Classical and Quantum Algorithms for Finding Cycles

Jill Cirasella
Computational Sciences Specialist
Brooklyn College Library
cirasella@brooklyn.cuny.edu

27 July 2006

What I Did

- **Not:** "how to make the intractable tractable"
- **Rather:** "how to make the tractable easier"
- **Specifically:** Examined classical and quantum algorithms for finding cycles in undirected graphs.
- **Clarification:** "Finding cycles" = "finding and/or determining the existence of cycles"
- **Why?:** Happy with decision algorithms, which can be transformed into example-finding algorithms.
- **Note:** Not comprehensive. Stuck to the most important, elegant, efficient.

Basic Definitions

- A **graph** G is a structure (V, E), where V is a finite set of vertices and E is a finite set of edges. We denote $|V|$ by n and $|E|$ by m.
- In **undirected graphs**, edges are unordered pairs of distinct vertices (i.e., no direction, no self-loops).
- Two vertices are **adjacent** if they are connected by an edge.
- A sequence of $k + 1$ adjacent vertices is a **path** of length k.
- A path is a **cycle** if the first and last vertices are the same and all other vertices are distinct.

Representations of Graphs

- The **adjacency matrix** of G is an $n \times n$ matrix M such that $M(u, v) = 1$ iff $(u, v) \in E$, and $M(u, v) = 0$ otherwise.
  - Determining existence of an edge takes constant time.
  - Storing M requires $O(n^2)$ space.
- The **adjacency list representation** of G is an array A of size n. Each element of A represents a vertex $u \in V$ and points to a linked list of vertices adjacent to u.
  - Determining existence of an edge involves searching an unordered list of vertices and can require $O(n)$ time.
  - Storing A requires $O(n + m)$ space.

Search

Common feature of graph algorithms:
They involve a search for something.

- **Unordered search problem:** Let $N = 2^n$ for some positive integer n. Given an arbitrary bit string $x = (x_0, x_1, \ldots, x_{N-1})$, we want to find an i such that $x_i = 1$ or to learn that there is no such i.
- When $x_i = 1$, then x_i is called a **solution**.

Search Algorithms

- **Classical:** No choice but to examine each element. No matter what order they're examined in, the process requires $O(N)$ queries.
- **Quantum:** Grover's algorithm, coming up soon...

Basic Quantum Concepts

Classical Computing =
Classical States + Transitions Between Them

Similarly:

Quantum Computing =
Quantum States + Transitions Between Them

Qubits

- **Quantum states** consist of one or more quantum bits (**qubits**), each of which can be $|0\rangle, |1\rangle$, or a superposition of the two.
- The state of a qubit is defined by:

  $$\alpha_0|0\rangle + \alpha_1|1\rangle$$

  where α_0 and α_1 are complex amplitudes. When both α_0 and α_1 are nonzero, the qubit is in a superposition.
- **Observation** jostles a qubit out of superposition and projects it onto either $|0\rangle$ or $|1\rangle$.
- The probability of observing $|0\rangle$ is $|\alpha_0|^2$, and the probability of observing $|1\rangle$ is $|\alpha_1|^2$.
- Always the case that $|\alpha_0|^2 + |\alpha_1|^2 = 1$.

Quantum States

- An **n-qubit quantum state** $|\phi\rangle$ is written:

  $$\alpha_0|0\rangle + \alpha_1|1\rangle + \cdots + \alpha_{2^n-1}|2^n-1\rangle$$

  where $\alpha_0, \alpha_1, \ldots, \alpha_{2^n-1}$ are complex amplitudes and $|0\rangle, |1\rangle, \ldots, |2^n-1\rangle$ are quantum basis states. Observation projects $|\phi\rangle$ onto state $|i\rangle$ with probability $|\alpha_i|^2$.
- Always the case that $|\alpha_0|^2 + |\alpha_1|^2 + \cdots + |\alpha_{2^n-1}|^2 = 1$.
- **Quantum states** can be expressed as **vectors**, e.g.:

$$\begin{pmatrix} \alpha_0 \\ \alpha_1 \\ \vdots \\ \alpha_{2^n-1} \end{pmatrix}$$

Quantum Operations

- **Quantum operations**, like classical operations, can be expressed as matrices. Of course, only some matrices define quantum operations.
- A matrix U defines a realizable quantum operation iff it is **unitary** (that is, iff it is a square matrix whose inverse U^{-1} equals its conjugate transpose U^*).
- **Calculate effect of a quantum operation** by multiplying the operation's matrix U by the state's vector $|\phi\rangle$: $U|\phi\rangle = |\phi'\rangle$.

