

Simulations of quantum computations based on a mixed states model

Jarosław Miszczak

Instytut Informatyki Teoretycznej i Stosowanej PAN, Gliwice
and

Instytut Fizyki, Uniwersytet Śląski, Katowice

November 17, 2004

1. Plan of presentation

1. Quantum programming languages and simulators
2. Introduction to QCL
3. Simulations of quantum games: Parrondo paradox
4. Package quantum-octave
5. Some examples: Grover's algorithm, distance measures, teleportation

2. Why to simulate quantum computers?

1. Currently useful quantum computer devices do not exist.
2. Quantum algorithms can be executed only by the mean of simulation. Additionally it could be studied and analyzed this way.
3. Even if quantum hardware exist it won't be possible to observe quantum state. Debugging of quantum programs would be much more easier with simulator.

3. Quantum programming languages and simulators

- **QCL** – simulator + programming language
- **quantum-octave** – based on mixed states model
- Fraunhofer Quantum Computing Simulator – simulation engine (up to 31 qubits) + XML-based language for interchanging information
- other packages: QuCalc, Qubiter, Quasi2, ...

4. Introduction to QCL

QCL (Quantum Computer Language) is a language developed by Bernard Ömer. It is distributed with quantum computer simulator. It allows to mix elements from classical programming languages (numerical operators, loops, control statements, procedures) and from quantum computation model (superposition, unitary operations, controlled operations, functions acting on quantum registers).

4.1. QCL – quantum elements

qcl> qureg a[2]; – allocate two qubits of quantum memory

qcl> H(a[1]); – operations on sub-registers

qcl> CNot(a[1],a[2]); – conditional gates

qcl> qufunct, operator – functions and operators acting on quantum data types

Matrix2x2(1,0,0,-I,qureg); – complex matrices

4.2. QCL – some examples: operators

Example 1 Diffusion operator in Grover's algorithm

```
operator diffuse(qureg q) {  
    H(q);           // Hadamard Transform  
    Not(q);        // Invert q  
    CPhase(pi,q); // Rotate if q=1111..  
    !Not(q);       // undo inversion  
    !H(q);        // undo Hadamard Transform  
}
```

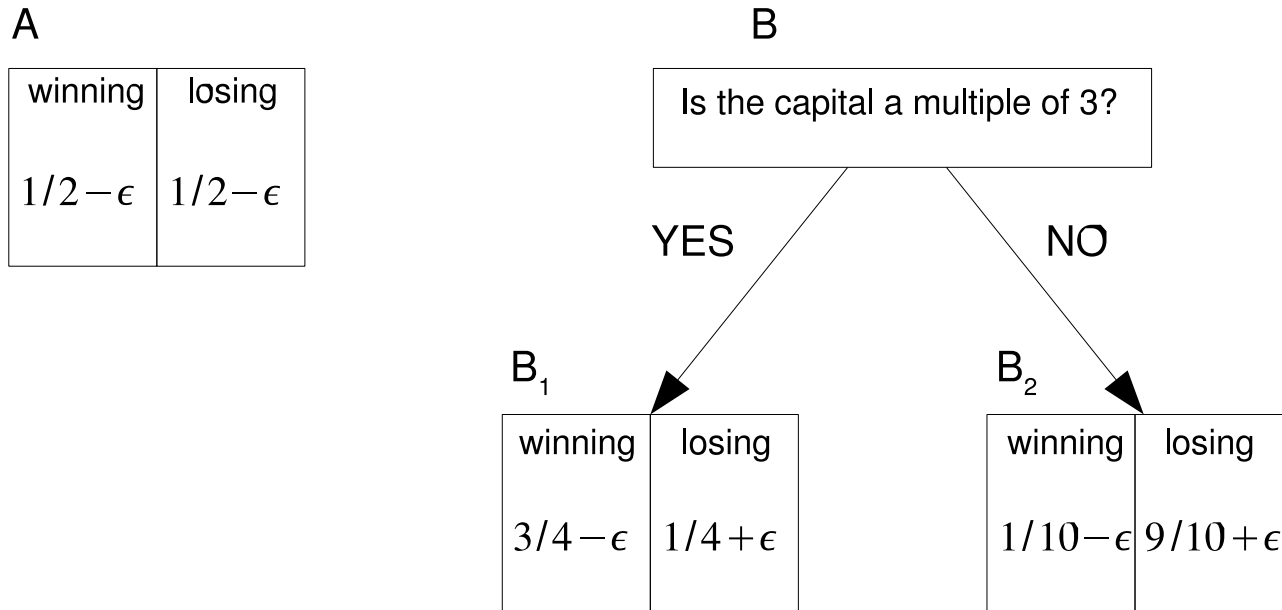
4.3. QCL – some examples: quantum conditions

