Constructing Hamiltonian Circuits

When all nodes have degree of at least n/2
(Also: an implementation in C++ using Boost)
Presented by Alan Hogan @ SUnMaRC, February 2008
Graphs

- A graph is a collection of vertices (or points) and edges (which connect the vertices).
Example Graph

Vertex

Edge
Paths and Circuits

• Paths are series of vertices connected by edges

• A circuit is a closed path (starts & ends at the same vertex)
Hamiltonian Circuits

- A Hamiltonian circuit is a closed path which visits every vertex in the graph exactly one time

- Also called “Hamiltonian Cycles”
Problem

- General algorithms to find Hamiltonian circuits are **slow**, running in non-polynomial time — it is an NP-complete problem

- We can use an efficient algorithm, however, in some cases, thanks to Dirac and Ore...
Dirac’s Theorem (1952)

- A simple graph with $n$ vertices ($n > 2$) is Hamiltonian if each vertex has degree $n/2$ or greater.

- (sufficient but not necessary)
Ore’s Theorem (1960)

- Generalization of Dirac’s Theorem

- If $G$ is a simple graph with $n$ vertices, where $n \geq 3$, and if for each pair of non-adjacent vertices $v$ and $w$, $\deg(v) + \deg(w) \geq n$, then $G$ is Hamiltonian
Also in Ore’s paper...

- Ore’s restatement of Dirac’s principle lends itself to an interesting & useful principle

- For graphs satisfying the pre-requisite condition, an existing almost-complete Hamiltonian circuit with a gap between vertices A and B where there “should” be the final edge can be “repaired.”
Using Ore’s Algorithm

1. Find two vertices $C$ & $D$ s.t. edges $(A,C)$ and $(B,D)$ exist; $(C,D)$ is in our almost-complete circuit; and $D$ lies between $C$ and $A$ on the partial circuit.

2. Connect vertex $A$ to vertex $C$.

3. Connect vertex $B$ to a vertex $D$.

4. Remove the edge between those two earlier vertices.
Using Ore’s Algorithm

1. Find a two vertices C & D s.t. edges (A,C) and (B,D) exist; (C,D) is in our almost-complete circuit; and D lies between C and A on the partial circuit.
2. Connect vertex A to vertex C
3. Connect vertex B to a vertex D
4. Remove the edge between those two earlier vertices.
Using Ore’s Algorithm

1. Find a two vertices $C$ & $D$ s.t. edges $(A,C)$ and $(B,D)$ exist; $(C,D)$ is in our almost-complete circuit; and $D$ lies between $C$ and $A$ on the partial circuit.

2. Connect vertex $A$ to vertex $C$

3. Connect vertex $B$ to a vertex $D$

4. Remove the edge between those two earlier vertices.
Using Ore's Algorithm

1. Find a two vertices C & D s.t. edges (A,C) and (B,D) exist; (C,D) is in our almost-complete circuit; and D lies between C and A on the partial circuit.

2. Connect vertex A to vertex C

3. Connect vertex B to a vertex D

4. Remove the edge between those two earlier vertices.
Using Ore’s Algorithm

1. Find two vertices $C$ & $D$ s.t. edges $(A,C)$ and $(B,D)$ exist; $(C,D)$ is in our almost-complete circuit; and $D$ lies between $C$ and $A$ on the partial circuit.

2. Connect vertex $A$ to vertex $C$.

3. Connect vertex $B$ to a vertex $D$.

4. Remove the edge between those two earlier vertices.
Using Ore’s Algorithm

1. Find two vertices C & D s.t. edges (A,C) and (B,D) exist; (C,D) is in our almost-complete circuit; and D lies between C and A on the partial circuit.