A Simple Quantum Gate

- A simple quantum gate is the "quantum coin flip":

$$QCF = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & -1 \\ 1 & 1 \end{pmatrix}.$$

- $QCF|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \to 50/50$ chance.
- $QCF|1\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \to 50/50$ chance.
- Uniform superposition $\to$ true randomness.
- If we don't observe but again apply QCF: Applying QCF to $QCF|1\rangle$ yields $|0\rangle$, and applying QCF to $QCF|0\rangle$ yields $-|1\rangle$.
- Two applications of QCF effect negation.
- QCF is sometimes called $\sqrt{NOT}$.

Hadamard Transform

- Hadamard transform is very similar to QCF. Rather than $\sqrt{NOT}$, can be thought of as $\sqrt{Identity}$.
- Like QCF, Hadamard transform puts a qubit into uniform superposition:

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}.$$

- $H|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \rightarrow 50/50$ chance.
- $H|1\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \rightarrow 50/50$ chance.
- Applying H to $H|0\rangle$ yields $|0\rangle$, and H to $H|1\rangle$ yields $|1\rangle$.
- Two applications of H effect identity function.
- H (and other quantum gates for single qubits) can be simultaneously applied to multiple qubits.

Time Complexity vs. Query Complexity

- **Time complexity:** # of operations an algorithm makes.
- **Query complexity:** # of times an algorithm accesses its worst-case input.
- Query complexity **says less** about an algorithm's efficiency than time complexity does.
- But, when we don't know the lower bound on a problem's classical time complexity, we sometimes do know the lower bound on its classical query complexity.
- Therefore, we can often **learn more** by comparing quantum and classical query complexities.

Milestones in Quantum Computing

- **R. Feynman, 1981:** Suggested that a quantum computer could do what no classical computer can: simulate quantum mechanics.
- **P. Benioff, 1982:** Showed reversible unitarity (i.e., quantum mechanics) to be at least as powerful as classical computation.
- **D. Deutsch, 1985:** Described a quantum Turing machine.
- **D. Deutsch and R. Jozsa, 1992:** First fully described quantum algorithm. Solved an oracle problem in p-time.
- **P. Shor, 1994:** Quantum algorithm for factoring numbers in p-time. (Good thing for quantum cryptography!)
- **L. Grover, 1996:** Quantum algorithm for searching an unordered list in $O(\sqrt{N})$ steps. First quantum algorithm for a "total" problem (i.e., no "promise" about the input).

What I Did, Again

Looked at classical and quantum algorithms for finding — or determining the existence of — triangles, quadrilaterals, and longer cycles in graphs.

Bag of Tricks

- **Three quantum algorithms become our "bag of tricks."** We combined and embedded them with each other and with classical operations to create the quantum cycle algorithms ahead.
  - Sometimes, a quantum cycle algorithm is made by inserting one or more of these tricks in place of steps in a classical cycle algorithm.
  - Other times, these tricks are used to make totally new algorithms that are not derived from classical algorithms.

Trick #1: Grover's Search Algorithm

- **Grover's Search Algorithm:** Solves the unordered search problem. Uses the **Grover iterate**, which involves a query (flips the phase of the solution state) and the diffusion transform (achieves "inversion about average"):
  1. Begin in the n-qubit state $|\vec{0}\rangle$.
  2. Apply the Hadamard transform H to every qubit in $|\vec{0}\rangle$.
  3. Apply the Grover iterate $O(\sqrt{N})$ times.
  4. Measure the resulting superposition, collapsing it into a single state.

  Queries: $O(\sqrt{N})$.

  See handout!

Trick #2: Amplitude Amplification

- **Amplitude Amplification:** Generalization of Grover's. Finds an x such $\chi(x) = 1$, where χ is a Boolean function.
  1. Begin in the n-qubit state $|\vec{0}\rangle$.
  2. Put $|\vec{0}\rangle$ in superposition. (Let a be the probability of observing "good" state at this point.)
  3. Apply the Q iterate $O(1/\sqrt{a})$ times.
  4. Measure the resulting superposition, collapsing it into a single state.

  Queries: $O(1/\sqrt{a})$.

  Can replace "for" loops. Instead of stepping through N elements, each with $1/N$ chance of being "good," perform $O(\sqrt{N})$ iterations of amplitude amplification.

Trick #3: Ambainis's Algorithm for Element Distinctness

- **Ambainis's Algorithm for Element Distinctness:** Determines whether a set of N elements contains two that equal each other. Examines the set using a quantum walk, which steps through the set's subsets of certain sizes. Each subset is similar to its predecessor, differing by just one element. Therefore, computation on each subset is similar to computation on its predecessor. Conserves queries.
  Queries: $O(N^{2/3})$.
- **Generalizes to k-element distinctness problem**, which asks whether there are k elements that equal each other.
  Queries: $O(N^{k/(k+1)})$.

