n_0}: We desire to calculate $\sum_{x^{-(n-n_0)}} P(x^{-n})$ evaluated at $x^{-n_0} = c^{n_0} \in Bool^{n_0}$.

a circuit that generates state $|\psi^-\rangle$: The state $|\psi^-\rangle_{\alpha^{-n}}$ acts on n qubits. The demonstration version of qMargi uses a trivial, inconsequential circuit for $|\psi^-\rangle$, but this can be changed easily by subclassing the class of qMargi that defines $|\psi^-\rangle$.

an estimate of $p = |\langle t|s\rangle|^2$

4.2 Output Files

Everything we said in Sec.2.2 also applies to the output files of qMargi.

4.3 Control Window

Fig.3 shows the **Control Window** for qMargi. This is the main and only window of qMargi (except for the occasional error or advice message window). This window is open if and only if qMargi is running.

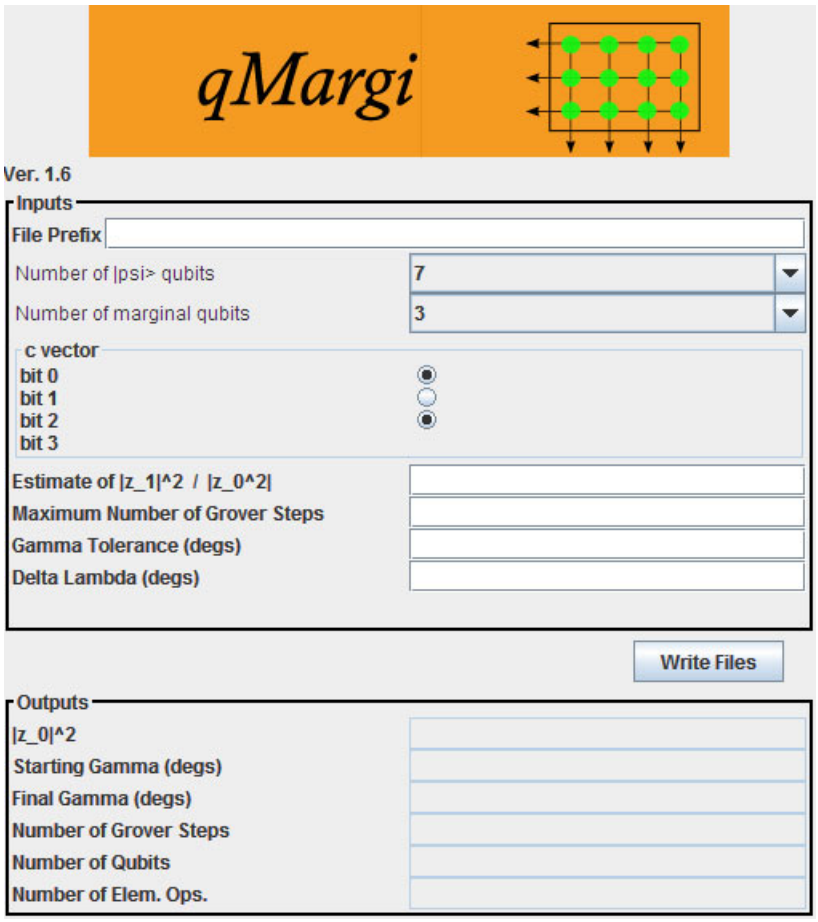

Figure 3: **Control Window** of qMargi

The **Control Window** allows the user to enter the following inputs:

File Prefix: Prefix to the 3 output files that are written when the user presses the **Write Files** button. For example, if the user inserts `test` in this text field, the following 3 files will be written:

- `test_qMargi_log.txt` This is a Log File.
- `test_qMargi_eng.txt` This is an English File
- `test_qMargi_pic.txt` This is a Picture File.

Number of |psi > qubits: This equals n.

Number of marginal qubits: This equals n_0.

c vector radio buttons: These radio buttons allow the user to specify the vector $c^{n_0} = (c_j)_{\forall j} \in Bool^{n_0}$. The j index of the radio buttons grows downward and is

indicated by **bit 0, bit 1, bit 2, bit 3**. For demonstration purposes, this Java applet only allows a maximum number of 7 $|\psi\rangle$ qubits and a maximum number of 4 marginal qubits. However, the applet is based on a class called `MargiMain` which does not have these limitations.

The number of rows of radio buttons that are visible equals the number chosen in the **Number of marginal qubits** menu.

Estimate of |z_1|^2/|z_0|^2:
Maximum Number of Grover Steps:
Gamma Tolerance (degs):
Delta Lambda (degs):
} Same as in Sec.2.3.

The **Control Window** displays the following output text boxes.

|z_0|^2:
Starting Gamma (degs):
Final Gamma (degs):
Number of Grover Steps:
Number of Qubits:
Number of Elementary Operations:
} Same as in Sec.2.3. (7)

5 qMean

The quantum circuit generated by qMean is described in detail in Ref.[9]. The circuit allows one to calculate

$$\overline{A} = \frac{1}{2^n} \sum_{x^n} A(x^n) \ , \tag{8}$$

where $A(x^n) = \langle x^n | \psi \rangle$ and $x^n \in Bool^n$.