2. Connect vertex A to vertex C

3. Connect vertex B to a vertex D

4. Remove the edge between those two earlier vertices.
Why does that work?

- We know that at least one pair of such desirable contiguous earlier vertices $C$ and $D$ exist because each vertex has at least half as many edges as there are vertices.
- Proof by the pigeonhole principle.
- Boxes = potential pairs of vertices $C$ & $D = n-3$
- Pigeons = edges from $A$ or $B = 2(n/2-1) = n-2$
Worst case
(Edges not connected to A or B and not on the circuit are not depicted)
Worst case
(Edges not connected to A or B and not on the circuit are not depicted)
Worst case

(Edges not connected to A or B and not on the circuit are not depicted)
Repeated Use

- Repeated use of the algorithm suggested by Ore’s paper allows us to find a Hamiltonian circuit for any graph in our scope (all vertices have at least $n/2$ edges)
Repeating the Algorithm

1. Pretend we have a circuit
2. Acknowledge one pretend edge does not really exist
3. Fix that edge. We have a pretend circuit again, but it’s closer to true
4. Go back to step 2. Repeat until all edges really exist
Repeating the Algorithm

1. Pretend we have a circuit
2. Acknowledge one pretend edge does not really exist
3. Fix that edge. We have a pretend circuit again, but it’s closer to true
4. Go back to step 2. Repeat until all edges really exist
1. Pretend we have a circuit
2. Acknowledge one pretend edge does not really exist
3. Fix that edge. We have a pretend circuit again, but it’s closer to true
4. Go back to step 2. Repeat until all edges really exist
Repeating the Algorithm

1. Pretend we have a circuit
2. Acknowledge one pretend edge does not really exist
3. Fix that edge. We have a pretend circuit again, but it’s closer to true
4. Go back to step 2. Repeat until all edges really exist
Repeating the Algorithm

1. Pretend we have a circuit
2. Acknowledge one pretend edge does not really exist
3. Fix that edge. We have a pretend circuit again, but it’s closer to true
4. Go back to step 2. Repeat until all edges really exist
1. Pretend we have a circuit
2. Acknowledge one pretend edge does not really exist
3. Fix that edge. We have a pretend circuit again, but it’s closer to true
4. Go back to step 2. Repeat until all edges really exist
Repeating the Algorithm

1. Pretend we have a circuit
2. Acknowledge one pretend edge does not really exist
3. Fix that edge. We have a pretend circuit again, but it’s closer to true
4. Go back to step 2. Repeat until all edges really exist
Repeating the Algorithm

1. Pretend we have a circuit

2. Acknowledge one pretend edge does not really exist

3. Fix that edge. We have a pretend circuit again, but it’s closer to true

4. Go back to step 2. Repeat until all edges really exist
Repeating the Algorithm

1. Pretend we have a circuit
2. Acknowledge one pretend edge does not really exist
3. Fix that edge. We have a pretend circuit again, but it’s closer to true
4. Go back to step 2. Repeat until all edges really exist
Repeating the Algorithm

1. Pretend we have a circuit
2. Acknowledge one pretend edge does not really exist
3. Fix that edge. We have a pretend circuit again, but it’s closer to true
4. Go back to step 2. Repeat until all edges really exist
Repeating the Algorithm

1. Pretend we have a circuit
2. Acknowledge one pretend edge does not really exist
3. Fix that edge. We have a pretend circuit again, but it's closer to true
4. Go back to step 2. Repeat until all edges really exist
Repeating the Algorithm

1. Pretend we have a circuit
2. Acknowledge one pretend edge does not really exist
3. Fix that edge. We have a pretend circuit again, but it’s closer to true
4. Go back to step 2. Repeat until all edges really exist
Repeating the Algorithm

1. Pretend we have a circuit
2. Acknowledge one pretend edge does not really exist
3. Fix that edge. We have a pretend circuit again, but it’s closer to true
4. Go back to step 2. Repeat until all edges really exist
Repeating the Algorithm

1. Pretend we have a circuit
2. Acknowledge one pretend edge does not really exist
3. Fix that edge. We have a pretend circuit again, but it’s closer to true
4. Go back to step 2. Repeat until all edges really exist
Repeating the Algorithm

1. Pretend we have a circuit
2. Acknowledge one pretend edge does not really exist
3. Fix that edge. We have a pretend circuit again, but it’s closer to true
4. Go back to step 2. Repeat until all edges really exist
Implementation

• The algorithm as discussed was slightly modified to use two graphs – the pretend circuit, and the true graph

• Implemented in C++ using the Boost graph library and Xcode

• Command-line only (but a GUI frontend could be constructed)
Thank you.

alanhogan.com/asu/hamiltonian-circuit

Read more about this project, download this presentation, or get a copy of the source code online.
Thank you.

alanhogan.com/asu/hamiltonian-circuit

Read more about this project, download this presentation, or get a copy of the source code online.