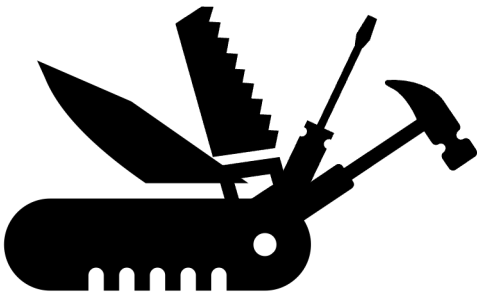


# A Multi-tool for your Quantum Algorithmic Toolbox



Shelby Kimmel  
Middlebury College

Based on work with

- **Chloe Ye, Da-Yeon Koh, Noel Anderson, Jay-U Chung** arXiv:2012.01276
- **Teal Witter** arXiv:2010.02324 (WADS 2021)
- **Kai DeLorenzo, Teal Witter**, arXiv:1904.05995 (TQC 2019)
- Michael Jarret, Stacey Jeffery, Alvaro Piedrafita, arXiv:1804.10591 (ESA 2018)
- Stacey Jeffery: arXiv: 1704.00765 (Quantum vol 1 p 26)
- **Bohua Zhan**, Avinatan Hassidim, arXiv:1101.0796 (ITCS 2012)

# Quantum Computers

Quantum computers use quantum particles – atoms, photons, phonons, electrons, etc. – to compute.

Famously fast for:

- factoring
- simulating quantum chemistry

# Quantum Computers

Quantum computers use quantum particles – atoms, photons, phonons, electrons, etc. – to compute.

Famously fast for:

- factoring
- simulating quantum chemistry



- **Specialized Problems**

# Quantum Computers

Quantum computers use quantum particles – atoms, photons, phonons, electrons, etc. – to compute.

Famously fast for:

- factoring
- simulating quantum chemistry



- **Specialized Problems**
- **Sophisticated design/analysis**

# Quantum Computers

Quantum computers use quantum particles – atoms, photons, phonons, electrons, etc. – to compute.

Famously fast for:

- factoring
- simulating quantum chemistry

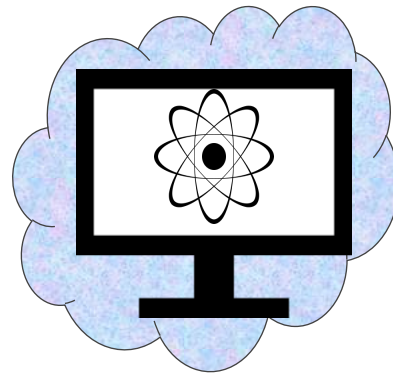


- **Specialized Problems**
- **Sophisticated design/analysis**

- **Everyday Problems?**
- **Accessible design/analysis?**

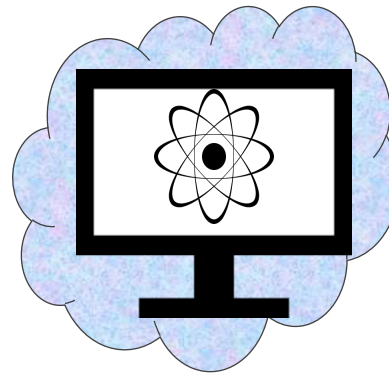
Imagine...

Year 2027



# Imagine...

## Year 2027



```
+18 average.m x aig.m x aigztest.m x KunED.m
V
10 - viID=zeros(2^bitsize,1); %initial vector st
11 - viUN=zeros(2^(bitsize+1),1); %one more bit
12 - viID(1)=1;
13 - viUN(1)=1;
14
15 %running to get the antinodes
16 %UnIDedUnitary is the unitary that represen
17 %IDedUnitary is the unitary for the identif
18 viantiUN=UnIDedUnitary(bitsize+1,pi/2)*viUN
19 viantiID=IDedUnitary(v,pi/2)*viID;
20
21 %Getting a superpoistion of node and anti-n
22 sum(viantiID);
23 sum(viantiUN);
24 a=1; %a allows us to change the phase betwe
25 vinitUN=viUN-abs(viantiUN); %setting up the
26 vinitID=viID-abs(viantiID);
27
28 %Now we run on the superposition of the ant

Command Window
7 0
ans =
0 0 0
1 0 0
0 1 1
1 1 1
fx >>
```

```
# else:
#     return -1.0*(abs(test1)+abs(test2))

def sim_mat_create(vec):
    return np.array([[1.0+vec[0],vec[1],0],[vec[2],1.0+vec[3],0],
#return np.array([[1.0+vec[0],vec[1],0],[0.0,1.0,0],[0.0,0.0,0.0]])

def sim_mat_create_full(vec):
    return np.array([[1.0,0.0,0.0,0.0],[0.0,1.0+vec[0],vec[1],0],
#return np.array([[1.0,0.0,0.0,0.0],[0.0,1.0+vec[0],vec[1],0],[0.0,0.0,1.0,0],[0.0,0.0,0.0,1.0+vec[3]])

def sim_transform(sim_mat,target_map):
    sim_mat_inv=inv(sim_mat)
    return dot(dot(sim_mat,target_map),sim_mat_inv)

def test_transform(vec,target_map):
    return SimpleCPTest(sim_transform(sim_mat_create(vec),target_map))

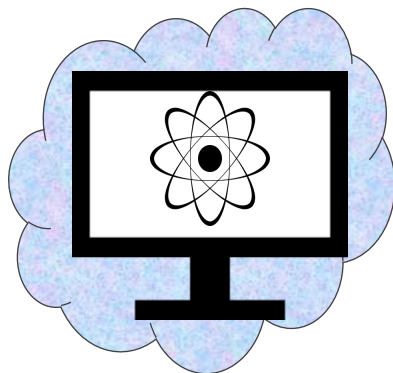
def maxfunc(vec,matx,matz):
    sim_mat=np.array([[1.0+vec[0],vec[1],0],[vec[2],1.0+vec[3],0],
#sim_mat=np.array([[1.0+vec[0],vec[1],0],[0.0,1.0,0],[0.0,0.0,1.0,0],[0.0,0.0,0.0,1.0+vec[3]])
    sim_mat_inv=inv(sim_mat)
    m1=dot(dot(sim_mat,matx),sim_mat_inv)
    m2=dot(dot(sim_mat,matz),sim_mat_inv)
    value = Two_CP_matrices(m1,m2)
    return -1*value

#li=np.array([.0,.0001,.0,.0])
li=np.array([0,0,0,0])
#li=np.array([0,0,0])
res = minimize(maxfunc, li, (gxsub, gzsub), method='nelder-mead')
output=res.x
print(test_transform(output,gzsub))
print(test_transform(output,gxsub))
print(output)

Optimization terminated successfully.
Current function value: -0.000123
Iterations: 233
Function evaluations: 415
0.000123102176507
```

# Imagine...

## Year 2027



### Row With all 1's?

```
011 ... 00001
011 ... 10111
111 ... 11111
010 ... 01001
```

```
else:
    return -1.0*(abs(test1)+abs(test2))

def im_mat_create(vec):
    return np.array([[1.0+vec[0],vec[1],0],[vec[2],1.0+vec[3],0],
                    [1.0+vec[0],vec[1],0],[0.0,1.0,0],[0.0,0.0,0.0]])

def im_mat_create_full(vec):
    return np.array([[1.0,0.0,0.0,0.0],[0.0,1.0+vec[0],vec[1],0],
                    [vec[2],1.0+vec[3],0],[0.0,0.0,0.0,1.0]])

def im_transform(sim_mat,target_map):
    im_mat_inv=inv(sim_mat)
    return dot(dot(sim_mat,target_map),sim_mat_inv)

def st_transform(vec,target_map):
    return SimpleCPTest(sim_transform(sim_mat_create(vec),target_map))

def xfunc(vec,matx,matz):
    im_mat=np.array([[1.0+vec[0],vec[1],0],[vec[2],1.0+vec[3],0],
                    [1.0+vec[0],vec[1],0],[0.0,1.0,0],[0.0,0.0,0.0,1.0]])
    im_mat_inv=inv(sim_mat)
    l=dot(dot(sim_mat,matx),sim_mat_inv)
    r=dot(dot(sim_mat,matz),sim_mat_inv)
    value = Two_CP_matrices(m1,m2)
    return -1*value

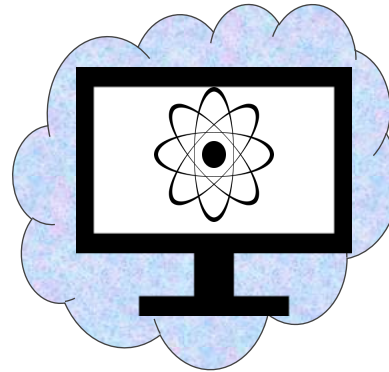
def minimize(maxfunc, li, (gxsub, gzsub), method='nelder-mead'):
    res,x = minimize(maxfunc, li, (gxsub, gzsub), method='nelder-mead')
    test_transform(output,gzsub)
    test_transform(output,gxsub)
    output

optimization terminated successfully.
    Current function value: -0.000123
    Iterations: 233
    Function evaluations: 415
122102176507
```



# Imagine...

Year 2027



Row With all 1's?

```
011 ... 00001
011 ... 10111
111 ... 11111
010 ... 01001
```

```
0 0 0
1 0 0
0 1 1
1 1 1
```

```
else:
    return -1.0*(abs(test1)+abs(test2))

lm_mat_create(vec):
return np.array([[1.0+vec[0],vec[1],0],[vec[2],1.0+vec[3],0],
return np.array([[1.0+vec[0],vec[1],0],[0.0,1.0,0],[0.0,0.0,1.0+vec[4],vec[5]]])

lm_mat_create_full(vec):
return np.array([[1.0,0.0,0.0,0.0],[0.0,1.0+vec[0],vec[1],0],[0.0,vec[2],1.0+vec[3],0],[0.0,0.0,vec[4],1.0+vec[5]]])

lm_transform(sim_mat,target_map):
lm_mat_inv=inv(sim_mat)
return dot(dot(sim_mat,target_map),sim_mat_inv)

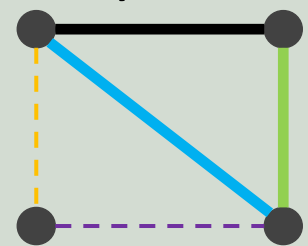
st_transform(vec,target_map):
return SimplexSolver(target_map,lm_mat_inv)

def simplex_solver(target_map,lm_mat_inv):
lm_mat=lm_mat_inv
sim_mat=np.zeros((4,4))
lm_mat_inv=inv(sim_mat)
l=dot(dot(sim_mat,target_map),lm_mat_inv)
value = TwoPhaseSimplexSolver(l,lm_mat_inv)
return -1*value

def main():
x=np.array([1,1,1,1])
array([0,1,1,1])
array([1,0,0,0])
minimize(fx,x)
:=res.x
test_trans
test_trans
output)

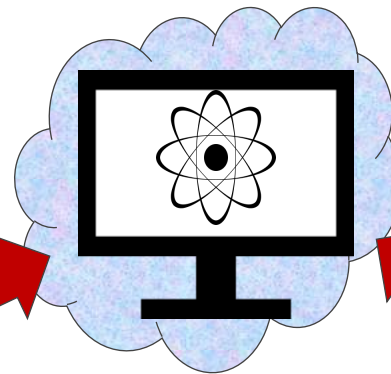
Optimization terminated successfully.
Current function value: -0.000123
Iterations: 233
Function evaluations: 415
122102176507
```

Cycle?



