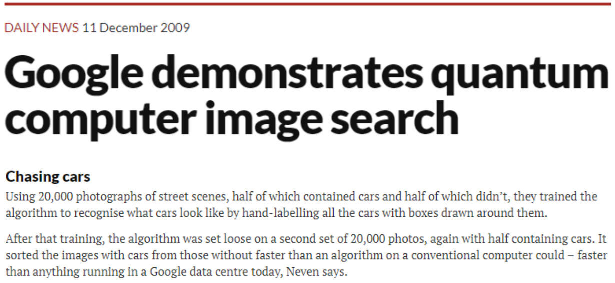
Implementing Quantum Pattern Matching

AP3421-PR Quantum Information Project
Aritra Sarkar & Naveen Kumar

Motivation

- Jigsaw Puzzles!!



DAILY NEWS 11 December 2009

Google demonstrates quantum computer image search

Chasing cars

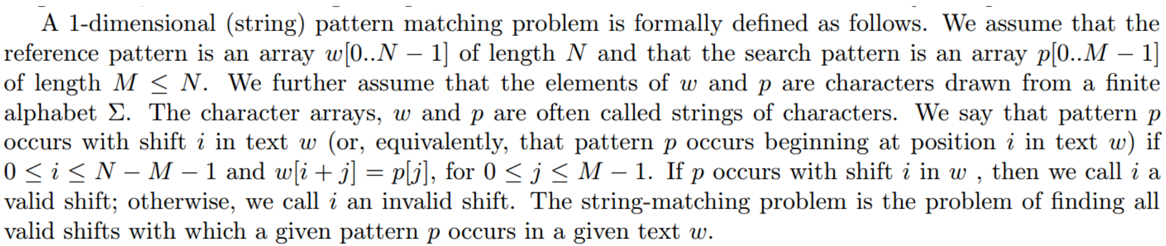
Using 20,000 photographs of street scenes, half of which contained cars and half of which didn’t, they trained the algorithm to recognise what cars look like by hand-labelling all the cars with boxes drawn around them.

After that training, the algorithm was set loose on a second set of 20,000 photos, again with half containing cars. It sorted the images with cars from those without faster than an algorithm on a conventional computer could – faster than anything running in a Google data centre today, Neven says.

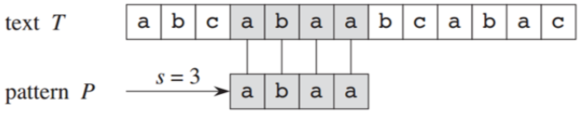
… a trip down memory lane (90s kid et. al)

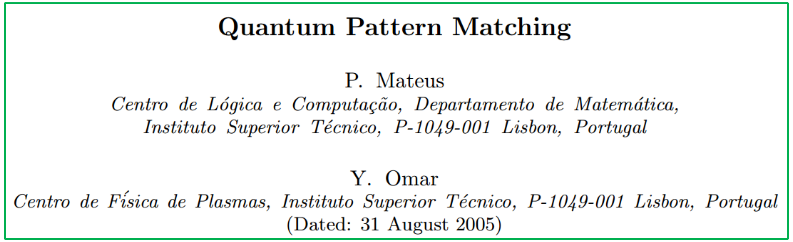
- Pattern matching for big data in an exa-scale computer
  - Pattern matching: classification & analysis
  - Big data sources: social media, genes, LHC
  - Exa-scale device: GPU clusters

Pattern Matching



A 1-dimensional (string) pattern matching problem is formally defined as follows. We assume that the reference pattern is an array $w[0..N-1]$ of length N and that the search pattern is an array $p[0..M-1]$ of length $M \leq N$. We further assume that the elements of w and p are characters drawn from a finite alphabet Σ. The character arrays, w and p are often called strings of characters. We say that pattern p occurs with shift i in text w (or, equivalently, that pattern p occurs beginning at position i in text w) if $0 \leq i \leq N-M-1$ and $w[i+j] = p[j]$, for $0 \leq j \leq M-1$. If p occurs with shift i in w , then we call i a valid shift; otherwise, we call i an invalid shift. The string-matching problem is the problem of finding all valid shifts with which a given pattern p occurs in a given text w.

