

Compiling Quantum Algorithms: Grover's Algorithm Realized

Main Paper:

From Boolean functions to quantum circuits:
A scalable quantum compilation flow *

Ismael Barzani:

ismael.ridha@mail.concordia.ca

ismael.talib.ridha.barzani@umontreal.ca

OVERVIEW

1. A problem is defined and casted to available quantum algorithms.
2. A quantum algorithm can be written in the form of multiple unitary transformations.
3. Logic circuit synthesis and optimization methods are then applied to obtain logic circuits.
4. An optimized logic circuit is then mapped and routed to qubits to run on actual architecture.

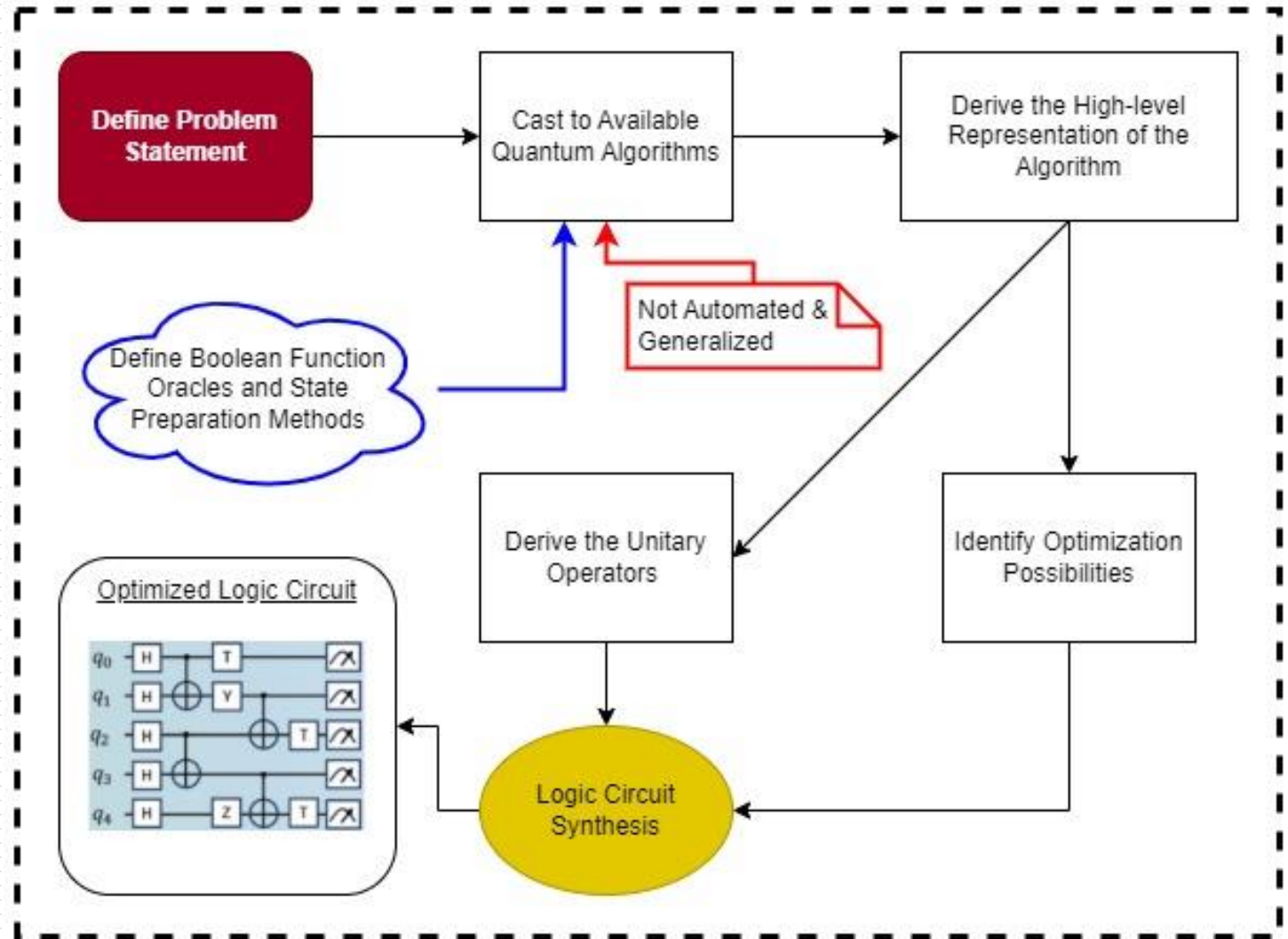


Fig. 1. Implementation Pipeline for Applying a Quantum Algorithm

VERTEX COLORING PROBLEM

OBJECTIVE:

Assign colors to vertices of a graph such that adjacent vertices are not of the same color.

- **For Example:** Vertices 0, 2, and 3 (v_0, v_2 , and v_3) cannot have the color **ORANGE** (A).

ZED CITY PROBLEM:

Assume Zed city is a new municipality in Japan with 4 convenience store chains **A**, **B**, **C**, and **D**.

- Zed city as an undirected graph
- Each district represented by a node,
- Two districts are adjacent if there is an edge.

Goal: Use vertex coloring to assign stores to districts without one, ensuring one store per district and no adjacent districts share a chain.

