SURGE 2012

QUANTUM GENETIC ALGORITHM IN SOLVING A MAZE

Niraj Kumar

Department of Physics , IIT Kanpur

Mentor – Dr. Debabrata Goswami

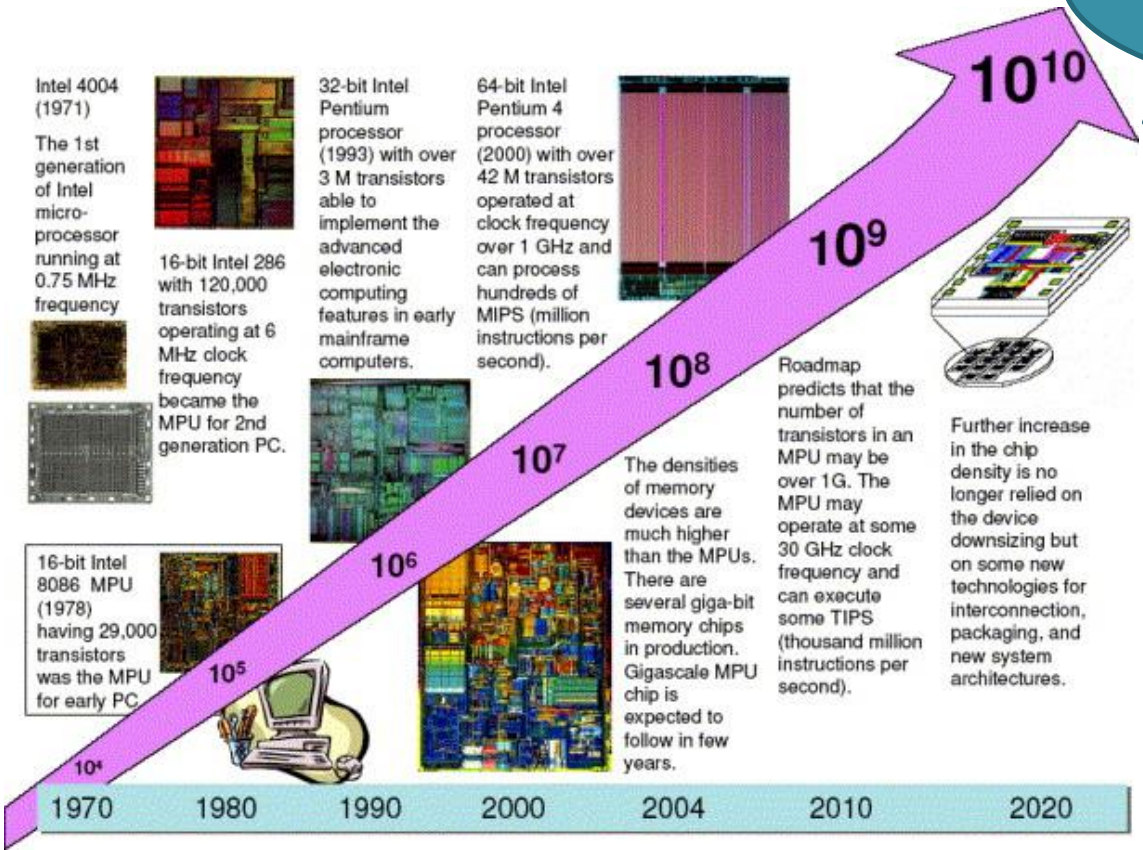
Dept. of Chemistry , IIT Kanpur

Moore's Law

Classical Computation

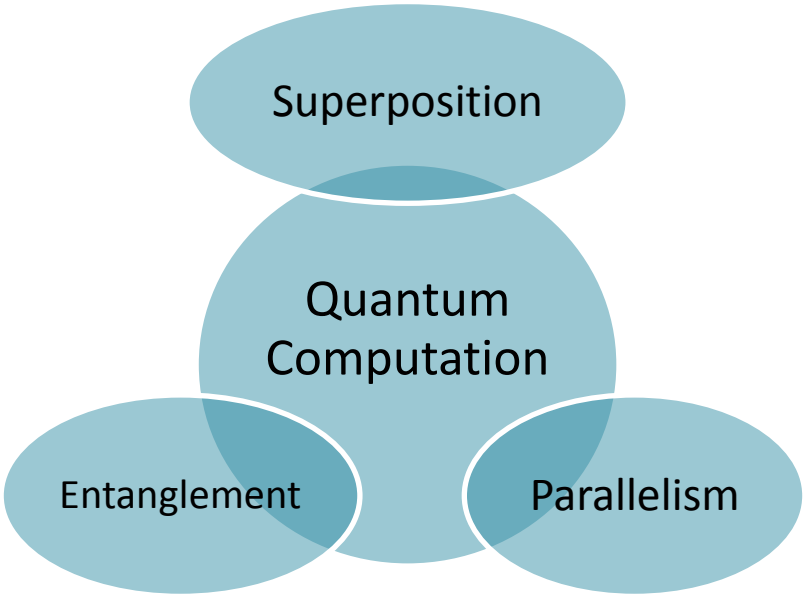
Quantum Computation

Figure Depicting number of transistors per chip



STRENGTH OF QUANTUM COMPUTATION

Elements of Quantum Comp.



Popular Quantum Algorithms

Shor Algo.

- Factoring Algorithm
- Exponential Speed-up over Classical Algorithm

Grover Algo

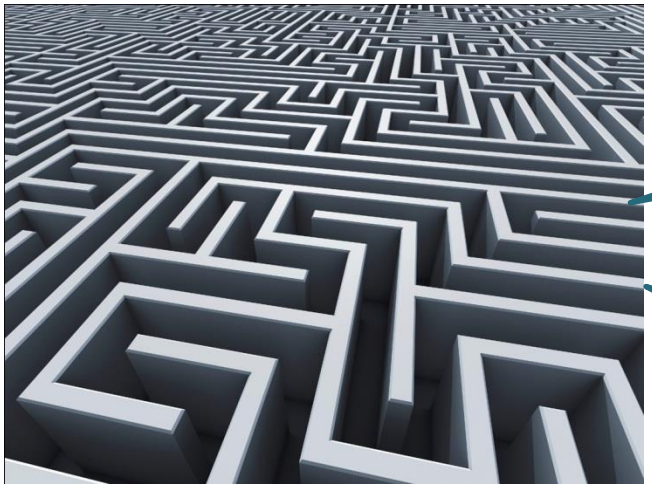
