Learning Graphs with Queries

Lev Reyzin
Clique talk
Fall 2006
Some Papers

- Angluin, Chen. **Learning a Hidden Graph Using $O(\log n)$ Queries Per Edge.** (COLT ’04)

- Bouvel, Grebinski, Kucherov. **Combinatorial Search on Graphs Motivated by Bioinformatics Applications: A Brief Survey** (WG ’05)

- Reyzin, Srivastava. **A Survey of Graph Learning with Queries** (Manuscript ’06)

- Hein. **An Optimal Algorithm to Reconstruct Trees from Additive Distance Data.** (Journal of Math Bio ’89)
Hidden Graphs

Oracle at Delphi
Queries

Edge
query pair of vertices in graph to detect presence of edge (social networks)

Edge Detection
query set of vertices in graph for presence of edges (chemical reactions)

Edge Counting
query set of vertices in graph for number of edges (DNA sequencing)

Connectedness
query pair of vertices in graph to discover if they’re in same component (electrical networks)

Shortest Path
query two vertices in graph for shortest path length between them (evolutionary trees)
Edge Queries

**Edge Query:** query pair of vertices in graph to detect presence of edge
Edge Detection Queries

**Edge Detection Query**: query a set of vertices in graph for presence of an edge
Edge Counting Queries

Edge Counting Query: query a set of vertices in a graph for number of edges in subgraph induced by the vertices
Shortest Path Queries

**Shortest Path:** query two vertices in graph for shortest path length between them
Connectedness Queries: query pair of vertices in graph to discover if they’re in same component
Relative Power of Queries

$E \leq ED \leq EC$

$E \leq SP$

$C \leq SP$
Target Classes of Graphs

- general graphs
- trees
- partitions
## Results Summary

\( n = \) number of nodes, \(|E| = \) number of edges, 
\( k = \) number of connected components, \( d = \) maximum degree

<table>
<thead>
<tr>
<th>Query</th>
<th>partition</th>
<th>graph</th>
<th>tree</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td>( \Theta(n^2) )</td>
<td>( \Theta(n^2) )</td>
<td>( \Theta(n^2) )</td>
</tr>
<tr>
<td>ED</td>
<td>( \Theta(n^2) )</td>
<td>( \Theta(</td>
<td>E</td>
</tr>
<tr>
<td>EC</td>
<td>( O(n \lg n) ) ( \Omega(n) )</td>
<td>( O(</td>
<td>E</td>
</tr>
<tr>
<td>SP</td>
<td>( \Theta(nk) )</td>
<td>( \Theta(n^2) )</td>
<td>( \Theta(n^2), \Theta(dn \lg n) )</td>
</tr>
<tr>
<td>C</td>
<td>( \Theta(nk) )</td>
<td>not possible</td>
<td>not possible</td>
</tr>
</tbody>
</table>

\( E = \) Edge Query, \( ED = \) Edge Detection Query, \( EC = \) Edge Counting Query,  
\( SP = \) Shortest Path Query, \( C = \) Connectedness Query

references omitted
Learning Partitions with ED

- **Upper bound**: $O(n^2)$ – trivial. query all pairs of vertices.
- **Lower bound**: $\Omega(n^2)$

requires $\Omega(n^2)$ by adversarial argument
Results Summary

\( n = \) number of nodes, \(|E| = \) number of edges, 
\( k = \) number of connected components, \( d = \) maximum degree

<table>
<thead>
<tr>
<th>Query</th>
<th>partition</th>
<th>graph</th>
<th>tree</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td>( \Theta(n^2) )</td>
<td>( \Theta(n^2) )</td>
<td>( \Theta(n^2) )</td>
</tr>
<tr>
<td>ED</td>
<td>( \Theta(n^2) )</td>
<td>( \Theta(</td>
<td>E</td>
</tr>
</tbody>
</table>
| EC    | \( O(n \log n) \)  \\
|       | \( \Omega(n) \)  | \( O(|E| \log n), O(\frac{n^2}{\log n}), O(dn) \) | \( \Theta(n) \) |
| SP    | \( \Theta(nk) \)  | \( \Theta(n^2) \) | \( \Theta(n^2), \Theta(dn \log n) \) |
| C     | \( \Theta(nk) \)  | not possible | not possible |

\( E = \) Edge Query, \( ED = \) Edge Detection Query, \( EC = \) Edge Counting Query, 
\( SP = \) Shortest Path Query, \( C = \) Connectedness Query

references omitted
Learning Partitions with C

- **Upper bound:** $O(nk)$ algorithm
  - Step 1: Place $v_1$ in its own component
  - Step $i > 1$: Query $C(v_i, v_w)$ for items $v_w$ from each existing component; if a "yes" is encountered, place $v_i$ in the corresponding component and continue to step $i+1$. Otherwise place $v_i$ in its own component.
An Example
Learning Partitions with C