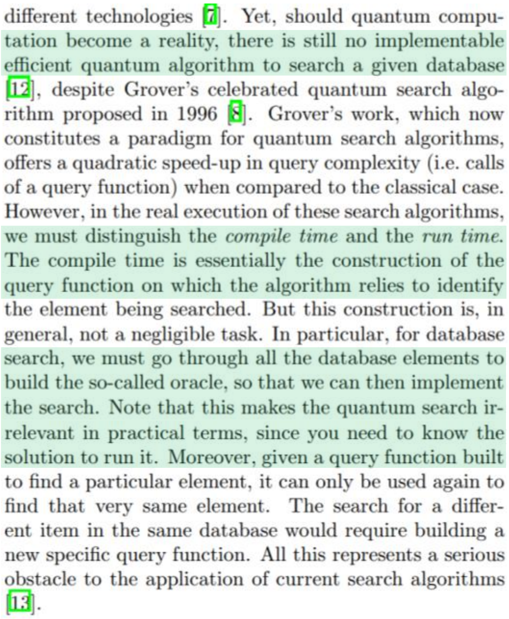




Quantum Pattern Matching

P. Mateus
Centro de Lógica e Computação, Departamento de Matemática,
Instituto Superior Técnico, P-1049-001 Lisbon, Portugal

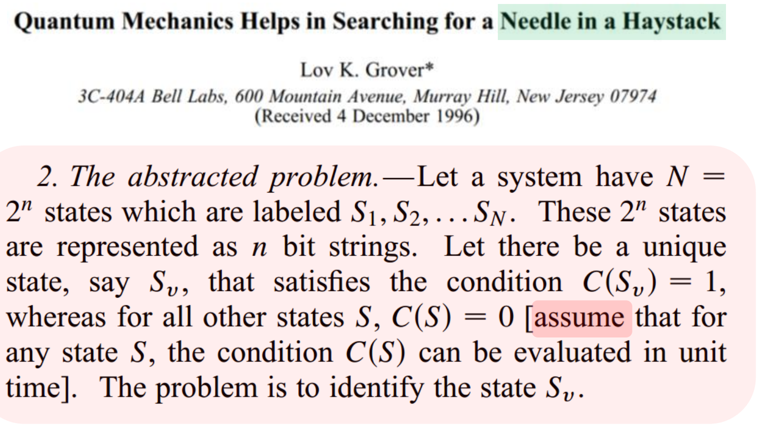
Y. Omar
Centro de Física de Plasmas, Instituto Superior Técnico, P-1049-001 Lisbon, Portugal
(Dated: 31 August 2005)



different technologies [7]. Yet, should quantum computation become a reality, there is still no implementable efficient quantum algorithm to search a given database [12], despite Grover's celebrated quantum search algorithm proposed in 1996 [8]. Grover's work, which now constitutes a paradigm for quantum search algorithms, offers a quadratic speed-up in query complexity (i.e. calls of a query function) when compared to the classical case. However, in the real execution of these search algorithms, we must distinguish the *compile time* and the *run time*. The compile time is essentially the construction of the query function on which the algorithm relies to identify the element being searched. But this construction is, in general, not a negligible task. In particular, for database search, we must go through all the database elements to build the so-called oracle, so that we can then implement the search. Note that this makes the quantum search irrelevant in practical terms, since you need to know the solution to run it. Moreover, given a query function built to find a particular element, it can only be used again to find that very same element. The search for a different item in the same database would require building a new specific query function. All this represents a serious obstacle to the application of current search algorithms [13].

Oracles

- Grover’s search
  - unstructured database
  - assumes existence of an Oracle
  - quadratic reduction in query complexity
- What is an Oracle?