# Imagine...

Year 2027



How?  
Worth it?

Row With all 1's?

```
011 ... 00001
011 ... 10111
111 ... 11111
010 ... 01001
```

```
0 0 0
1 0 0
0 1 1
1 1 1
```

```
else:
    return -1.0*(abs(test1)+abs(test2))

lm_mat_create(vec):
return np.array([[1.0+vec[0],vec[1],0],[vec[2],1.0+vec[3],0],
return np.array([[1.0+vec[0],vec[1],0],[0.0,1.0,0],[0.0,0.0,1.0+vec[2],vec[3]]])

lm_mat_create_full(vec):
return np.array([[1.0,0.0,0.0,0.0],[0.0,1.0+vec[0],vec[1],0],[0.0,0.0,1.0+vec[2],vec[3]]])

lm_transform(sim_mat,target_map):
lm_mat_inv=inv(sim_mat)
return dot(dot(sim_mat,target_map),sim_mat_inv)

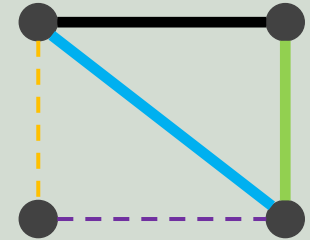
st_transform(vec,target_map):
return Simplex(target_map,vec)

def func(vec):
    sim_mat=lm_mat_create(vec)
    lm_mat_inv=inv(sim_mat)
    target_map=np.array([0.0,0.0,0.0,0.0])
    test_trans=dot(dot(sim_mat,target_map),lm_mat_inv)
    value = TwoNorm(test_trans)
    return -1*value

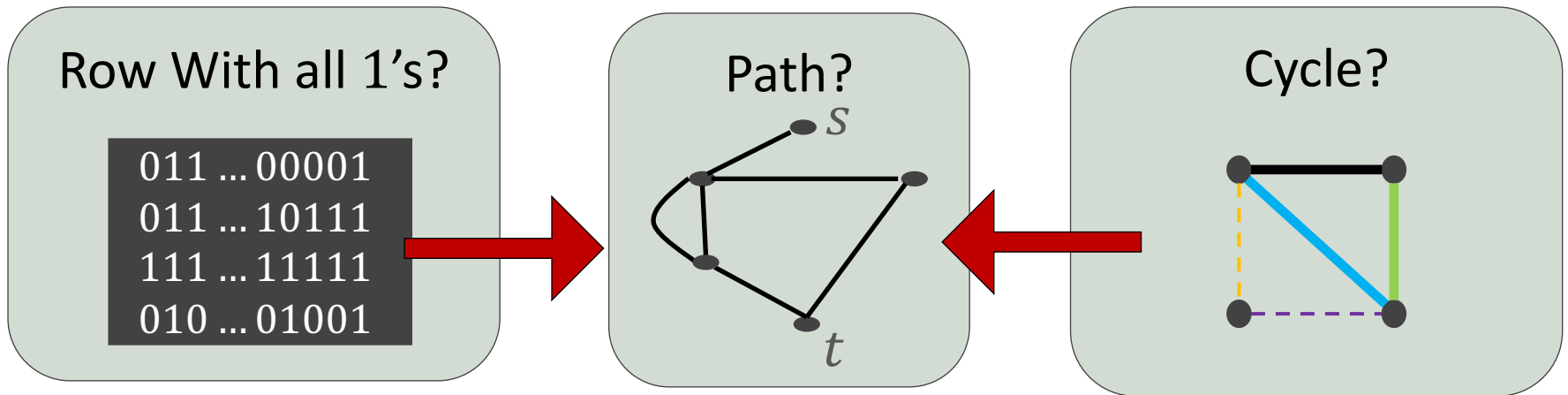
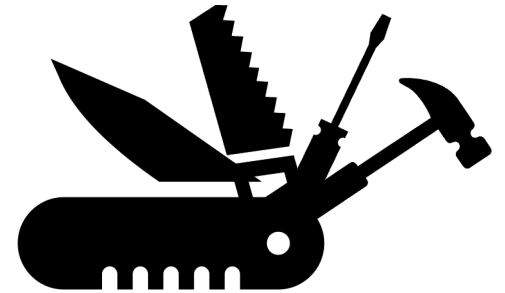
res = minimize(func,vec)
res.x
test_trans
test_trans
output)

Optimization terminated successfully.
Current function value: -0.000123
Iterations: 233
Function evaluations: 415
122102176507
```

Cycle?

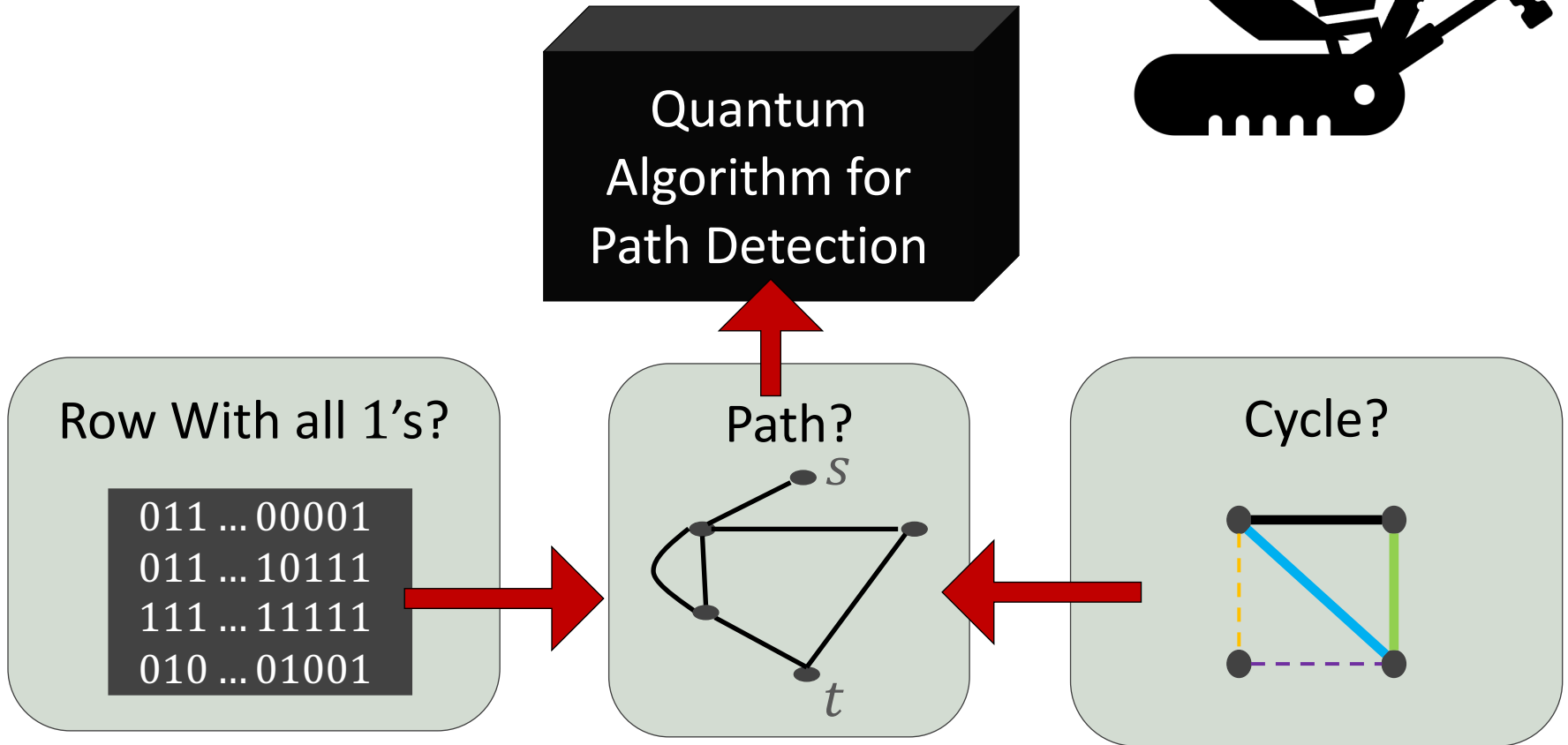
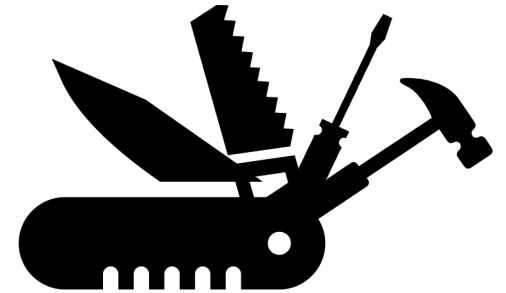