- Search Algorithm
- Quadratic Speed-up over Classical Algorithm

QGA

- Optimizational Algorithm
- Considerable Speed-up over Classical Algorithm

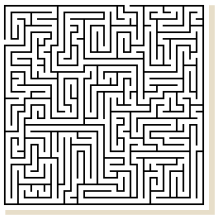
PROBLEM STATEMENT

Solving a 2-dimension perfect maze using Classical and Quantum Genetic Algorithm



Classical Genetic Algorithm

Quantum Genetic Algorithm



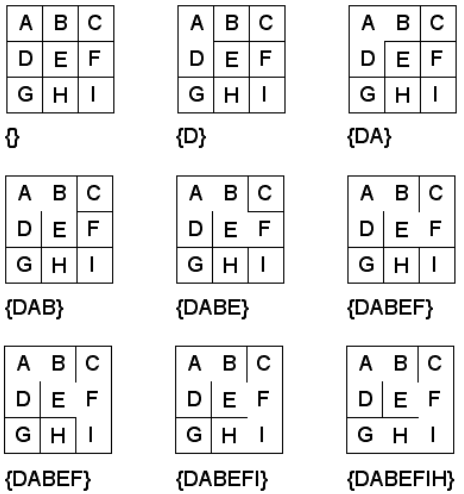
MAZE GENERATION

Recursive Backtracker Algorithm

Salient Feaures

- Two dimensional Maze
- Perfect Mazes

Illustration on 3x3 Maze



Recursive Backtracker Algo. for 3x3 maze:

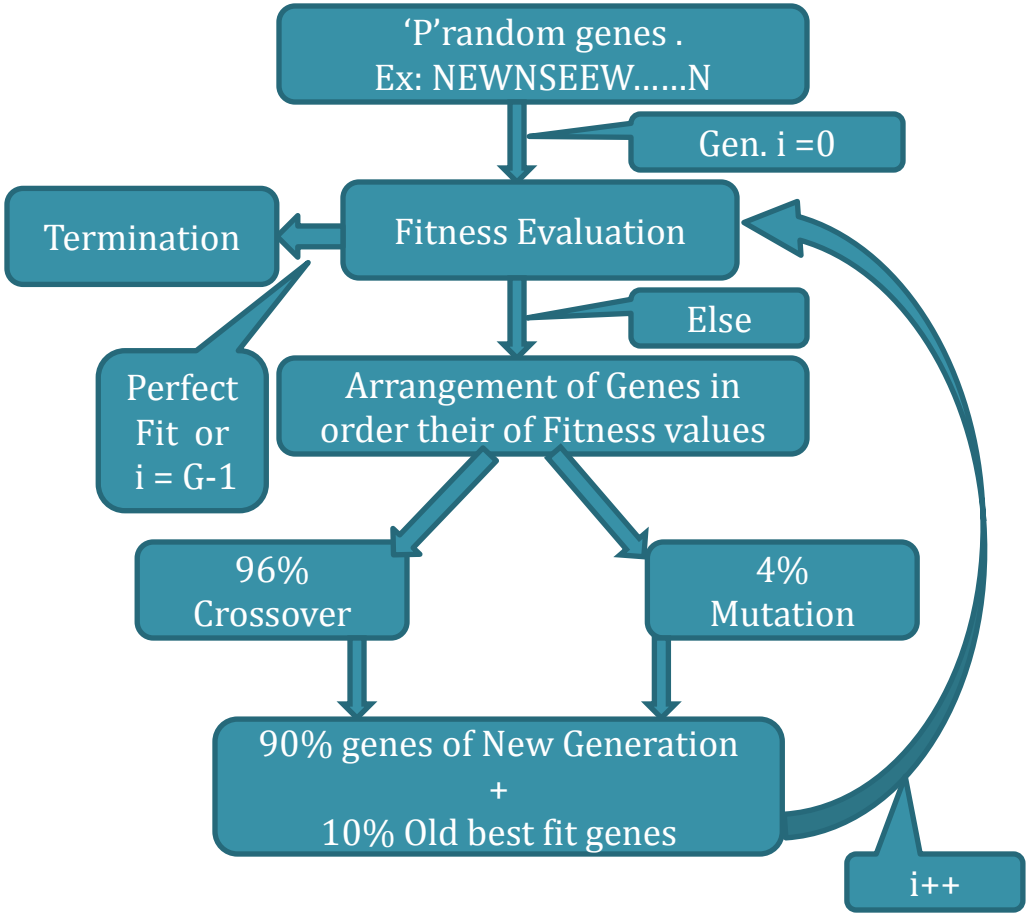
1. Pick a room randomly, and push it onto the stack. 2. This is the starting room.
3. Pick a room adjacent to the first room, and open a door between them. This room is the current room.
4. Push the starting room onto the stack.
5. While there's at least one room left on the stack, repeat the following steps:
    a. If there are any unconnected rooms next to the current room,
        1. Pick one randomly, and open a door between it and the current room.
        2. Push the current room onto the stack; the new room becomes the current room.
        3. Go back to the top of the loop.
    b. If there are no unconnected rooms next to the current room,
        1. Pop a room off of the stack; it becomes the current room.
        2. Go back to the top of the loop.
6. When the stack is empty, the maze is complete.

CLASSICAL GENETIC ALGORITHM

Size of Maze (nxm)	Population of genes (P)
Input	
Number of generations G	End Point

Points to note :

1. The maze consists of nxm grids.
2. Each grid has one or more of the walls north(N), south(S), east(E), west(W).
3. The walls are assigned a number i.e. n =1, e =2, s =4, w = 8.
4. Population is the number of genes in one generation.
5. Generation is number of times of crossover and mutation before within which the solution is ascertained.
6. Fitness is a measure of how close is the particular gene from the perfect solution.
7. Fitness is measured at the end of each generation and best fit genes are chosen in further generations.



Implementing CGA in Python

cga.py (~) - gedit

File Edit View Search Tools Documents Help

Open Save Undo

cga.py

```
def fit(fitness):
    if fitness == ((grid_len)**2)*2:
        return 0
    return 1

def fit_cal(gene):
    row = 0
    col = 0
    i = 0
    x = gate(grid[row][col])
    while comp(gene[i],x) == 0 and final(row,col) == 1 and i < l_gen:
        row,col = next_rc(row,col,gene[i])
        i = i+1
        x = gate(grid[row][col])
    fitness = ((grid_len)**2)*2 - ((row_f - row)**2 + (col_f - col)**2)
    return fitness

def mutate(gene):
    x = random.randint(0,l_gen-1)
    g = gate(no+e+s+w - gene[x])
    gene[x] = random.sample(g,1)[0]
    return gene

def crossover(gene1,gene2):
    # random index cut
    gene3 = np.zeros(l_gen)
    gene4 = np.zeros(l_gen)
    x = random.randint(1,l_gen-1)
    y = l_gen - x
    for i in xrange(x):
        gene3[i] = gene1[i]
        gene4[i] = gene2[i]
```

