A Multi-tool for your Quantum Algorithmic Toolbox (With a New and Improved Timer!)

Shelby Kimmel

Middlebury College

Based on work with

- Noel Anderson and Jay-U Chung (up yesterday on arxiv)
- Teal Witter arXiv:2010.02324
- 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 Algorithms for Everyday* Problems

*Formula evaluation, search, graph problems, etc

- (problems that take polynomial time in the worst case)

Quantum Algorithms for Everyday* Problems

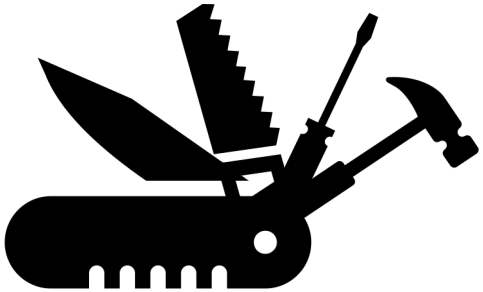
*Formula evaluation, search, graph problems, etc

- (problems that take polynomial time in the worst case)

Plethora of existing techniques:

- Grover Search
- Quantum walks (discrete/continuous)
- Span progams
- **Learning graphs (Belovs)**
- QFT

Multi-tool

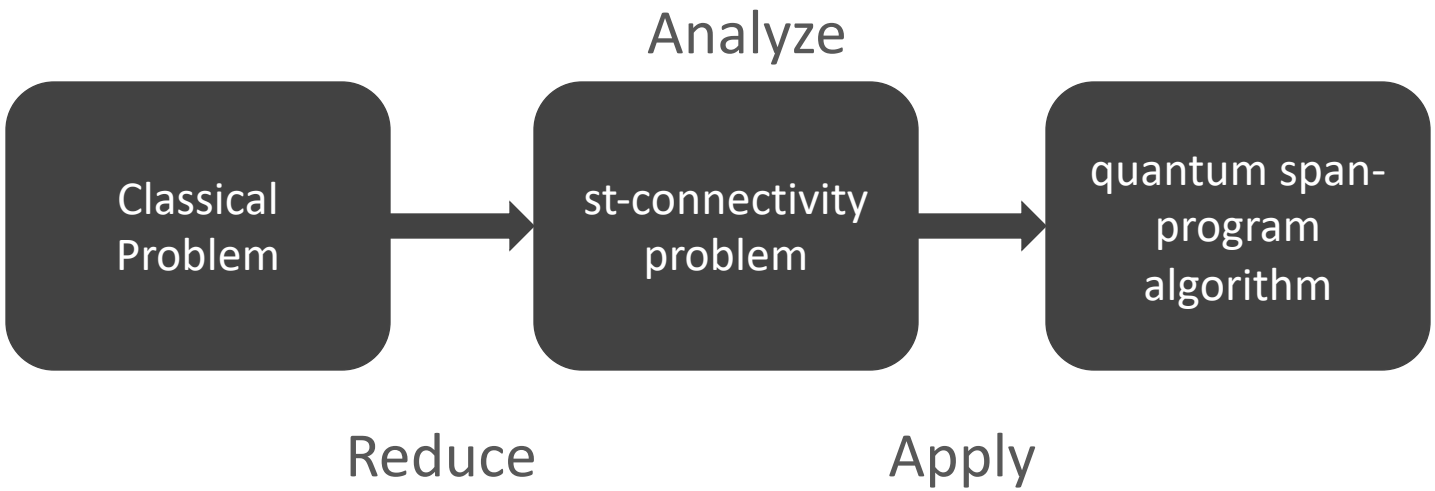


- ✓ Structured
- ✓ Unstructured
- ✓ Easy creation
- ✓ Easy, provable, non-quantum analysis

- ❖ Query model

Multi-tool

- ✓ Structured
- ✓ Unstructured
- ✓ Easy creation
- ✓ Easy, provable, non-quantum analysis

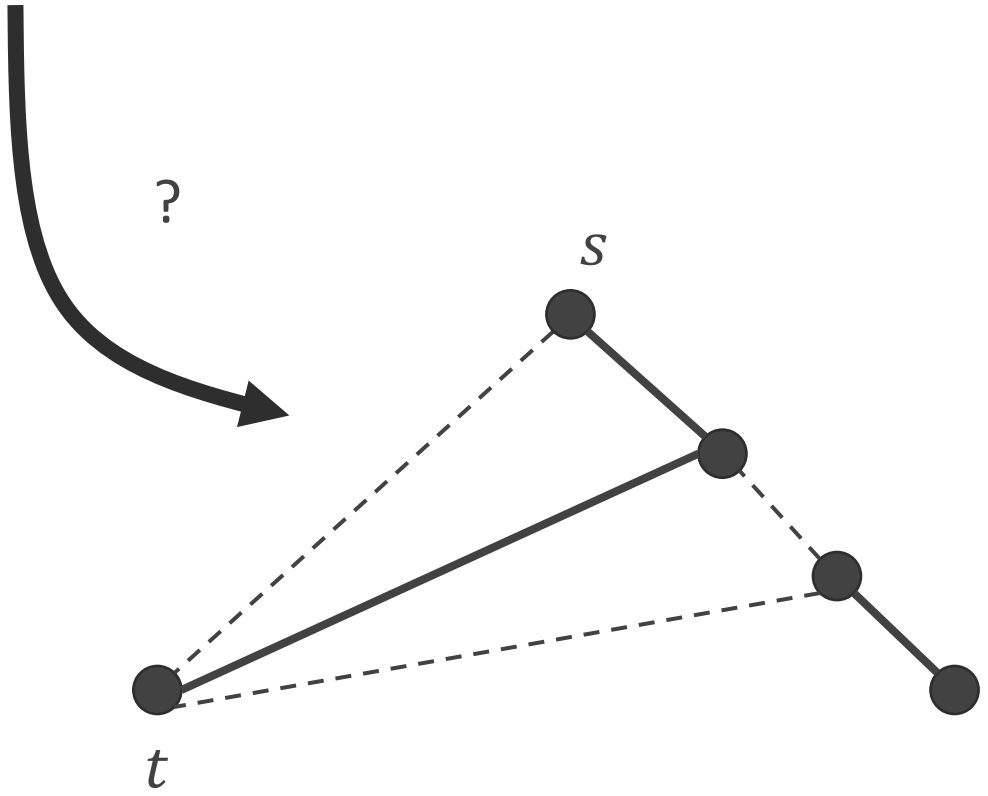


Reduction

Problem: Bit string $x = x_1 x_2 x_3$ contains a 1?

Reduction

Problem: Bit string $x = x_1 x_2 x_3$ contains a 1?



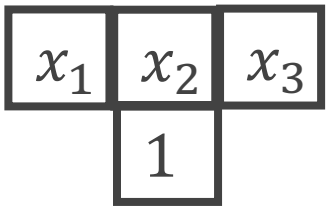
Reduction

Problem: Does a bit string $x = x_1x_2x_3$ have a property?

Initially unknown string

Reduction

Problem: Does a bit string $x = x_1x_2x_3$ have a property?

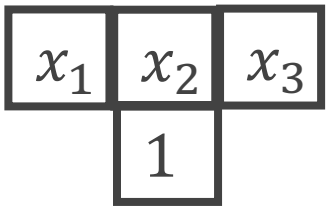


Can ask value of ith bit

$$O_x|i\rangle|b\rangle = |i\rangle|b + x_i\rangle$$

Reduction

Problem: Does a bit string $x = x_1x_2x_3$ have a property?



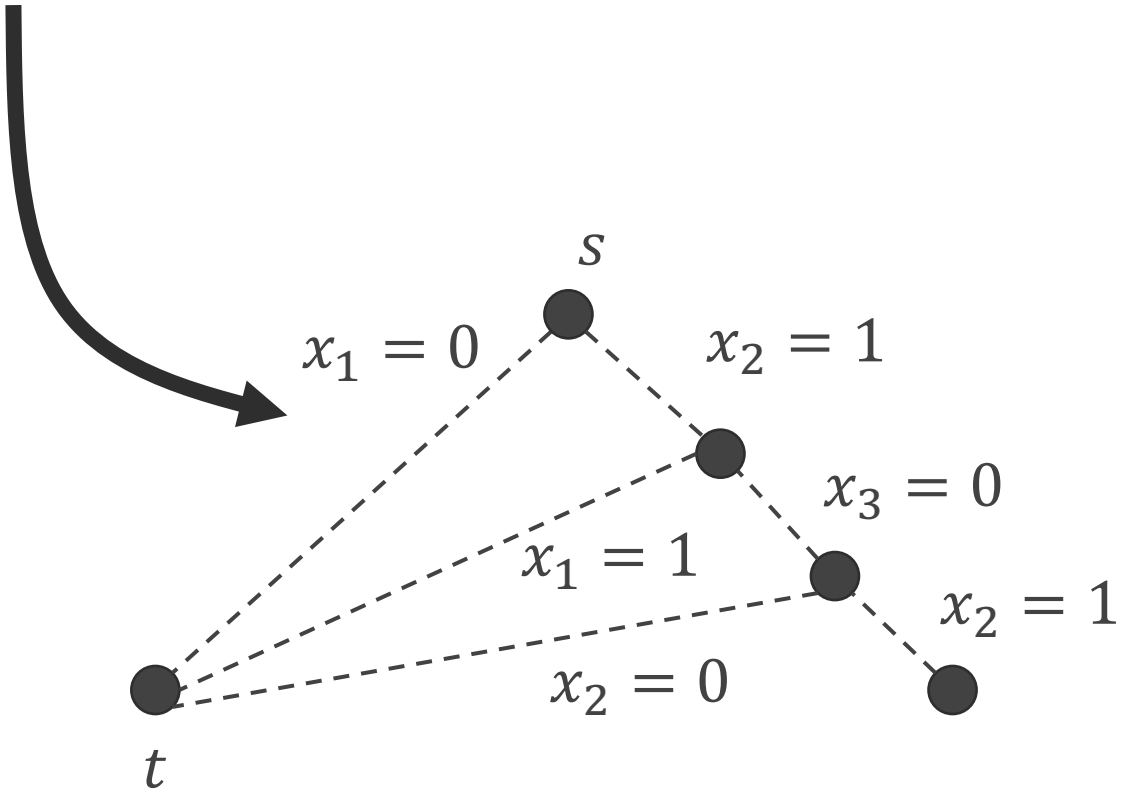
Can ask value of ith bit

$$O_x|i\rangle|b\rangle = |i\rangle|b + x_i\rangle$$

Query complexity is # of queries before can answer problem (with high probability for any input)