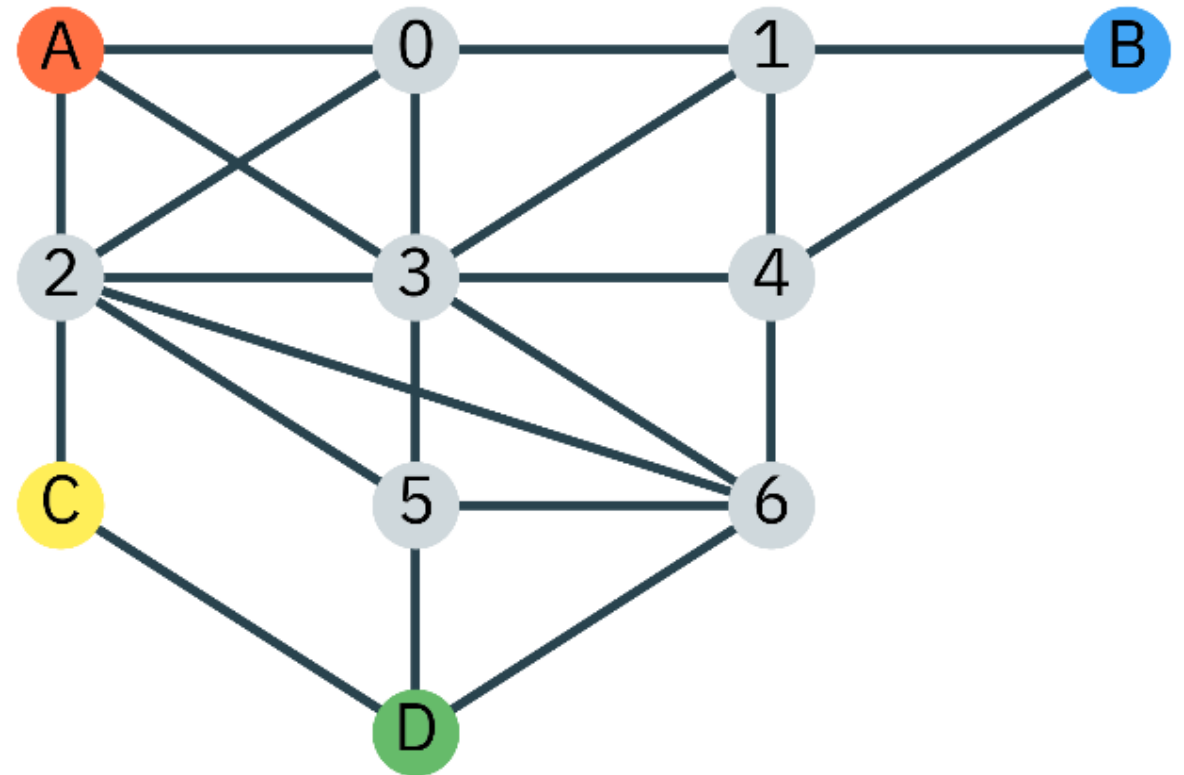
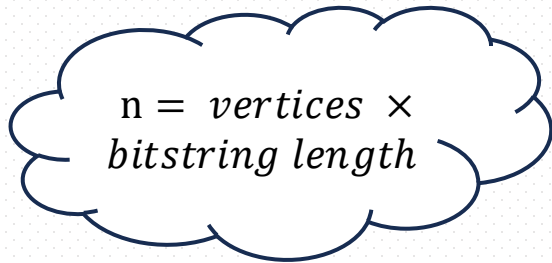


Fig. 2. An Instance of Graph Vertex Coloring; IBM's *ZED city problem challenge**

CASTING TO SEARCHING ALGORITHMS

Classical Algorithms: **Brute-Forcing**



Any classical algorithm will require $O(2^n)$ queries to perform the task.

Quantum Algorithms: **Grover's Search**

Using quantum mechanics, a database of unsorted data can be searched quadratically faster than any classical search [L. K. Grover].

→ **Grover's algorithm requires $O(\sqrt{2^n})$ queries to perform the task.**

- The basic idea of Grover's algorithm is to invert the phase of the desired basis state, and then invert all the basis states about the average amplitude of all the states.
- The algorithm uses $n + 1$ qubits.
- The optimal number of iterations is $\left\lfloor \frac{\pi}{4} \sqrt{2^n} \right\rfloor$ iterations

