CS 4884: Introduction to Graphs

T. M. Murali

January 29, 2019
The Oracle of Bacon
Introduction

Euler Tours

Heilholzer’s Algorithm

Hamiltonian Cycles

T. M. Murali

January 29, 2019

CS 4884: Computing the Brain
Graphs

Graph $\equiv$ Network

- Model pairwise relationships (edges) between objects (nodes).
Graphs

Graph ≡ Network

- Model pairwise relationships (edges) between objects (nodes).
- Useful in a large number of applications: gene and protein networks, our bodies (nervous and circulatory systems, brains).
- Other examples:
Graphs

Graph \equiv \text{Network}

- Model pairwise relationships (edges) between objects (nodes).
- Useful in a large number of applications: gene and protein networks, our bodies (nervous and circulatory systems, brains).
- Other examples: computer networks, the World Wide Web, ecology (food webs), social networks, software systems, job scheduling, VLSI circuits, cellular networks, transportation networks, . . .
Introduction Euler Tours Heilholzer’s Algorithm Hamiltonian Cycles

Graphs

Graph \equiv \text{Network}

- Model pairwise relationships (edges) between objects (nodes).
- Useful in a large number of applications: gene and protein networks, our bodies (nervous and circulatory systems, brains).
- Other examples: computer networks, the World Wide Web, ecology (food webs), social networks, software systems, job scheduling, VLSI circuits, cellular networks, transportation networks, . . .
- Problems involving graphs have a rich history dating back to Euler.
Devise a walk through the city that crosses each of the seven bridges exactly once.
Euler and Graphs
Definition of an Undirected Graph

- **Undirected graph** $G = (V, E)$: set $V$ of nodes and set $E$ of edges.
  - Each element of $E$ is an unordered pair of nodes.
  - Edge $(u, v)$ is *incident* on $u$, $v$; $u$ and $v$ are *neighbours* of each other.
  - $G$ contains no self loops.
Definition of an Undirected Graph

- **Undirected graph** $G = (V, E)$: set $V$ of nodes and set $E$ of edges.
  - Each element of $E$ is an unordered pair of nodes.
  - Edge $(u, v)$ is *incident* on $u, v$; $u$ and $v$ are *neighbours* of each other.
  - $G$ contains no self loops.
Definition of an Undirected Graph

- **Undirected graph** \( G = (V, E) \): set \( V \) of nodes and set \( E \) of edges.
  - Each element of \( E \) is an unordered pair of nodes.
  - Edge \((u, v)\) is *incident* on \( u, v \); \( u \) and \( v \) are *neighbours* of each other.
  - \( G \) contains no self loops.
A \textit{v}_1-\textit{v}_k \textit{ path} in an undirected graph \( G = (V, E) \) is a sequence of nodes \( v_1, v_2, \ldots, v_{k-1}, v_k \in V \) such that for every \( i, 1 \leq i < k \), \((v_i, v_{i+1})\) is an edge in \( E \).
A \( v_1 - v_k \) path in an undirected graph \( G = (V, E) \) is a sequence of nodes \( v_1, v_2, \ldots, v_{k-1}, v_k \in V \) such that for every \( i, 1 \leq i < k \), \((v_i, v_{i+1})\) is an edge in \( E \).
A $v_1$-$v_k$ path in an undirected graph $G = (V, E)$ is a sequence of nodes $v_1, v_2, \ldots, v_{k-1}, v_k \in V$ such that for every $i, 1 \leq i < k$, $(v_i, v_{i+1})$ is an edge in $E$. 
A $v_1-v_k$ \textit{path} in an undirected graph $G = (V, E)$ is a sequence of nodes $v_1, v_2, \ldots, v_{k-1}, v_k \in V$ such that for every $i, 1 \leq i < k$, $(v_i, v_{i+1})$ is an edge in $E$.