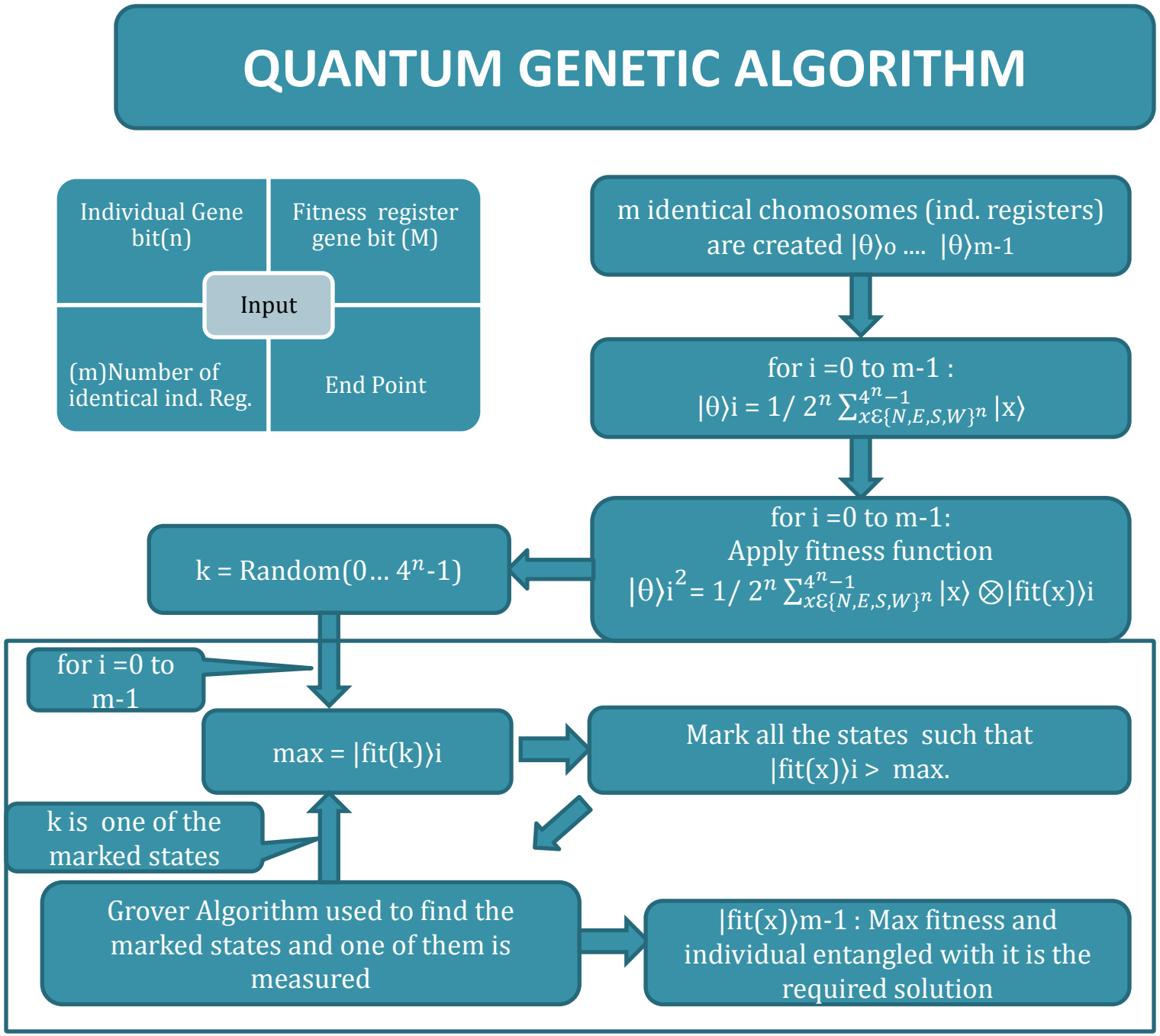
Pytho

fit_cal(gene) : Takes gene as input and returns its fitness value.
mutate(gene) : Inputs gene and returns new gene with one bit altered
crossover(gene1,gene2): intermingles two genes with breakage at one random point and returns new ones

```
iraj@niraj-laptop:~$ ipython
Enthought Python Distribution -- www.enthought.com

Python 2.7.3 |EPD 7.3-1 (32-bit)| (default, Apr 11 2012, 18:02:54)
Type "copyright", "credits" or "license" for more information.

IPython 0.12.1 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

In [1]: run cga.py
Enter population size:10
Enter genetation size:4
Enter finish row:5
Enter finish column:6
[ 2.  2.  1.  2.  8.  2.  1.  4.  1.  1.  4.  1.  2.  1.  4.  4.  2.  4.
  4.  2.  8.  1.  8.  1.  4.  8.  4.  4.  8.  2.  4.  1.  1.  2.  2.  4.
  8.  2.  2.  8.  4.  2.  2.  1.  8.  4.  8.  8.  1.  4.  2.  4.  1.  2.
  2.  8.  8.  1.  1.  2.  8.  2.  8.  8.  4.  2.  8.  8.  8.  8.  8.  4.
  4.  2.  1.  2.  1.  2.  4.  4.  1.  4.  4.  8.  2.  4.  1.  1.  1.  2.
  2.  8.  2.  8.  1.  4.  1.  1.  8.  2.  8.  8.  8.  4.  4.  4.  1.  1.
  4.  8.  2.  4.  2.  1.  8.  2.  1.  1.  8.  4.]
1739.0

[ 4.  1.  4.  4.  8.  4.  1.  2.  1.  2.  4.  4.  1.  8.  4.  8.  2.  8.