GROVER'S ALGORITHM FOR ZED CITY

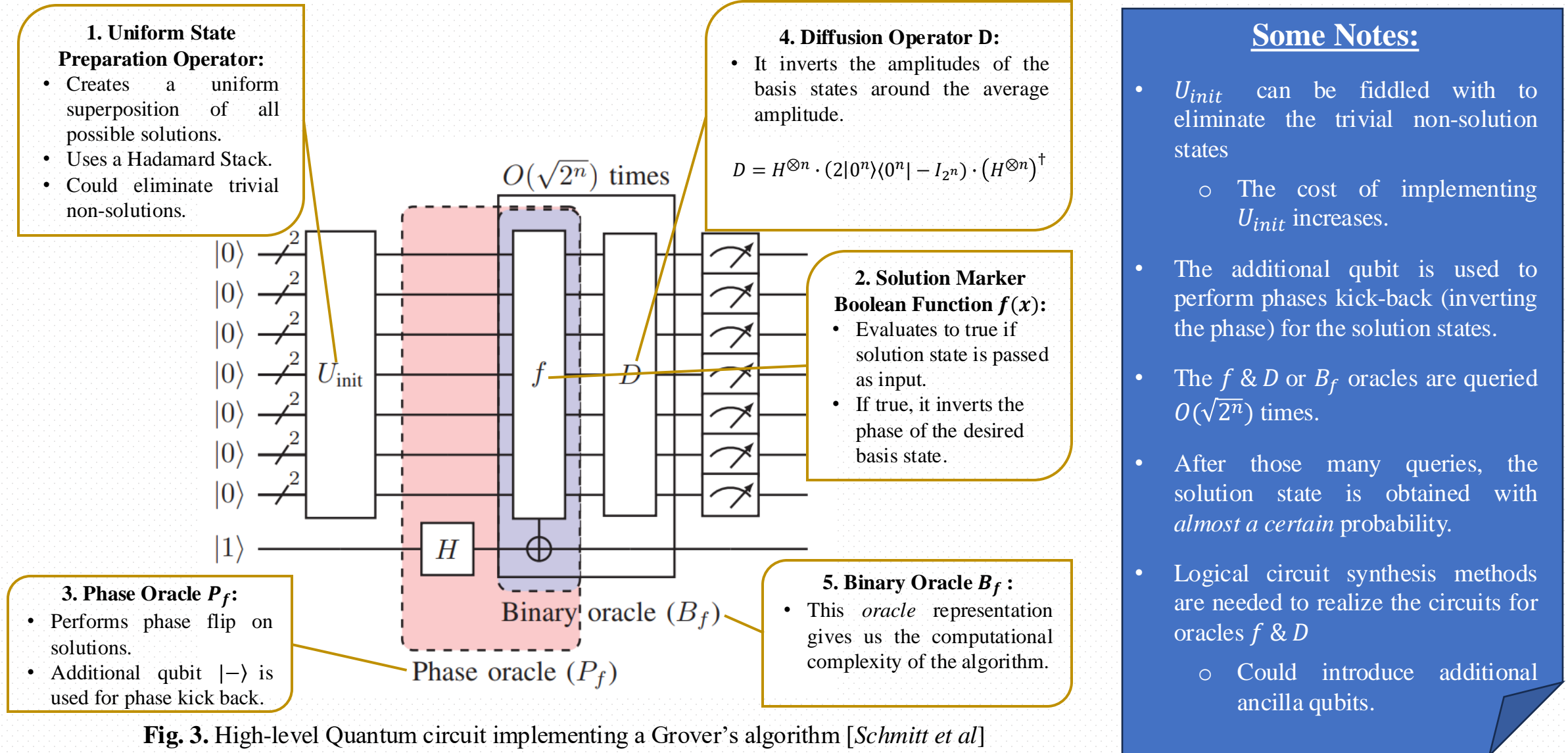


Fig. 3. High-level Quantum circuit implementing a Grover's algorithm [Schmitt et al]

GROVER'S SEARCH ALGORITHM DEMO

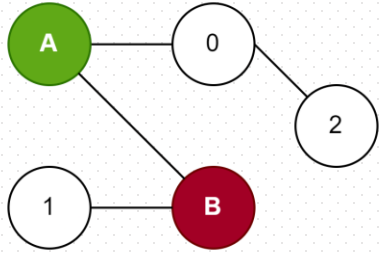


Fig. 4. Scaled-down Vertex Coloring Problem

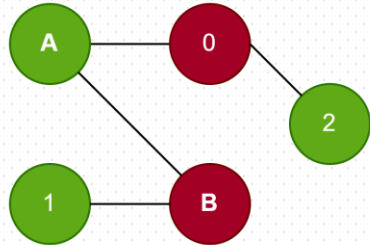
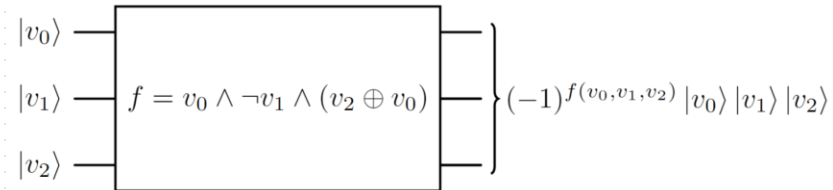


Fig. 5. Solution by observation

- We want to color vertices v_0, v_1 , and v_2 such that they don't have the same color as their neighbors.
- Let color *green* = '0' and *red* = '1'.
- We can concatenate the vertices as $|v_1\rangle|v_2\rangle|v_3\rangle$ or $|v_1v_2v_3\rangle$ to run it through Grover's circuit.
- List of possible solutions: $2^3 = 8$; This also means it will take 8 queries if classical method is used.
- We assume that Grover's algorithm will take $\left\lceil \frac{\pi}{4} \sqrt{8} \right\rceil = 2$ queries (*iterations*) to get the solution:
 - This means the depth of our Grover's circuit should be no more than 2 Binary oracles
- Let's define a Boolean function f satisfying the vertex coloring:

$$\begin{aligned} f(v_0, v_1, v_2) &= (v_0 \neq 0) \wedge (v_1 \neq 1) \wedge (v_2 \neq v_0) \\ &= v_0 \wedge \neg v_1 \wedge (v_2 \oplus v_0) \end{aligned}$$

- We will use the above Boolean function as Boolean oracle f in our circuit as follows:



GROVER'S SEARCH ALGORITHM DEMO (*cont.*)