A path is \textit{simple} if all its nodes are distinct.
A \( v_1 \)-\( v_k \) path in an undirected graph \( G = (V, E) \) is a sequence of nodes \( v_1, v_2, \ldots, v_{k-1}, v_k \in V \) such that for every \( i, 1 \leq i < k \), \( (v_i, v_{i+1}) \) is an edge in \( E \).

A path is simple if all its nodes are distinct.

A cycle is a path where the first \( k - 1 \) nodes are distinct and \( v_1 = v_k \).
A $v_1$-$v_k$ path in an undirected graph $G = (V, E)$ is a sequence of nodes $v_1, v_2, \ldots, v_{k-1}, v_k \in V$ such that for every $i, 1 \leq i < k$, $(v_i, v_{i+1})$ is an edge in $E$.

A path is **simple** if all its nodes are distinct.

A **cycle** is a path where the first $k-1$ nodes are distinct and $v_1 = v_k$.

An undirected graph $G$ is **connected** if for every pair of nodes $u, v \in V$, there is a $u$-$v$ path in $G$. 

Eulerian tour
Given an undirected graph $G(V, E)$, construct an *Eulerian tour*, i.e., a path in $G$ that traverses each edge in $E$ exactly once.
Eulerian tour

Given an undirected graph $G(V, E)$, construct an *Eulerian tour*, i.e., a path in $G$ that traverses each edge in $E$ exactly once, if such a tour exists.
What Euler Proved

§ 19. Praeterea si duo tantum numeri litteris A, B, C etc. adscripti fuerint impares, reliqui vero omnes pares, tum semper desideratus transitus succedet, si modo cursus ex regione ad quam pontium impar numerus tendit incipiatur. Si enim pares numeri bifcentur atque etiam impares nunitate auxi, ut praeceptum est, summa harum medietarum nunitate erit maior quam numerus pontium, ideoque aequalis ipse numero praefixo. Ex hocque porro perspicitur, si quatuor vel sex vel octo etc. fuerint numeri impares in secunda columna, tum tamen numerorum tertiae columnae maiorem fore numero praefixo, cunque excedere vel nunitate, vel binario vel ternario etc. et idcirco transitus fieri nequit.

§ 20. Caeo ergo quocunque proposito flatim faciillime poterit cognosciri, vtrum transitus per omnes pontes semel institutus quat an non, ope huius regulae. Si fuerint plures duabus regiones, ad quas ducentium pontium numerus est impar, tum certo affirmari potest, talem transitum non dari. Si autem ad duas tantum regiones ducentium pontium numerus est impar, tum transitus fieri potest, si modo cursus in altera harum regionum incipiatur. Si denique nulla omnino fuerit regio, ad quam pontes numero impares conducant, tum transitus desiderato modo institui poterit, in quacunque regione ambulantia initium ponatur. Hae igitur data regula problematii proposito plenissime satisfit.

§ 2  § 21.
What Euler Proved (in English)

Degree $d(v)$ of a node $v$ is the number of edges incident on it.
Degree $d(v)$ of a node $v$ is the number of edges incident on it.

Euler’s conclusion:
1. If there are more than two nodes with odd degree, then the graph has no Eulerian tour.
2. If exactly two nodes in the graph have odd degree, then there exists a tour that starts at one of these nodes and ends at the other node.
3. If all nodes have even degree, then there exists a tour starting at any node.
What Didn’t Euler Prove?

- Implicit assumption: $G$ is connected.
What Didn’t Euler Prove?

- Implicit assumption: $G$ is connected.
- Euler’s conditions (1741) were necessary. Hierholzer proved their sufficiency (1873).
What Didn’t Euler Prove?

- Implicit assumption: $G$ is connected.
- Euler’s conditions (1741) were necessary. Hierholzer proved their sufficiency (1873).
- What about constructing such a tour if it exists?
What Didn’t Euler Prove?