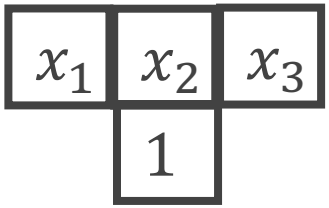
Reduction

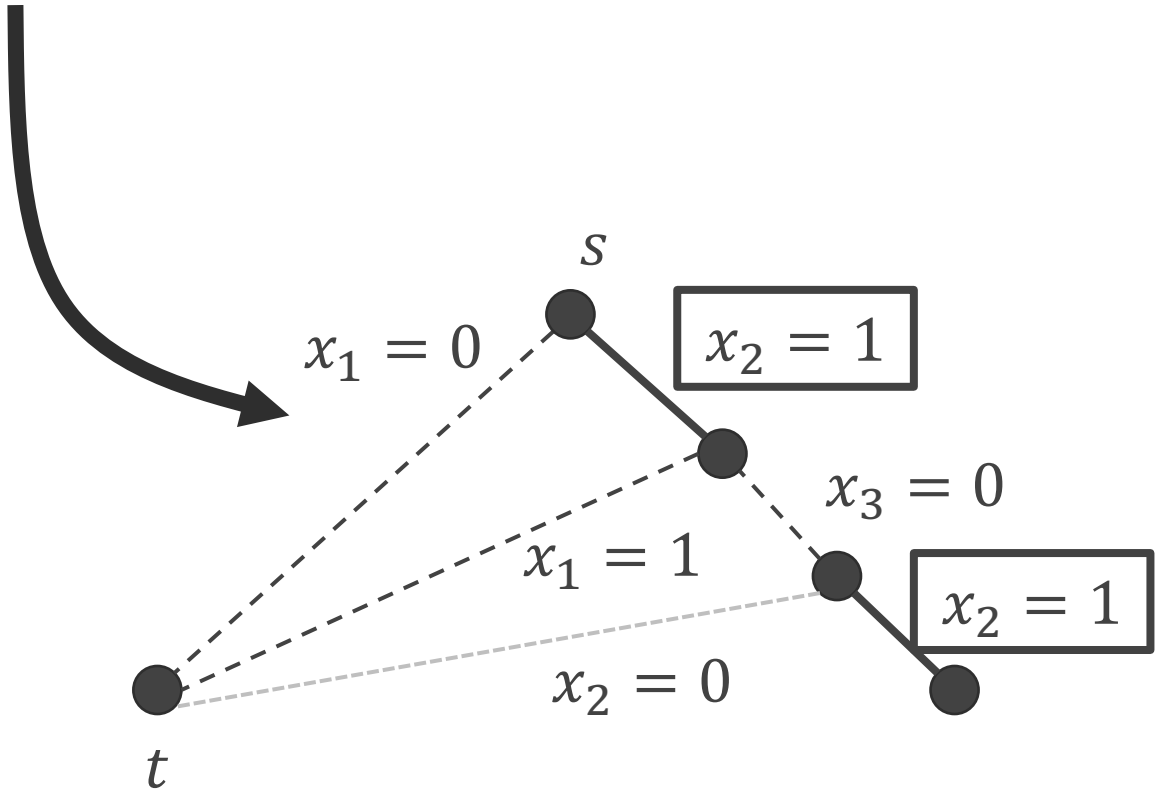
Problem: Does a bit string $x = x_1x_2x_3$ have a property?



Reduction

Problem: Does a bit string $x = x_1x_2x_3$ have a property?





Reduction (Decision Tree Approach)

Problem: Bit string $x = x_1 x_2 x_3$ contains a 1?

Idea: Turn classical algorithm into decision tree:

[Lin and Lin ‘16, Beigi and Taghavi ‘20]

Reduction (Decision Tree Approach)

Problem: Bit string $x = x_1x_2x_3$ contains a 1?

Idea: Turn classical algorithm into decision tree:

Algorithm:

- For each bit:
  -If 1: Output 1, end
- Output 0

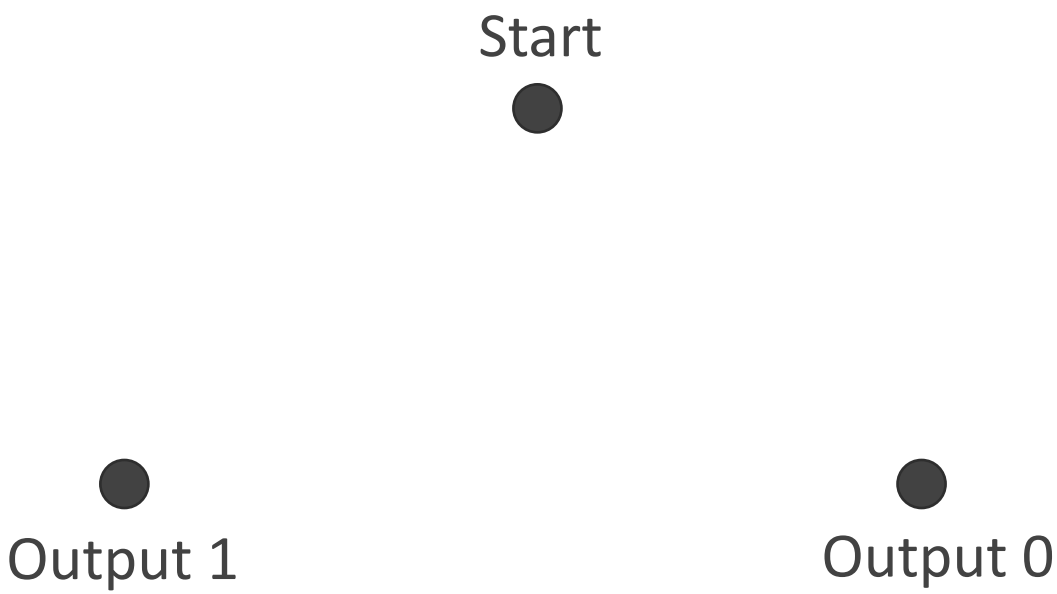
Reduction (Decision Tree Approach)

Problem: Bit string $x = x_1x_2x_3$ contains a 1?

Idea: Turn classical algorithm into decision tree:

Algorithm:

- For each bit:
  -If 1: Output 1, end
- Output 0



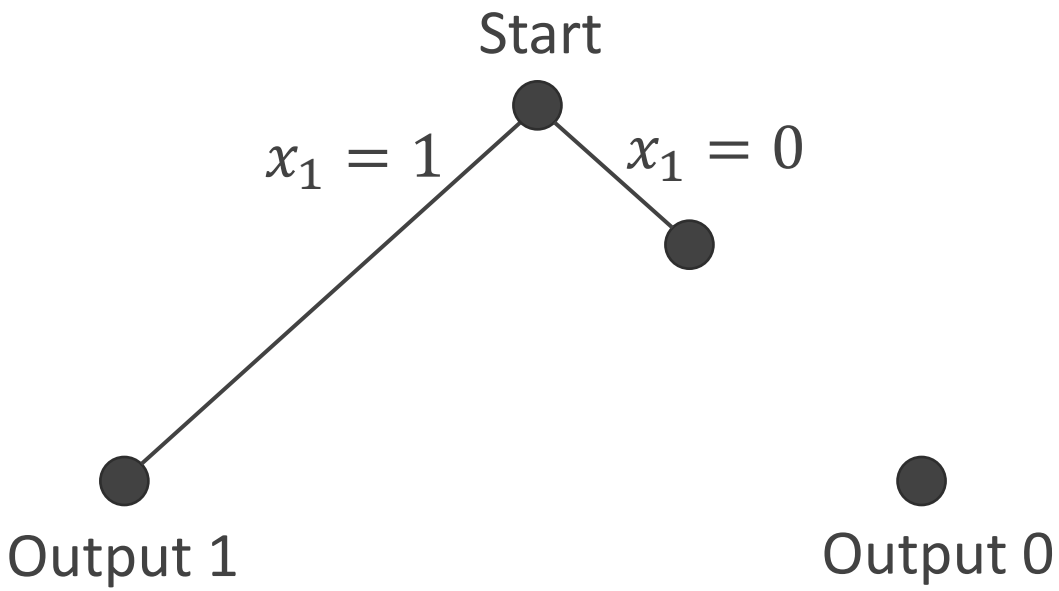
Reduction (Decision Tree Approach)

Problem: Bit string $x = x_1x_2x_3$ contains a 1?