5.1 Input Parameters

Using the notation of Ref.[9], the circuit depends on the following inputs:

***n*:** The number of $|\psi\rangle$ qubits. $x^n \in Bool^n$.

a circuit that generates state $|\psi\rangle$: The state $|\psi\rangle_{\alpha^n}$ acts on n qubits. The demonstration version of qMean uses a trivial, inconsequential circuit for $|\psi\rangle$, but this can be changed easily by subclassing the class of qMean that defines $|\psi\rangle$.

an estimate of $p = |\langle t|s\rangle|^2$

5.2 Output Files

Everything we said in Sec.2.2 also applies to the output files of qMean.

5.3 Control Window

Fig.4 shows the **Control Window** for qMean. This is the main and only window of qMean (except for the occasional error or advice message window). This window is open if and only if qMean is running.

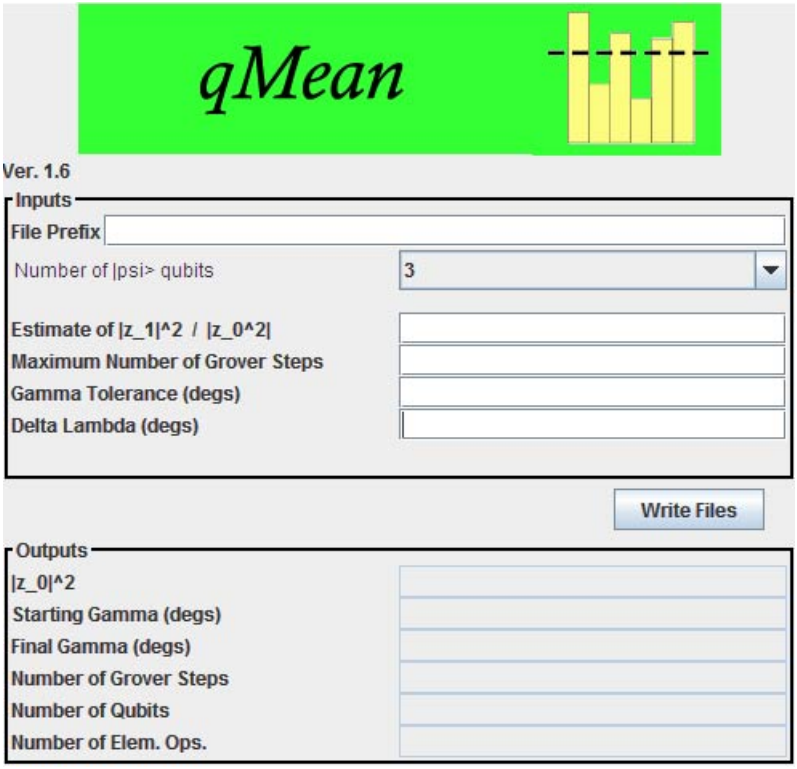

Figure 4: **Control Window** of qMean

The **Control Window** allows the user to enter the following inputs:

File Prefix: Prefix to the 3 output files that are written when the user presses the **Write Files** button. For example, if the user inserts `test` in this text field, the following 3 files will be written:

- `test_qMean_log.txt` This is a Log File.
- `test_qMean_eng.txt` This is an English File
- `test_qMean_pic.txt` This is a Picture File.

Number of |psi > Qubits: This equals n. For demonstration purposes, this Java applet only allows a maximum number of 4 $|\psi\rangle$ qubits. However, the applet is based on a class called `MeanMain` which does not have these limitations.

Estimate of |z_1|^2/|z_0|^2:
Maximum Number of Grover Steps:
Gamma Tolerance (degs):
Delta Lambda (degs):
} Same as in Sec.2.3.

The **Control Window** displays the following output text boxes.

|z_0|^2:
Starting Gamma (degs):
Final Gamma (degs):
Number of Grover Steps:
Number of Qubits:
Number of Elementary Operations:

} Same as in Sec.2.3. (9)

6 qJennings

The quantum circuit generated by qJennings is described in detail in Ref.[10]. The circuit allows one to calculate

$$\sum_{\sigma \in Sym_n} \prod_{j=0}^{n-1} h(j^\sigma | \{< j\}^\sigma) \tag{10}$$

for ordered modular models.

6.1 Input Parameters

Using the notation of Ref.[10], the circuit depends on the following inputs:

$\boldsymbol{n}$**:** The number of nodes of the graph G being discovered.

$\boldsymbol{\ell_{max}}$**:** An upper bound on the number of parents any node of G is allowed to have.

a subroutine that returns the value of $h(j|S)$ **given its arguments:** For the function $h(j|S)$, $j \in \{0..n-1\}$ and $S \subset \{0..n-1 \backslash j\}$. The demonstration version of qJennings uses a trivial, inconsequential function $h(j|S)$, but this can be changed easily by rewriting or overriding the method that defines $h(j|S)$.

an estimate of $p = |\langle t|s\rangle|^2$

6.2 Output Files

Everything we said in Sec.2.2 also applies to the output files of qJennings.