Example 2 CNot as a quantum condition

```
qcl> qureg c[1];
qcl> H(c);
[1/32] 0.70711 |0> + 0.70711 |1>
qcl> qureg t[1];
qcl> if c { Not(t); }
[2/32] 0.70711 |0,0> + 0.70711 |1,1>
qcl> dump c&t;
: SPECTRUM c&t: <0,1>
0.5 |0>, 0.5 |3>
```

5. Example: Parrondo paradox

We have two games – game A and game B . Game B consists of two games B_1 and B_2 , which are executed depending on amount of collected money.



Paradox Even when games A and B are losing, one can find combination of them which is the winning one.

5.1. Simulaton of Parrondo paradox

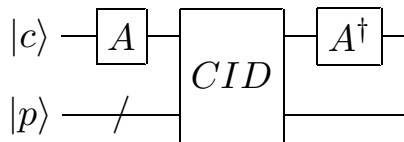
Algorithm proposed in [1]: Usage of new qubit for coin for every step of game
↳ large amount of memory needed for simulation.

Our idea: Use of one qubit for storage of coin. We need also 3 auxiliary qubits for temporary results of calculations.

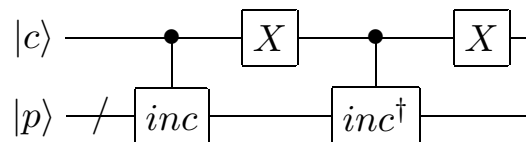
Problem: History of games is stored in coin state.

5.2. Implementation of Parrondo paradox

Game A

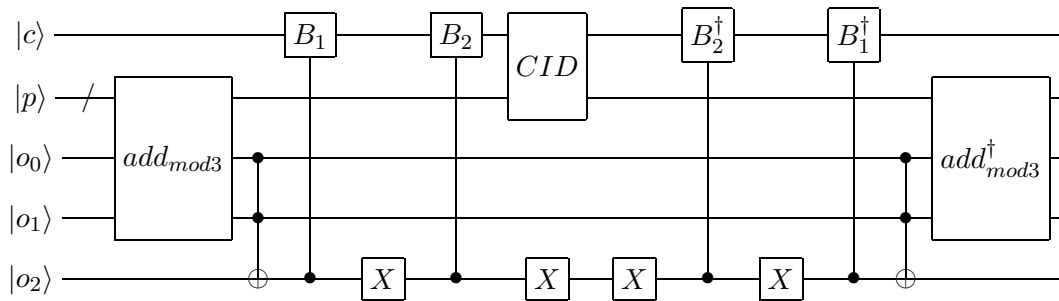


Conditional increment/decrement operation



5.3. Implementation of Parrondo paradox – continuation

Game B



5.4. QCL implementation

Game A

($\alpha, \delta, \theta, \beta$ – parameters of game)

```
operator Ai(real de, real al,
            real th, real be,
            qureg c)
{ // c - coin register
  V(de,c);
  RotZ(al,c);
  RotY(th,c);
  RotZ(be,c);
}
```

Conditional increment/decrement operation

```
operator CID(qureg p,
             quconst c)
{
  // if c==0 p-- else p++
  if (c==1) {
    inc(p);
  }
  else {
    !inc(p);
  }
}
```