Idea: Turn classical algorithm into decision tree:

Algorithm:

- For each bit:
  - -If 1: Output 1, end
- Output 0



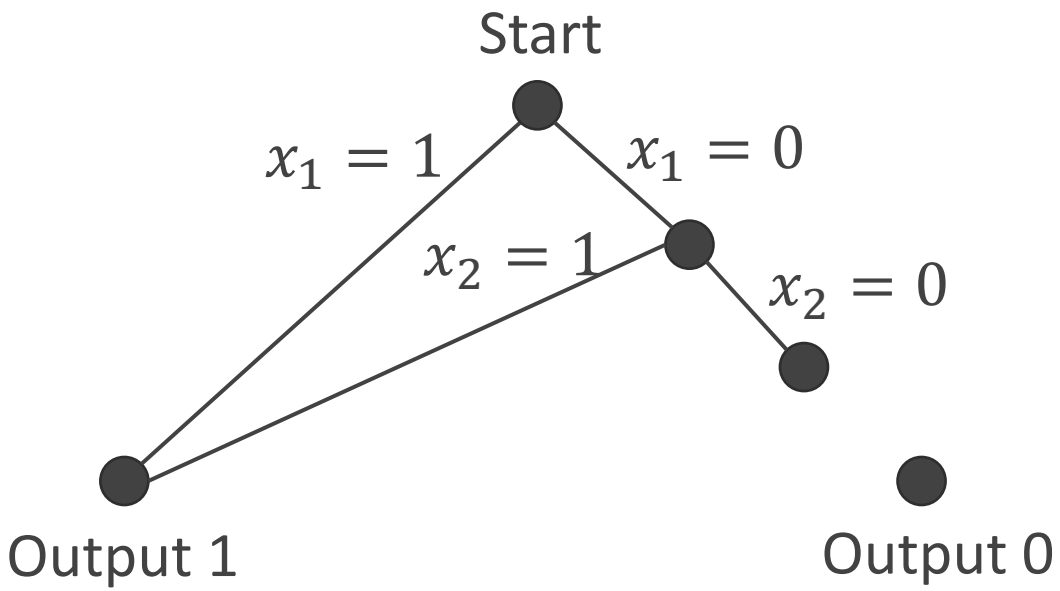
Reduction (Decision Tree Approach)

Problem: Bit string $x = x_1x_2x_3$ contains a 1?

Idea: Turn classical algorithm into decision tree:

Algorithm:

- For each bit:
  -If 1: Output 1, end
- Output 0



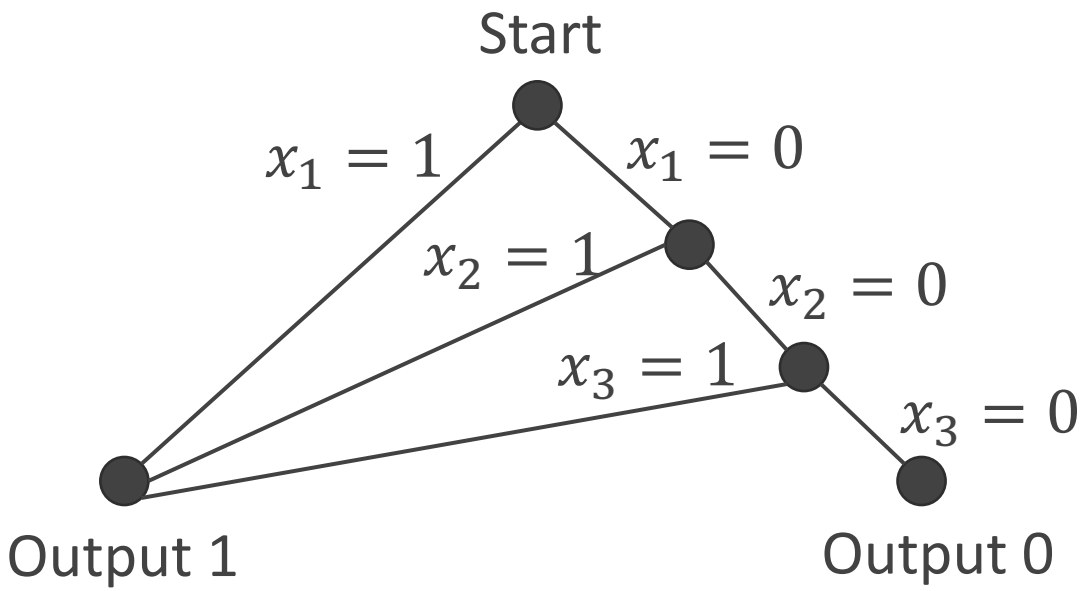
Reduction (Decision Tree Approach)

Problem: Bit string $x = x_1x_2x_3$ contains a 1?

Idea: Turn classical algorithm into decision tree:

Algorithm:

- For each bit:
  - -If 1: Output 1, end
- Output 0



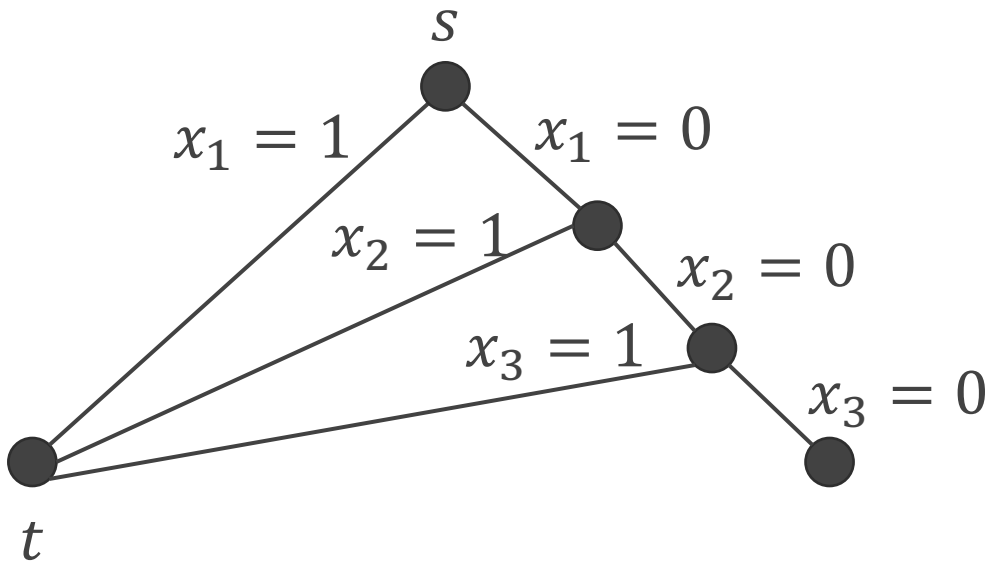
Reduction (Decision Tree Approach)

Problem: Bit string $x = x_1x_2x_3$ contains a 1?

Idea: Turn classical algorithm into decision tree:

Algorithm:

- For each bit:
  - -If 1: Output 1, end
- Output 0

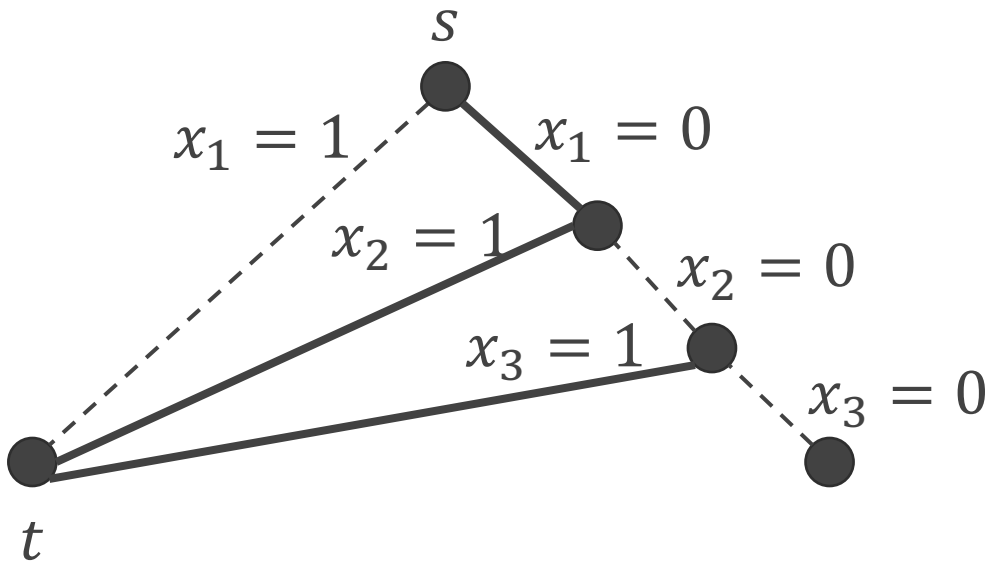


Reduction (Decision Tree Approach)

Problem: Bit string $x = x_1x_2x_3$ contains a 1?

Idea: Turn classical algorithm into decision tree:

$x = 011$

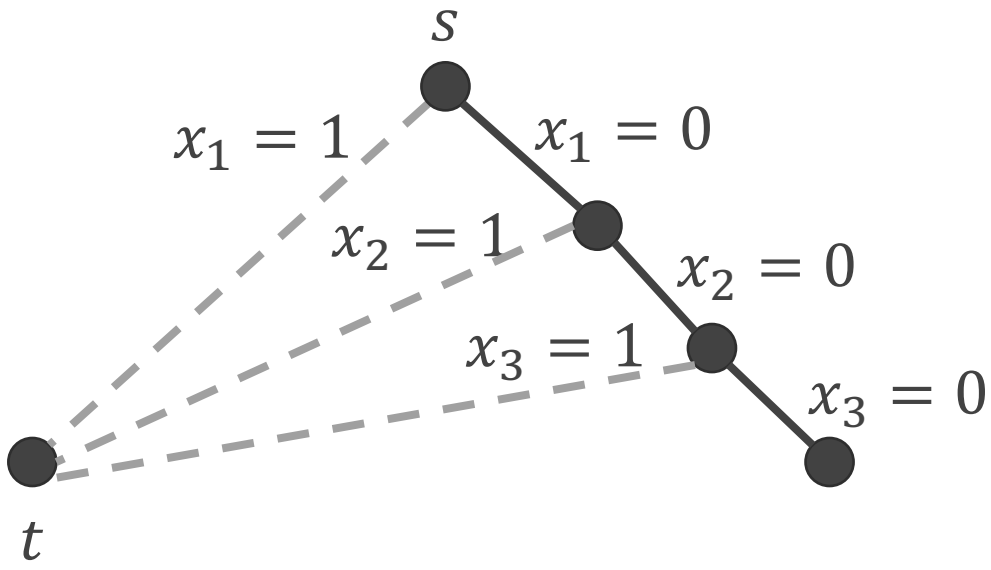


Reduction (Decision Tree Approach)

Problem: Bit string $x = x_1x_2x_3$ contains a 1?