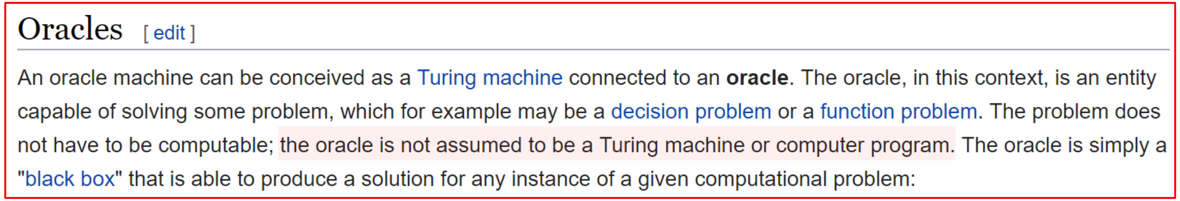


Quantum Mechanics Helps in Searching for a Needle in a Haystack

Lov K. Grover*

3C-404A Bell Labs, 600 Mountain Avenue, Murray Hill, New Jersey 07974
(Received 4 December 1996)

2. *The abstracted problem.*—Let a system have $N = 2^n$ states which are labeled $S_1, S_2, \ldots S_N$. These 2^n states are represented as n bit strings. Let there be a unique state, say S_ν, that satisfies the condition $C(S_\nu) = 1$, whereas for all other states S, $C(S) = 0$ [assume that for any state S, the condition $C(S)$ can be evaluated in unit time]. The problem is to identify the state S_ν.



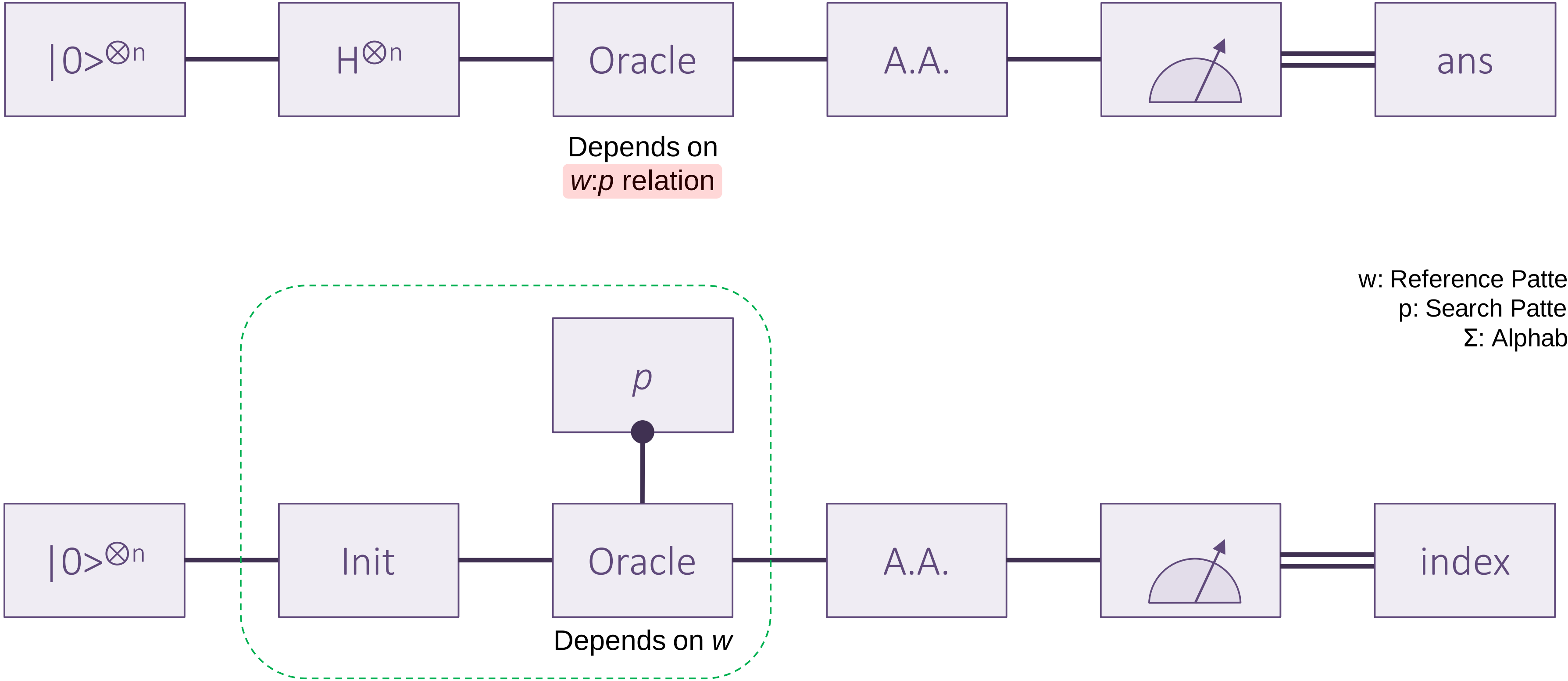
Oracles [edit]