6.3 Control Window

Fig.5 shows the **Control Window** for qJennings. This is the main and only window of qJennings (except for the occasional error or advice message window). This window is open if and only if qJennings is running.

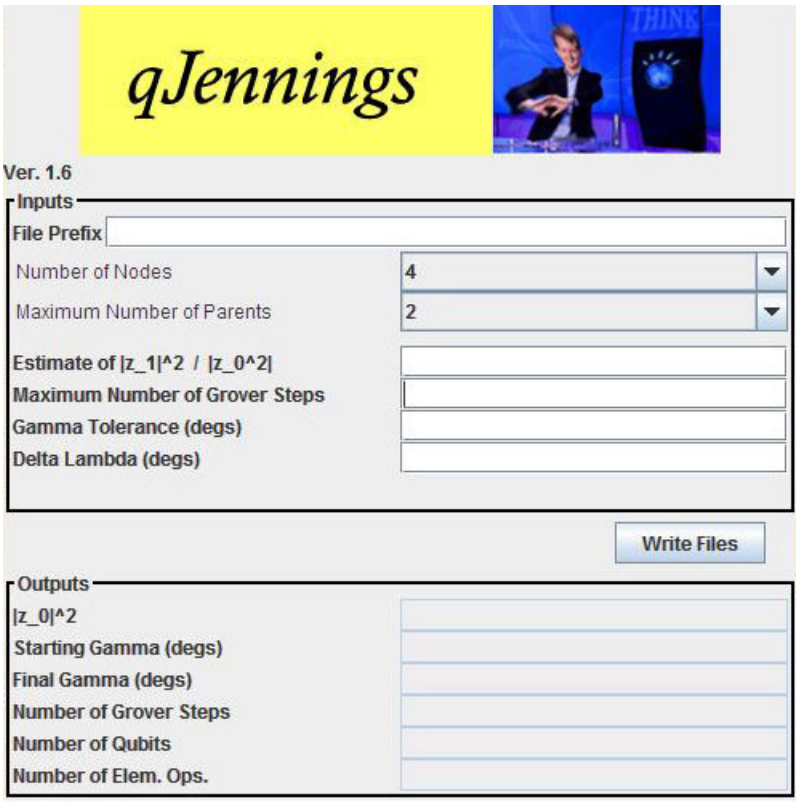

Figure 5: **Control Window** of qJennings

The **Control Window** allows the user to enter the following inputs:

File Prefix: Prefix to the 3 output files that are written when the user presses the **Write Files** button. For example, if the user inserts `test` in this text field, the following 3 files will be written:

- `test_qJen_log.txt` This is a Log File.
- `test_qJen_eng.txt` This is an English File
- `test_qJen_pic.txt` This is a Picture File.

Number of Nodes: This equals n.

Maximum Number of Parents: This equals ℓ_{max}. Must have $\ell_{max} \leq n-1$. For demonstration purposes, this Java applet only allows a maximum n of 5 and a maximum ℓ_{max} of 4. However, the applet is based on a class called `JenMain` which does not have these limitations.

Estimate of |z_1|^2/|z_0|^2:
Maximum Number of Grover Steps:
Gamma Tolerance (degs):
Delta Lambda (degs):
} Same as in Sec.2.3.

The **Control Window** displays the following output text boxes.

|z_0|^2:
Starting Gamma (degs):
Final Gamma (degs):
Number of Grover Steps:
Number of Qubits:
Number of Elementary Operations:
} Same as in Sec.2.3. (11)

References

[1] R.R. Tucci, "QuanTree and QuanLin, Two Special Purpose Quantum Compilers", arXiv:0712.3887

[2] R.R. Tucci, "QuanFou, QuanGlue, QuanOracle and QuanShi, Four Special Purpose Quantum Compilers", arXiv:0802.2367

[3] R.R. Tucci, "Java Application that Outputs Quantum Circuit for Some NAND Formula Evaluators", arXiv:0802.2370

[4] R.R. Tucci, "Code Generator for Quantum Simulated Annealing", arXiv:0908.1633

[5] R.R. Tucci, "Quibbs, a Code Generator for Quantum Gibbs Sampling", arXiv:1004.2205

[6] R.R. Tucci, "QOperAv, a Code Generator for Generating Quantum Circuits for Evaluating Certain Quantum Operator Averages", arXiv:1010.4926

[7] R.R. Tucci, "Quantum Circuit for Calculating Symmetrized Functions Via Grover-like Algorithm", arXiv:1403.6707

[8] R.R. Tucci, "Quantum Circuit for Calculating Mobius-like Transforms Via Grover-like Algorithm", arXiv:1403.6910

[9] R.R. Tucci, "Quantum Circuit for Calculating Mean Values Via Grover-like Algorithm", arXiv:1404.0668

[10] R.R. Tucci, "Quantum Circuit For Discovering from Data the Structure of Classical Bayesian Networks", arXiv:1404.0055

[11] R.R. Tucci, "An Adaptive, Fixed-Point Version of Grover's Algorithm", arXiv:1001.5200