Idea: Turn classical algorithm into decision tree:

$x = 000$



Reduction (Decision Tree Approach)

Example application: maximum matching in generic graphs

$M \leftarrow \emptyset$
loop
  /* Phase 1 */
  for *every edge* e **do** $w(e) \leftarrow$ **if** $e \in M$ **then** 2 **else** 0
  execute a search of Edmonds' weighted matching algorithm
  if *no augmenting path is found* **then halt** /* M has maximum cardinality */
  form the graph H of permissible edges

  /* Phase 2 */
  $\mathcal{P} \leftarrow$ a maximal set of vertex-disjoint augmenting paths in H
  augment M by the paths of $\mathcal{P}$

Figure 1: The high-level cardinality matching algorithm.

Gabow's classical maximum matching algorithm

procedure *find_ap_set*
initialize $\mathcal{S}$ to an empty graph and $\mathcal{P}$ to an empty set
for *each vertex* $v \in V$ **do** $b(v) \leftarrow v$ /* $b(v)$ maintains the base vertex of B_v */
for *each free vertex* f **do**
  if $f \notin V(\mathcal{P})$ **then**
    add f to $\mathcal{S}$ as the root of a new search tree
    find_ap(f)

procedure *find_ap*(x) /* x is an outer vertex */
1 **for** *each edge* $xy \notin M$ **do** /* scan xy from x */
  if $y \notin V(\mathcal{S})$ **then**
    if y *is free* **then** /* y completes an augmenting path */
2   add xy to $\mathcal{S}$ and add path $yP(x)$ to $\mathcal{P}$
      terminate every currently executing recursive call to *find_ap*
    else /* grow step */
      add xy, yy' to $\mathcal{S}$, where $yy' \in M$
      find_ap(y')
3   **else if** $b(y)$ *is an outer proper descendant of* $b(x)$ *in* $\mathcal{S}^-$ **then** /* blossom step */
    /* equivalent test: $b(y)$ became outer strictly after $b(x)$ */
    let u_i, $i = 1, \ldots, k$ be the inner vertices in $P(y, b(x))$, ordered so u_i precedes u_{i-1}
    for $i \leftarrow 1$ **to** k **do**
4   **for** *every vertex* v *with* $b(v) \in \{u_i, u_i'\}$, *where* $u_i u_i' \in M$ **do** $b(v) \leftarrow b(x)$
      /* this executes the blossom step for xy. each u_i is now outer. */
    for $i \leftarrow 1$ **to** k **do** *find_ap*(u_i)
5   **return**

Figure 4: Path-preserving depth-first search.

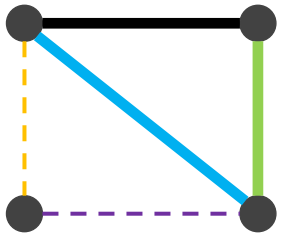
procedure *find_ap_set*
initialize $\mathcal{S}$ to an empty graph and $\mathcal{P}$ to an empty set
for *each vertex* $v \in V$ **do** $b(v) \leftarrow v$ /* $b(v)$ maintains the base vertex of B_v */
for *each free vertex* f **do**
  if $f \notin V(\mathcal{P})$ **then**
    add f to $\mathcal{S}$ as the root of a new search tree
    find_ap(f)

procedure *find_ap*(x) /* x is an outer vertex */
1 **for** *each edge* $xy \notin M$ **do** /* scan xy from x */
  if $y \notin V(\mathcal{S})$ **then**
    if y *is free* **then** /* y completes an augmenting path */
2   add xy to $\mathcal{S}$ and add path $yP(x)$ to $\mathcal{P}$
      terminate every currently executing recursive call to *find_ap*
    else /* grow step */
      add xy, yy' to $\mathcal{S}$, where $yy' \in M$
      find_ap(y')
3   **else if** $b(y)$ *is an outer proper descendant of* $b(x)$ *in* $\mathcal{S}^-$ **then** /* blossom step */
    /* equivalent test: $b(y)$ became outer strictly after $b(x)$ */
    let u_i, $i = 1, \ldots, k$ be the inner vertices in $P(y, b(x))$, ordered so u_i precedes u_{i-1}
    for $i \leftarrow 1$ **to** k **do**
4   **for** *every vertex* v *with* $b(v) \in \{u_i, u_i'\}$, *where* $u_i u_i' \in M$ **do** $b(v) \leftarrow b(x)$
      /* this executes the blossom step for xy. each u_i is now outer. */
    for $i \leftarrow 1$ **to** k **do** *find_ap*(u_i)
5   **return**

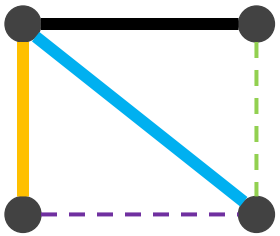
Figure 4: Path-preserving depth-first search.

Reduction (Parallel/Series Approach)

Problem: Is there a cycle?



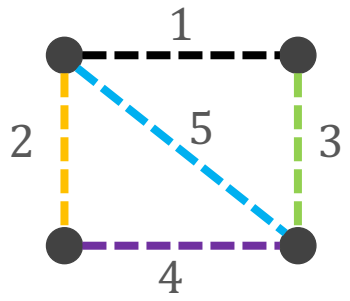
Yes



No

Reduction (Parallel/Series Approach)

Problem: Is there a cycle?



Hidden bit string

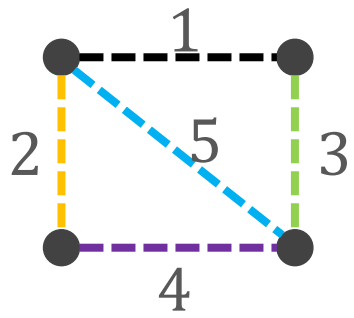
$$x_1 x_2 x_3 x_4 x_5$$

$x_i = 1 \leftrightarrow$ edge i is present

Reduction (Parallel/Series Approach)

Problem: Is there a cycle?

Subproblem: Is there a cycle through edge 1?



There is a cycle through Edge 1 iff

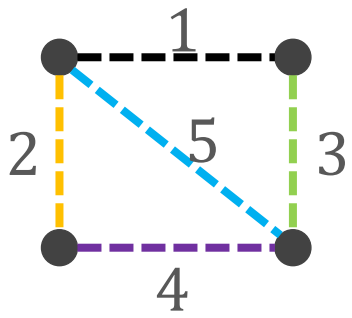
- Edge 1 is present
- Path between the endpoints of Edge 1 not using Edge 1

AND

Reduction (Parallel/Series Approach)

Problem: Is there a cycle?

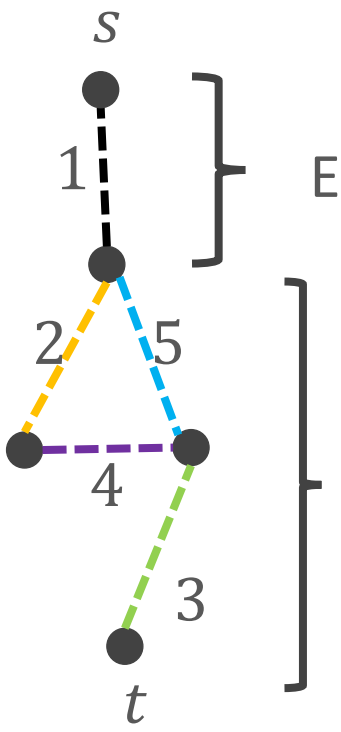
Subproblem: Is there a cycle through edge 1?



There is a cycle through Edge 1 iff

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

AND

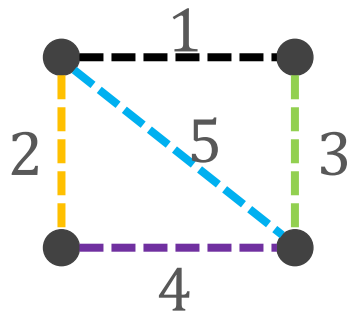


Edge 1 is present

Path between the endpoints of Edge 1 not using Edge 1

Reduction (Parallel/Series Approach)

Problem: Is there a cycle?



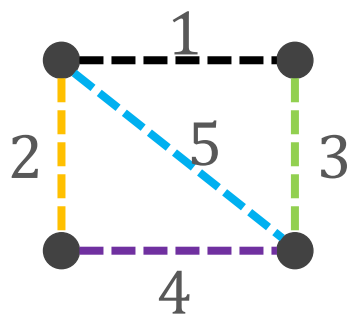
There is a cycle if

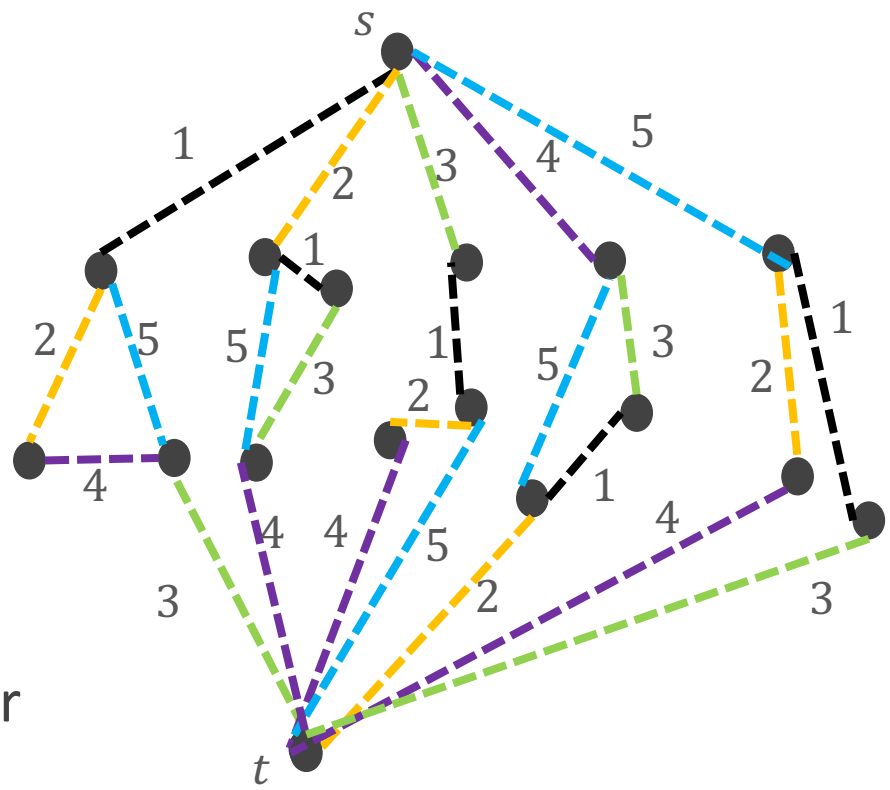
- Cycle through edge 1
- Cycle through edge 2
- …
- Cycle through edge n

Or

Reduction (Parallel/Series Approach)

Problem: Is there a cycle?