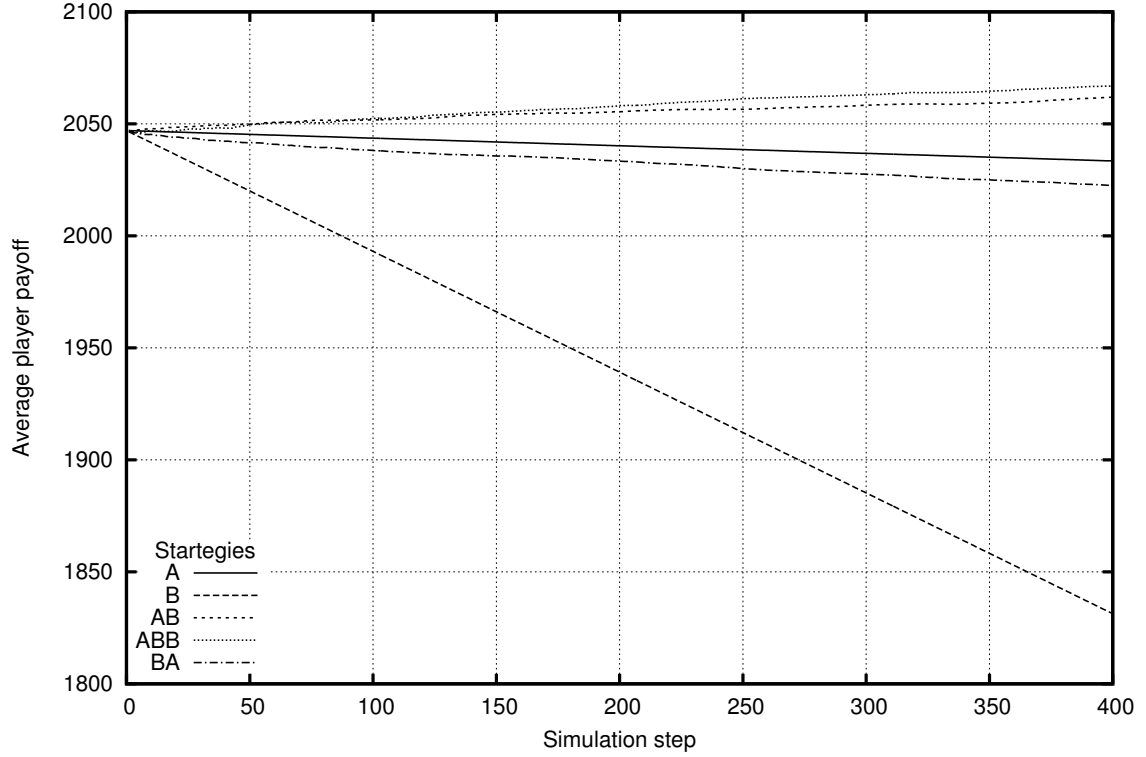
6. Parrondo paradox – results of simulations

Initial state of the coin is set to $H|0\rangle$. Each step of the simulation consists of *one execution of strategy*.

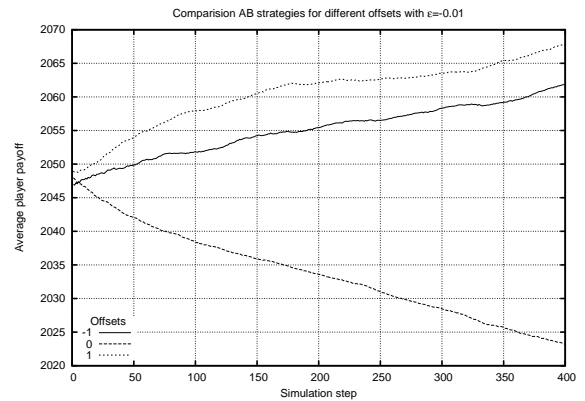
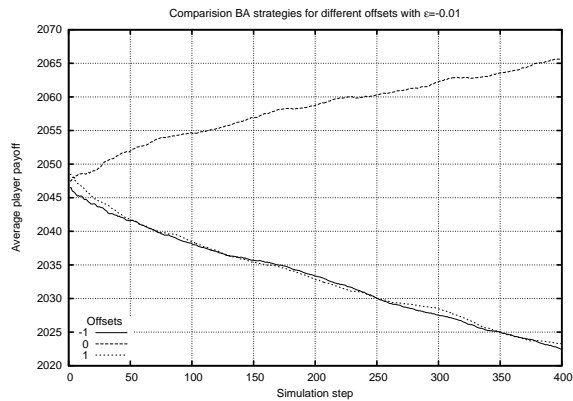
We perform simulation for different starting state of pay-off register. That fact has some influence on behavior of mean value.

We need 12 qubits to perform 400 games (in each game pay-off can be only increased/decreased by 1).

Comparison of strategies with $\epsilon=0.01$



6.2. Results of simulations – different strategies



7. Package quantum-octave

Package quantum-octave is a set of functions written in GNU Octave language. It was designed as meta-language for development of new algorithms and protocols.

Goals of the project

- Creation of the software to rapidly perform simple simulations of mixed states model.
- Enlargement of the set of the libraries of the GNU Octave package.

