Use of Simulated Annealing in Quantum Circuit Synthesis

Manoj Rajagopalan
17 Jun 2002

Outline
• Overview
• Simulated Annealing: the idea used
• Implementation
• Results
• Conclusions and Future Work

Quantum Circuit Synthesis
• Input-output transformation specified
• Objective: Gates and their arrangement into a circuit to achieve this transformation
• Methods:
  – Factorization <vshende> (Cybenko)
  – Enumeration <gmathew, akprasad> (exhaustive, B&B)
  – Genetic Algorithms <smaddipa> (Williams+, Yabuki+)
  – Simulated Annealing <rmanoj> ( )

Synthesis by SA
• Choose whole circuit every time
• Use equivalent transformations (optimization)
• Incremental modification (ends of circuit)
  – Computationally efficient!
  – But is it good enough?
• Hey, what is Simulated Annealing?
Optimization Problems

- State variables:
  - Reflect the system state
  - May not be directly modified
- Control variables (degrees of freedom):
  - Affect the state
  - Directly modified
- Constraints on state and control (equality/inequality)
- Objective: Solve for optimum of scalar objective function

Optimization Heuristics

- Search space is too large for exhaustive enumeration, too complex to visualize
- Pick random solutions and evaluate performance index (PI)
- Filter good/best solution(s)
- Perturb these and re-evaluate PI
- Incorporate means to avoid local minima

Simulated Annealing: Basic Idea

- Objective: minimize scalar function subject to given constraints
- Select one initial solution and evaluate cost
- Perturb the solution and calculate new cost
- Improvement in cost?
  - Yes: Copy perturbed solution to initial solution
  - No: Probabilistically accept perturbed solution (to avoid local minima)

Quantum Circuit Synthesis as an Optimization Problem

- Select type, number and location of gates (control)
- Evaluate equivalent unitary operator (state)
- Constraint: Operator == given unitary
- Objective: Minimize number of gates
Outline

- Overview
- Simulated Annealing: the idea used
  - Incremental perturbation
- Implementation
- Results
- Conclusions and Future Work

SA: Quantum Circuit Synthesis

- Assume given operator can be synthesized
- Number of qubits known for given operator
- Choose entire circuits in each perturbation?
  - How many gates?
  - What location?
  - Need to ‘multiply’ all gates to get equivalent operator each time!
- Alternative: Incremental modification (at ends of circuit each time): NOP, ADD, REM, REP
- Qubits handled independently

SA: Incremental Perturbation

```
#0
H
H
#1
T
H
#2
#3
ADD: Hadamard Gate to #1

#0
H
H
#1
T
#2
#3
ADD: Hadamard Gate to #1
```
SA: Incremental Perturbation

- Equivalent Operator =

\[
\begin{bmatrix}
1 \\
H \\
H \\
I \\
I
\end{bmatrix}
\otimes
\begin{bmatrix}
\text{Original Operator}
\end{bmatrix}
\]

SA: Incremental Perturbation

- Equivalent Operator =

\[
\begin{bmatrix}
H' \\
\otimes \\
I \\
\otimes \\
I \\
\otimes \\
I
\end{bmatrix}
\otimes
\begin{bmatrix}
\text{Original Operator}
\end{bmatrix}
\]

REMover: Hadamard Gate from #0
**SA: Incremental Perturbation**

**Replace:** T Gate on #2 with X Gate

**Equivalent Operator =**

\[
\begin{bmatrix}
1 & 0 \\
0 & 1 \\
0 & 1 \\
1 & 0
\end{bmatrix}
\times
\begin{bmatrix}
1 & 0 \\
0 & 1 \\
1 & 0 \\
1 & 0
\end{bmatrix}
\times
\begin{bmatrix}
1 & 0 \\
0 & 1 \\
0 & 1 \\
1 & 0
\end{bmatrix}
\]

**Replace:** CNOT on #1-3 with CNOT on #3-2
**SA: Incremental Perturbation**

- **Equivalent Operator:**

\[
\begin{bmatrix}
I \\
C \\
\text{NOT} \\
I
\end{bmatrix} \times
\begin{bmatrix}
I \\
H \\
I \\
I
\end{bmatrix} =
\begin{bmatrix}
\otimes \\
\otimes \\
\otimes \\
\otimes
\end{bmatrix}
\]

**Outline**

- **Overview**
- Simulated Annealing: the idea used
- **Implementation**
  - Data Structures
  - Algorithm
  - Annealer Configuration
  - Hardware and Software Platforms
- **Results**
- Conclusions and Future Work
Data Structures

- Class \textit{qgate}: Quantum gates & operators
- Qubit: \texttt{list<qgate>}
- Circuit: Array of qubits
- \texttt{qgate} instance for CNOTs duplicated on control and target qubits

SA Algorithm

- Initial circuit = empty
- Initial operator = I
- For each qubit
  - For head and tail of qubit, each
    - Choose one out of 4 moves: NOP, ADD, REM, REP in a non-conflicting manner.
    - Choose gates required for these, if applicable
  - Evaluate new operator, guarding special cases
  - Calculate (frobenius) norm of deviation from given unitary