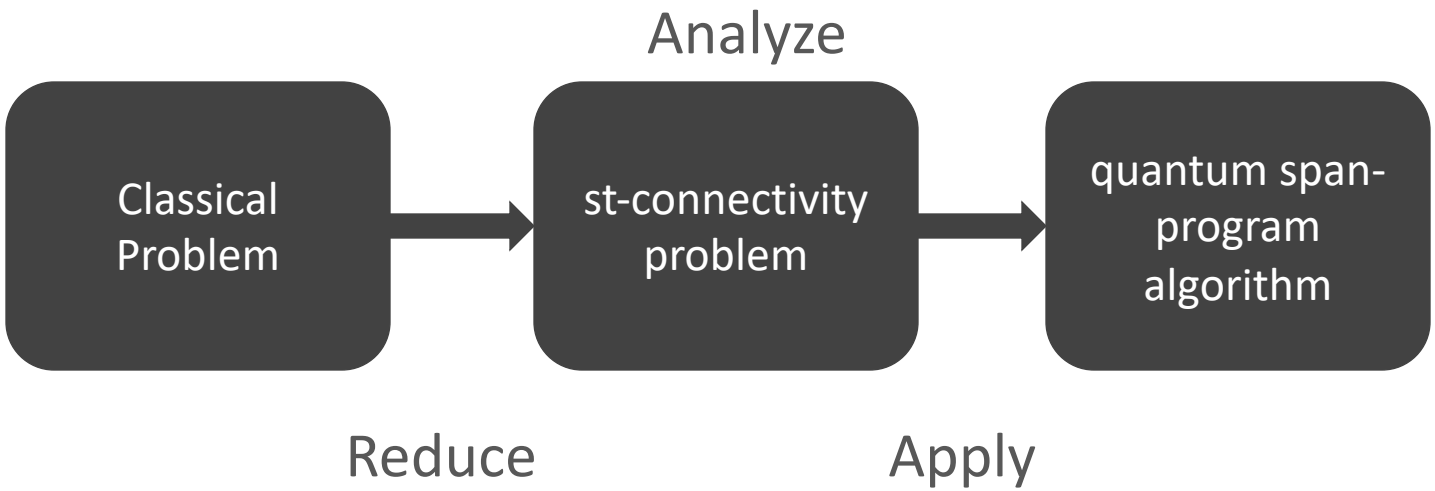
There is a cycle if

- Cycle through edge 1
- Cycle through edge 2
- …
- Cycle through edge n

Or

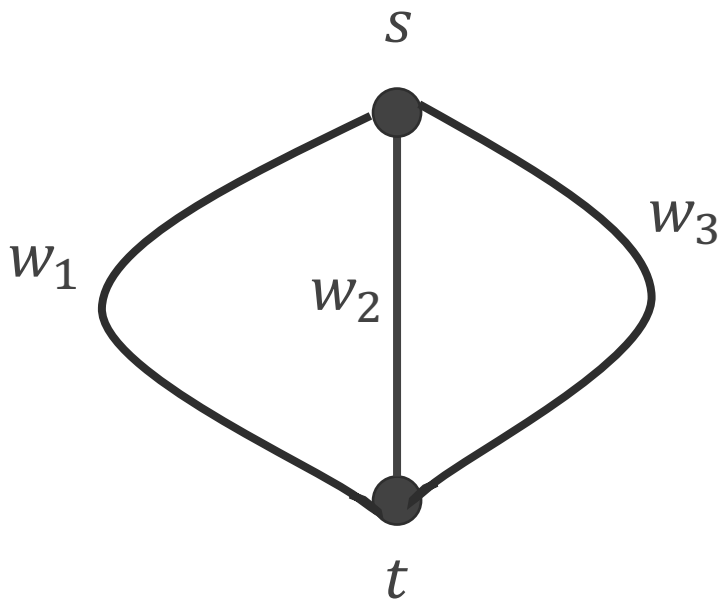
Multi-tool

- ✓ Structured
- ✓ Unstructured
- ✓ Easy creation
- ✓ Easy, provable, non-quantum analysis



Analysis

Assign a weight to each edge (start with 1!):

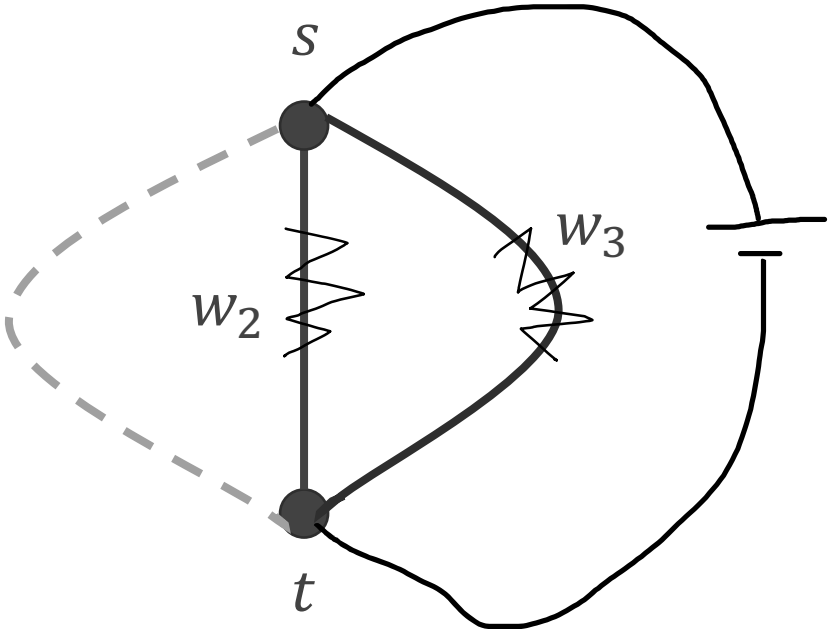


Analysis

For each possible input, can calculate

- effective resistance (if path)

$x = 011$

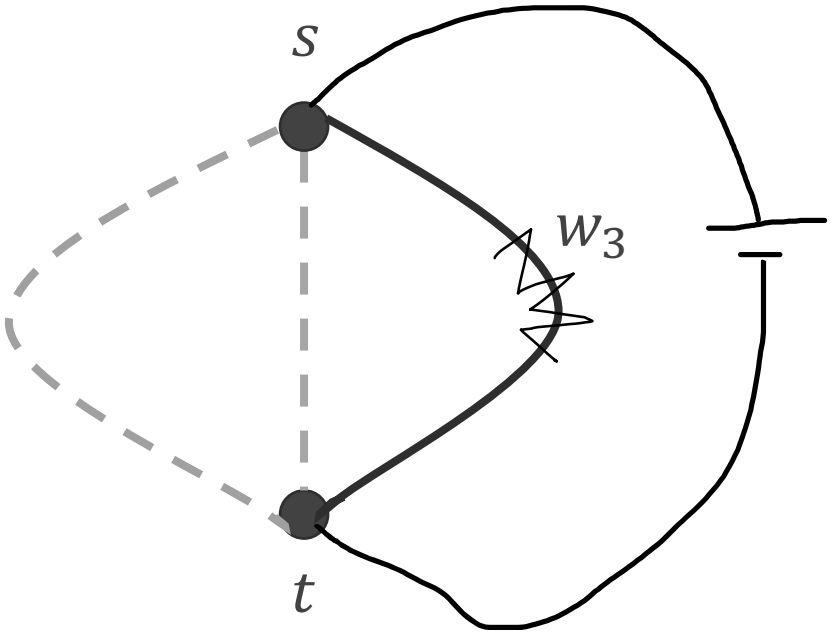


Analysis

For each possible input, can calculate

- effective resistance (if path)