Classical Triangle Algorithms

- **Classical Triangle Algorithm #1:** Brute force with an adjacency matrix. **Time:** $O(n^3)$. **Queries:** $O(n^2)$.
- **Classical Triangle Algorithm #2:** Brute force with an adjacency list. **Time:** $O(n^3)$. **Queries:** $O(n+m)$.
- **Classical Triangle Algorithm #3:**
  1. Compute M^2 using Boolean matrix multiplication.
  2. Compute $M^2 \wedge M$.
  3. If $M^2 \wedge M$ contains a 1, then G contains a $\triangle$.

  Time: $O(n^\alpha)$, where α is the exponent of Boolean matrix multiplication algorithm. **Queries:** $O(n^2)$.

Quantum Triangle Algorithms #1 and #2

- **Quantum Triangle Algorithm #1:** Brute force with Grover's:
  1. Perform Grover's to find a $\triangle$ among the $\binom{n}{3}$ triples of vertices.

  Queries: $O(n^{3/2})$.

- **Quantum Triangle Algorithm #2:** Uses both Grover's and amplitude amplification:
  1. Perform Grover's to find an edge (u, v) among the $\binom{n}{2}$ potential edges.
  2. Perform Grover's to find a vertex w such that u, v, w is a $\triangle$.
  3. Perform amplitude amplification on steps 1 and 2 (to find desired edge (u, v)).

  Queries: $O(n + \sqrt{nm})$ — better than $O(n^{3/2})$ if G is sparse.

Quantum Triangle Algorithm #3

- **Quantum Triangle Algorithm #3:** Adapts classical algorithm that uses brute force on adjacency list. Uses Grover's and **two** layers of amplitude amplification.

  1. Choose random vertex $u \in V$.
  2. Query all elements in $A[u]$ and make list T of these neighbors.
  3. Choose random vertex $v \in T$.
  4. Perform Grover's to find vertex $w \in A[v]$ that is also in T. If such a w is found, u, v, w is a $\triangle$.
  5. Perform ampl. ampl. on steps 3–4 (to find a desired v).
  6. Perform ampl. ampl. on steps 1–5 (to find a desired u).

  Queries: $O(n^{3/2})$.

Quantum Triangle Algorithms #4 and #5

Quickly. . .

- **Quantum Triangle Algorithm #4:** Uses Grover's and lots of combinatorial tricks. Complicated.
  Queries: $\widetilde{O}(n^{10/7})$.

- **Quantum Triangle Algorithm #5:**
  - Shows that triangle problem reduces to "graph collision" $\rightarrow$
  - which reduces to "collision" $\rightarrow$
  - which reduces to "unique collision" $\rightarrow$
  - which can be solved by the "Generic Algorithm" $\rightarrow$
  - which is a generalization of Ambainis's Algorithm for element distinctness.

  Queries: $\widetilde{O}(n^{1.3})$ — later improved to $O(n^{1.3})$.

Classical Quadrilateral Algorithms

- **Obvious Adaptations:** Several modifications of $\triangle$ algorithms, none very interesting. However, there is a noteworthy modification of the matrix multiplication algorithm, coming up soon...

- **Classical Quadrilateral Algorithm #1:** Takes in adjacency list and cleverly creates a matrix:

```
1. Create an $n \times n$ matrix $C$ and initialize all entries to 0.
2. For each vertex $u \in V$,
3.    For each pair of vertices $v, w \in A[u]$ such that $v < w$,
4.       If $C(v, w) = 0$, then $C(v, w) \leftarrow u$.
         Else, $u, v, C(v, w), w$ is a $\square$.
```

  Time: $O(n^2)$. **Queries:** $O(n + m)$.

Quantum Quadrilateral Algorithm #1

- **Quantum Quadrilateral Algorithm #1:** Takes in adjacency matrix and uses Grover's and amplitude amplification:
  1. Choose two random vertices $u, v \in V$.
  2. Perform Grover's search to find two vertices $u', v' \in V$ such that u, u', v, v' is a $\square$.
  3. Perform amplitude amplification on steps 1–2 (to find good u and v).

  Queries: $O(n^{3/2})$.