§ 21: Quando autem invenitum fuerit talem transitum instituit posse, quae sit superest quomodo curfus sit dirigendus. Pro hoc sequenti vtor regul; tolluntur cogitatione quoties fieri potest, bini pontes, qui ex una regione in aliam ducunt, quo pacto pontium numerus vehementer plerunque diminuetur, tum quaeratur, quod facile fier, curfus desideratus per pontes reliquos, quo inuenit pontes cogitatione sublati hunc ipsum curfum non multum turbabunt, id quod paululum attenti statim patebit; neque opus esse indeo plura ad curfus reipfa formandos praecipere.
What Didn’t Euler Prove?

- Implicit assumption: $G$ is connected.
- Euler’s conditions (1741) were necessary. Hierholzer proved their sufficiency (1873).
- What about constructing such a tour if it exists?
  - We must go through the effort to write out a path that is correct.
  - Method to accomplish this was trivial, and Euler did not want to spend a great deal of time on it.
What Didn’t Euler Prove?

- Implicit assumption: $G$ is connected.
- Euler’s conditions (1741) were necessary. Hierholzer proved their sufficiency (1873).
- What about constructing such a tour if it exists?
  - We must go through the effort to write out a path that is correct.
  - Method to accomplish this was trivial, and Euler did not want to spend a great deal of time on it.
- Hierholzer provided an algorithm.
Hierholzer’s Algorithm

If there are two nodes in $G$ with odd degree, call them $s$ and $t$. Otherwise, let $s$ be any node in $G$.

Set $u ← s$.

while $d(u) > 0$

Output $u$.

Let $v$ be a neighbour of $u$.

Delete the edge $(u, v)$ from $G$.

Set $u ← v$.

end while
Hierholzer’s Algorithm

- If there are two nodes in $G$ with odd degree, call them $s$ and $t$.
- Otherwise, let $s$ be any node in $G$.

T. M. Murali January 29, 2019 CS 4884: Computing the Brain
Hierholzer’s Algorithm

If there are two nodes in $G$ with odd degree, call them $s$ and $t$.
Otherwise, let $s$ be any node in $G$.

$u \leftarrow s$ # $u$ denotes the currently-visited node.

while $d(u) > 0$ do
  Output $u$.
  Let $v$ be a neighbour of $u$.
  Delete the edge $(u, v)$ from $G$.
  $u \leftarrow v$
end while
Hierholzer’s Algorithm

If there are two nodes in $G$ with odd degree, call them $s$ and $t$.
Otherwise, let $s$ be any node in $G$.

$u \leftarrow s \ #u$ denotes the currently-visited node.

**while** $d(u) > 0$ **do**

Output $u$.
Let $v$ be a neighbour of $u$.
Delete the edge $(u, v)$ from $G$.

$u \leftarrow v$

**end while**
Properties of Heilholzer’s Algorithm

```
u ← s
while d(u) ≥ 0 do
    Output u.
    Let v be a neighbour of u.
    Delete the edge (u, v) from G.
    u ← v
end while
```

If $G$ had no nodes of odd degree, then $u = s$.

If $G$ had two nodes of odd degree, then $u = t$.
### Properties of Heilholzer’s Algorithm

- **Will the algorithm terminate?**
- **If it terminates, what can we say about node \( u \) at termination?**
- **Will all edges of \( G \) have been traversed upon termination?**

---

**Algorithm**

\[ u \leftarrow s \]

**while** \( d(u) > 0 \) **do**

- Output \( u \).
- Let \( v \) be a neighbour of \( u \).
- Delete the edge \((u, v)\) from \( G \).
- \( u \leftarrow v \)

**end while**

---

T. M. Murali January 29, 2019 CS 4884: Computing the Brain
Properties of Heilholzer’s Algorithm

- Will the algorithm terminate? Yes, because we traverse a new edge in each iteration.
- If it terminates, what can we say about node $u$ at termination?

- Will all edges of $G$ have been traversed upon termination?
Properties of Heilholzer’s Algorithm

Will the algorithm terminate? Yes, because we traverse a new edge in each iteration.

If it terminates, what can we say about node \( u \) at termination?
- If \( G \) had no nodes of odd degree, then \( u = s \).
- If \( G \) had two nodes of odd degree, then \( u = t \).