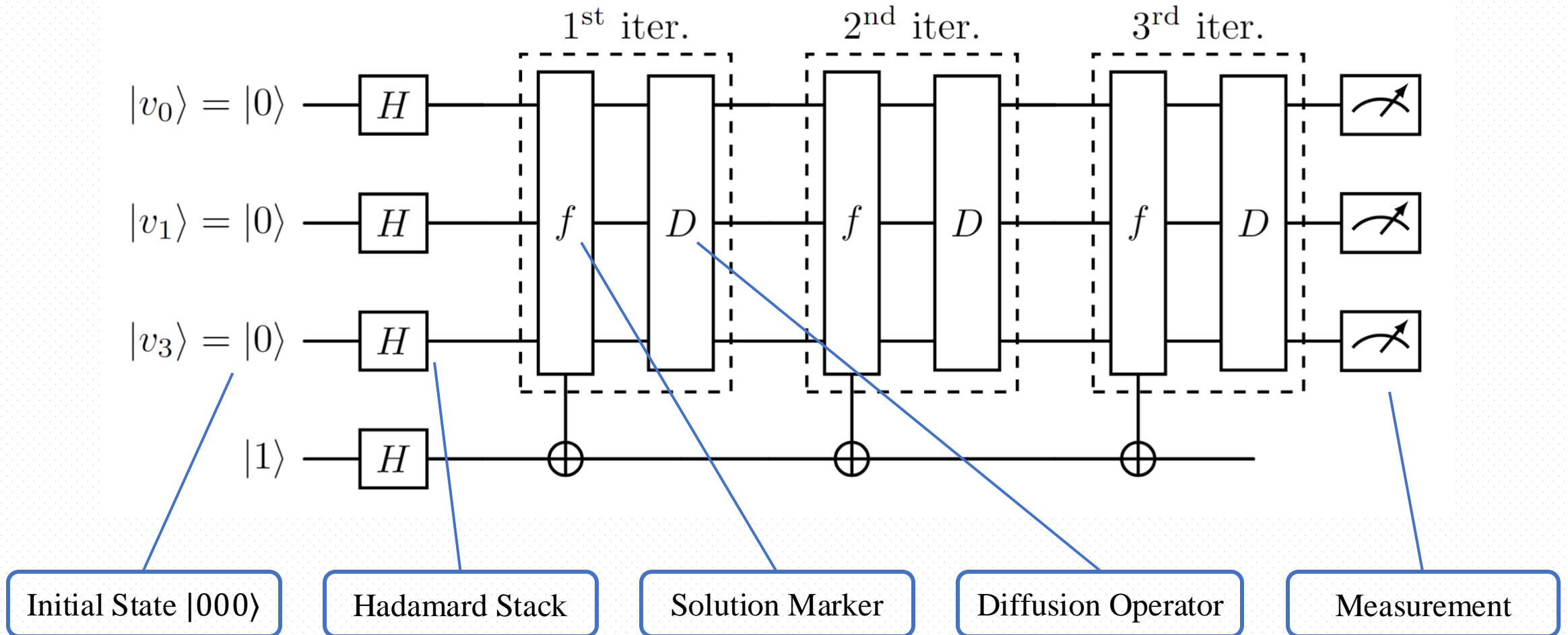


Fig. 6. Derived Grover's Circuit for the Vertex Coloring Instance of 3 vertices for the Graph in Fig. 4.

GROVER'S SEARCH ALGORITHM DEMO (*cont.*)

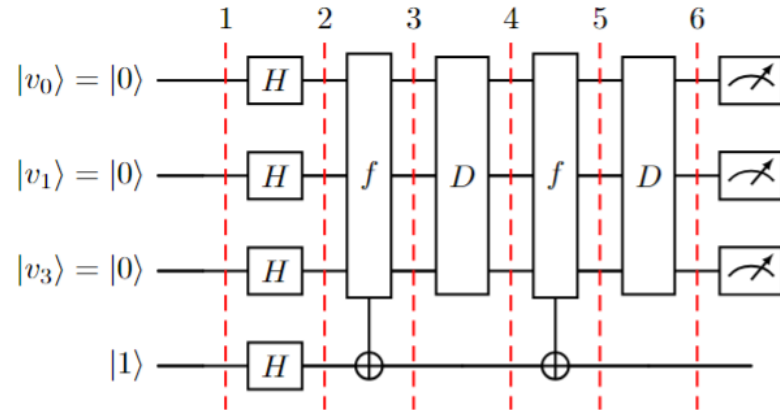


Fig. 7. Grover's circuit after performing 2 iterations.

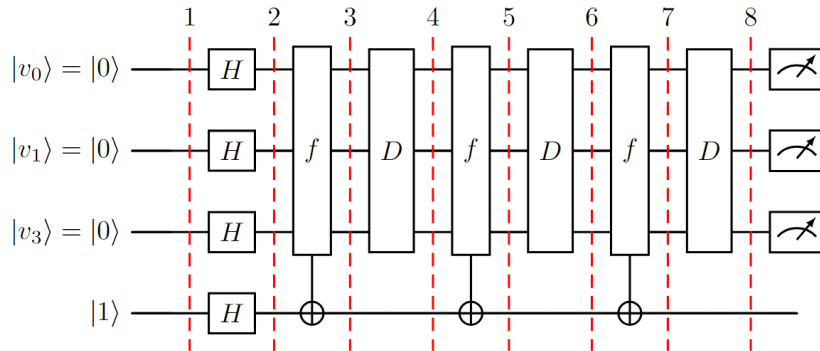


Fig. 8. Grover's circuit after performing 3 iterations.

$$1 : |000\rangle \quad (\text{Avg. Amp.} = 1)$$

$$2 : \frac{1}{\sqrt{8}} |000\rangle + \frac{1}{\sqrt{8}} |001\rangle + \frac{1}{\sqrt{8}} |010\rangle + \frac{1}{\sqrt{8}} |011\rangle + \frac{1}{\sqrt{8}} |100\rangle + \frac{1}{\sqrt{8}} |101\rangle + \frac{1}{\sqrt{8}} |110\rangle + \frac{1}{\sqrt{8}} |111\rangle \quad (\text{Desired State Prob.} = 12.5\%)$$

$$3 : \frac{1}{\sqrt{8}} |000\rangle + \frac{1}{\sqrt{8}} |001\rangle + \frac{1}{\sqrt{8}} |010\rangle + \frac{1}{\sqrt{8}} |011\rangle - \frac{1}{\sqrt{8}} |100\rangle + \frac{1}{\sqrt{8}} |101\rangle + \frac{1}{\sqrt{8}} |110\rangle + \frac{1}{\sqrt{8}} |111\rangle \quad (\text{Avg. Amp.} = \frac{3\sqrt{2}}{16})$$

$$4 : \frac{\sqrt{2}}{8} |000\rangle + \frac{\sqrt{2}}{8} |001\rangle + \frac{\sqrt{2}}{8} |010\rangle + \frac{\sqrt{2}}{8} |011\rangle + \frac{5\sqrt{2}}{8} |100\rangle + \frac{\sqrt{2}}{8} |101\rangle + \frac{\sqrt{2}}{8} |110\rangle + \frac{\sqrt{2}}{8} |111\rangle \quad (\text{Desired State Prob.} = 78.125\%)$$