$x = 001$

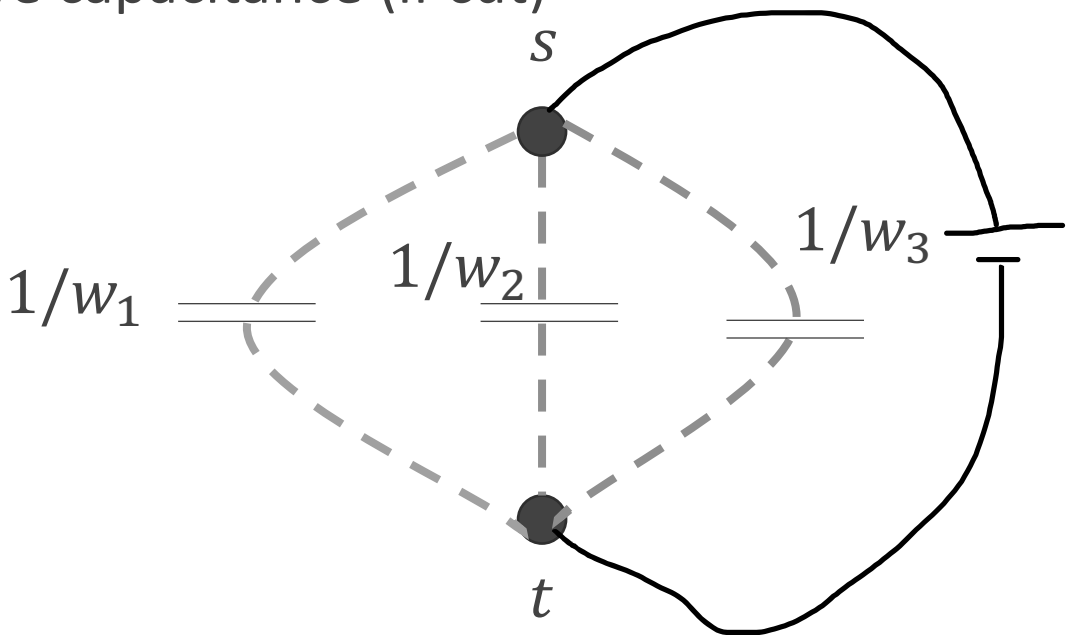


Analysis

For each possible input, can calculate

- effective resistance (if path)
- effective capacitance (if cut)

$x = 000$



Analysis

Then Quantum Query Complexity is:

$$O(\sqrt{(\max C)(\max R)})$$

$\max R$: max effective resistance over connected instances
$\max C$: max effective capacitance over not connected instances

[Belovs, Reichardt, `12], [JJKP, `18]

Analysis

Then Quantum Query Complexity is:

$$O(\sqrt{(\max C)(\max R)})$$

$\max R$: max effective resistance over connected instances
$\max C$: max effective capacitance over not connected instances

[Belovs, Reichardt, `12], [JJKP, `18]

Note:

- R is always less than the shortest path
- C is always less than the size of the smallest cut

Analysis

Then Quantum Query Complexity is:

$$O(\sqrt{(\max C)(\max R)})$$

$\max R$: max effective resistance over connected instances
$\max C$: max effective capacitance over not connected instances

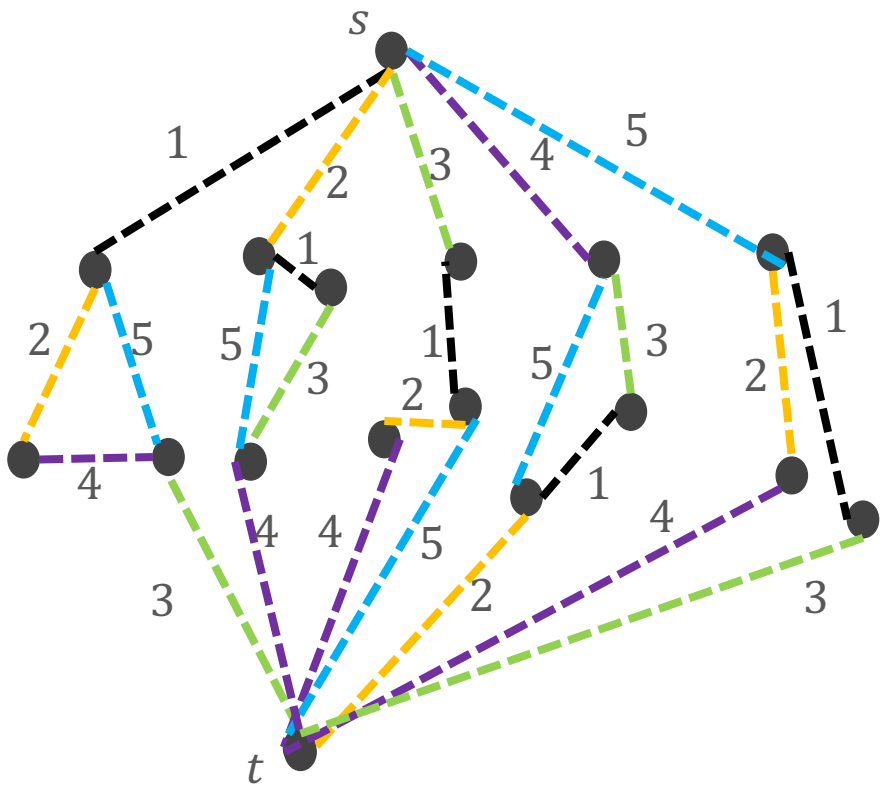
[Belovs, Reichardt, `12], [JJKP, `18]

Space complexity scales log(no. edges, vertices) in graph.
[Belovs, Reichardt `12]

Example

For capacitance, bound cut:

n vertices, $O(n^2)$ edges

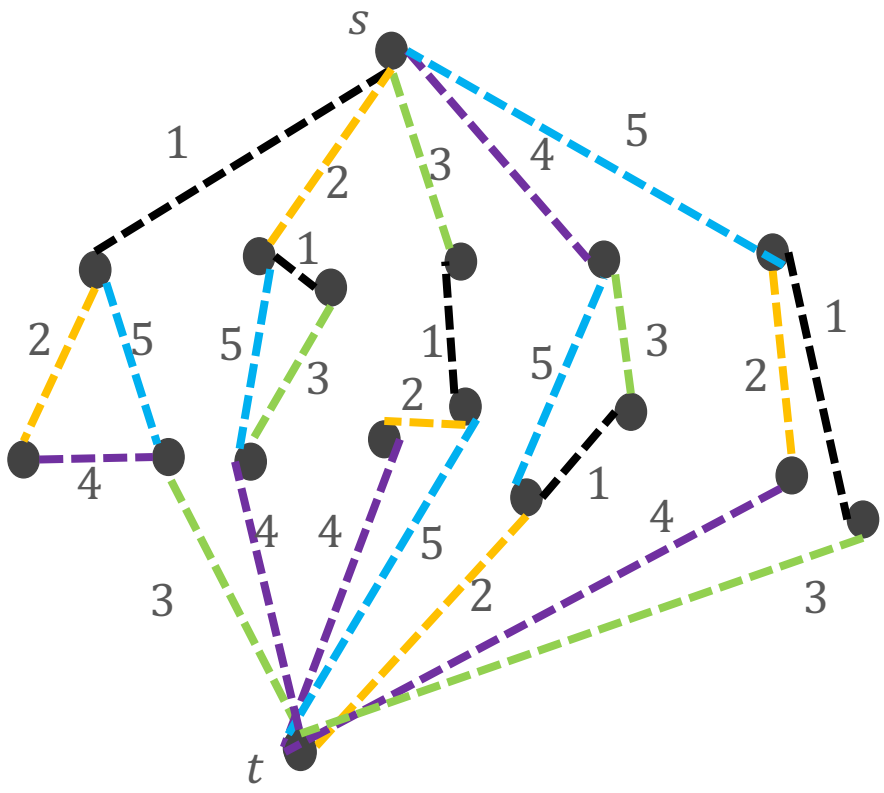


Example

For capacitance, bound cut:

No cycles -> at most n edges

n vertices, $O(n^2)$ edges



Example

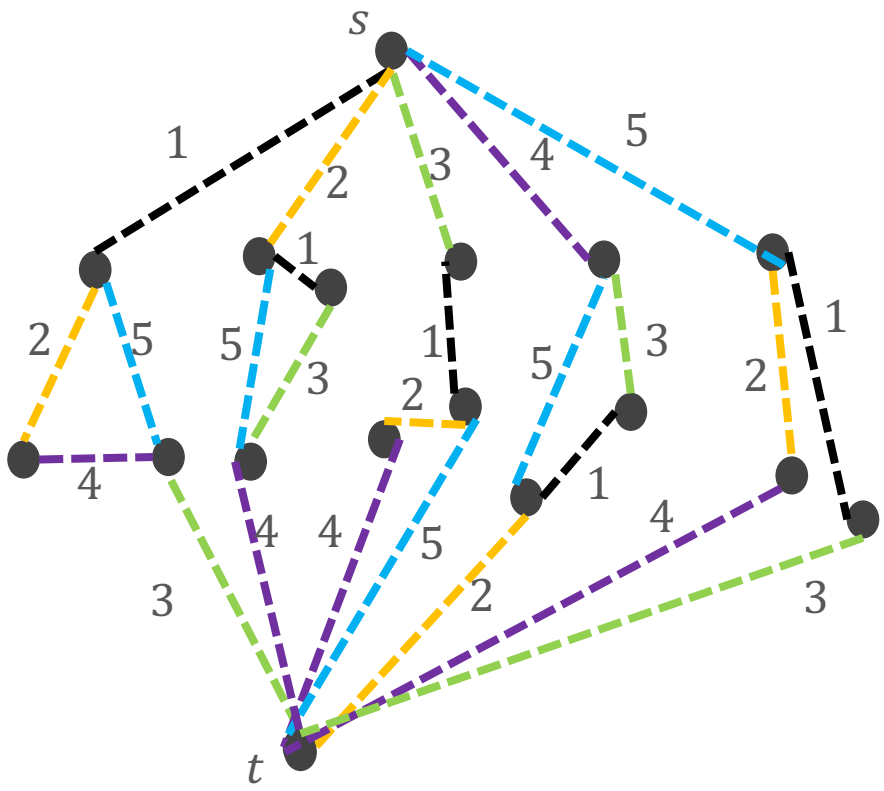
For capacitance, bound cut:

No cycles -> at most n edges

- cut at top edge when possible
- otherwise might need $O(n^2)$ cuts within subgraph