Will all edges of \( G \) have been traversed upon termination?

```plaintext
u ← s

while \( d(u) > 0 \) do
  Output \( u \).
  Let \( v \) be a neighbour of \( u \).
  Delete the edge \((u, v)\) from \( G \).
  \( u ← v \)
end while
```
Properties of Heilholzer’s Algorithm

- Will the algorithm terminate? Yes, because we traverse a new edge in each iteration.
- If it terminates, what can we say about node $u$ at termination?
  - If $G$ had no nodes of odd degree, then $u = s$.
  - If $G$ had two nodes of odd degree, then $u = t$.
- Will all edges of $G$ have been traversed upon termination? No!

```plaintext
u ← s
while $d(u) > 0$ do
    Output $u$.
    Let $v$ be a neighbour of $u$.
    Delete the edge $(u, v)$ from $G$.
    $u ← v$
end while
```
Properties of Heilholzer’s Algorithm

Will the algorithm terminate? Yes, because we traverse a new edge in each iteration.

If it terminates, what can we say about node $u$ at termination?
- If $G$ had no nodes of odd degree, then $u = s$.
- If $G$ had two nodes of odd degree, then $u = t$.

Will all edges of $G$ have been traversed upon termination? No! Set $u$ to be any node on the path output so far and repeat.

Algorithm's running time is $O(|V| + |E|)$, i.e., linear in the size of $G$.

```
u ← s
while $d(u) > 0$ do
    Output $u$.
    Let $v$ be a neighbour of $u$.
    Delete the edge $(u, v)$ from $G$.
    $u ← v$
end while
```
Properties of Heilholzer’s Algorithm

- Will the algorithm terminate? Yes, because we traverse a new edge in each iteration.
- If it terminates, what can we say about node $u$ at termination?
  - If $G$ had no nodes of odd degree, then $u = s$.
  - If $G$ had two nodes of odd degree, then $u = t$.
- Will all edges of $G$ have been traversed upon termination? No! Set $u$ to be any node on the path output so far and repeat.

```plaintext
u ← s
while $d(u) > 0$ do
  Output $u$.
  Let $v$ be a neighbour of $u$.
  Delete the edge $(u, v)$ from $G$.
  $u ← v$
end while
```
Properties of Heilholzer’s Algorithm

Will the algorithm terminate? Yes, because we traverse a new edge in each iteration.

If it terminates, what can we say about node $u$ at termination?

- If $G$ had no nodes of odd degree, then $u = s$.
- If $G$ had two nodes of odd degree, then $u = t$.

Will all edges of $G$ have been traversed upon termination? No! Set $u$ to be any node on the path output so far and repeat.
### Properties of Heilholzer’s Algorithm

- **Will the algorithm terminate?** Yes, because we traverse a new edge in each iteration.
- **If it terminates, what can we say about node $u$ at termination?**
  - If $G$ had no nodes of odd degree, then $u = s$.
  - If $G$ had two nodes of odd degree, then $u = t$.
- **Will all edges of $G$ have been traversed upon termination?** No! Set $u$ to be any node on the path output so far and repeat.
- **Algorithm’s running time is $O(|V| + |E|)$**, i.e., linear in the size of $G$. 

**Algorithm**

\[
\begin{align*}
  &u \leftarrow s \\
  &\textbf{while } d(u) > 0 \textbf{ do} \\
  &\quad \text{Output } u. \\
  &\quad \text{Let } v \text{ be a neighbour of } u. \\
  &\quad \text{Delete the edge } (u, v) \text{ from } G. \\
  &\quad u \leftarrow v \\
  &\textbf{end while}
\end{align*}
\]
Eulerian tour
Given an undirected graph $G(V, E)$, construct an *Eulerian tour*, i.e., a path in $G$ that traverses each edge in $E$ exactly once, if such a tour exists.
Hamiltonian cycle
Given an undirected graph $G(V, E)$, construct an Hamiltonian cycle, i.e., a cycle in $G$ that traverses each node in $V$ exactly once, if such a tour exists.
Conditions for Existence of Hamiltonian Cycle