An oracle machine can be conceived as a Turing machine connected to an **oracle**. The oracle, in this context, is an entity capable of solving some problem, which for example may be a decision problem or a function problem. The problem does not have to be computable; the oracle is not assumed to be a Turing machine or computer program. The oracle is simply a "black box" that is able to produce a solution for any instance of a given computational problem:

Bran Stark, Three Eyed Raven
Legendary Creature — Shapeshifter

Learned
Introvert
Mystical
Fictional

Quantum Pattern Matching



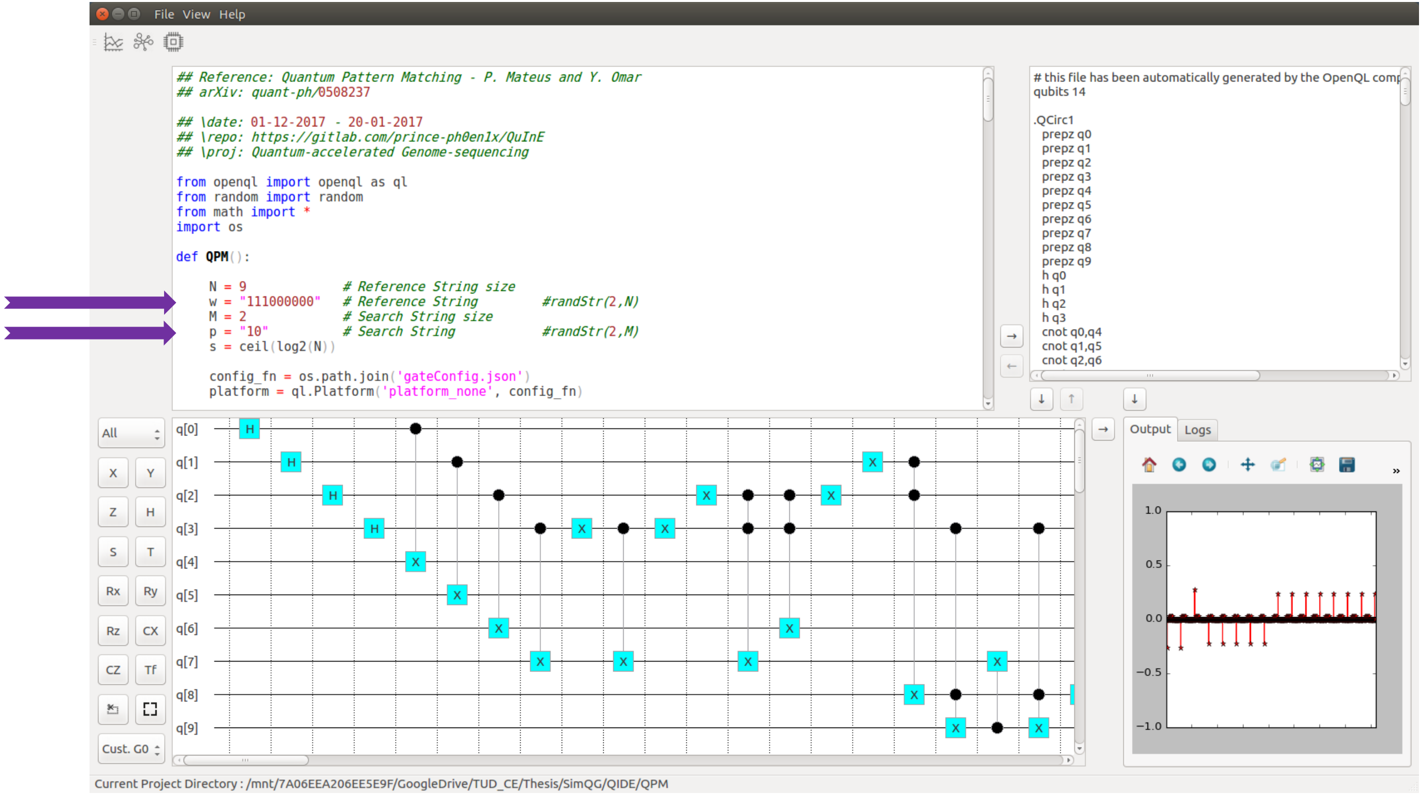
Depends on *w*:*p* relation

Depends on *w*

w: Reference Pattern
p: Search Pattern
Σ: Alphabet

Tools

- MATLAB
- QuInE
- OpenQL
- QX simulator



```
## Reference: Quantum Pattern Matching - P. Mateus and Y. Omar
## arXiv: quant-ph/0508237

## \date: 01-12-2017 - 20-01-2017
## \repo: https://gitlab.com/prince-ph0en1x/QuInE
## \proj: Quantum-accelerated Genome-sequencing

from openql import openql as ql
from random import random
from math import *
import os

def QPM():

    N = 9               # Reference String size
    w = "111000000"     # Reference String          #randStr(2,N)
    M = 2               # Search String size