SA Algorithm

- Out of a few (10) such moves, choose move with minimum deviation norm
- Is this below tolerance (10^{-6})?
  - Yes: Synthesis complete! Return.
  - No: Is this ‘cost’ better than that previously accepted?
    - Yes: Accept this move into circuit
    - No: Accept this move with probability \( e^{(-\Delta \text{cost} / T)} \)

SA Algorithm

- Repeat this procedure for a few (10) trials.
- Change temperature according to schedule.
- Iterate whole procedure till temperature lower limit is reached.
Annealer Configuration

- Moves:
  - 4 equiprobable changes at each end of qubit: NOP, ADD, REM, REP
- 1 or 10 moves per trial
- 1 or 10 trials per iteration
- 1001 iterations:
  - \(T_{\text{start}} = 1\)
  - \(T_{\text{end}} = 0.001\)
  - Temperature schedule = linear with step 0.001
- Objective: Simply minimize deviation norm (no circuit size reduction yet) (tolerance = \(10^{-6}\))

Platforms

- proton.eecs.umich.edu
- AMD Athlon @ 1194 MHz 256kB cache
- Debian linux (kernel v2.4.18)
- Coded in C++
- g++ 2.95.4 with –O3 optimization
- Timing and peak memory tracking using getrusage()

Outline

- Overview
- Simulated Annealing: the idea used
- Implementation
- Results
  - Single qubit circuits
  - Circuits with CNOT gates
- Conclusions and Future Work

Results: simple \(\{H,X,Z\}\) circuits

- TEST I
  - Randomly generated, 3 qubits, 30 gates
  - Optimal equivalent:
    
    \[
    \begin{array}{c}
    X \\
    X \quad Z \\
    Z
    \end{array}
    \]
Results(1/2): TEST I

<table>
<thead>
<tr>
<th>Using {H, X, Z}</th>
<th>Using {H, X}</th>
</tr>
</thead>
<tbody>
<tr>
<td># gates</td>
<td>Time(s)</td>
</tr>
<tr>
<td>Min</td>
<td>4</td>
</tr>
<tr>
<td>Max</td>
<td>22</td>
</tr>
<tr>
<td>Avg</td>
<td>9</td>
</tr>
</tbody>
</table>

97% success rate  61% success rate

Results(2/2): TEST I

<table>
<thead>
<tr>
<th>Using {H, Z}</th>
<th>Using {H, X, S}</th>
</tr>
</thead>
<tbody>
<tr>
<td># gates</td>
<td>Time(s)</td>
</tr>
<tr>
<td>Min</td>
<td>6</td>
</tr>
<tr>
<td>Max</td>
<td>47</td>
</tr>
<tr>
<td>Avg</td>
<td>10</td>
</tr>
</tbody>
</table>

17% success rate  54% success rate

Results: Simple {H, X, Z} circuits

- TEST II
  - Randomly generated
  - 5 qubits, 300 gates

Results(1/3): TEST II

<table>
<thead>
<tr>
<th>Using {H, X, Z}</th>
<th>Using {H, X, S}</th>
</tr>
</thead>
<tbody>
<tr>
<td># gates</td>
<td>Time(s)</td>
</tr>
<tr>
<td>Min</td>
<td>6</td>
</tr>
<tr>
<td>Max</td>
<td>74</td>
</tr>
<tr>
<td>Avg</td>
<td>17</td>
</tr>
</tbody>
</table>

100% success rate  82% success rate
### Results(2/3): TEST II

<table>
<thead>
<tr>
<th></th>
<th>Using {H, Z}</th>
<th></th>
<th>Using {H, X}</th>
</tr>
</thead>
<tbody>
<tr>
<td># gates</td>
<td>8</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>Time(s)</td>
<td>0.06</td>
<td>0.05</td>
<td></td>
</tr>
<tr>
<td>Min</td>
<td>8</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>Max</td>
<td>58</td>
<td>178</td>
<td></td>
</tr>
<tr>
<td>Avg</td>
<td>23</td>
<td>52</td>
<td></td>
</tr>
<tr>
<td>99% success rate</td>
<td>1.82</td>
<td>3.48</td>
<td></td>
</tr>
<tr>
<td>81% success rate</td>
<td>81% success rate</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Results(3/3): TEST II

<table>
<thead>
<tr>
<th></th>
<th>Using {H, S}</th>
<th></th>
<th>Using {H, T}</th>
</tr>
</thead>
<tbody>
<tr>
<td># gates</td>
<td>12</td>
<td>28</td>
<td></td>
</tr>
<tr>
<td>Time(s)</td>
<td>0.46</td>
<td>19.86</td>
<td></td>
</tr>
<tr>
<td>Min</td>
<td>12</td>
<td>28</td>
<td></td>
</tr>
<tr>
<td>Max</td>
<td>94</td>
<td>28</td>
<td></td>
</tr>
<tr>
<td>Avg</td>
<td>29</td>
<td>28</td>
<td></td>
</tr>
<tr>
<td>81% success rate</td>
<td>4.23</td>
<td>19.86</td>
<td></td>
</tr>
<tr>
<td>1 % success rate</td>
<td>28</td>
<td>19.86</td>
<td></td>
</tr>
</tbody>
</table>

### {H, X, Z}: Conclusions
- Easily synthesized
- Optimal equivalents detected
- Average number of gates is impressive!
- Versatility of annealer: wide variety of gate libraries
- Fast!