7.1. Motivation to create quantum-octave package

- Lack of tools which are able to operate on mixed states in an easy way.
- Usage of mixed states gives the opportunity to analyse such quantum effects as entanglement and decoherence. This is impossible in existing quantum computer simulators.

Model and simulation

- Construction of complex gates from basic ones.
- Possibility to operate on subregisters.
- Preparation of mixture of pure states.
- Unitary and non-unitary evolution and measurement.

Additional features

- Wide range of analytical functions: partial trace, negativity entropy, fidelities, distance measures.
- Possibility of graphical observation of pure states amplitudes and probability distributions of mixed states.

7.2. Basic constants

- commonly used states: Φ_{iP} , Φ_{iM} , Ψ_{iP} , Ψ_{iM} , Werner, $\text{MaxMix}(\text{dim})$
- common operators: S_x , S_y , S_z , CNot
- MeasureZ – measurement of σ_z

7.3. Basic operations

- Ket, KetN, State – build ket vector and its representation as a density operator
- MixState – prepare mixtures of states
- ProductGate – build arbitrary product gate
- ControlledGate – build controlled gate with many target and many controlled qubits
- QFT(n), Id(n)

7.4. Preparation, evolution, measurement

Preparation of states is performed using `Ket`, `KetN` or `MixStates` functions. Evolution is performed using `Evolve (gate, state)` function. It returns density matrix.

Measurement is performed using `Measure` or `MeasureZ` functions and results are returned as a structure describing state or random outcome.

All operations are acting on mixed states.

7.5. Partial operations

- PTrace – partial trace with respect to the chosen subregisters
- PTranspose – partial transposition with respect to the chosen subregisters

$$\text{PhiP} = \frac{1}{2}(|00\rangle + |11\rangle)$$

```
octave:1> PTrace(State(PhiP), [1])
```

```
ans =
```

```
0.50000  0.00000  
0.00000  0.50000
```

7.6. Entanglement measures

- only for pure states: Entropy
- for mixed states: Negativity, Concurrence

7.7. Examples of usage

- Execution of Grover's algorithm with non-pure initial state.
- Dynamics of different distance measures in Shor's and trivial algorithms.
- Influence of error to quality of teleported state.

7.8. Execution of Grover's algorithm with non-pure initial state

In this example Grover's algorithm is performed on 3 qubits with non-pure initial state, that is chosen as mixture of base states.

Examples:

```
sm1 = MixStates(0.9,s0,0.1,s1,0.1,s2,0.1,s3,0.1,s4)
```

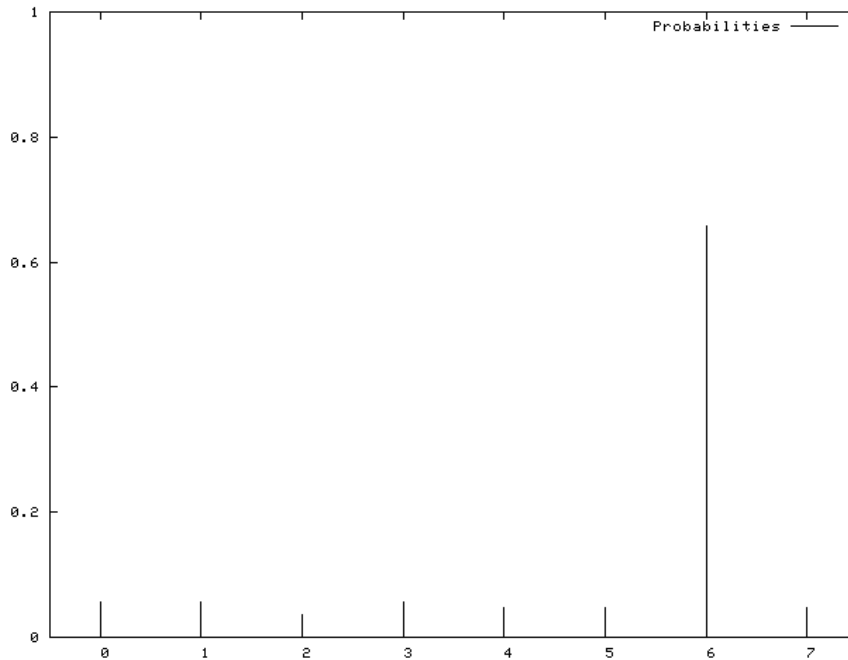
and

```
sm2 = MixStates(0.5,s0,0.2,s1,0.2,s2,0.2,s3,0.2,s4)
```

where s_0, s_1, s_2, s_4 are base states.