    p = "10"            # Search String             #randStr(2,M)
    s = ceil(log2(N))

    config_fn = os.path.join('gateConfig.json')
    platform = ql.Platform('platform_none', config_fn)
```

```
# this file has been automatically generated by the OpenQL comp
qubits 14

.QCirc1
   prepz q0
   prepz q1
   prepz q2
   prepz q3
   prepz q4
   prepz q5
   prepz q6
   prepz q7
   prepz q8
   prepz q9
   h q0
   h q1
   h q2
   h q3
   cnot q0,q4
   cnot q1,q5
   cnot q2,q6
```

Current Project Directory : /mnt/7A06EEA206EE5E9F/GoogleDrive/TUD_CE/Thesis/SimQG/QIDE/QPM

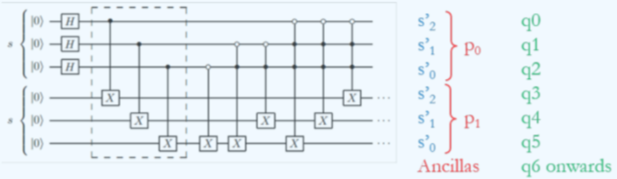
Algorithm

Initialize

- 111000000
- Sequential copy and increment

$$|\psi_0\rangle = \sum_{i=1}^{N-M+1} \frac{1}{\sqrt{N-M+1}} |i, i+1, ..., i+M-1\rangle$$

$$|\psi_0\rangle = \frac{1}{3}\{|0,1\rangle + |1,2\rangle + ... + |7,8\rangle\}$$



$$\frac{1}{4}(|0000,0001\rangle + |0001,0010\rangle + ... + |1110,1111\rangle + |1111,1111\rangle)$$

Make/Call Oracles

- Arbitrary Boolean function

$$f : \{0,1\}^{2^{s'}} \rightarrow \{-1,1\}$$

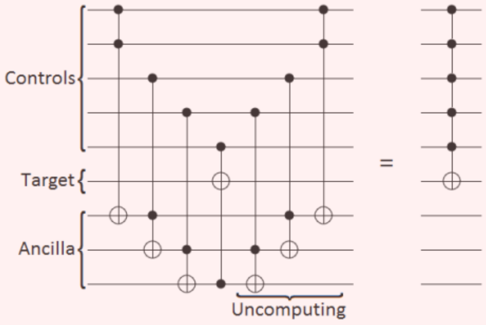
$$f_1 = [1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0]$$

need to mark the positions of $0, 1, 2$
qubit states of $|0000\rangle$, $|0001\rangle$, $|0010\rangle$
using 3 CPhase