# Our Strategy



Reduce our problem to path detection

# Our Strategy

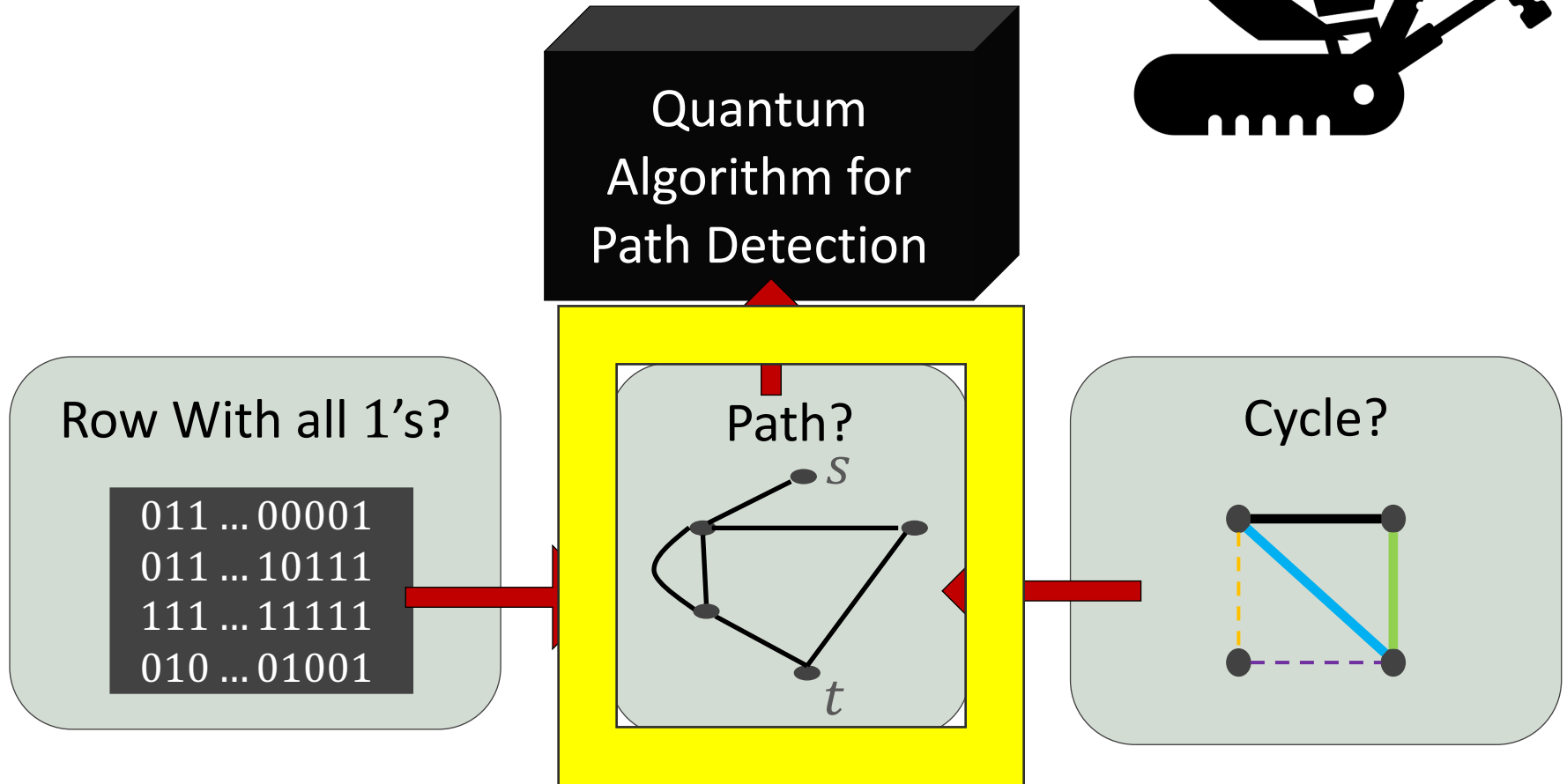
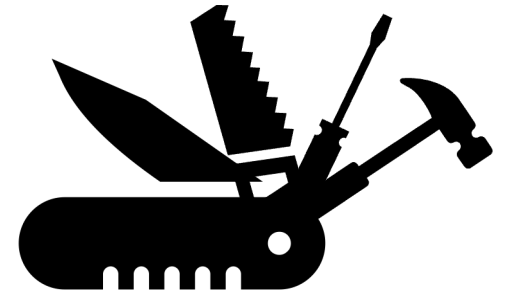


Reduce our problem to path detection

# Outline

1. Path Detection (set-up)
2. Reductions
3. Quantum Path Detection Algorithm (complexity)

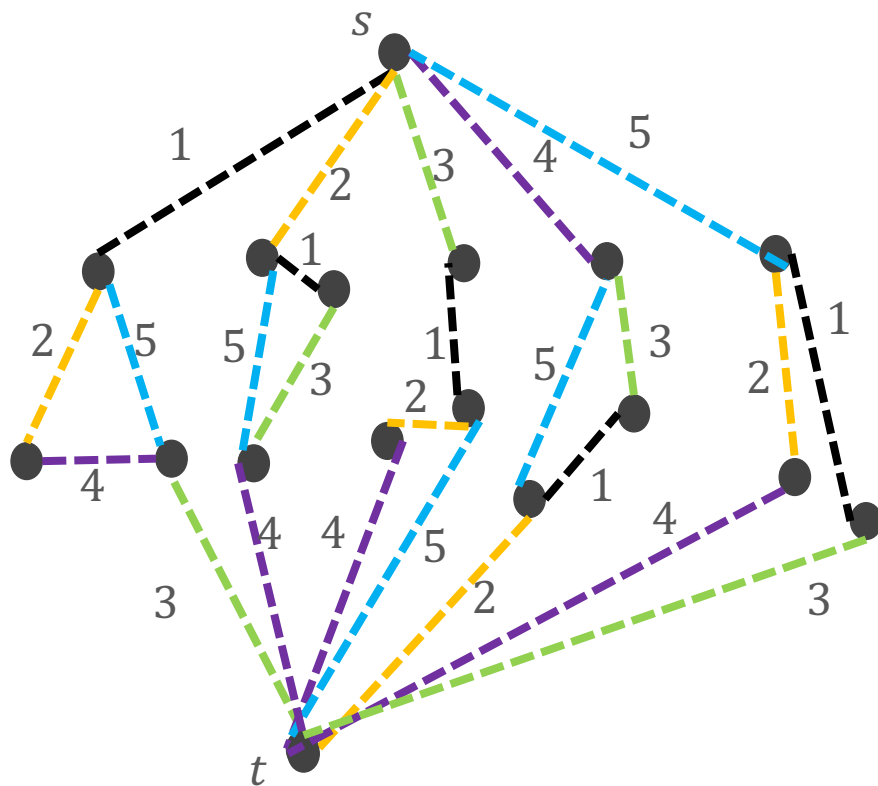
# Path Detection



Reduce our problem to path detection

# Path Detection

Is there a path from  $s$  to  $t$ ?



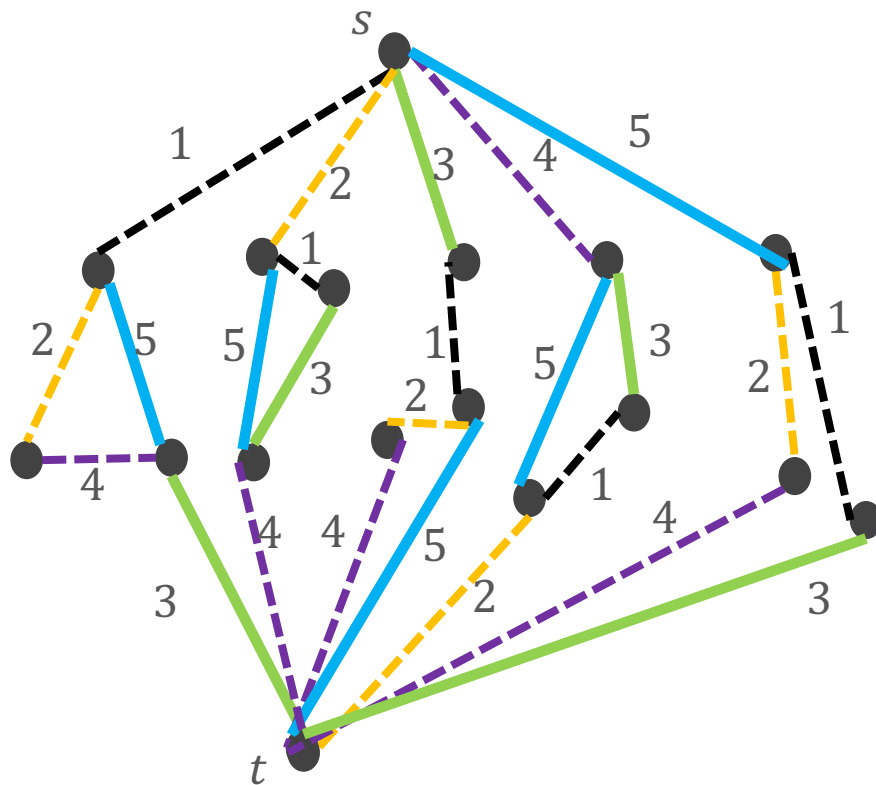
“Skeleton graph”  $G$

Bit String Input  $x$ :



# Path Detection

Is there a path from  $s$  to  $t$ ?



$G(x)$

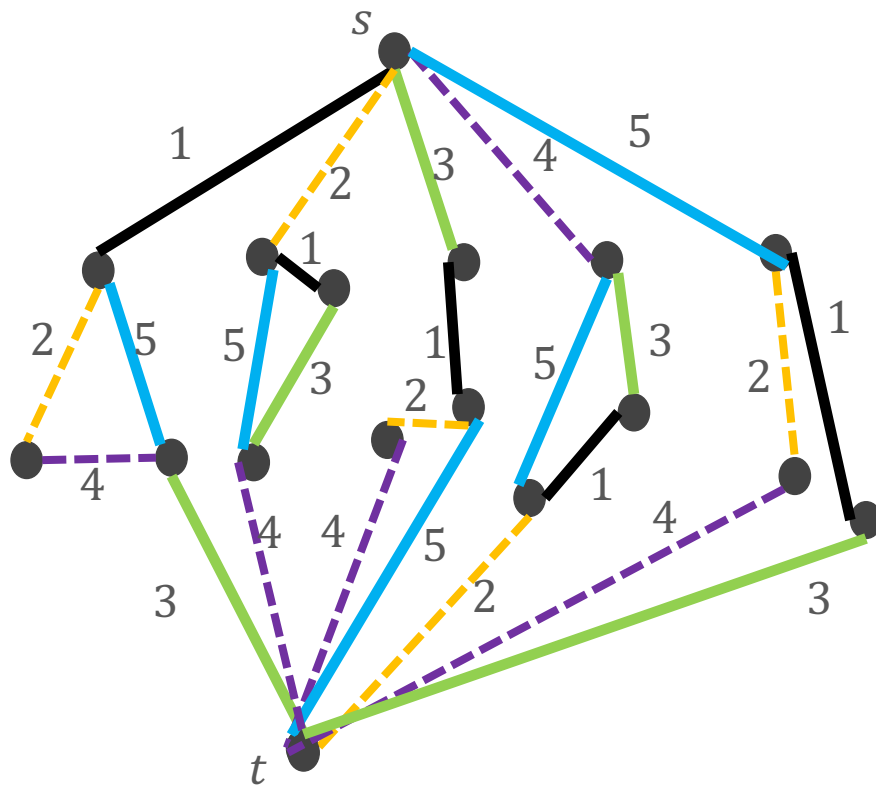
Bit String Input  $x$ :

$x_1$	$x_2$	$x_3$	$x_4$	$x_5$
0	0	1	0	1



# Path Detection

Is there a path from  $s$  to  $t$ ?