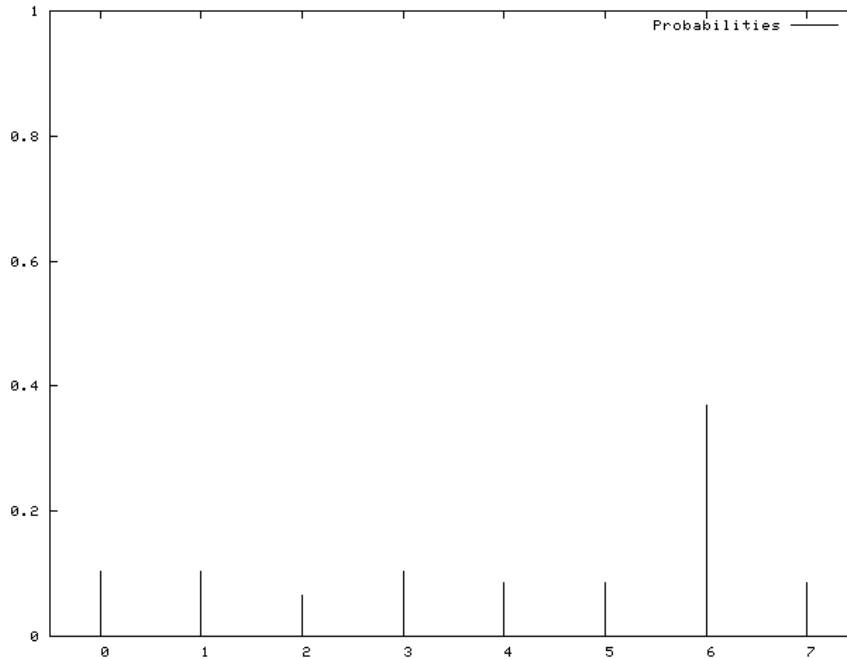
7.8.1. Distribution of the results – “almost” pure state

```
PlotProbs(grover(6, sm1));
```



7.8.2. Distribution of the results – highly mixed state

`PlotProbs(grover(6, sm2));`



7.9. Dynamics of different distance measures in Shor's and trivial algorithms

The idea is to observe how quantum state, obtained by performing quantum algorithm on non-pure state, differs from state obtained in ideal (pure state) case. On each step of simulation Initial state ρ_p is prepared as mixture of pure state $\rho_0 = |\underbrace{00 \dots 0}_d\rangle \langle \underbrace{00 \dots 0}_d|$ and maximally mixed state I_d/d

$$\rho_p = (1 - p)\rho_0 + pI_d/d, \quad (1)$$

where $p \in \{0.00, 0.01, 0.02, \dots, 1.00\}$ and I_d is d -dimensional identity matrix.

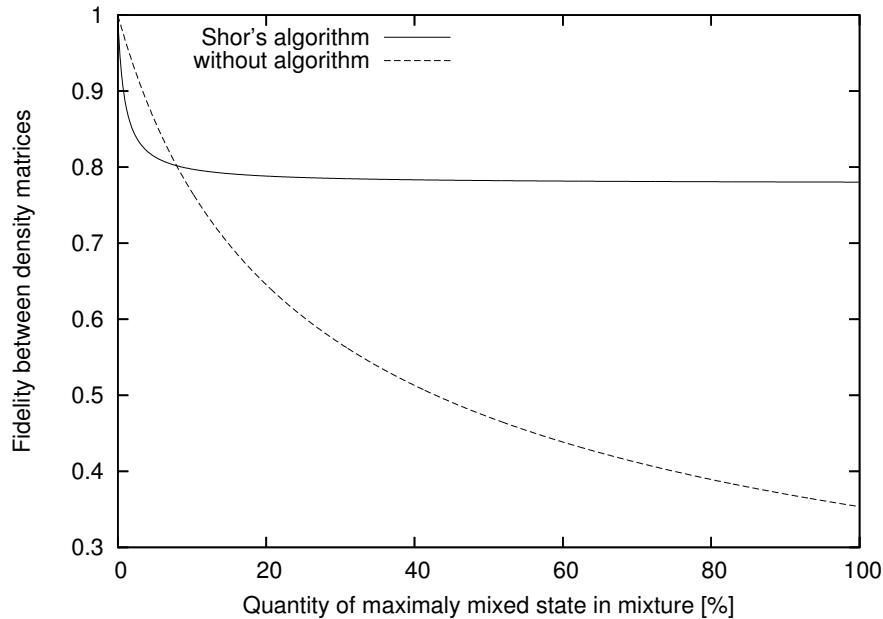
For each state obtained in such way the trivial and Shor's circuits are applied and for each obtained outcome two distances and two fidelity measures are calculated.