- **Upper bound**: $O(nk)$
  - **correctness**: trivial
  - **running time**: For complexity, note that there are at most $k$ components at any step (since there are at most $k$ components at phase $n$ and components are never destroyed); hence $n$ vertices take at most $nk$ queries.
Back to Our Table

\( n = \) number of nodes, \(|E| = \) number of edges, \\
\( k = \) number of connected components, \( d = \) maximum degree

<table>
<thead>
<tr>
<th>Query</th>
<th>partition</th>
<th>graph</th>
<th>tree</th>
</tr>
</thead>
<tbody>
<tr>
<td>( E )</td>
<td>( \Theta(n^2) )</td>
<td>( \Theta(n^2) )</td>
<td>( \Theta(n^2) )</td>
</tr>
<tr>
<td>( ED )</td>
<td>( \Theta(n^2) )</td>
<td>( \Theta(</td>
<td>E</td>
</tr>
<tr>
<td>( EC )</td>
<td>( O(n \lg n) ) ( \Omega(n) )</td>
<td>( O(</td>
<td>E</td>
</tr>
<tr>
<td>( SP )</td>
<td>( \Theta(nk) )</td>
<td>( \Theta(n^2) )</td>
<td>( \Theta(n^2), \Theta(dn \lg n) )</td>
</tr>
<tr>
<td>( C )</td>
<td>( \Theta(nk) )</td>
<td>not possible</td>
<td>not possible</td>
</tr>
</tbody>
</table>

\( E \) = Edge Query, \( ED \) = Edge Detection Query, \( EC \) = Edge Counting Query, \\
\( SP \) = Shortest Path Query, \( C \) = Connectedness Query

references omitted
Learning Partitions with EC

- **Lower bound**: $\Omega(n)$

  information theoretic argument:
  - The number of partitions of an element set is given by the Bell number $B_n$.
  - $\lg(B_n) = n \lg n$ (de Bruijn ’81)
  - each EC query gives $\lg(C(n,2)) = \lg n$ bits.
  - we need $\Omega((n \lg n) / (\lg n)) = \Omega(n)$
An Algorithm for the Upper Bound

- **Upper bound:** $O(n \log n)$
  - Phase 1: set $C = \{c_1\}$ with $c_1=1$
  - Phase $i$: let $v = (v_{i+1})$ query $EC(C+v)$
    - if $EC(C+v)=EC(C)$ add a new component $c=v$ to $C$
    - else split $C$ into roughly equal halves $C_1$ and $C_2$ and query $EC(C_1+v),EC(C_2+v)$. Recurse until $EC(\{c_j\}+v) > EC(c_j)$ for a single component $c_j$. Call $c_j$ a live component. Repeat recursively on $C/c_j$ until all live components are found. Merge them and $v$ into 1 component in $C$. 

An Example

partitions

1 2 3 4 5 6 7 8

1 2 3 4 5 6 7 8

1 2 3 4 5 6 7 8

1 2 3 4 5 6 7 8

1 2 3 4 5 6 7 8

1 2 3 4 5 6 7 8

1 2 3 4 5 6 7 8

1 2 3 4 5 6 7 8

1 2 3 4 5 6 7 8

1 2 3 4 5 6 7 8

1 2 3 4 5 6 7 8

1 2 3 4 5 6 7 8

1 2 3 4 5 6 7 8

1 2 3 4 5 6 7 8

1 2 3 4 5 6 7 8

1 2 3 4 5 6 7 8

1 2 3 4 5 6 7 8

1 2 3 4 5 6 7 8

1 2 3 4 5 6 7 8
Proof Sketch

• **Correctness** of the algorithm is simple by induction on the phase. C contains the components of G[1 . . . i] at end of phase i

• **Running time** is bounded by $O(n \lg n)$ because we do $O(\lg n)$ queries to find each live component. But each time we find a live component, the number of components in our set decreases by 1. Since we can have at most n partitions, the total running time is bounded by $O(n \lg n)$
Back to Our Table

$n =$ number of nodes, $|E| =$ number of edges, 
$k =$ number of connected components, $d =$ maximum degree