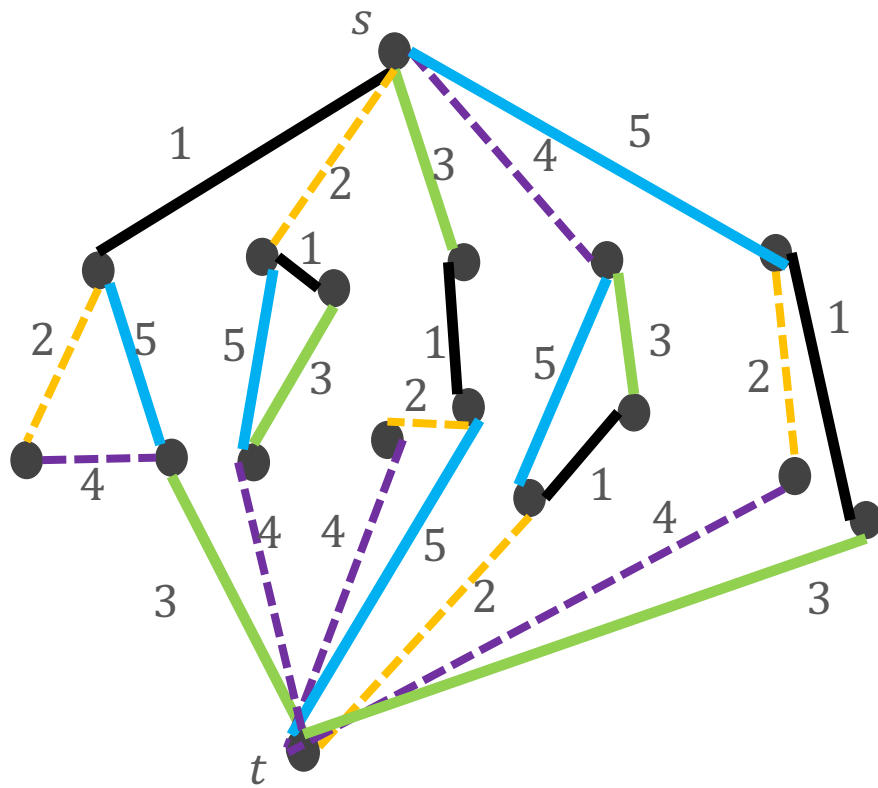
$G(x)$

Bit String Input  $x$ :

$x_1$	$x_2$	$x_3$	$x_4$	$x_5$
1	0	1	0	1

# Path Detection

Is there a path from  $s$  to  $t$ ?



$G(x)$

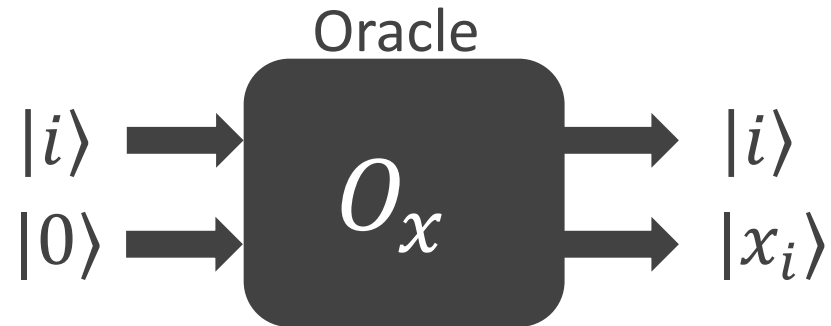
Bit String Input  $x$ :

$x_1$	$x_2$	$x_3$	$x_4$	$x_5$
1	0	1	0	1

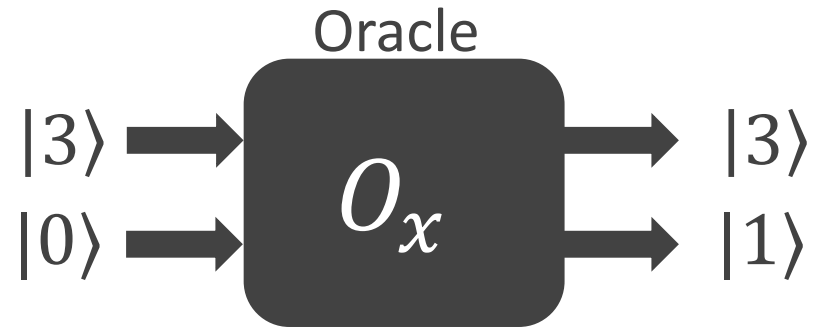
Catch:

- Bit string initially hidden

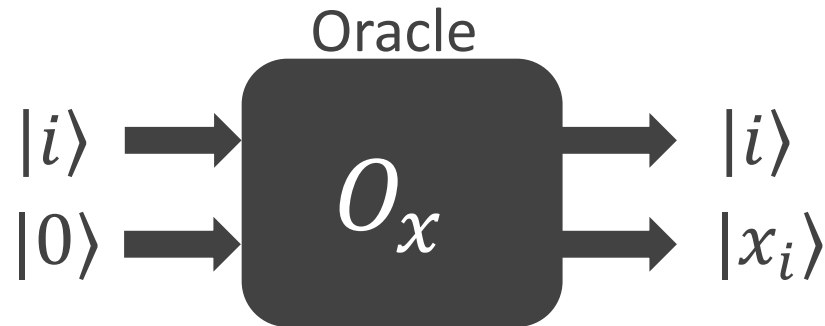
# Quantum Query



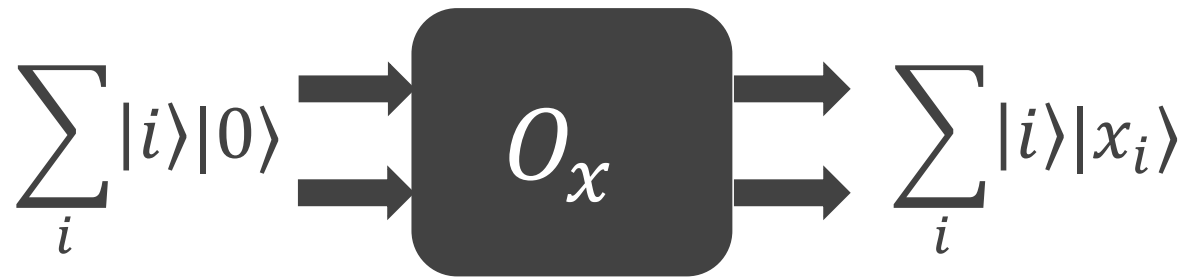
# Quantum Query



# Quantum Query

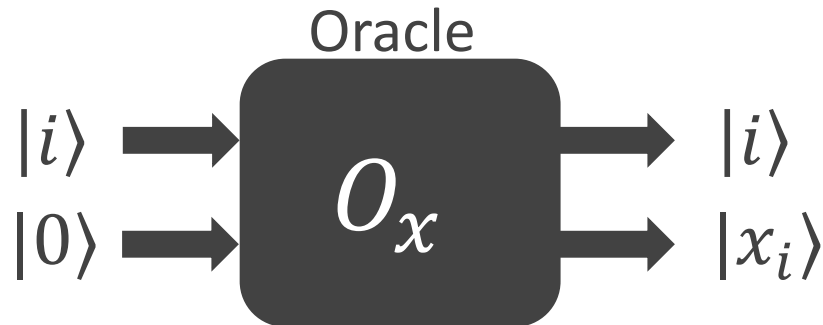


Query in superposition



(states not normalized)

# Quantum Query



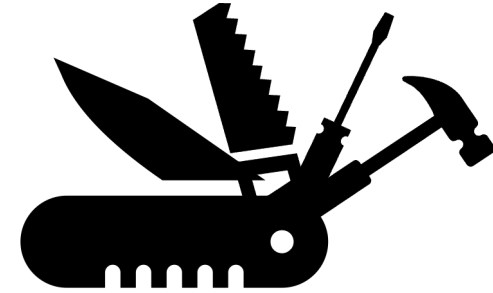
Goal: Figure out if path while using the oracle as few times as possible (while revealing as few bits as possible)

-> minimize **Query Complexity**

For our algorithms, runtime scales like query complexity, times the time it takes to do one step of a quantum walk on the skeleton graph (for most graphs with structure: log time)

# Reduction to Path Detection

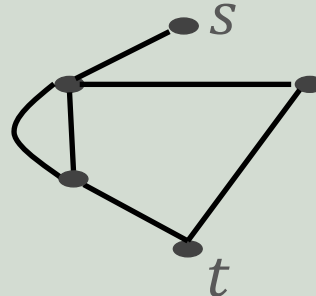
Quantum  
Algorithm for  
Path Detection



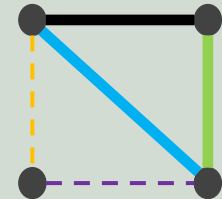
Row With all 1's?

```
011 ... 00001
011 ... 10111
111 ... 11111
010 ... 01001
```

Path?



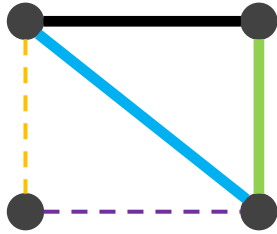
Cycle?



