



POLITECNICO
DI TORINO



e-Lite



Graphs: Cycles

Tecniche di Programmazione – A.A. 2018/2019

Summary

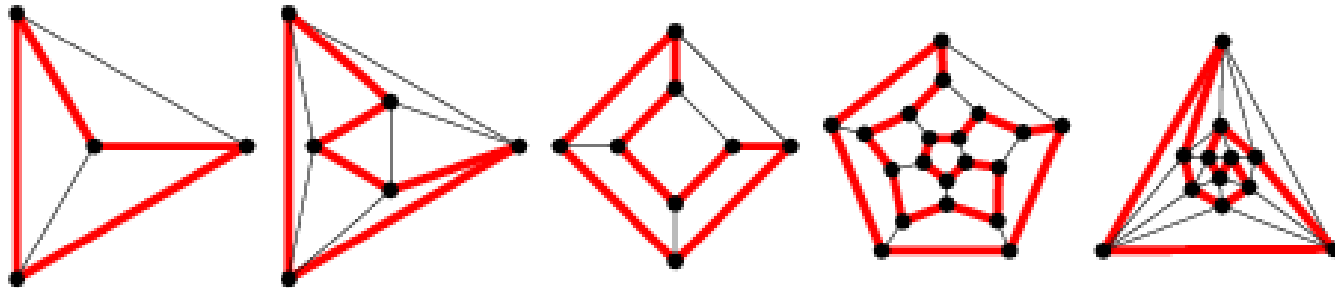
- ▶ Definitions
- ▶ Algorithms

Cycle

- ▶ A **cycle** of a graph, sometimes also called a circuit, is a subset of the edge set of that forms a path such that the first node of the path corresponds to the last.

Hamiltonian cycle

- ▶ A cycle that uses each graph vertex of a graph exactly once is called a Hamiltonian cycle.



Hamiltonian path

- ▶ A Hamiltonian path, also called a Hamilton path, is a path between two vertices of a graph that visits each vertex exactly once.
 - ▶ N.B. does not need to return to the starting point

Eulerian Path and Cycle

- ▶ An **Eulerian path**, also called an Euler chain, Euler trail, Euler walk, or "Eulerian" version of any of these variants, is a walk on the graph edges of a graph which **uses each graph edge** in the original graph **exactly once**.
- ▶ An **Eulerian cycle**, also called an Eulerian circuit, Euler circuit, Eulerian tour, or Euler tour, is a trail which starts and ends at the **same** graph vertex.

Theorem

- ▶ A connected graph has an Eulerian **cycle** if and only if it **all vertices have even degree**.
- ▶ A connected graph has an Eulerian **path** if and only if it has **at most two graph vertices of odd degree**.
- ▶ ...easy to check!

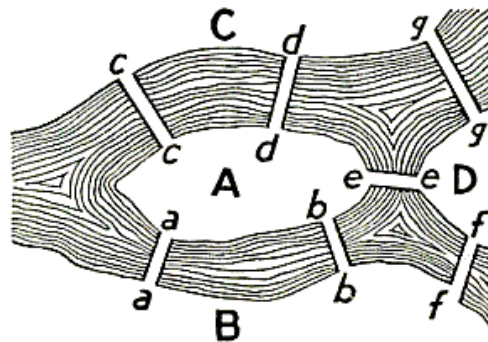
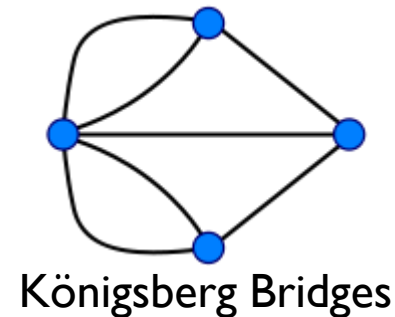


FIGURE 98. *Geographic Map:
The Königsberg Bridges.*



Königsberg Bridges

Weighted vs. Unweighted

- ▶ Classical versions defined on Unweighted graphs
- ▶ Unweighted:
 - ▶ Does such a cycle exist?
 - ▶ If yes, find at least one
 - ▶ Optionally, find all of them
- ▶ Weighted
 - ▶ Does such a cycle exist?
 - ▶ Often, the graph is complete 😊
 - ▶ If yes, find at least one
 - ▶ If yes, find **the best one** (with **minimum** weight)

Eulerian cycles: Hierholzer's algorithm (1)

- ▶ Choose **any** starting vertex v , and **follow a trail** of edges from that vertex until returning to v .
 - ▶ It is **not** possible to get stuck at any vertex other than v , because the even degree of all vertices ensures that, when the trail enters another vertex w there must be an unused edge leaving w .
 - ▶ The tour formed in this way is a **closed** tour, but may **not** cover all the vertices and edges of the initial graph.