Shor's algorithm is performed on 7 qubits, but only the first 3 qubits are taken into account during calculations of fidelity and distance measures because the last 4 qubits decohere and normally are not measured.

To compare different methods of calculating distance between states we have chosen three functions that operate on probability distributions. Probability distributions p_1, p_2 are obtained by performing measurement of observable $\hat{Z} \otimes \hat{Z} \otimes \hat{Z}$.

7.9.1. Fidelity between density

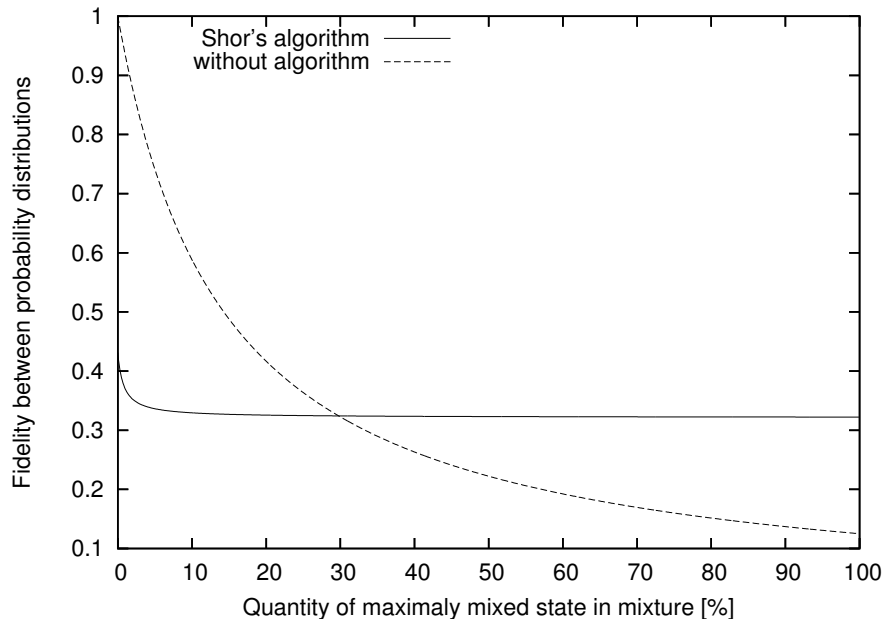
(as a function of 'purity' of state)



$$F(\rho_1, \rho_2) = \text{tr} \sqrt{\sqrt{\rho_1} \rho_2 \sqrt{\rho_1}}$$

7.9.2. Fidelity between probability distributions

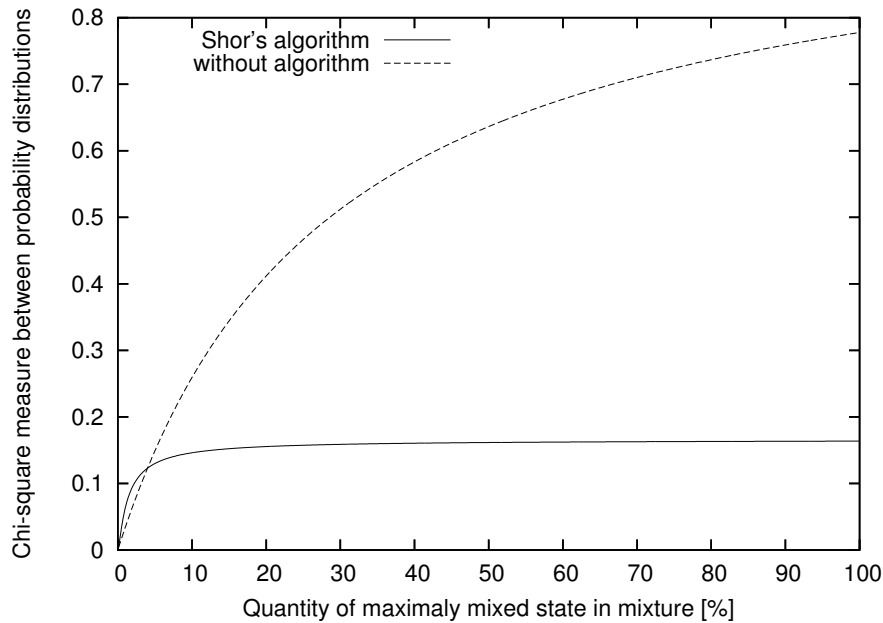
(as a function of 'purity' of state)