- $G$ has a Hamiltonian cycle if $G$ is a cycle.
- An $n$-node graph $G$ has a Hamiltonian cycle if each node has degree $n - 2$.
- Each node has degree $\geq n/2$ (Dirac, 1952).
- Two disconnected nodes with sum of degrees $\geq n$ (Ore, 1952).
Conditions for Existence of Hamiltonian Cycle

- $G$ has a Hamiltonian cycle if $G$ is a cycle.
- An $n$-node graph $G$ has a Hamiltonian cycle if each node has degree $n - 1$. 
Conditions for Existence of Hamiltonian Cycle

- $G$ has a Hamiltonian cycle if $G$ is a cycle.
- An $n$-node graph $G$ has a Hamiltonian cycle
  - if each node has degree $n - 1$.
  - each node has degree $n - 2$. 
Conditions for Existence of Hamiltonian Cycle

- $G$ has a Hamiltonian cycle if $G$ is a cycle.
- An $n$-node graph $G$ has a Hamiltonian cycle
  - if each node has degree $n - 1$.
  - each node has degree $n - 2$.
  - each node has degree $\geq n/2$ (Dirac, 1952).
**Conditions for Existence of Hamiltonian Cycle**

- $G$ has a Hamiltonian cycle if $G$ is a cycle.
- An $n$-node graph $G$ has a Hamiltonian cycle
  - if each node has degree $n - 1$.
  - each node has degree $n - 2$.
  - each node has degree $\geq n/2$ (Dirac, 1952).
  - two disconnected nodes with sum of degrees $\geq n$ (Ore, 1952).
Status of Hamiltonian Cycle Problem

**Hamiltonian cycle**

Given an undirected graph \( G(V, E) \), construct an *Hamiltonian cycle*, i.e., a cycle in \( G \) that traverses each node in \( V \) exactly once, if such a tour exists.

- The Hamiltonian cycle problem is NP-complete,
Hamiltonian cycle

Given an undirected graph $G(V, E)$, construct an Hamiltonian cycle, i.e., a cycle in $G$ that traverses each node in $V$ exactly once, if such a tour exists.

- The Hamiltonian cycle problem is NP-complete, it is very unlikely that we will find a polynomial time algorithm to check if an undirected graph contains such a cycle.
Status of Hamiltonian Cycle Problem

Hamiltonian cycle

Given an undirected graph $G(V, E)$, construct an Hamiltonian cycle, i.e., a cycle in $G$ that traverses each node in $V$ exactly once, if such a tour exists.

- The Hamiltonian cycle problem is NP-complete, it is very unlikely that we will find a polynomial time algorithm to check if an undirected graph contains such a cycle.
- Algorithms for computing Hamiltonian cycle:
Status of Hamiltonian Cycle Problem

Hamiltonian cycle

Given an undirected graph $G(V, E)$, construct an Hamiltonian cycle, i.e., a cycle in $G$ that traverses each node in $V$ exactly once, if such a tour exists.

- The Hamiltonian cycle problem is NP-complete, it is very unlikely that we will find a polynomial time algorithm to check if an undirected graph contains such a cycle.
- Algorithms for computing Hamiltonian cycle:
  - Brute force: try all permutations. Running time is $O(n^2 n!)$. 
Status of Hamiltonian Cycle Problem

**Hamiltonian cycle**

Given an undirected graph $G(V, E)$, construct an *Hamiltonian cycle*, i.e., a cycle in $G$ that traverses each node in $V$ exactly once, if such a tour exists.

- The Hamiltonian cycle problem is NP-complete, it is very unlikely that we will find a polynomial time algorithm to check if an undirected graph contains such a cycle.

- Algorithms for computing Hamiltonian cycle:
  - Brute force: try all permutations. Running time is $O(n^2n!)$.
  - Dynamic programming: running time of $O(n^22^n)$ (Held and Karp 1962).
  - Fastest known algorithm runs in time $O(1.657^n)$ (Björklund 2010).