  8.  1.  1.  1.  8.  1.  4.  1.  2.  4.  4.  8.  8.  4.  4.  2.  1.  8.
  8.  4.  1.  8.  1.  8.  8.  4.  4.  8.  8.  1.  2.  2.  4.  2.  2.  8.
  4.  1.  2.  2.  2.  2.  8.  8.  4.  8.  1.  8.  2.  8.  4.  2.  4.  2.
  8.  4.  1.  8.  8.  4.  8.  1.  2.  2.  4.  4.  4.  2.  1.  1.  8.  4.
  2.  2.  1.  4.  8.  1.  8.  8.  8.  2.  1.  4.  2.  2.  2.  1.  4.  2.
  2.  2.  4.  2.  2.  8.  2.  2.  4.  2.  8.  2.]
1755.0

[ 4.  2.  8.  1.  8.  2.  1.  4.  4.  8.  4.  2.  8.  8.  1.  2.  1.  8.
  1.  4.  1.  1.  1.  1.  4.  1.  4.  4.  1.  8.  2.  2.  2.  4.  4.  8.
```

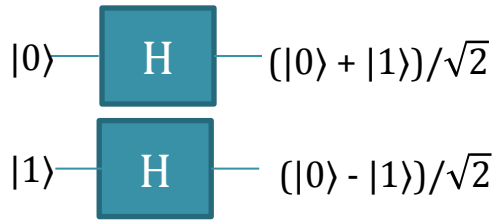
1. Running cga.py on Ipython
2. Values for population,generation, finish row, finish column are given by user.
3. The gene is displayed with its corresponding fitness value.

QUANTUM GENETIC ALGORITHM



Superposition

- For the basis states $|0\rangle$ and $|1\rangle$, Hadamard Gate looks like :



- The four basis gates look like:

$|N\rangle = |00\rangle$, $|E\rangle = |01\rangle$, $|S\rangle = |10\rangle$, $|W\rangle = |11\rangle$

- for i = 0 to m-1

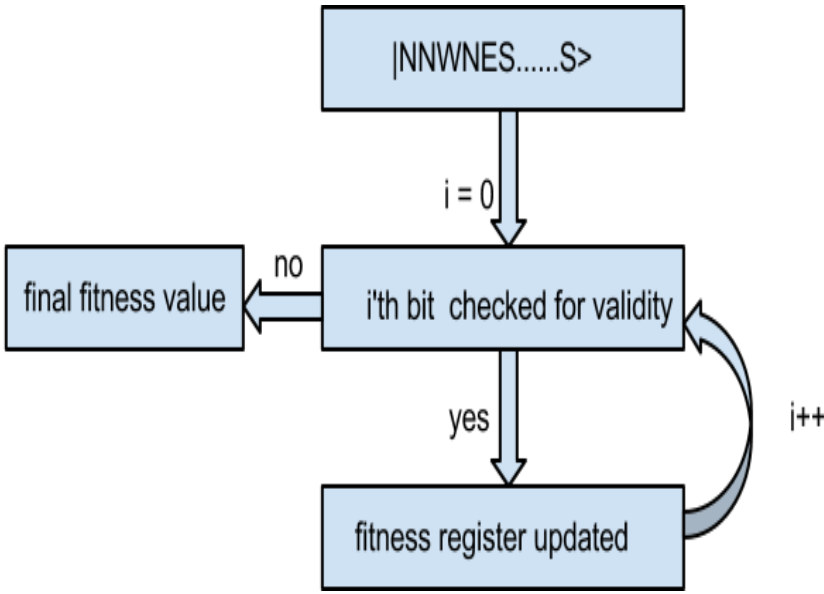
$$|\varphi\rangle_i = 1/\,2^n \sum_{x\varepsilon\{N,E,S,W\}^n}^{4^n-1} |x\rangle \otimes |fit(x)\rangle i$$