$$F(p_1, p_2) = \sum_{x \in X} \sqrt{p_1(x)} \sqrt{p_2(x)}$$

7.9.3. Chi-square measure

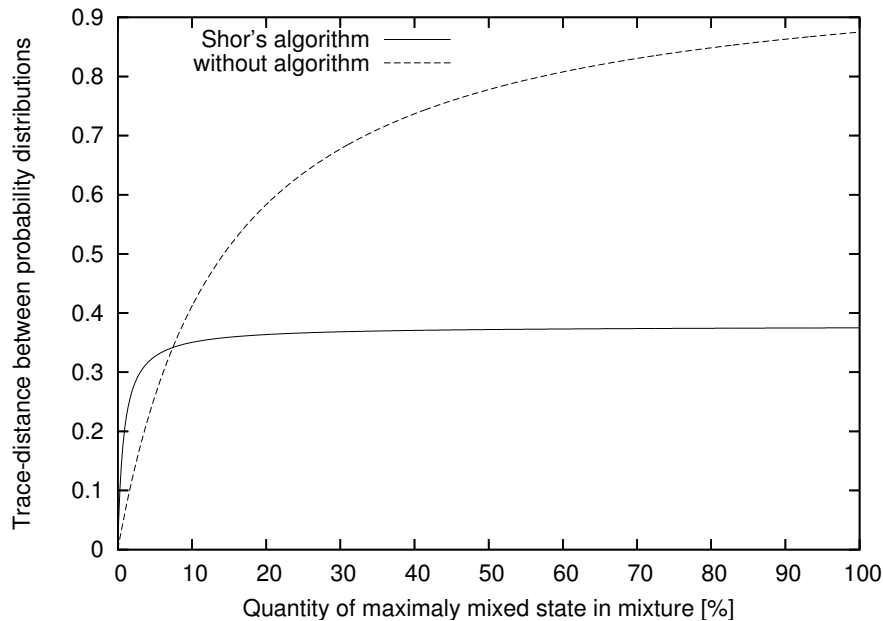
(as a function of 'purity' of state)



$$\chi^2(p_1, p_2) = \sum_{x \in X} \frac{(p_1(x) - p_2(x))^2}{p(x)}$$

7.9.4. Trace-distance

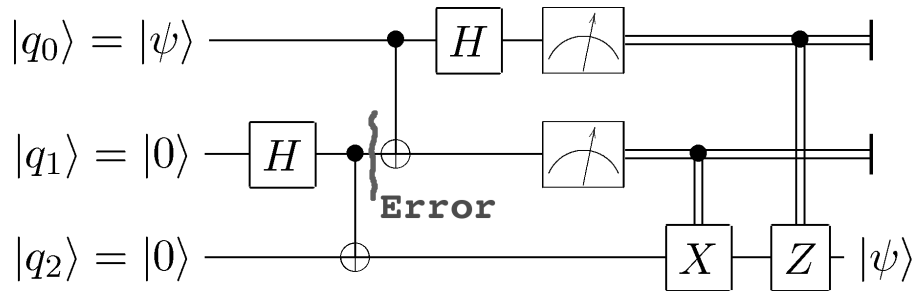
(as a function of 'purity' of state)