Max Cut size is $O(n^3)$

n vertices, $O(n^2)$ edges

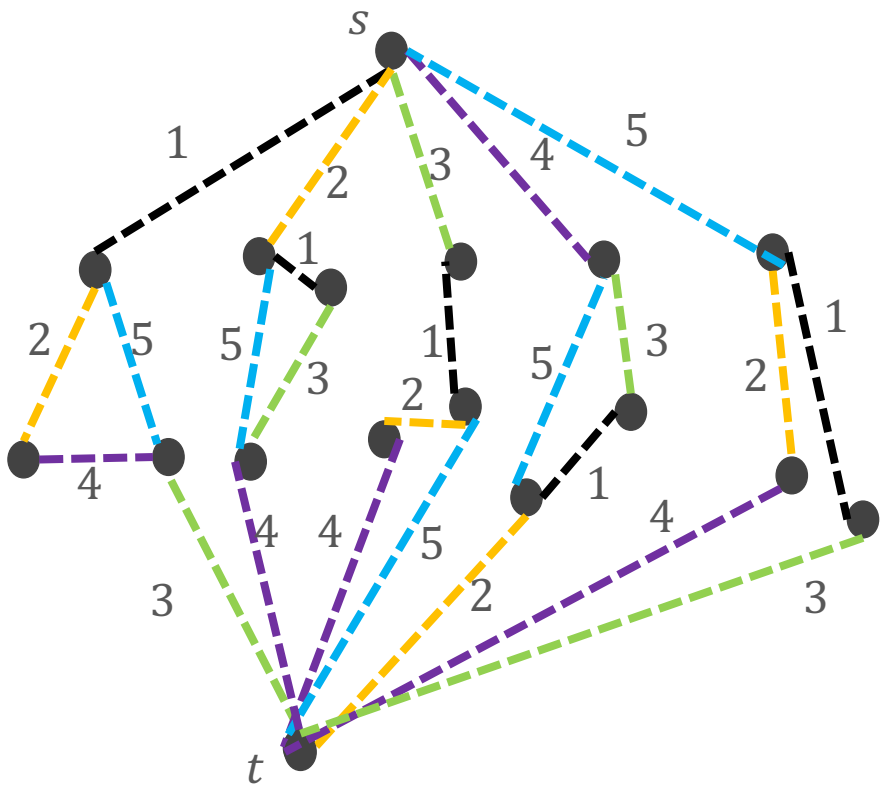


Example

For resistance:

Cycle with k edges -> paths through each k subgraphs, each of length k.

n vertices, $O(n^2)$ edges



Example

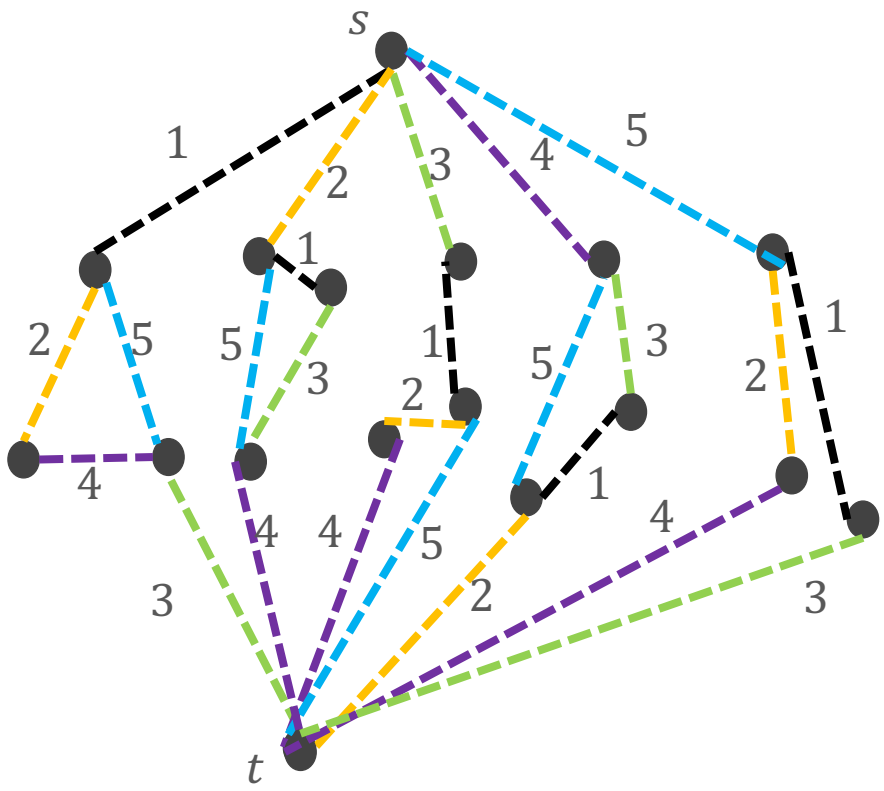
For resistance:

Cycle with k edges -> paths through each k subgraphs, each of length k.

- In series, resistances add
- In parallel, inverse resistances add

Effective resistance is $O(1)$

n vertices, $O(n^2)$ edges

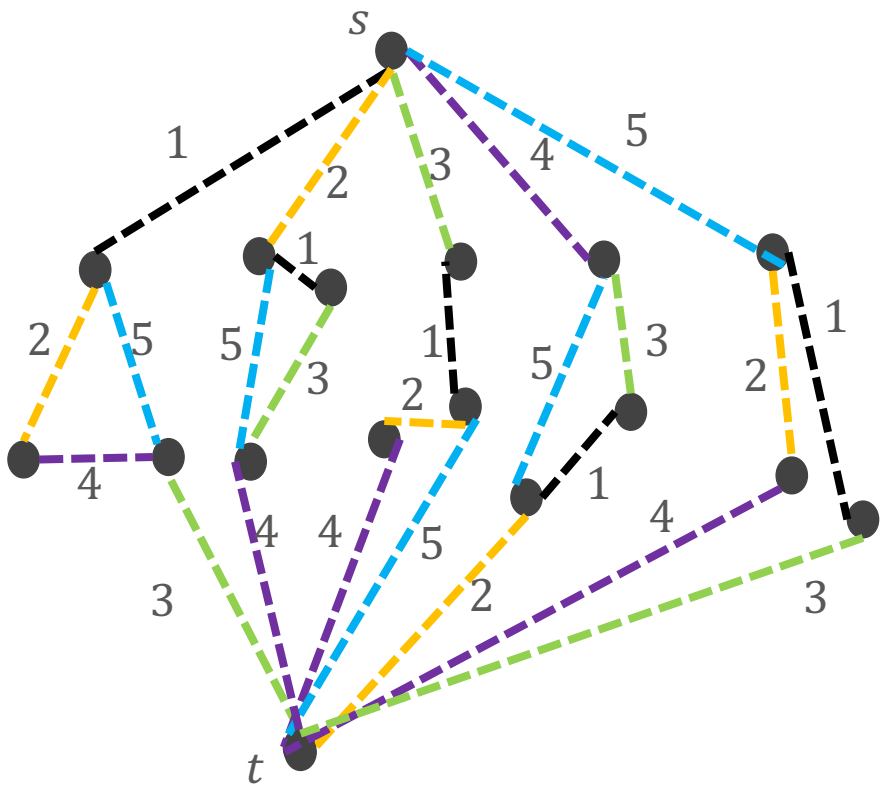


Example

Effective resistance is $O(1)$
Max Cut size is $O(n^3)$

Query complexity is
$O(n^{3/2})$

n vertices, $O(n^2)$ edges



Performance

*There are alternative optimal algorithms for some problems

- Read-once Boolean formulas (query optimal) [J**K**]
- Total connectivity (query optimal) [JJ**K**P]
- Cycle detection (query optimal) [D**K**W]
- Even length cycle detection [D**K**W]
- Bipartiteness (query optimal) [D**K**W]
- 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
- Maximum matching (**K**W)
- Super-polynomial speed-up for game evaluation [ZH**K**]

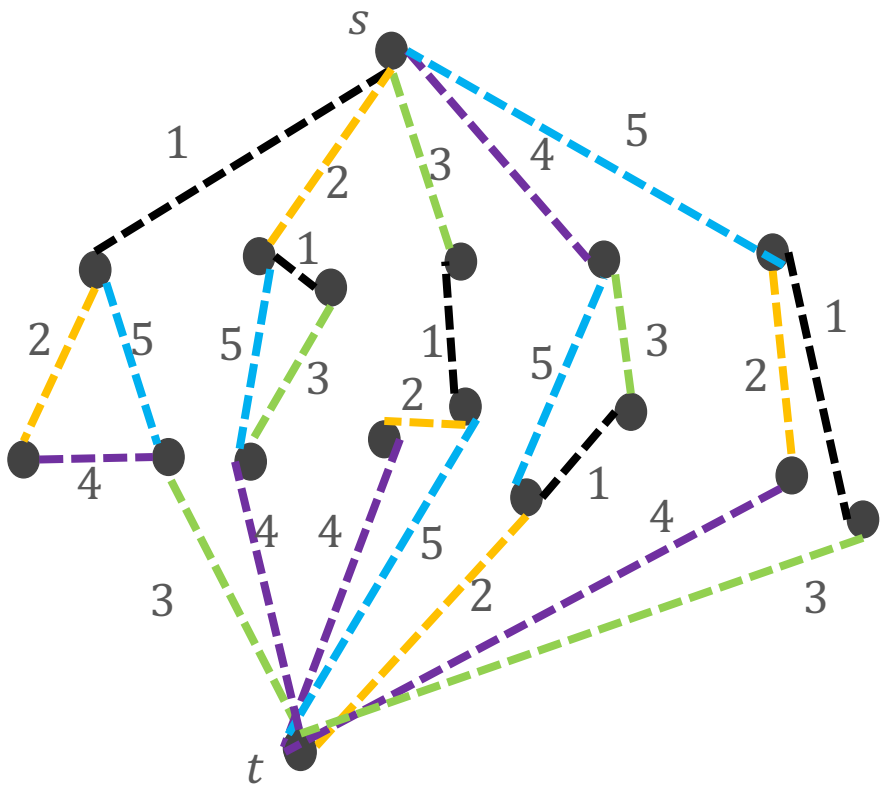
Promise

Effective resistance is equal to the inverse of the cycle rank.