<table>
<thead>
<tr>
<th>Query</th>
<th>partition</th>
<th>graph</th>
<th>tree</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td>$\Theta(n^2)$</td>
<td>$\Theta(n^2)$</td>
<td>$\Theta(n^2)$</td>
</tr>
<tr>
<td>ED</td>
<td>$\Theta(n^2)$</td>
<td>$\Theta(</td>
<td>E</td>
</tr>
<tr>
<td>EC</td>
<td>$O(n\lg n)$, $\Omega(n)$</td>
<td>$O(</td>
<td>E</td>
</tr>
<tr>
<td>SP</td>
<td>$\Theta(nk)$</td>
<td>$\Theta(n^2)$</td>
<td>$\Theta(n^2), \Theta(dn\lg n)$</td>
</tr>
<tr>
<td>C</td>
<td>$\Theta(nk)$</td>
<td>not possible</td>
<td>not possible</td>
</tr>
</tbody>
</table>

$E =$ Edge Query, $ED =$ Edge Detection Query, $EC =$ Edge Counting Query, 
$SP =$ Shortest Path Query, $C =$ Connectedness Query

references omitted
Learning Graphs with ED
(Angluin, Chen COLT ’04)

- Upper bound: $O(|E| \log n)$

- Lemma 1: if $S_1$ and $S_2$ are two non-empty independent sets of vertices in $G$. We can find $s$ edges between $S_1$ and $S_2$ in $O(s \log n)$
Learning Graphs w/ ED - continued
(Angluin, Chen COLT ’04)

- Upper bound: $O(|E| \log n)$

  - **Fact**: A graph with $m$ edges can be $O(m)^{1/2}$ colored
    - we can collapse pairs of vertices not joined by an edge, until we get a clique of $m$ edges. It can be $O(t)$-colored and has $O(t^2)$ vertices.

- **Lemma 2**: if $S_1$ and $S_2$ ($|S_1| < |S_2|$ are two non-empty sets of vertices in $G$ (w/ $s_1$ and $s_2$ edges respectively). We can find $s$ edges between $S_1$ and $S_2$ in $O(s \times \log|S_2| + s_1 + s_2)$
  - we color both $S_1$ and $S_2$ with $(s_1)^{1/2}$ and $(s_2)^{1/2}$ colors
  - for each pair of color classes in $S_1$ and $S_2$, we query the union
  - recurse
Learning Graphs w/ ED - continued
(Angluin, Chen COLT ’04)

• **Upper bound:** $O(|E| \log n)$

– The Algorithm:
  • 1) If $|V| = 2$, mark pair of vertices as edge and return
  • 2) Divide $V$ into halves $S_1$ and $S_2$. Ask $ED(S_1)$ and $ED(S_1)$
  • 3) Recursively solve the problem for $S_i$ if $ED(S_i) = 1$
  • 4) Using lemma 2, find edges between $S_1$ and $S_2$
Back to Our Table

\( n = \) number of nodes, \( |E| = \) number of edges, \\
\( k = \) number of connected components, \( d = \) maximum degree

<table>
<thead>
<tr>
<th>Query</th>
<th>partition</th>
<th>graph</th>
<th>tree</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td>( \Theta(n^2) )</td>
<td>( \Theta(n^2) )</td>
<td>( \Theta(n^2) )</td>
</tr>
<tr>
<td>ED</td>
<td>( \Theta(n^2) )</td>
<td>( \Theta(</td>
<td>E</td>
</tr>
<tr>
<td>EC</td>
<td>( O(n \lg n) ) ( \Omega(n) )</td>
<td>( O(</td>
<td>E</td>
</tr>
<tr>
<td>SP</td>
<td>( \Theta(nk) )</td>
<td>( \Theta(n^2) )</td>
<td>( \Theta(n^2), \Theta(dn \lg n) )</td>
</tr>
<tr>
<td>C</td>
<td>( \Theta(nk) )</td>
<td>not possible</td>
<td>not possible</td>
</tr>
</tbody>
</table>

\( E \) = Edge Query, \( ED \) = Edge Detection Query, \( EC \) = Edge Counting Query, \\
\( SP \) = Shortest Path Query, \( C \) = Connectedness Query

references omitted
Learning Trees with SP Queries
(Hein, ’89)

- **Upper bound:** $O(d \cdot n \log n)$
  - **Proof intuition:**

![Diagram of a tree structure]
Open Problems and Future Work

Close the gap for partition with EC queries

Consider other queries: traceroute queries, beacon queries,…

Graph Verification. Given a result from a certain class, how hard is it to verify. Very interesting: verifying general graphs with EC queries.
Beacon and Traceroute Queries

- **Beacon**
  - Nodes labeled 1, 2, 3, and \( \infty \)

- **Traceroute**
  - Route from source to target

- **Target**
  - Complex network with multiple nodes and connections
Verification
Thank You

Questions?