- If state of an index to be marked
  - Take Boolean value of Index
  - Apply CPhase on all s' qubits
  - X Control on qubits with value = 0

Inversion about mean

- Grover Gate on all *s'*M* qubits
- Inversion about Mean
- Amplitude Amplification

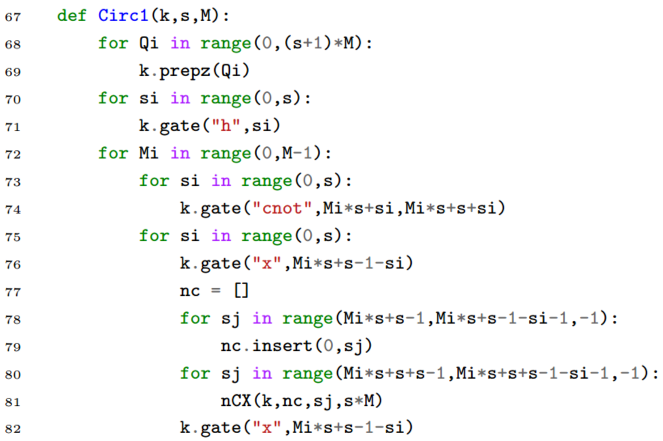


Kernels

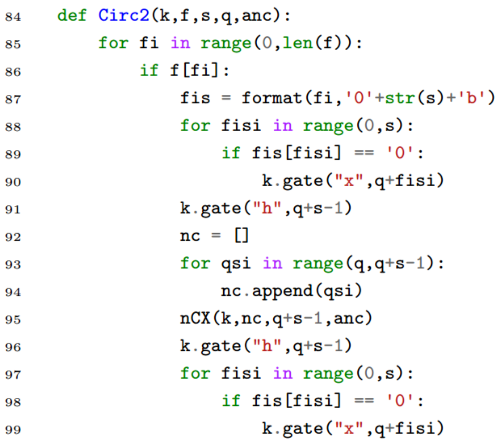
Initialize

Make/Call Oracles

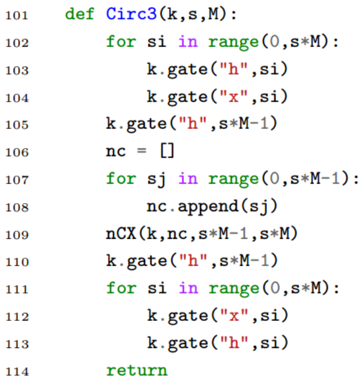
Inversion about mean



```
def Circ1(k,s,M):
    for Qi in range(0,(s+1)*M):
        k.prepz(Qi)
    for si in range(0,s):
        k.gate("h",si)
    for Mi in range(0,M-1):
        for si in range(0,s):
            k.gate("cnot",Mi*s+si,Mi*s+s+si)
        for si in range(0,s):
            k.gate("x",Mi*s+s-1-si)
            nc = []
            for sj in range(Mi*s+s-1,Mi*s+s-1-si-1,-1):
                nc.insert(0,sj)
            for sj in range(Mi*s+s+s-1,Mi*s+s+s-1-si-1,-1):
                nCX(k,nc,sj,s*M)
            k.gate("x",Mi*s+s-1-si)
```