- All $|\varphi\rangle_i'$s are equivalent
- Contain superposition of all possible individuals

Fitness Evaluation

Salient features:

- Fitness register is an M bit register
- Evaluation formula is similar to the classical counterpart

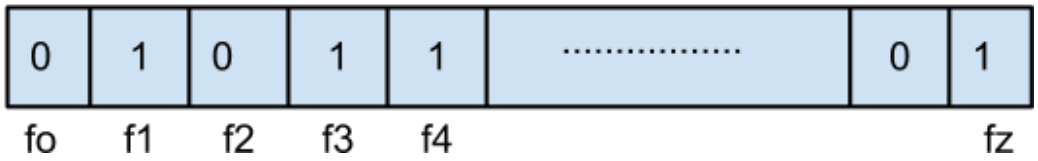


Oracle

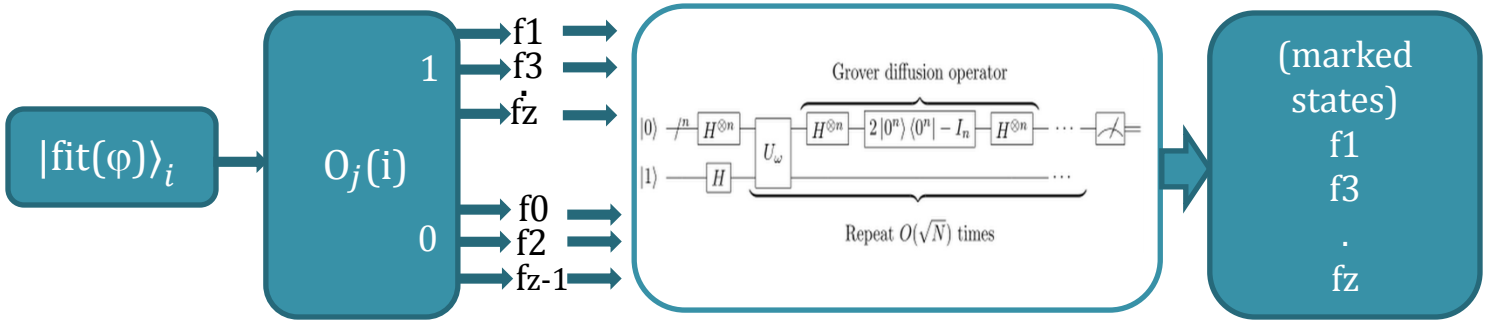
Array
- A[0],A[1],... A[N]

$O_j(i)$ → 1 : A[i] > A[j]; 0 : else

- If oracle is implemented with respect to j in the above array.
- Hence all the array elements greater than A[j] is marked by the oracle
- The individual fitness register $|fit(\varphi)\rangle_i \equiv f[0],..... f[z]$, where $z = 2^M - 1$.
- The figure below shows the result of applying oracle on $|fit(\varphi)\rangle_i$ with respect to f[k] :



Grover Algorithm for finding marked states



Conclusion

Complexity Assessment :

The complexity of maximum finding algorithm is $O(\sqrt{n})$. But the search space is $n = 4^n$ size. Hence the number of steps may not be enough to achieve efficient algorithm. This can be further refined by using better logarithmic search algorithm.

- The project was on Quantum Genetic Algorithm(specifically reduced QGA)
- The word 'reduced' because only one generation sufficed to find out the optimum result.
- As the only thing that slows down QGA is the search for the marked states, hence this algorithm has the potential of being efficient if a better search algorithm is used compared to Grover algorithm.

Future Work

- Effort on coming up with better maximum finding algorithm (of a logarithmic complexity).
- Comimg up with the implementation of Oracle and the maximum finding algorithm in a quantum computer.