Eulerian cycles: Hierholzer's algorithm (2)

- ▶ As long as there exists a vertex v that belongs to the current tour but that has adjacent edges not part of the tour, **start another trail** from v , following **unused** edges until returning to v , **and join** the tour formed in this way to the previous tour.

Finding Eulerian circuits

Hierholzer's Algorithm

Given: an Eulerian graph G

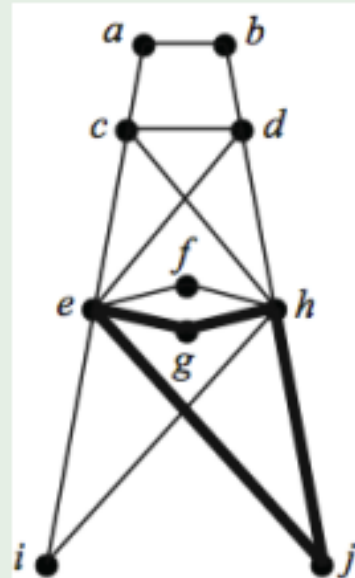
Find an Eulerian circuit of G .

- 1 Identify a circuit in G and call it R_1 . Mark the edges of R_1 . Let $i = 1$.
- 2 If R_i contains all edges of G , then stop (since R_i is an Eulerian circuit).
- 3 If R_i does not contain all edges of G , then let v_i be a node on R_i that is incident with an unmarked edge, e_i .
- 4 Build a circuit, Q_i , starting at node v_i and using edge e_i . Mark the edges of Q_i .
- 5 Create a new circuit, R_{i+1} , by patching the circuit Q_i into R_i at v_i .
- 6 Increment i by 1, and go to step (2).

Finding Eulerian circuits

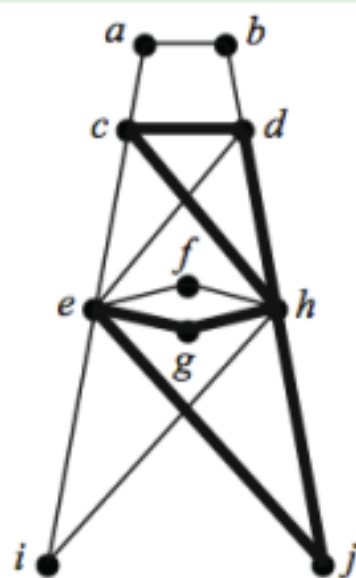
Hierholzer's Algorithm

Example



$R_1: e, g, h, j, e$

$Q_1: h, d, c, h$



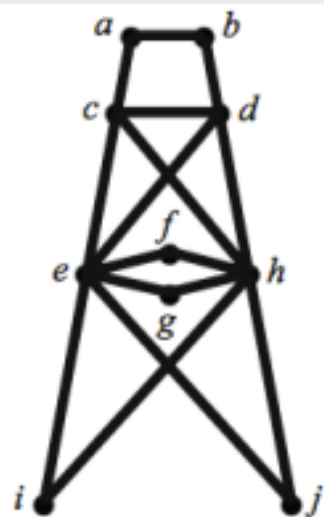
$R_2: e, g, h, d, c, h, j, e$

$Q_2: d, b, a, c, e, d$

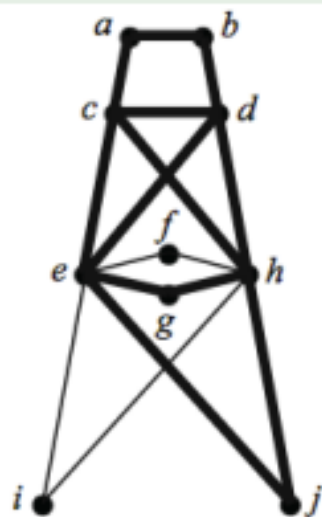
Finding Eulerian circuits

Hierholzer's Algorithm

Example (continued)



R_4 : $e, g, h, f, e, i, h, d, b, a,$
 c, e, d, c, h, j, e



R_3 : $e, g, h, d, b, a, c, e, d, c, h, j, e$
 Q_3 : h, f, e, i, h

Eulerian Circuits in JGraphT

org.jgrapht.alg.cycle

The screenshot shows the JGraphT API documentation for the `HierholzerEulerianCycle` class. The left sidebar lists various packages and classes, with `org.jgrapht.alg.cycle` selected. The main content area displays the class signature, type parameters, implemented interfaces, and a detailed description of the algorithm. The `Constructor Summary` section is highlighted.