$$5 : \frac{\sqrt{2}}{8} |000\rangle + \frac{\sqrt{2}}{8} |001\rangle + \frac{\sqrt{2}}{8} |010\rangle + \frac{\sqrt{2}}{8} |011\rangle - \frac{5\sqrt{2}}{8} |100\rangle + \frac{\sqrt{2}}{8} |101\rangle + \frac{\sqrt{2}}{8} |110\rangle + \frac{\sqrt{2}}{8} |111\rangle \quad (\text{Avg. Amp.} = \frac{\sqrt{2}}{32})$$

$$6 : -\frac{\sqrt{2}}{16} |000\rangle - \frac{\sqrt{2}}{16} |001\rangle - \frac{\sqrt{2}}{16} |010\rangle - \frac{\sqrt{2}}{16} |011\rangle + \frac{11\sqrt{2}}{16} |100\rangle - \frac{\sqrt{2}}{16} |101\rangle - \frac{\sqrt{2}}{16} |110\rangle - \frac{\sqrt{2}}{16} |111\rangle \quad (\text{Desired State Prob.} = 94.53\%)$$

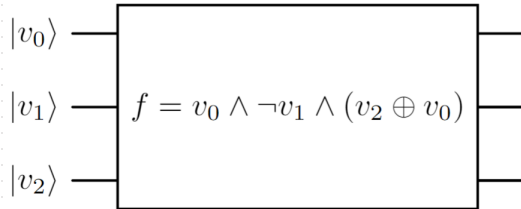
$$7 : -\frac{\sqrt{2}}{16} |000\rangle - \frac{\sqrt{2}}{16} |001\rangle - \frac{\sqrt{2}}{16} |010\rangle - \frac{\sqrt{2}}{16} |011\rangle - \frac{11\sqrt{2}}{16} |100\rangle - \frac{\sqrt{2}}{16} |101\rangle - \frac{\sqrt{2}}{16} |110\rangle - \frac{\sqrt{2}}{16} |111\rangle \quad (\text{Avg. Amp.} = -\frac{9\sqrt{2}}{64})$$

$$8 : -\frac{7\sqrt{2}}{32} |000\rangle - \frac{7\sqrt{2}}{32} |001\rangle - \frac{7\sqrt{2}}{32} |010\rangle - \frac{7\sqrt{2}}{32} |011\rangle + \frac{13\sqrt{2}}{32} |100\rangle - \frac{7\sqrt{2}}{32} |101\rangle - \frac{7\sqrt{2}}{32} |110\rangle - \frac{7\sqrt{2}}{32} |111\rangle \quad (\text{Desired State Prob.} = 33\%)$$

- Querying more than the optimal number of iterations has **negative effects**.
- The desired state probability increases exponentially as ***n* increases**.

FROM ORACLE TO A REALISTIC CIRCUIT

Oracle + Boolean Function



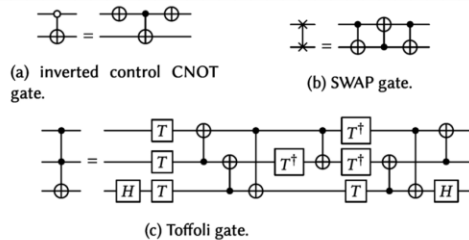
Logic Network

- Boolean function \rightarrow Logic Network
- Logic Network is graph for a given gate basis
- *Xor-And-Inverter Graphs* ¹
- (XAIG or XAG): $\{\wedge, \oplus, \neg\}$
- Pseudo-Kronecker Expressions used ²

Optimization

- *AND* gate operations are usually expensive due to native *T* gates ^{3,4}
- No. of *AND* gates is called Multiplicative Complexity \tilde{c} ⁵
- Boolean functions Affine Equivalence is used to reduce \tilde{c} ⁶

Realistic Quantum Circuits



Universal Gate Set

- Usually targeted for Fault-Tolerant Quantum Computing
- Clifford + *T* Gate Set ⁸
- Used to implement Toffoli gates

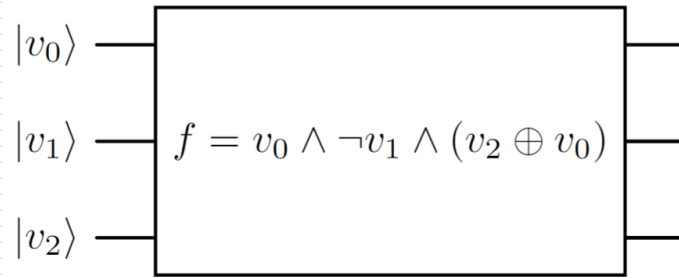
Reversible Circuit

- Traverse the XAG graph bottom-up to generate a reversible circuit ⁷
- Realizes Boolean Oracles:
 $|x\rangle|y\rangle \mapsto |x\rangle|y \oplus f(x)\rangle$

1. I. Håleček, P. Fíšer and J. Schmidt, "Are XORs in logic synthesis really necessary?"
 2. R. Drechsler, "Pseudo-Kronecker expressions for symmetric functions"
 3. G. J. Mooney, C. D. Hill, and L. C. L. Hollenberg, "Cost-optimal single-qubit gate synthesis in the Clifford hierarchy,"
 4. B. Eastin and E. Knill, "Restrictions on Transversal Encoded Quantum Gate Sets"
 5. J. Boyar, R. Peralta, and D. Pochuev, "On the multiplicative complexity of boolean functions over the basis $(\wedge, \oplus, 1)$ "
 6. E. Testa, M. Soeken, L. Amarù and G. D. Micheli, "Reducing the Multiplicative Complexity in Logic Networks for Cryptography and Security Applications"
 7. G. Meuli, M. Soeken, E. Campbell, M. Roetteler, and G. De Micheli, "The role of multiplicative complexity in compiling low *T*-count oracle circuits"
 8. D. Gottesman, "Theory of fault-tolerant quantum computation"