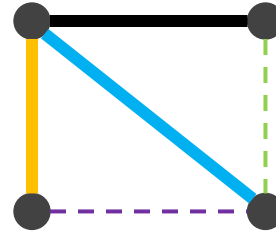
Reduce our problem to path detection

# Cycle Detection to Path Detection

Problem: Is there a cycle?



Yes

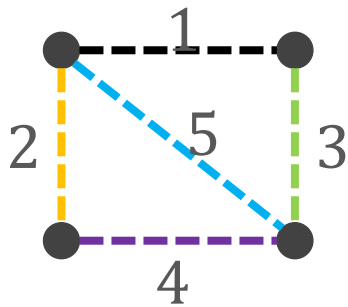


No



# Cycle Detection to Path Detection

Problem: Is there a cycle?



Unknown bit string

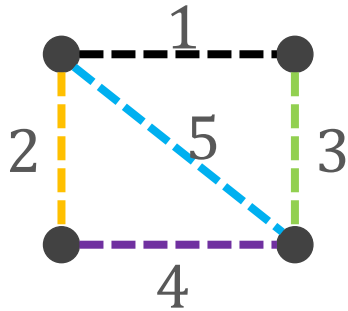
$$x_1 x_2 x_3 x_4 x_5$$

$x_i = 1 \leftrightarrow$  edge  $i$  is present

# Cycle Detection to Path Detection

Problem: Is there a cycle?

Subproblem: Is there a cycle through edge 1?



There is a cycle through

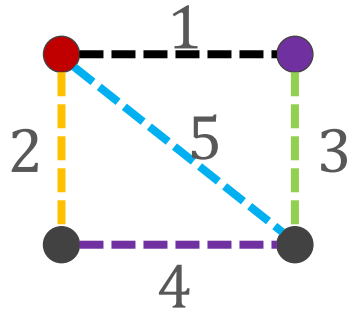
Edge 1 iff

- Edge 1 is present
  - Path between the endpoints of Edge 1 not using Edge 1
- AND

# Cycle Detection to Path Detection

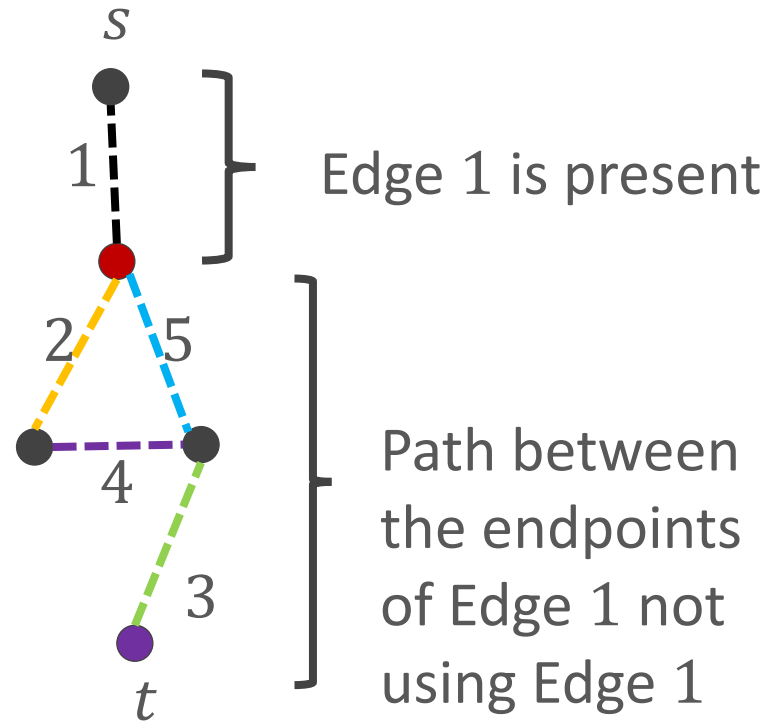
Problem: Is there a cycle?

Subproblem: Is there a cycle through edge 1?



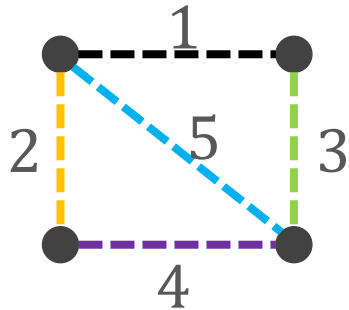
There is a cycle through  
Edge 1 iff

- Edge 1 is present
  - Path between the endpoints of Edge 1 not using Edge 1
- AND



# Cycle Detection to Path Detection

Problem: Is there a cycle?

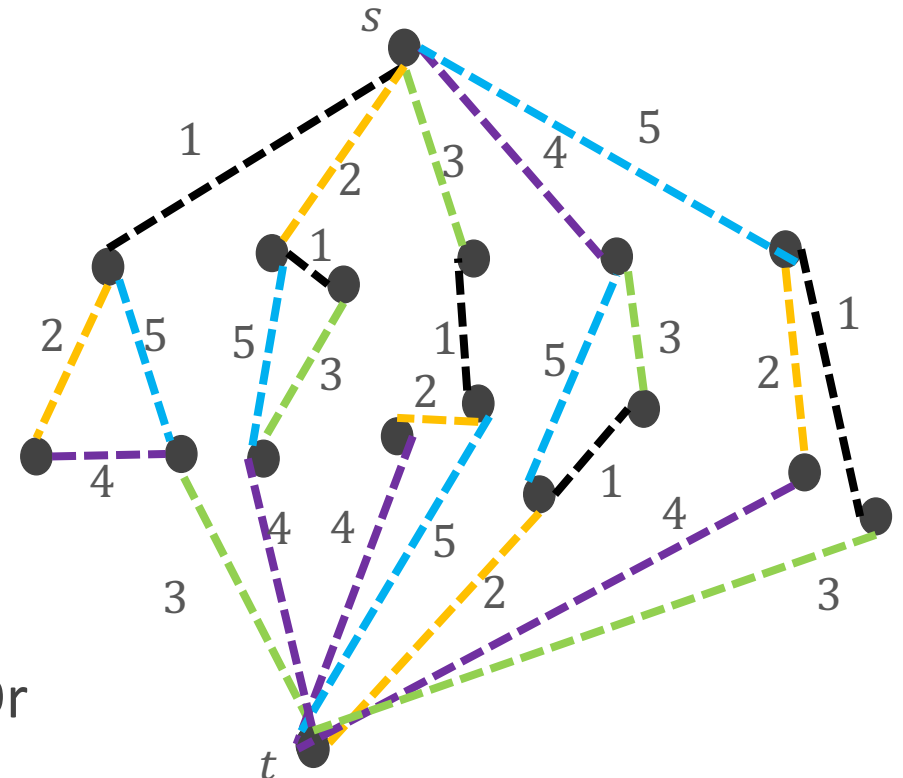
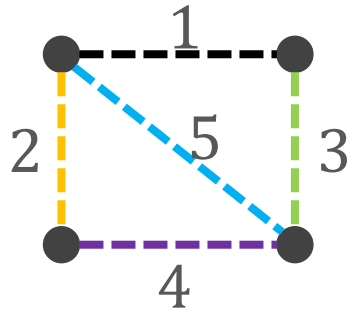


There is a cycle if

- Cycle through edge 1
  - Cycle through edge 2
  - ...
  - Cycle through edge n
- Or

# Cycle Detection to Path Detection

Problem: Is there a cycle?



There is a cycle if

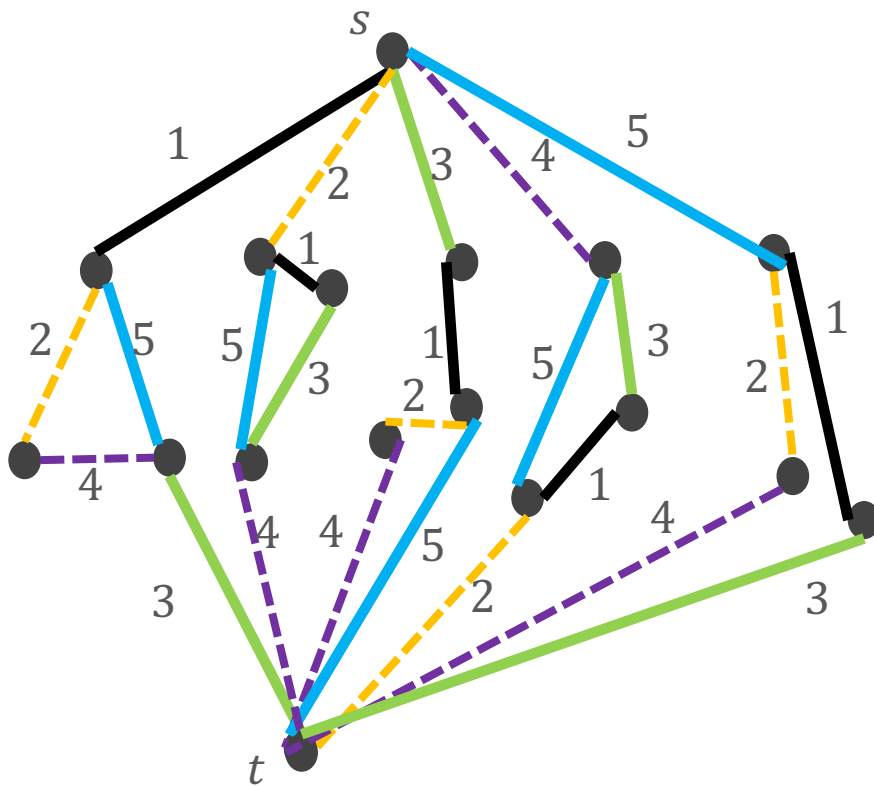
- Cycle through edge 1
  - Cycle through edge 2
  - ...
  - Cycle through edge n
- Or

# Cycle Detection to Path Detection

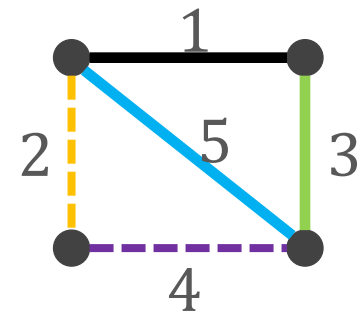
Is there a path from  $s$  to  $t$ ?