If promised that if there is a cycle, then cycle rank is at least r, then query complexity is

$$O\left(\sqrt{n^3/r}\right)$$

n vertices, $O(n^2)$ edges

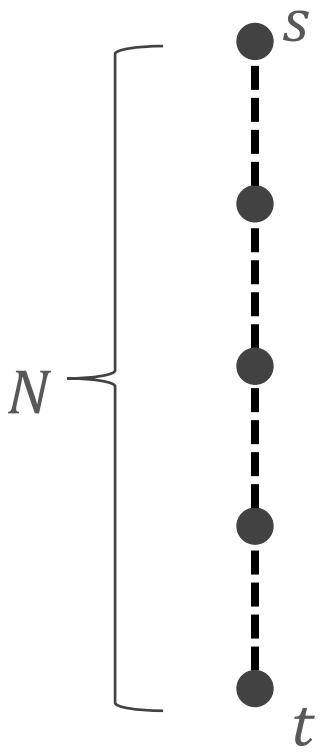


Promise

Promise Search: all N bits are 1, or at least r are 0

Quantum query complexity is

$$O\left(\sqrt{N/r}\right)$$



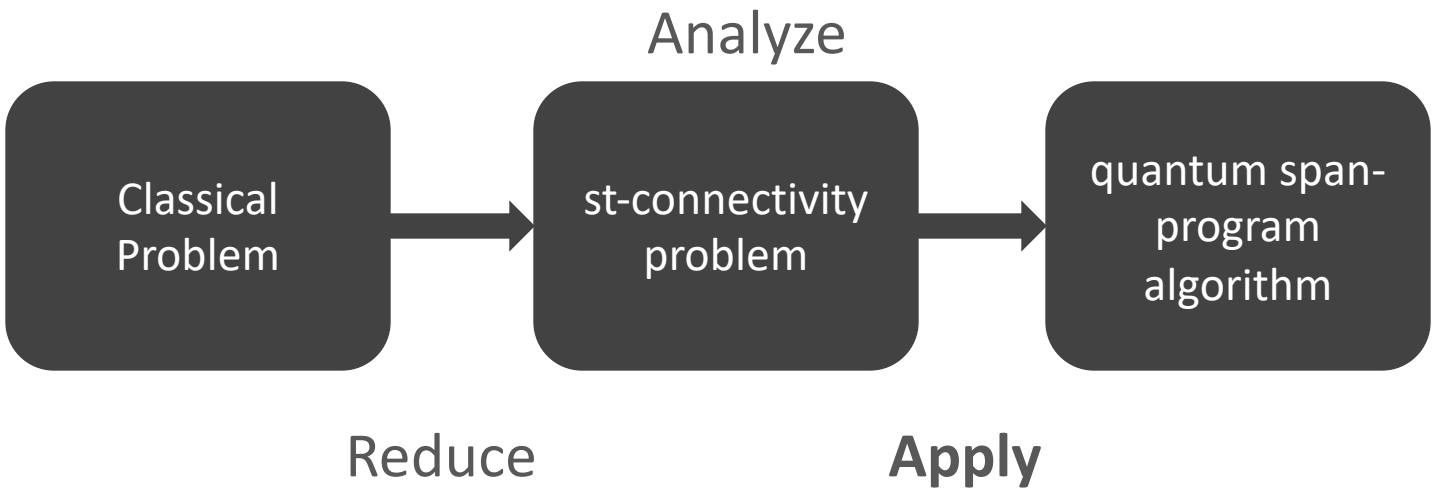
No Promise? No problem!

We'd like algorithm to be fast if instance has small effective resistance or capacitance, even if don't know promise ahead of time.

Multi-tool

- ✓ Structured
- ✓ Unstructured
- ✓ Easy creation
- ✓ Easy, provable, non-quantum analysis

New and Improved Timer!



Algorithm

Apply phase estimation with a unitary U that is a product of two reflections,

- One depends on input x
- One depends on underlying graph

To a specific state

Algorithm

Apply phase estimation with a unitary U that is a product of two reflections,

- One depends on input x
- One depends on underlying graph

To a specific state

Measurement Outcome	Phase 0	Non-Zero Phase
Conclusion	Path	No Path

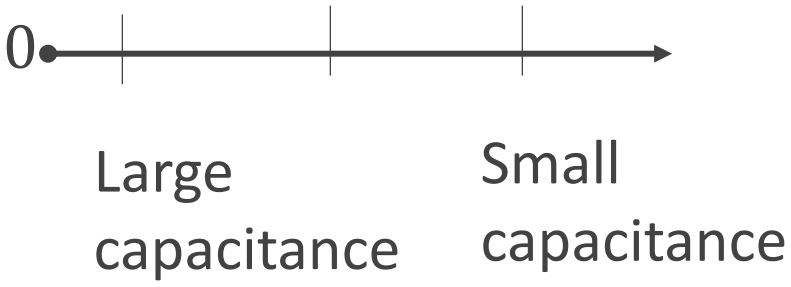
Algorithm

If no path, U doesn't have a zero-valued eigenvector:

Algorithm

If no path, U doesn't have a zero-valued eigenvector:

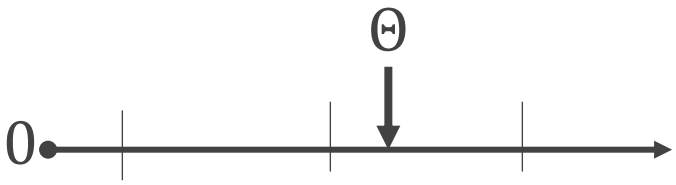
Smallest eigenphase if no path



Algorithm

If no path, U doesn't have a zero-valued eigenvector:

Smallest eigenphase if no path



Large capacitance

Small capacitance

- Non-zero phase: no path
- Zero-phase: not sure if path or just large capacitance

Algorithm

Apply phase estimation with a unitary U that is a product of two reflections,

- One depends on input x
- One depends on underlying graph

To a specific state

Measurement Outcome	Phase 0	Non-Zero Phase
Alg 1 Conclusion	??	No Path

Algorithm

Apply phase estimation with a unitary U that is a product of two reflections,

- One depends on input x
- One depends on underlying graph

To a specific state

Measurement Outcome	Phase 0	Non-Zero Phase
Alg 1 Conclusion	??	No Path
Alg 2 Conclusion	Path	??

Algorithm

- Repeatedly run algorithm for exponentially increasing runtime until do not get a ?? result, or until reach max runtime.

Algorithm

Previously:

$$O\left(\sqrt{(\max C)(\max R)}\right)$$

Query Complexity:

If there is a path:

$$\tilde{O}\left(\sqrt{R(x)\max C}\right)$$

Where $R(x)$ is resistance of actual input

If there is no path:

$$\tilde{O}\left(\sqrt{C(x)\max R}\right)$$

Where $C(x)$ is capacitance of actual input

Algorithm

Query Complexity:

Analogous results for generic span program and dual adversary style algorithm for state conversion

If there is a path:

$$\tilde{O}\left(\sqrt{R(x)\max C}\right)$$

Where $R(x)$ is resistance of actual input

If there is no path:

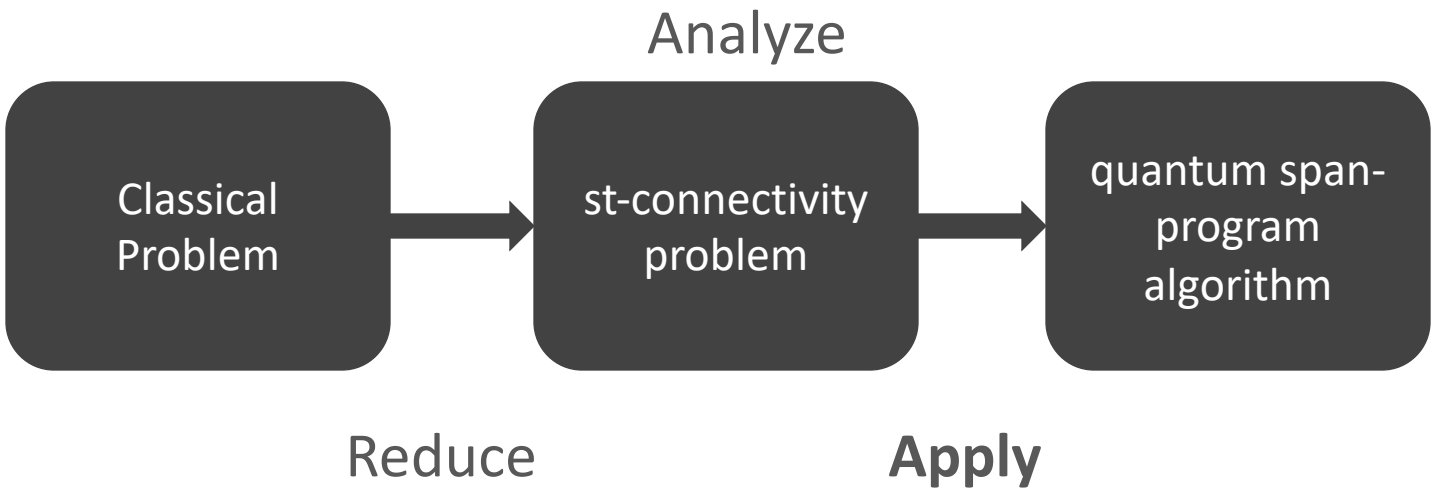
$$\tilde{O}\left(\sqrt{C(x)\max R}\right)$$

Where $C(x)$ is capacitance of actual input

Multi-tool

- ✓ Structured
- ✓ Unstructured
- ✓ Easy creation
- ✓ Easy, provable, non-quantum analysis

New and Improved Timer!



Open Problems

- How to set weights?
- When is this approach optimal?
- Remove log factors in our timer? Prove optimal?

Funding:

Middlebury

ARO

People:

Stacey Jeffery

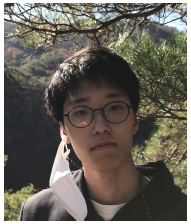
Michael Jarret

Alvaro Piedrafita

Teal Witter

Kai De Lorenzo

Noel Anderson



Jay-U Chung