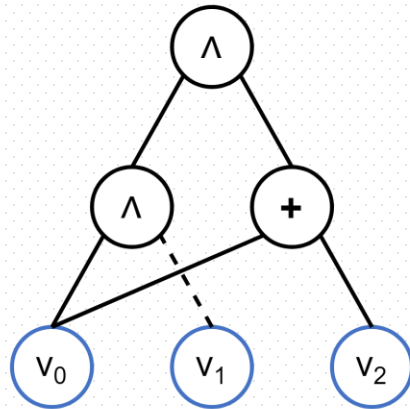
XAG TO REVERSIBLE CIRCUIT

Oracle with Access to Boolean Function:

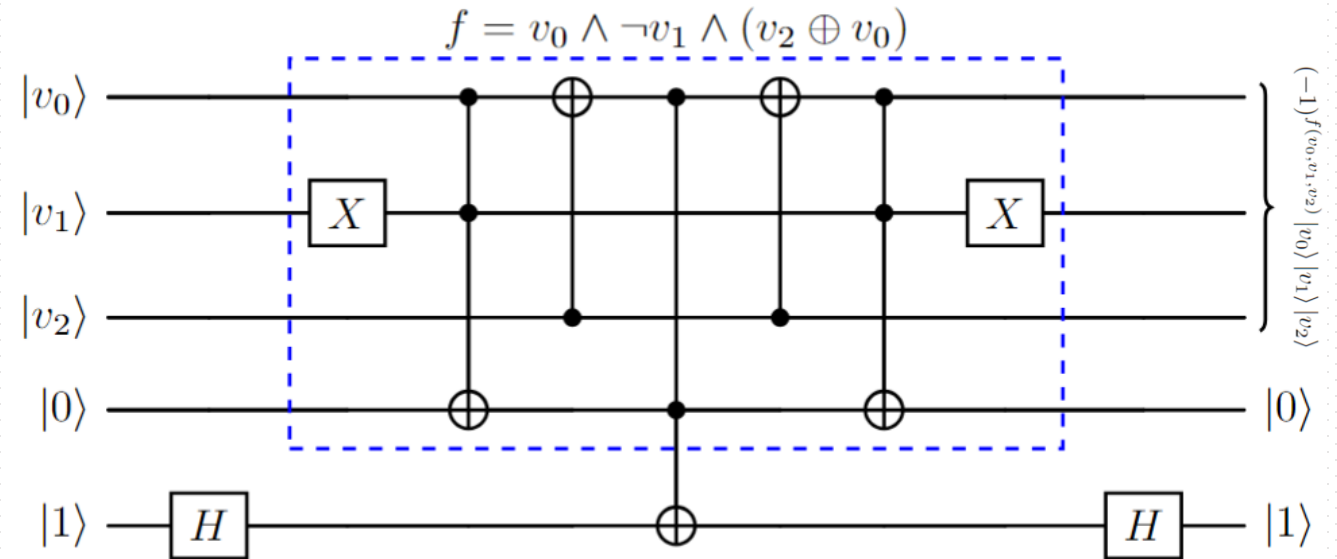


Generated XAG Network Boolean Function:

$$f(v_0, v_1, v_2) = v_0 \wedge \neg v_1 \wedge (v_2 \oplus v_0) :$$







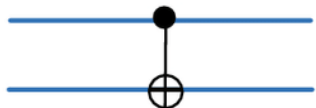


Generated Reversible Circuit



CLIFFORD + T GATE SET AND TOFFOLI GATES

CLIFFORD+T GATE SET

Hadamard Gate		$\frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$
T Gate		$\begin{bmatrix} 1 & 0 \\ 0 & e^{-i\frac{\pi}{4}} \end{bmatrix}$
Hermitian of T Gate		$\begin{bmatrix} 1 & 0 \\ 0 & e^{i\frac{\pi}{4}} \end{bmatrix}$
Phase Gate		$\begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}$
Hermitian of Phase Gate		$\begin{bmatrix} 1 & 0 \\ 0 & -i \end{bmatrix}$
NOT Gate		$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$
CNOT Gate		$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$