OVERVIEW PACKAGE **CLASS** USE TREE DEPRECATED INDEX HELP

PREV CLASS NEXT CLASS FRAMES NO FRAMES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

org.jgrapht.alg.cycle

Class HierholzerEulerianCycle<V,E>

java.lang.Object
org.jgrapht.alg.cycle.HierholzerEulerianCycle<V,E>

Type Parameters:

- V - the graph vertex type
- E - the graph edge type

All Implemented Interfaces:

- EulerianCycleAlgorithm<V,E>

```
public class HierholzerEulerianCycle<V,E>  
    extends Object  
    implements EulerianCycleAlgorithm<V,E>
```

An implementation of Hierholzer's algorithm for finding an Eulerian cycle in Eulerian graphs. The algorithm works with directed and undirected graphs which may contain loops and/or multiple (parallel) edges. The running time is linear, i.e. $O(|E|)$ where $|E|$ is the cardinality of the edge set of the graph.

See the Wikipedia article for details and references about Eulerian cycles and a short description of Hierholzer's algorithm for the construction of an Eulerian cycle. The original presentation of the algorithm dates back to 1873 and the following paper: Carl Hierholzer: Über die Möglichkeit, einen Linienzug ohne Wiederholung und ohne Unterbrechung zu umfahren. *Mathematische Annalen* 6(1), 30–32, 1873.

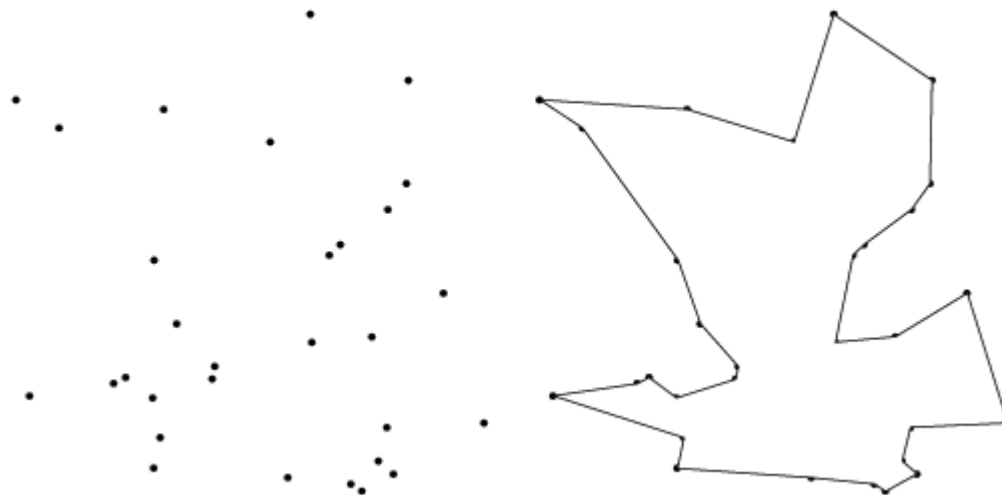
Author:
Dimitrios Michail

Constructor Summary

Constructors

Hamiltonian Cycles

- ▶ There are theorems to identify **whether** a graph is Hamiltonian (i.e., whether it contains at least one Hamiltonian Cycle)
- ▶ **Finding** such a cycle has **no** known efficient solution, in the general case
- ▶ Example: the **Traveling Salesman Problem (TSP)**



The Traveling Salesman Problem (TSP)

Weighted or
unweighted

Given a collection of cities connected by roads

Find the shortest route that visits each city exactly once.

About TSP

- Most notorious NP-complete problem.
- Typically, it is solved with a backtracking algorithm:
 - The best tour found to date is saved.
 - The search backtracks unless the partial solution is cheaper than the cost of the best tour.