Quantum Quadrilateral Algorithm #2

- **Quantum Quadrilateral Algorithm #2:** Takes in adjacency list and uses Grover's, amplitude amplification, and Ambainis's:
    1. Choose a random vertex $u \in V$, which has k neighbors $v_1, \ldots, v_k$ with adjacency lists $A[v_1], \ldots, A[v_k]$.
    2. Perform Grover's search on $v_1, \ldots, v_k$ to find up to $2n^{1-c} + 1$ neighbors v_L such that $|A[v_L]| \geq n^c$ (c to be determined later). We say that these v_L have "long" lists.
    3. If there are more than $2n^{1-c}$ neighbors with long lists, then graph G must contain a quadrilateral. Truncate each long list to length n^c, and concatenate these truncated lists into a new list L. Perform Ambainis's on L to find a vertex $w \neq u$ that is adjacent to at least two neighbors v_L.

Quantum Quadrilateral Algorithm #2

- **Quantum Quadrilateral Algorithm #2:** Continued:
  4. Otherwise, pad all short lists with unique dummy elements to bring their lengths up to n^c. Concatenate all lists into a new list L. Perform Ambainis's on L to find a vertex $w \in V$ that is adjacent to at least two vertices that are adjacent to u.
  5. Perform amplitude amplification on steps 1–4.

  Queries: $O(n^{19/14})$.

Classical Longer-Cycle Algorithms

**For all longer-cycle algorithms,
assume that k is fixed and not inputted.**

- **Obvious Adaptations:** Obvious k-cycle algorithms all have time complexities around $O(n^k)$.
- **Slight Improvements:** One finds k-cycles in $O(n^{k/2})$ time. Another finds k-cycles in $O((k-1)! \cdot nm) = O(nm)$ time.

Classical Even/Odd Cycle Algorithm #3

- **Classical Even/Odd Cycle Algorithm #3:** Modification of matrix-multiplication triangle algorithm:

```
1. Repeat until a $k$-cycle is found or until all acyclic orientations
   have been tried:
2.    Choose a random ordering of the vertices in $V$.
3.    Create an adjacency matrix $\vec{M}$ for the acyclic orientation $\vec{G}$
      corresponding to the ordering.
4.    Compute $\vec{M}^{k-1} \wedge M$.
```

Time: $O(k! \log k \cdot n^{\alpha}) = O(n^{\alpha})$. **Queries:** $O(n^2)$.

Classical Even Cycle Algorithm #4

- **Classical Even Cycle Algorithm #4:** Takes in adjacency list and performs a series of breadth-first searches:
  1. For each vertex $s \in V$,
  2. Do a BFS, stopping after a certain stage or after a certain number of edges have been found.
  3. Check whether subgraph created by BFS has a $2k$-cycle. (How to check depends on why BFS stopped.)

  Time: $O((2k)! \cdot n^2) = O(n^2)$. **Queries:** $O(n^2)$.

- **Quantum Adaptation:** Replace the "for" loop with amplitude amplification. Though tempting, cannot replace the BFS with Grover's, as BFS is used not to search but to generate a subgraph.
  Queries: $O(n^{3/2})$.

Generalization of Ambainis's Algorithm

- **Generalization of Ambainis's Algorithm:** Childs and Eisenberg showed that Ambainis's $O(n^{k/(k+1)})$ algorithm for k-element distinctness can be repurposed to find a subset of size k that satisfies any given property P.

  A k-cycle is a subset of the $\binom{n}{2}$ possible edges in a graph G, so the query complexity of finding a k-cycle is:

$$O\left(\binom{n}{2}^{k/(k+1)}\right) = O(n^{2k/(k+1)}) < O(n^2).$$

Generalization of $O(n^{1.3})$ Triangle Algorithm

- **Generalization of $O(n^{1.3})$ Triangle Algorithm:** Magniez, Santha, and Szegedy showed that their $O(n^{1.3})$ triangle algorithm generalizes to a $O(n^{2-(2/k)})$ algorithm for finding a copy of a specified k-vertex subgraph. The specified subgraph could of course be a k-cycle.

This is the best query complexity yet for arbitrary-length cycles.

Decision-to-Example-Finding Algorithms

- As mentioned, **decision algorithms are always sufficient.** If we want an example of a cycle in addition to the assertion that one exists, we can transform the decision algorithm into an example-finding algorithm.

- **Embed the decision algorithm in a recursion.** Just maintain a list L of "active" vertices. At each level of recursion, randomly reorder L, split it in half, and recur. (Need to repeat each level several times.)

- If it uses an adjacency matrix, example-finding algorithm has same query complexity as decision algorithm. (If it uses an adjacency list, the "upkeep" of the list dominates the query complexity of the decision algorithm.)

Conclusion

- In many ways the generalization of Ambainis's algorithm and the generalization of the $O(n^{1.3})$ triangle algorithm are the ultimate in cycle algorithms.
- My project, by surveying the many classical and quantum cycle algorithms that came before, allows us to appreciate the new findings with a sense of history, a sense of drama, and a sense of the cumulative nature of algorithmic research.