Bit String Input  $x$ :

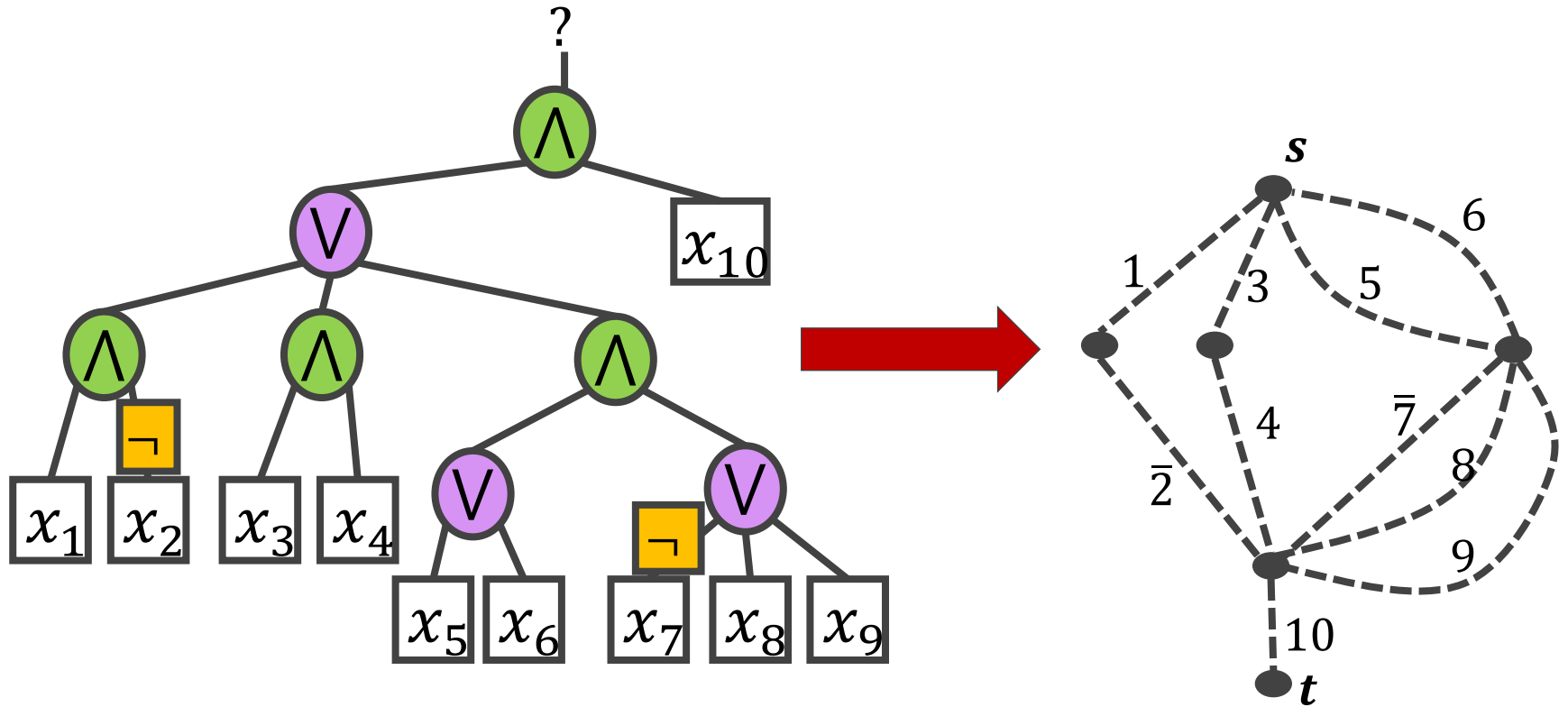
$x_1$	$x_2$	$x_3$	$x_4$	$x_5$
1	0	1	0	1



$G(x)$



# Formula Evaluation to Path Detection



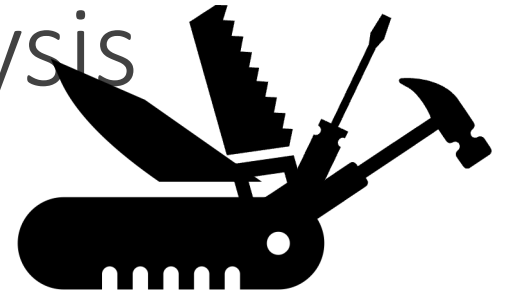
[Nisan, Ta-Shma, '95]

# Quantum Path Detection Reductions

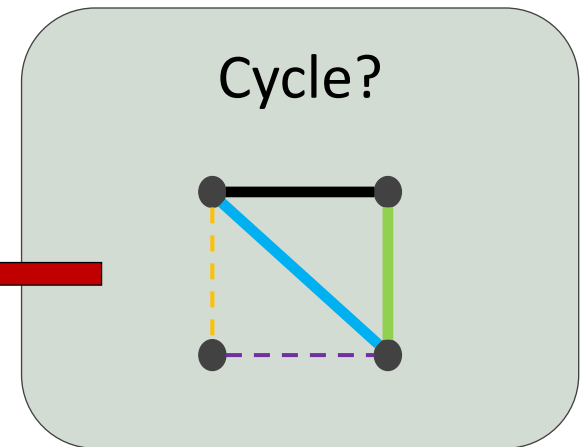
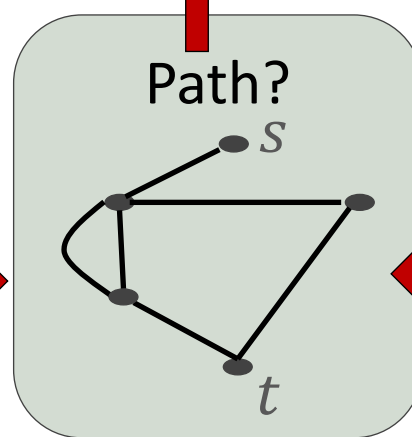
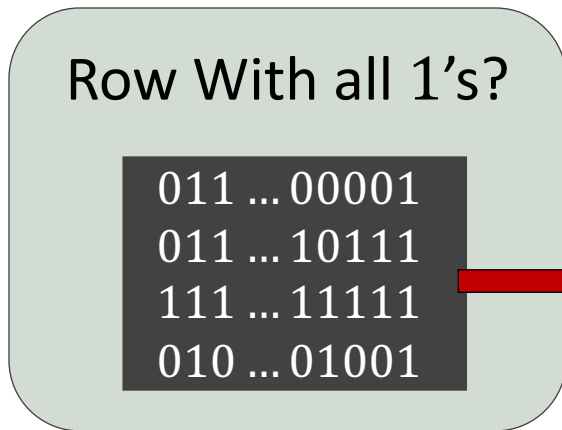
- Read-once Boolean formulas [Jeffery, K, '17]
- Total connectivity [Jarret, Jeffery, K, Piedrafita, '18]
- Cycle detection [Delorenzo, K, Witter, '19]
- Even length cycle detection [Delorenzo, K, Witter, '19]
- Bipartiteness [Delorenzo, K, Witter, '19]
- Maximum matching (K, Witter, '21)
- 2-player game evaluation [Zhan, Hassidim, K, '12]
- Directed st-connectivity (Beigi, Taghavi `19)
- Directed smallest cycle (Beigi, Taghavi `19)
- Topological sort (Beigi, Taghavi `19)
- Connected components (Beigi, Taghavi `19)
- Strongly connected components (Beigi, Taghavi `19)
- k-cycle at vertex v (Beigi, Taghavi `19)
- st-connectivity (Reichardt, Belovs `12)
- Maximum bipartite matching (Lin, Lin `16; Beigi and Taghavi `19)



# Quantum Algorithm Analysis

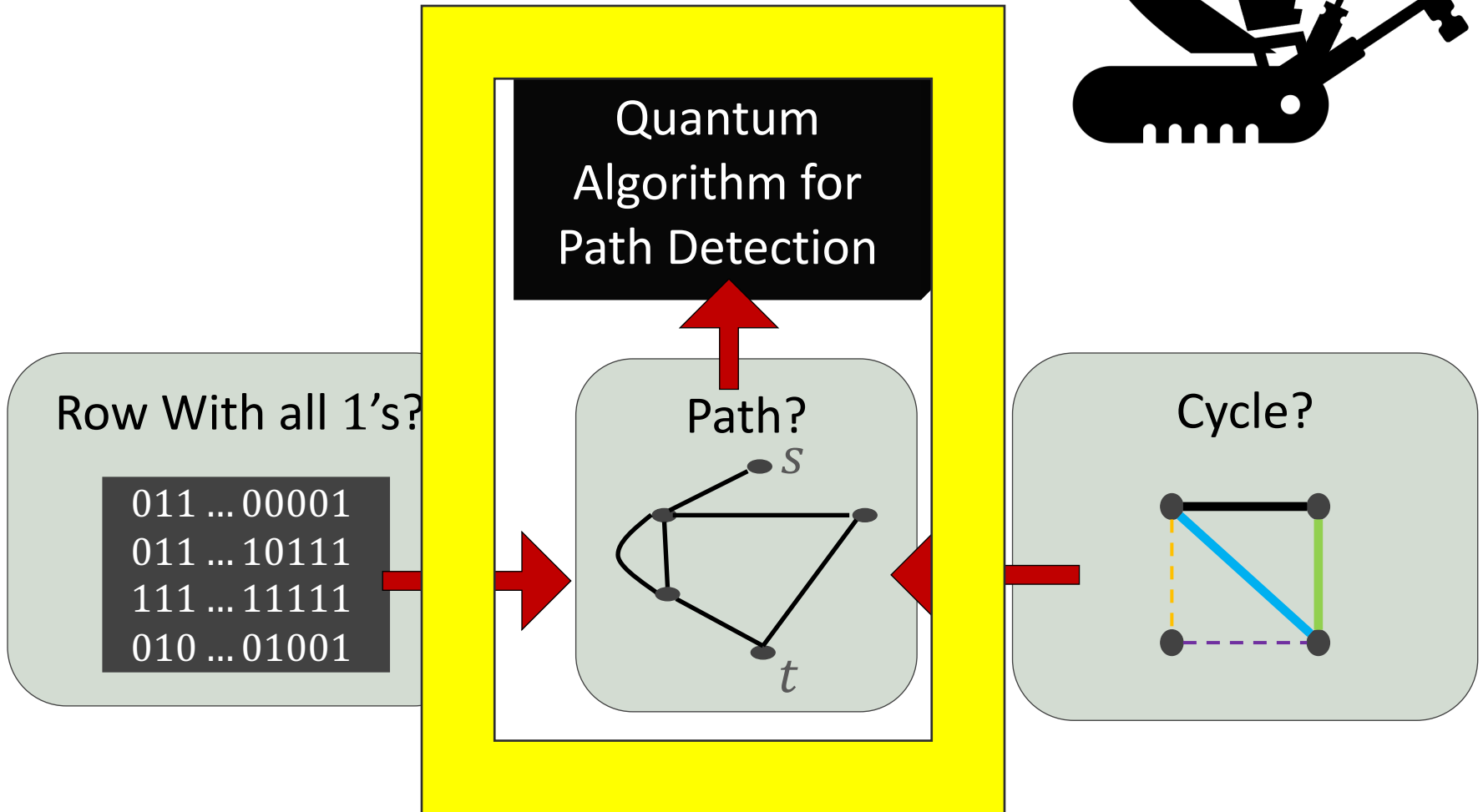
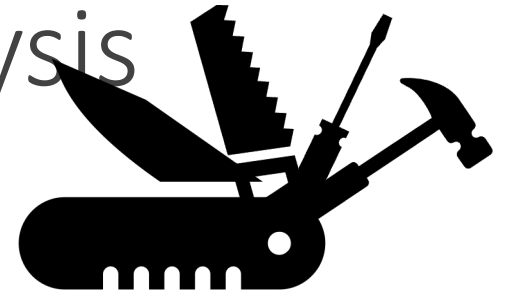