Hamiltonian Cycles in JGraphT

org.jgrapht.alg.interfaces

The screenshot shows the JavaDoc page for the `org.jgrapht.alg.interfaces.HamiltonianCycleAlgorithm` interface. The page includes a navigation menu on the left with 'All Classes' and 'Packages' sections. The main content area has tabs for 'OVERVIEW', 'PACKAGE', 'CLASS', 'USE', 'TREE', 'DEPRECATED', 'INDEX', and 'HELP'. Below the tabs, there are links for 'PREV CLASS', 'NEXT CLASS', 'FRAMES', and 'NO FRAMES'. The 'SUMMARY' section shows the interface name and its type parameters: `V` for the graph vertex type and `E` for the graph edge type. It lists all known implementing classes, including `ChristofidesThreeHalvesApproxMetricTSP`, `HeldKarpTSP`, `PalmerHamiltonianCycle`, `TwoApproxMetricTSP`, and `TwoOptHeuristicTSP`. A description of the algorithm is provided, along with the author's name, Alexandru Valeanu. A 'Method Summary' section is also visible, showing a table with columns for 'All Methods', 'Instance Methods', and 'Abstract Methods'. The table lists the `getTour` method with its signature and description.

Overview: **Interface HamiltonianCycleAlgorithm<V,E>**

Type Parameters:
V - the graph vertex type
E - the graph edge type

All Known Implementing Classes:
<https://jgrapht.org/javadoc/org/jgrapht/alg/interfaces/HamiltonianCycleAlgorithm.html>

An algorithm solving the Hamiltonian cycle problem.

A Hamiltonian cycle, also called a Hamiltonian circuit, Hamilton cycle, or Hamilton circuit, is a graph cycle (i.e., closed loop) through a graph that visits each node exactly once (Skiena 1990, p. 196).

Author:
Alexandru Valeanu

Method Summary

All Methods	Instance Methods	Abstract Methods
Modifier and Type	Method and Description	
GraphPath<V, E>	getTour(Graph<V, E> graph)	Computes a tour.

<https://jgrapht.org/javadoc/org/jgrapht/alg/interfaces/HamiltonianCycleAlgorithm.html>

All Known Implementing Classes:
ChristofidesThreeHalvesApproxMetricTSP, HeldKarpTSP, PalmerHamiltonianCycle, TwoApproxMetricTSP, TwoOptHeuristicTSP

Limitations...

- ▶ **No exact solution (Approximate algorithms)**
 - ▶ Class `TwoApproxMetricTSP<V,E>`
 - ▶ Class `ChristofidesThreeHalvesApproxMetricTSP<V,E>`
 - ▶ Class `TwoOptHeuristicTSP<V,E>`
- ▶ **Or complete under extra conditions**
 - ▶ Class `PalmerHamiltonianCycle<V,E>`
- ▶ **Or complete but $O(2^N)$**
 - ▶ Class `HeldKarpTSP<V,E>`

The Metric Traveling Salesman Problem

An approximation algorithm

ASSUMPTION: G is a metric graph.

- 1 Compute a minimum weight spanning tree T for G .
- 2 Perform a depth-first traversal of T starting from any node, and order the nodes of G as they were discovered in this traversal.

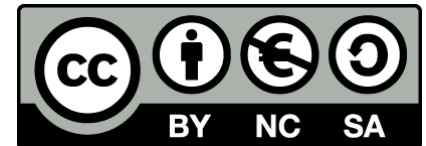
⇒ a tour that is at most twice the optimal tour in G .





ClassTwoApproxMetricTSP<V,E>

Resources

- ▶ <http://mathworld.wolfram.com/>
- ▶ http://en.wikipedia.org/wiki/Euler_cycle
- ▶ Mircea MARIN, Graph Theory and Combinatorics, Lectures 9 and 10, <http://web.info.uvt.ro/~mmarin/>

Licenza d'uso



- ▶ Queste diapositive sono distribuite con licenza Creative Commons “Attribuzione - Non commerciale - Condividi allo stesso modo (CC BY-NC-SA)”
- ▶ Sei libero:
 - ▶ di riprodurre, distribuire, comunicare al pubblico, esporre in pubblico, rappresentare, eseguire e recitare quest'opera 
 - ▶ di modificare quest'opera 
- ▶ Alle seguenti condizioni:
 - ▶ Attribuzione — Devi attribuire la paternità dell'opera agli autori originali e in modo tale da non suggerire che essi avallino te o il modo in cui tu usi l'opera. 
 - ▶ Non commerciale — Non puoi usare quest'opera per fini commerciali. 
 - ▶ Condividi allo stesso modo — Se alteri o trasformi quest'opera, o se la usi per crearne un'altra, puoi distribuire l'opera risultante solo con una licenza identica o equivalente a questa.
- ▶ <http://creativecommons.org/licenses/by-nc-sa/3.0/>