Optimized Toffoli Gate Circuits using Clifford + T Gate Set

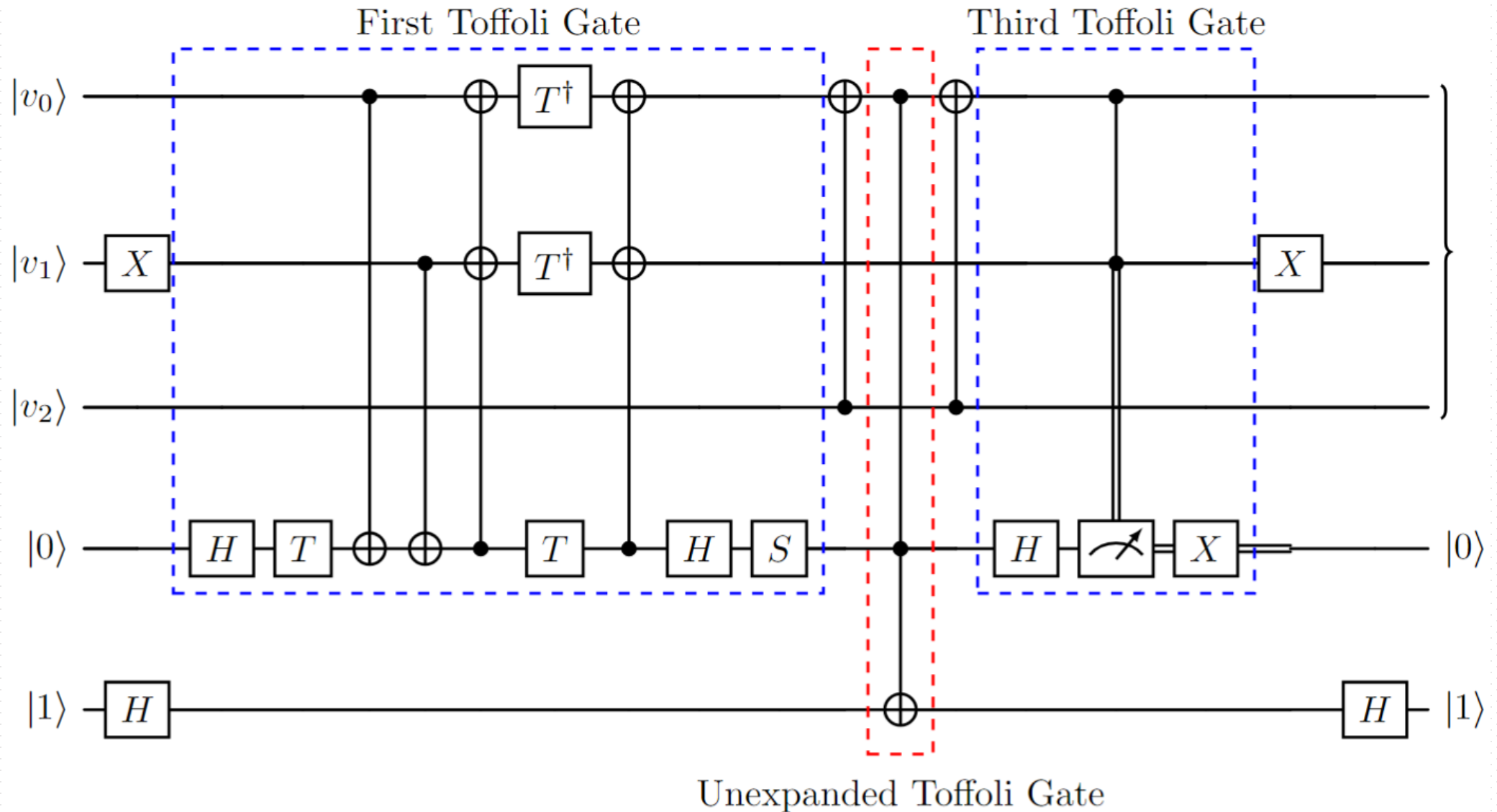
$$\begin{array}{c}
 |x_1\rangle \\
 |x_2\rangle \\
 |x_3\rangle \oplus
 \end{array}
 =
 \begin{array}{c}
 \begin{array}{c} \text{---} [T] \text{---} \oplus \text{---} \bullet \text{---} \bullet \text{---} [T^\dagger] \text{---} \bullet \text{---} \oplus \text{---} |x_1\rangle \end{array} \\
 \begin{array}{c} \text{---} [T] \text{---} \bullet \text{---} \oplus \text{---} [T^\dagger] \text{---} \oplus \text{---} [T^\dagger] \text{---} \oplus \text{---} \bullet \text{---} |x_2\rangle \end{array} \\
 \begin{array}{c} \text{---} [H] [T] \text{---} \bullet \text{---} \oplus \text{---} [T] \text{---} \bullet \text{---} \oplus \text{---} [H] \text{---} |x_3 \oplus x_1 x_2\rangle \end{array}
 \end{array}$$

$$\begin{array}{c}
 |x_1\rangle \\
 |x_2\rangle \\
 |0\rangle
 \end{array}
 \begin{array}{c}
 |x_1\rangle \\
 |x_2\rangle \\
 |x_1 x_2\rangle
 \end{array}
 =
 \begin{array}{c}
 |x_1\rangle \text{---} \bullet \text{---} \oplus \text{---} [T^\dagger] \text{---} \oplus \text{---} |x_1\rangle \\
 |x_2\rangle \text{---} \bullet \text{---} \oplus \text{---} [T^\dagger] \text{---} \oplus \text{---} |x_2\rangle \\
 |T\rangle \text{---} \oplus \text{---} \oplus \text{---} \bullet \text{---} [T] \text{---} \bullet \text{---} [H_Y] \text{---} |x_1 x_2\rangle
 \end{array}$$

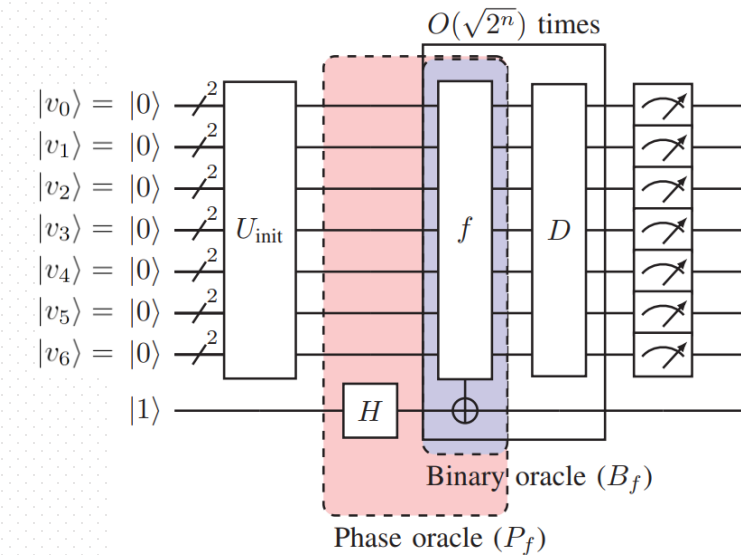
where $H_Y = SH$ and $|T\rangle = TH|0\rangle$

$$\begin{array}{c}
 |x_1\rangle \\
 |x_2\rangle \\
 |x_1 x_2\rangle
 \end{array}
 \begin{array}{c}
 |x_1\rangle \\
 |x_2\rangle \\
 |0\rangle
 \end{array}
 =
 \begin{array}{c}
 |x_1\rangle \text{---} \text{---} \text{---} \text{---} \text{---} \text{---} \text{---} \text{---} \text{---} |x_1\rangle \\
 |x_2\rangle \text{---} \text{---} \text{---} \text{---} \text{---} \text{---} \text{---} \text{---} |x_2\rangle \\
 |x_1 x_2\rangle \text{---} [H] \text{---} \text{---} \text{---} \text{---} \text{---} \text{---} \text{---} [X] \text{---} |0\rangle
 \end{array}$$