Quantum  
Algorithm for  
Path Detection



Reduce our problem to path detection

# Quantum Algorithm Analysis



Reduce our problem to path detection

# Quantum Path Detection Algorithm Complexity

Space Complexity:  $O(\log(\# \text{ edges in skeleton graph}))$

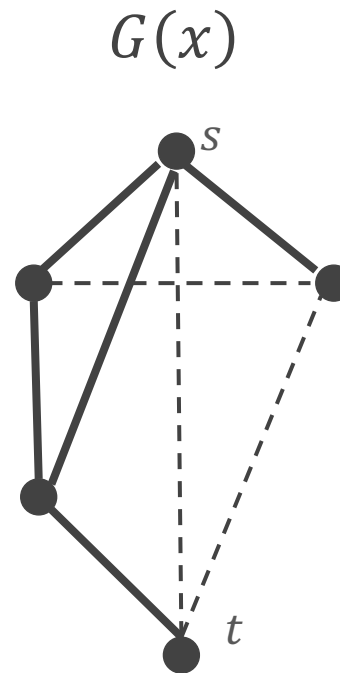
# Quantum Path Detection Algorithm Complexity

Space Complexity:  $O(\log(\# \text{ edges in skeleton graph}))$

Query Complexity: Depends on

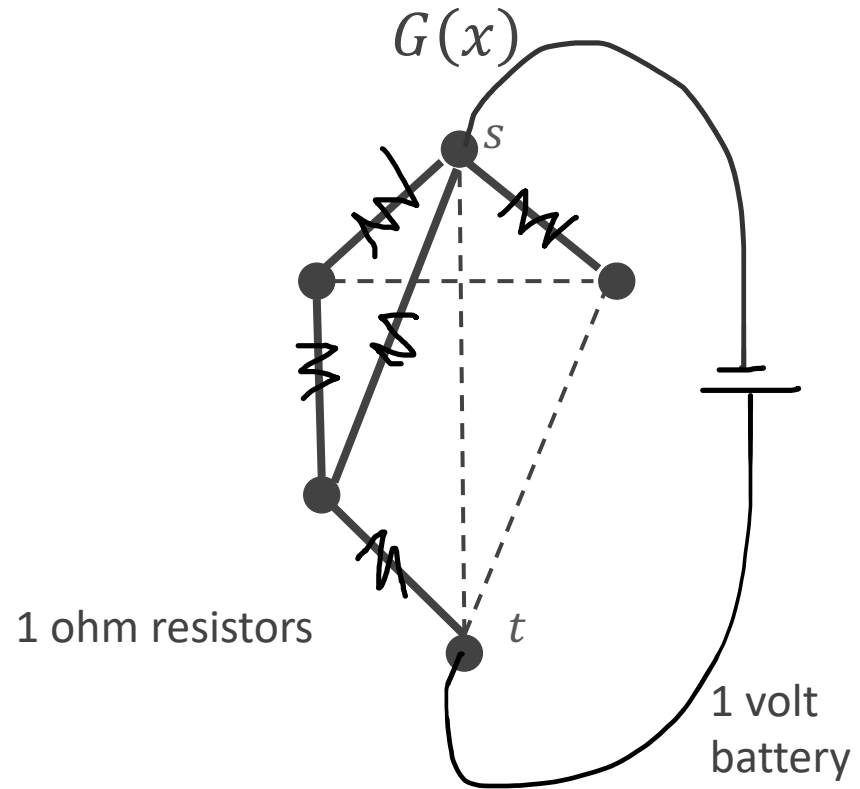
- effective resistance
- effective capacitance

# Effective Resistance



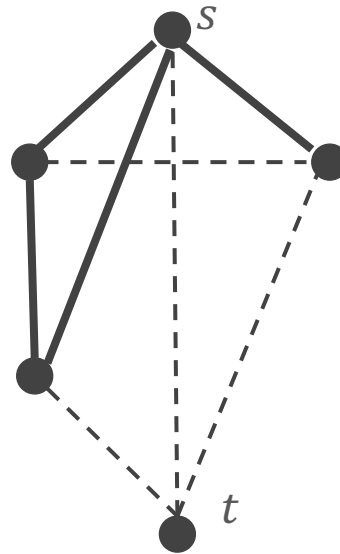
# Effective Resistance

$R(G(x))$  is effective resistance of this circuit



# Effective Capacitance

$G(x)$



# Effective Capacitance

$C(G(x))$  is effective capacitance of this circuit

0-resistance wires

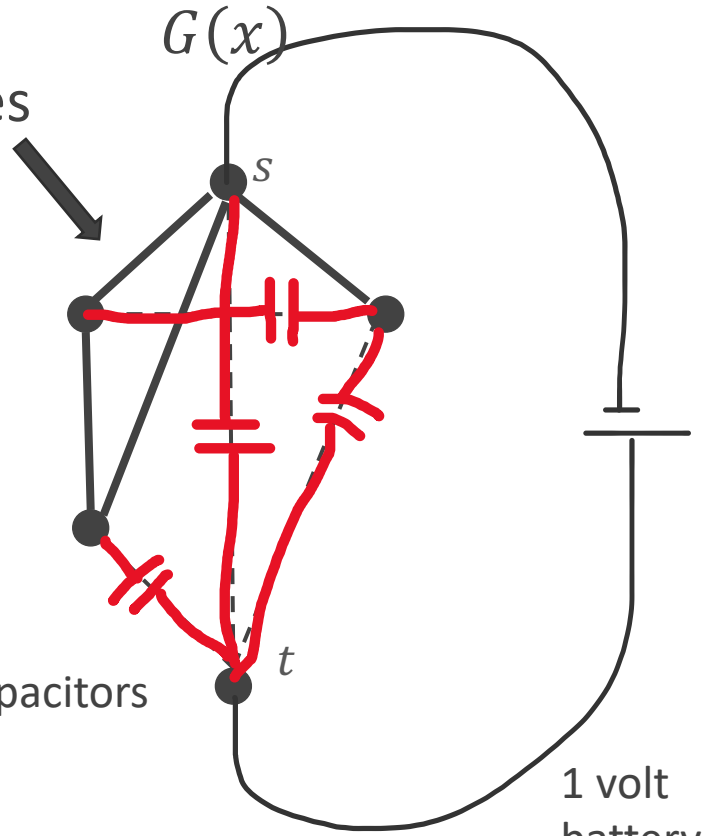
1 farad capacitors

$G(x)$

$s$

$t$

1 volt battery





# Quantum Path Detection Algorithm Complexity

Quantum Query Complexity of Path Detection:

- If path in  $G(x)$ :

$$\tilde{O} \left( \sqrt{R(G(x)) \times \max_{\substack{y:G(y) \\ \text{has no path}}} C(G(y))} \right)$$

# Quantum Path Detection Algorithm Complexity

Quantum Query Complexity of Path Detection:

- If path in  $G(x)$ :

$$\tilde{O} \left( \sqrt{R(G(x)) \times \max_{\substack{y:G(y) \\ \text{has no path}}} C(G(y))} \right)$$

- If no path in  $G(x)$ :

$$\tilde{O} \left( \sqrt{C(G(x)) \times \max_{\substack{y:G(y) \\ \text{has a path}}} R(G(y))} \right)$$

# Let's Design a Quantum Algorithm!

Search: Is there a 1 in  $x = x_1x_2 \dots x_n$ ?

# Let's Design a Quantum Algorithm!

Search: Is there a 1 in  $x = x_1x_2 \dots x_n$ ?

Is there a 1 at bit 1?

OR

Is there a 1 at bit 2?

OR

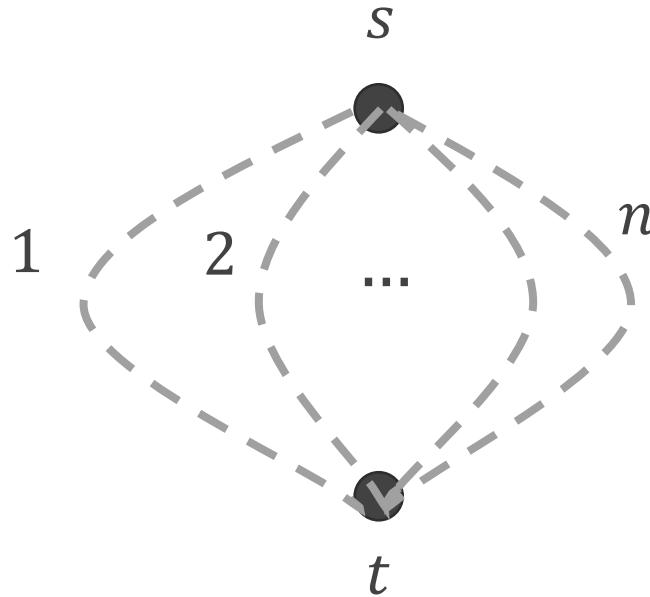
Is there a 1 at bit 3?