$$F(p_1, p_2) = \frac{1}{2} \sum_{x \in X} |p_1(x) - p_2(x)|$$

7.10. Influence of errors on quality of teleported state.

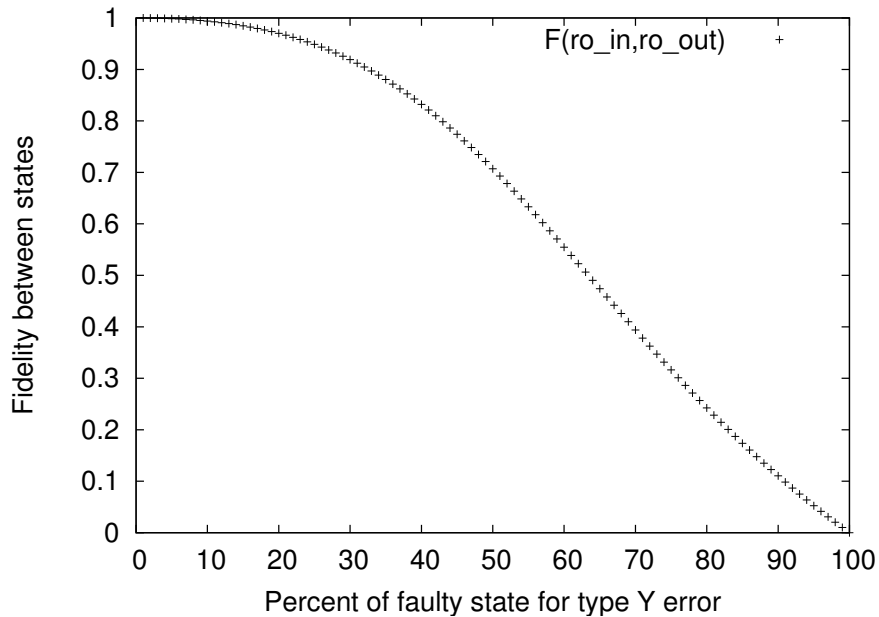
In this example we simulate influence of error on teleported state in case where initial state is pure or mixed. In our model we assume that error occurs during transmission of entangled qubit as shown below.



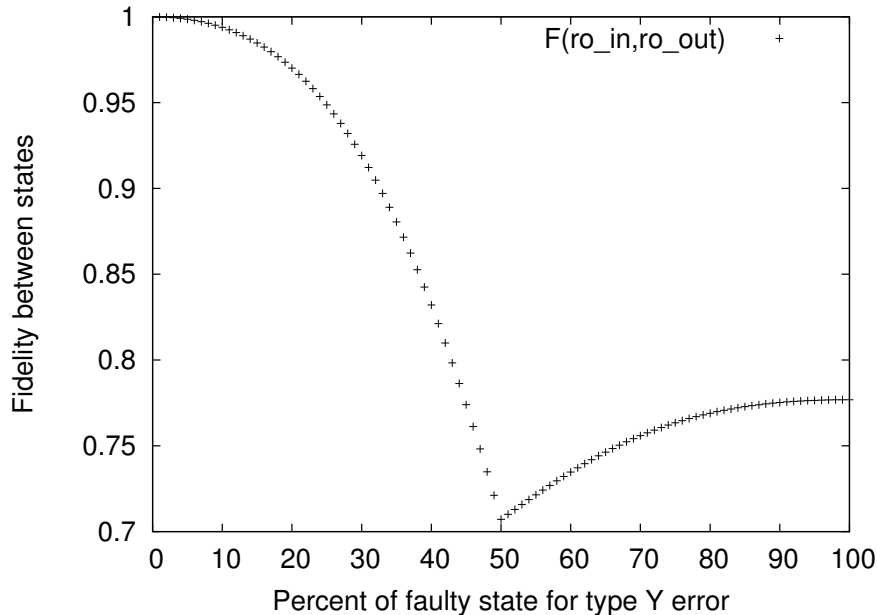
To simulate this situation we use non-unitary evolution:

$$\rho_{n+1} = \frac{E\rho_n E^\dagger}{\text{tr}(E\rho_n E^\dagger)}, \text{ with } E = pI + (p-1)\sigma_y.$$

7.10.1. Fidelity between input and output state as the function of influence of error – pure state case.



7.10.2. Fidelity between input and output state as the function of influence of error – mixed state case.



8. Conclusions

Simulations can be used for analysis and testing of algorithms and protocols. Using simulations we can observe influence of errors on algorithms, behavior of characteristic quantities like fidelity or entanglement.

9. References

References

- [1] D. Abbot, J. Ng, A. P. Flitney, *Quantum Parrondo's Games*, *Physica A*, **314** (2002), pp. 35-42.
- [2] B. Ömer. *Structured Quantum Programming*, PhD thesis, TU Vienna, 2003.
- [3] P. Gawron, J. A. Miszczak, *Simulations*, *Int. J. Quantum Information*, in press.
- [4] quantum-octave, web page <http://quantum-octave.sourceforge.net>
- [5] Fraunhofer Quantum Computing Services, web page <http://www.qc.fraunhofer.de>.

10. Who are We?

Zespół Kwantowych Systemów Informatyki, IITIS PAN



Thank you for your attention