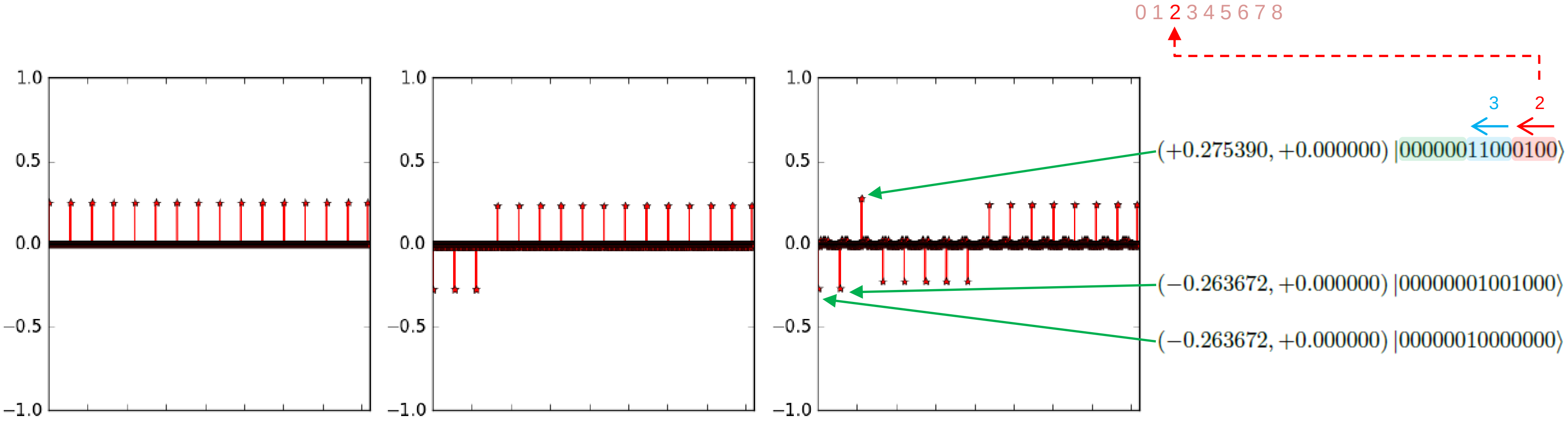
```
def Circ2(k,f,s,q,anc):
    for fi in range(0,len(f)):
        if f[fi]:
            fis = format(fi,'0'+str(s)+'b')
            for fisi in range(0,s):
                if fis[fisi] == '0':
                    k.gate("x",q+fisi)
            k.gate("h",q+s-1)
            nc = []
            for qsi in range(q,q+s-1):
                nc.append(qsi)
            nCX(k,nc,q+s-1,anc)
            k.gate("h",q+s-1)
            for fisi in range(0,s):
                if fis[fisi] == '0':
                    k.gate("x",q+fisi)
```



```
def Circ3(k,s,M):
    for si in range(0,s*M):
        k.gate("h",si)
        k.gate("x",si)
    k.gate("h",s*M-1)
    nc = []
    for sj in range(0,s*M-1):
        nc.append(sj)
    nCX(k,nc,s*M-1,s*M)
    k.gate("h",s*M-1)
    for si in range(0,s*M):
        k.gate("x",si)
        k.gate("h",si)
    return
```

Results

- Different stages of the run for the search string p = "10" in reference string w = "111000000"

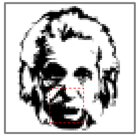


(a) after Kernel 1 (b) after Kernel 2_1 (c) after Kernel 2_0

Figure 7: Qubit state vector at difference phases of the algorithm's example run

Future Work

Dimension



Quantum Machine Learning

Alphabet

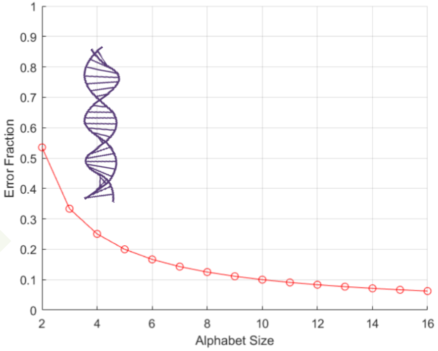


Figure 8: Failure probability for various alphabet sizes

$$eq_A = \frac{N!}{\lceil N/A \rceil !^A}$$

$$fp_A = \frac{eq_A + \frac{A^N - eq_A}{A}}{A^N} = \frac{(A-1) * eq_A}{A^{N-1}} + \frac{1}{A} \approx \frac{1}{A}$$

Implementing Quantum Pattern Matching

AP3421-PR Quantum Information Project
Aritra Sarkar & Naveen Kumar