OR

...

# Let's Design a Quantum Algorithm!

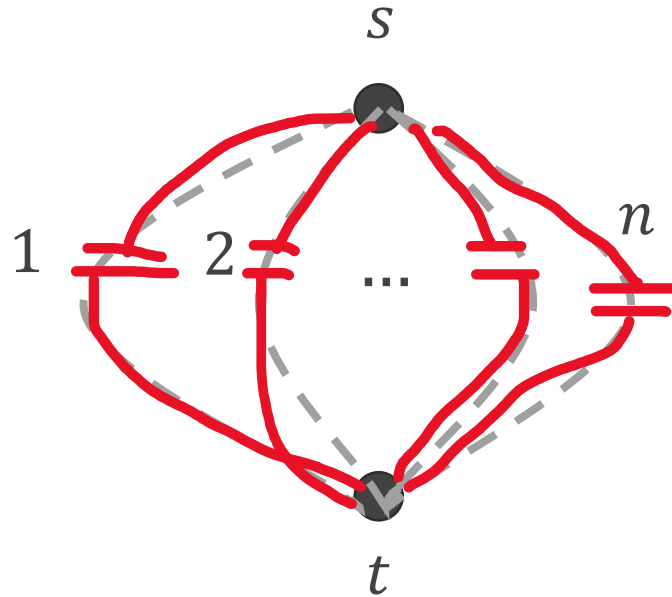
Search: Is there a 1 in  $x = x_1x_2 \dots x_n$ ?



# Let's Design a Quantum Algorithm!

No path:

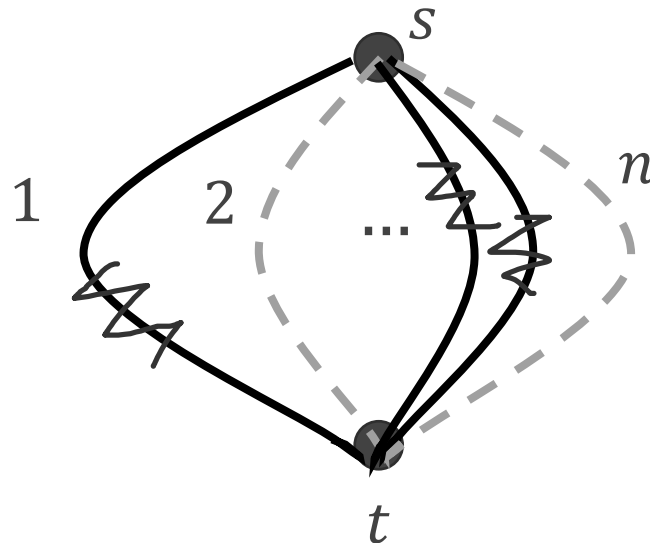
- $C(G(x)) = n$



# Let's Design a Quantum Algorithm!

Path:

- $R(G(x)) = 1/k$  ( $k$  is number of bits with value 1)



# Quantum Search Algorithm Complexity

Quantum Query Complexity of Path Detection:

- If path in  $G(x)$ :

$$\tilde{O} \left( \sqrt{R(G(x)) \times \max_{\substack{y:G(y) \\ \text{has no path}}} C(G(y))} \right)$$

- If no path in  $G(x)$ :

$$\tilde{O} \left( \sqrt{C(G(x)) \times \max_{\substack{y:G(y) \\ \text{has a path}}} R(G(y))} \right)$$



# Quantum Search Algorithm Complexity

Quantum Query Complexity of Path Detection:

- If path in  $G(x)$ :

$$\tilde{O}\left(\sqrt{R(G(x)) \times \max_{\substack{y:G(y) \\ \text{has no path}}} C(G(y))}\right) = \tilde{O}\left(\sqrt{\frac{n}{k}}\right)$$

- If no path in  $G(x)$ :

$$\tilde{O}\left(\sqrt{C(G(x)) \times \max_{\substack{y:G(y) \\ \text{has a path}}} R(G(y))}\right) = \tilde{O}(\sqrt{n})$$


Not new result (variant of Grover's Algorithm), but you designed it! (Within log factors of optimal)

# Performance

\*There are alternative optimal algorithms for some problems

- Read-once Boolean formulas (query optimal) [Jeffery, K, '17]
- Total connectivity (query optimal) [Jarret, Jeffery, K, Piedrafita, '18]
- Cycle detection (query optimal) [Delorenzo, K, Witter, '19]
- Even length cycle detection [Delorenzo, K, Witter, '19]
- Bipartiteness (query optimal) [Delorenzo, K, Witter, '19]
- Maximum matching (K, Witter, '21)
- 2-player game evaluation [Zhan, Hassidim, K, '12]
- Directed st-connectivity (query optimal) (Beigi, Taghavi '19)
- Directed smallest cycle (query optimal) (Beigi, Taghavi '19)
- Topological sort (Beigi, Taghavi '19)
- Connected components (Beigi, Taghavi '19)
- Strongly connected components (Beigi, Taghavi '19)
- k-cycle at vertex v (Beigi, Taghavi '19)
- st-connectivity (query optimal) (Reichardt, Belovs '12)
- Maximum bipartite matching (Lin, Lin '16; Beigi and Taghavi '19)

# Path Detection Algorithm:

$$U = R_1 R_2$$


Reflection that encodes quantum walk on the skeleton graph  
(Often  $O(\log n)$  time, no queries)

Reflection that encodes where edges are in graph  
(Takes  $O(1)$  time and  $O(1)$  queries)

# Path Detection Algorithm:

*Basic Algorithm (no speed up for easy instances):*

Do eigenvalue estimation of  $U$  for some initial state with high enough precision

- If initial state has eigenvalue 1 -> decide path
- Otherwise -> decide no path

# Path Detection

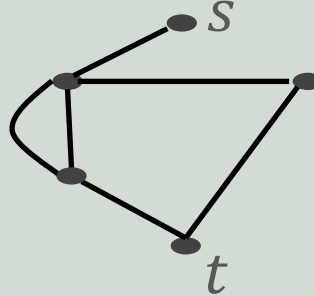


Quantum  
Algorithm for  
Path Detection

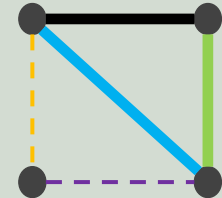
Row With all 1's?

```
011 ... 00001
011 ... 10111
111 ... 11111
010 ... 01001
```

Path?



Cycle?



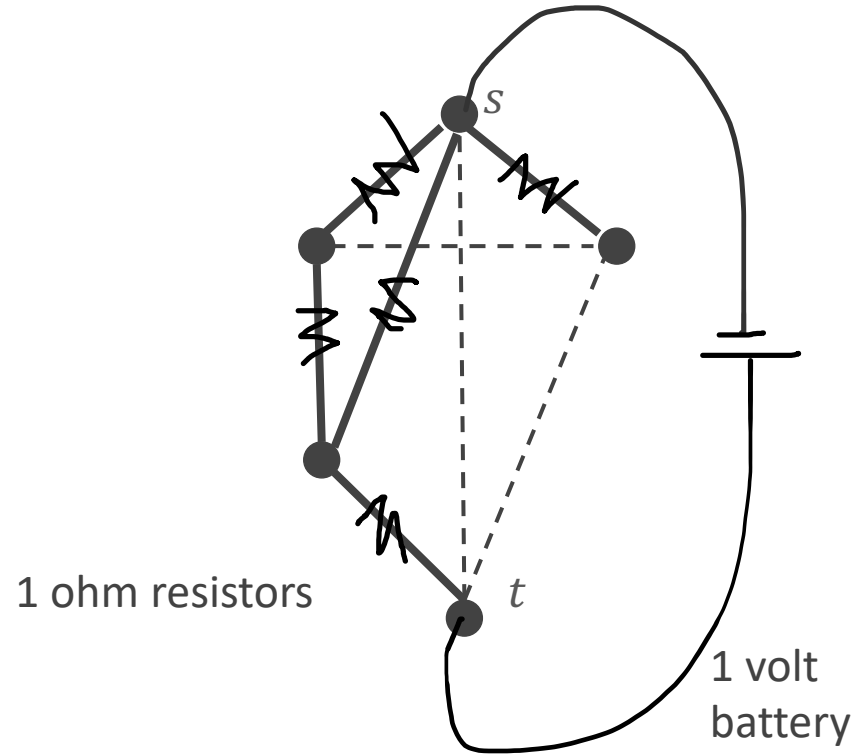
Reduce our problem to path detection

# Future/Current Work

- Recent work (with Stacey Jeffery and Alvaro Piedrafita): algorithm to sample an edge in the flow

# Effective Resistance

$R(G(x))$  is effective resistance of this circuit



# Future/Current Work

- Recent work (with Stacey Jeffery and Alvaro Piedrafita): algorithm to sample an edge in the flow
  - Can find a path faster than existing algs for graphs with only short paths
  - Can figuring out how to sabotage connections in graphs with high throughput bridges
  - Other applications?



# Future/Current Work

- Recent work (with Stacey Jeffery and Alvaro Piedrafita): algorithm to sample an edge in the flow
  - Can find a path faster than existing algs for graphs with only short paths
  - Can figuring out how to sabotage connections in graphs with high throughput bridges
  - Other applications?
- When is path detection reduction approach optimal?
- Way to find optimal weights?

# Thank you!

## Funding:

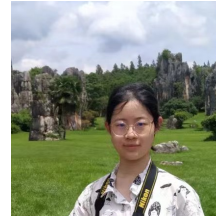


ARO

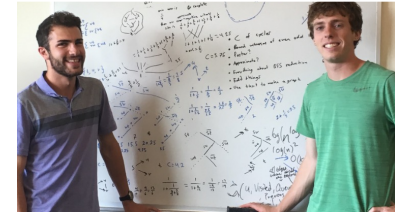
## Collaborators:



Da-Yeon Koh



Chloe Ye



Teal  
Witter

Kai  
DeLorenzo



Stacey Jeffery



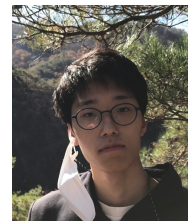
Michael  
Jarret



Alvaro  
Piedrafita



Noel  
Anderson



Jay-U  
Chung