REVERSIBLE CIRCUIT TO QUANTUM CIRCUIT

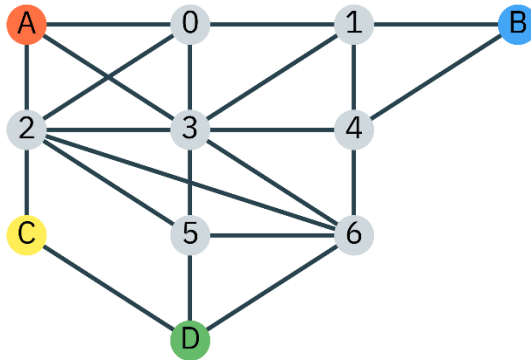


REVISITING THE ZED CITY PROBLEM



Let:

- $A = '00'$
- $B = '01'$
- $C = '10'$
- $D = '11'$



- Encoding:

$$|v_0 v_1 v_2 v_3 v_4 v_5 v_6\rangle = |v_{0_0} v_{0_1} v_{1_0} v_{1_1} v_{2_0} v_{2_1} v_{3_0} v_{3_1} v_{4_0} v_{4_1} v_{5_0} v_{5_1} v_{6_0} v_{6_1}\rangle$$

Listing 1. Python implementation of f

```
def f(v0, ..., v6 : BitVec(2)) -> BitVec(1):
    c0 = (v0 != '00')
    c1 = (v1 != '01') and (v1 != v0)
    c2 = (v2 != '00') and (v2 != '10') and
        (v2 != v0)
    c3 = (v3 != '00') and (v3 != v0) and
        (v3 != v1) and (v3 != v2)
    c4 = (v4 != '01') and (v4 != v1) and (v4 != v3)
    c5 = (v5 != '11') and (v5 != v2) and (v5 != v3)
    c6 = (v6 != '11') and (v6 != v2) and
        (v6 != v3) and (v6 != v4) and (v6 != v5)
    return c0 and c1 and c2 and
        c3 and c4 and c5 and c6
```

- Definition of the Solution Marker Function in Python
- Reed-Muller and Shannon's Expansion

Logic Circuit Synthesis

- Automated Boolean Oracle Synthesis and Circuit Generation
- No. of Ancilla Qubits Identification and Circuit Optimization
- Target Architecture Requirements

OPTIMIZING SOLUTION MARKER FUNCTION f

Recall: U_{init} can be fiddled with to eliminate the trivial non-solution states.

- *In Boolean satisfiability, there is often a degree of commonality between various non-solutions which can be eliminated*

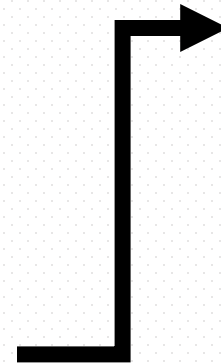
1. Define Function g to eliminate trivial non-solution states in U_{init} :

```
def g(v0, ..., v6 : BitVec(2)) -> BitVec(1):  
    return (v0 != '00') and (v1 != '01') and  
           (v2 != '00') and (v2 != '10') and  
           (v3 != '00') and (v4 != '01') and  
           (v5 != '11') and (v6 != '11')
```



2. Derive an Optimized Solution Marker Function f that ignores the non-trivial solutions:

```
def f(v0, ..., v6 : BitVec(2)) -> BitVec(1):  
    c1 = (v1[0] == v1[1]) and (v3 != v1)  
    c023 = ((v0 ^ v2 ^ v3) == '00')  
    c4 = (v4 != n1) and (v4 != v3)  
    c5 = (v5 != n2) and (v5 != v3)  
    c6 = ((v2 ^ v3 ^ v5 ^ v6) == '00') and  
         (v6 != v4)  
    return c1 and c023 and c4 and  
           c5 and c6
```



Optimized Logic Circuit:

- Reduced Circuit Cost
- Reduced Number of Ancilla Qubits

RESULTS

Top Submission to IBM's Quantum Challenge 2019:

- *These submissions were handcrafted solutions*

TABLE I
QUALITY OF RESULTS FOR B_f (HAND-OPTIMIZED AND NON-OPTIMIZED)

	Hand-optimized		Non-optimized	
	Qubits	cost	Qubits	cost
IBM's solution	32	5004		
Whit3z solution	32	2474		
XAG-based flow	31	2202	56	4347
XAG-based flow with pebbling	21	4497	30	7737

COST:

$$cost = n_{1_q} + 10n_{2_q}$$

where n_{1_q} is the number one-qubit operators and n_{2_q} the number of two-qubit operators.

Reduced Cost and No. Qubits
Compared to the Top Submission

Significant No. Qubit Reduction but
Higher Cost

Passable Solution without Hand-
Optimization

THANK YOU

December 19th, 2024

XAG OPTIMIZATION FOR LOGIC SYNTHESIS

- **Consider the majority-of-three Boolean Function:** $\langle x_1 x_2 x_3 \rangle = x_1 x_2 \oplus x_1 x_3 \oplus x_2 x_3$
- **Observing the function, we can see that there is 3 *AND* gates, making it costly to implement**