### Results: Circuits with CNOTs
- Brassard’s teleportation circuit
  - Careful with the gates (especially S and T)!

---

Sender

Receiver
### Results (1/3): Send Circuit

<table>
<thead>
<tr>
<th>Using {L, CNOT, R}</th>
<th>Using {CNOT, H, \text{S}_\text{NC}}</th>
</tr>
</thead>
<tbody>
<tr>
<td># gates</td>
<td>Time(s)</td>
</tr>
<tr>
<td>Min</td>
<td>4</td>
</tr>
<tr>
<td>Max</td>
<td>213</td>
</tr>
<tr>
<td>Avg</td>
<td>53</td>
</tr>
<tr>
<td>95 % success rate</td>
<td>8 % success rate</td>
</tr>
</tbody>
</table>

### Results (2/3): Send Circuit

<table>
<thead>
<tr>
<th>Using {CNOT, H, X}</th>
<th>Using {CNOT, H, Z}</th>
</tr>
</thead>
<tbody>
<tr>
<td># gates</td>
<td>Time(s)</td>
</tr>
<tr>
<td>Min</td>
<td>20</td>
</tr>
<tr>
<td>Max</td>
<td>301</td>
</tr>
<tr>
<td>Avg</td>
<td>125</td>
</tr>
<tr>
<td>51 % success rate</td>
<td>45 % success rate</td>
</tr>
</tbody>
</table>

### Results (3/3): Send Circuit

<table>
<thead>
<tr>
<th>Using {CNOT, H, X, \text{S}<em>\text{NC}, \text{T}</em>\text{NC}}</th>
<th>Using {R, S, T, L, X, CNOT}</th>
</tr>
</thead>
<tbody>
<tr>
<td># gates</td>
<td>Time(s)</td>
</tr>
<tr>
<td>Min</td>
<td>-</td>
</tr>
<tr>
<td>Max</td>
<td>-</td>
</tr>
<tr>
<td>Avg</td>
<td>-</td>
</tr>
<tr>
<td>0 % success rate</td>
<td>? % success rate</td>
</tr>
</tbody>
</table>

### Send Circuit: Conclusions

- Difficult to synthesize with overspecified gate library
- Using 10 trials per iteration instead of the usual 1 improves chances of getting an equivalent circuit and may even detect optimal one but takes much more time and may show worse average performance
Results (1/2): Receiver Circuit

<table>
<thead>
<tr>
<th>Using {S, CNOT, T}</th>
<th>Using {CNOT, H, S_{NC}}</th>
</tr>
</thead>
<tbody>
<tr>
<td># gates</td>
<td>Time(s)</td>
</tr>
<tr>
<td>Min</td>
<td>4</td>
</tr>
<tr>
<td>Max</td>
<td>20</td>
</tr>
<tr>
<td>Avg</td>
<td>5</td>
</tr>
</tbody>
</table>

100% success rate
83% find optimum

100% success rate
68% find optimum

Results (2/2): Receiver Circuit

<table>
<thead>
<tr>
<th>Using {S_{NC}, CNOT, T_{NC}, X, H}</th>
<th>Using {R, S, T, L, X, CNOT}</th>
</tr>
</thead>
<tbody>
<tr>
<td># gates</td>
<td>Time(s)</td>
</tr>
<tr>
<td>Min</td>
<td>3</td>
</tr>
<tr>
<td>Max</td>
<td>16</td>
</tr>
<tr>
<td>Avg</td>
<td>5</td>
</tr>
</tbody>
</table>

100% success rate
44% find optimum

Teleportation Circuit: Previous Work

- Williams and Gray
  - Objective: minimize discrepancy, sum of absolute value of matrix of differences
  - Send and receive minimal circuits have 4 gates each. Achieved. 3 using N&C gates
- Yabuki and Iba
  - Receive circuit needs minimum of 3 gates. We get 4 using given library and 3 using N&C gates

Receiver Circuit: Conclusions

- A lot easier to synthesize than the send circuit
- Variety of gate libraries can be used
- Overspecified gate library not a problem
- Annealer finds optimum quite often.
Outline

- Overview
- Simulated Annealing: the idea used
- Implementation
- Results
- Conclusions and Future Work

Overall Conclusions

- Annealer is versatile over a range of discrete gate sets using a very simple configuration
- Incremental perturbation works very well (Igor rules!)

Future Work

- Optimizations
  - Compose single qubit gates into one operator

- Ideas
  - Optimal synthesis: include # gates in PI
  - Circuits equivalent upto a nonzero global phase
  - Annealer configurations (types and probabilities of moves, temperature schedule) based on plots of quality of solution against iterations
  - More challenging problems: Toffoli gates (increased interqubit interaction), single qubit rotations (continuous values)