Graph Theory

Euler tours and Chinese postmen

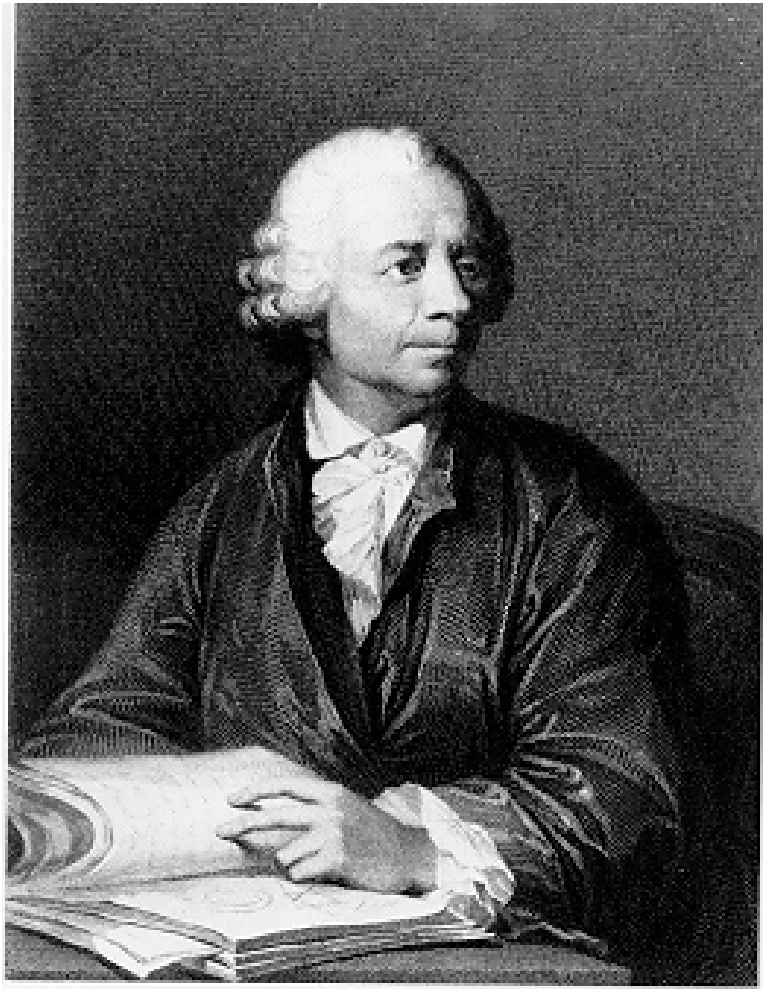
John Quinn

Week 5

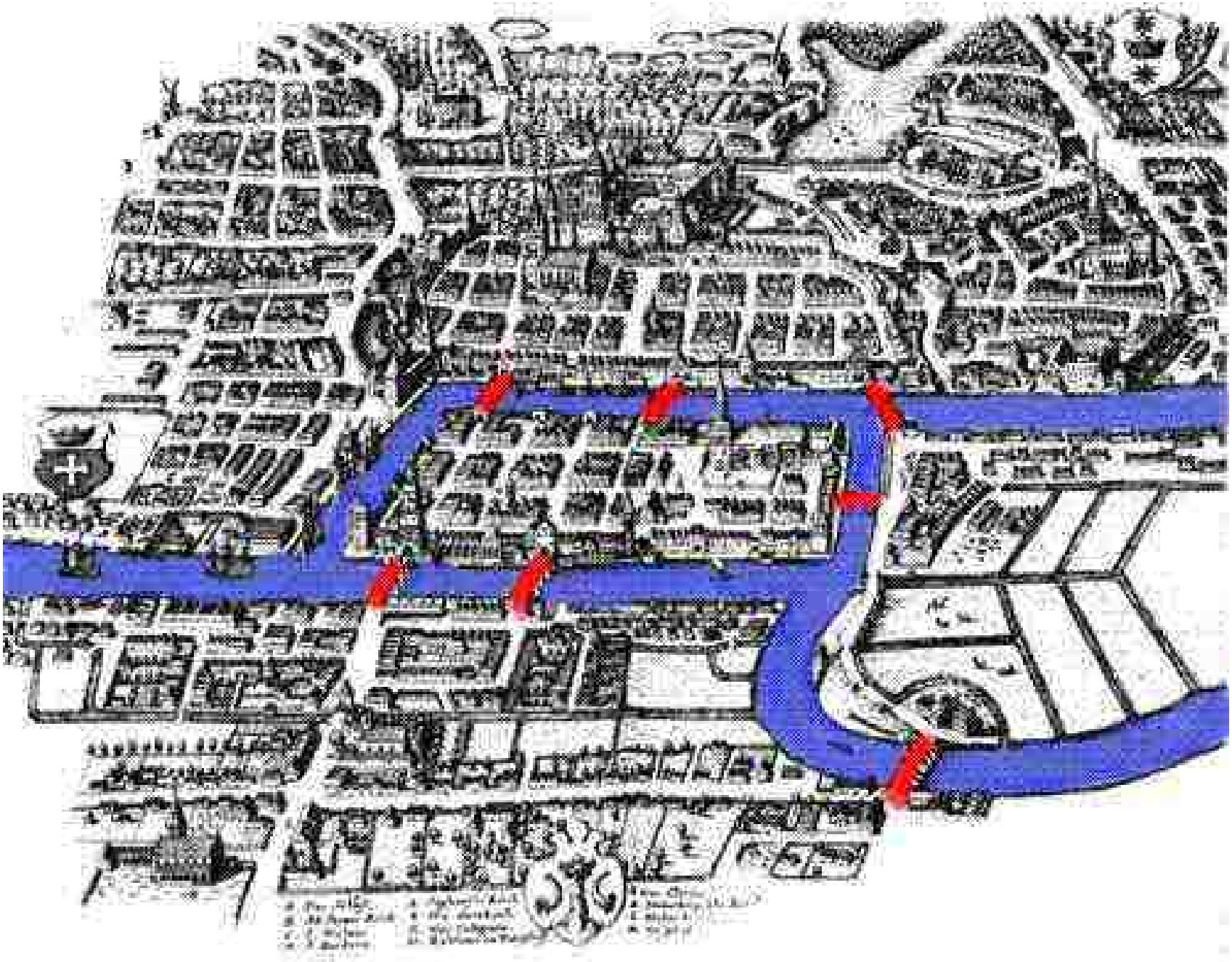
Recap: connectivity

- Connectivity and edge-connectivity of a graph
- Blocks
- Kruskal's algorithm

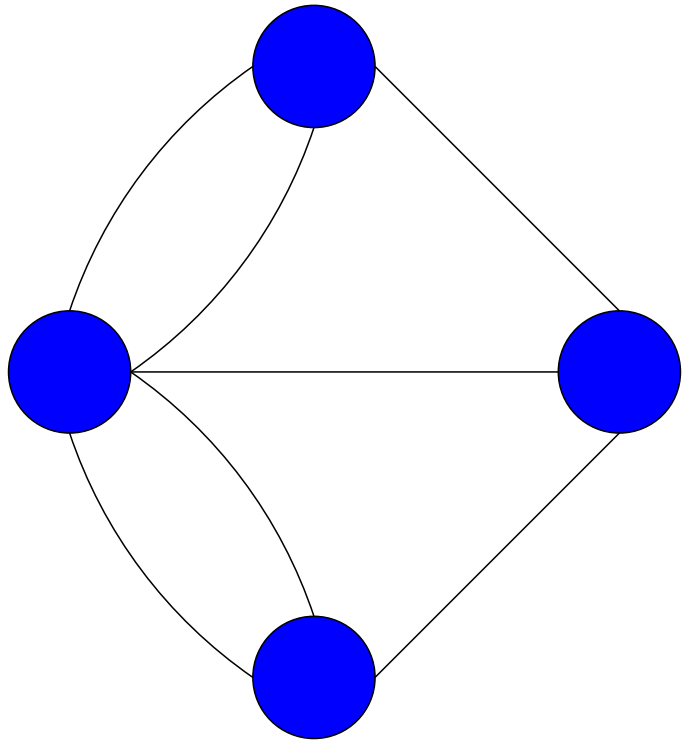
Königsberg, Prussia



The Seven Bridges of Königsberg



Graph of the bridge network



- Is it possible to find a tour of this graph such that each bridge is crossed exactly once?

Euler tours

- *Trails* are walks which never traverse the same edge twice.
- *Tours* or *cycles* are trails which end up at the place they started.
- An *Euler tour* traverses every edge of a graph exactly once.
- If a graph contains an Euler tour, it is *eulerian*.
- Graphs are eulerian iff all vertices are of even degree.
  Why?

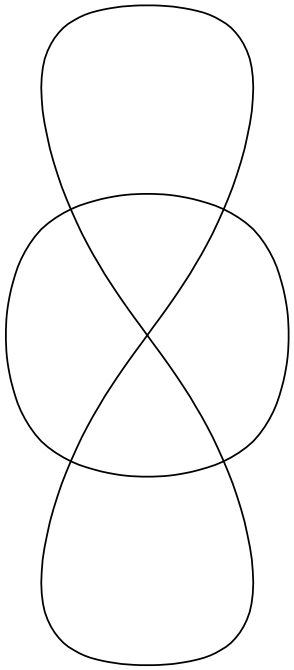
Proof: an eulerian graph must have all vertices of even degree

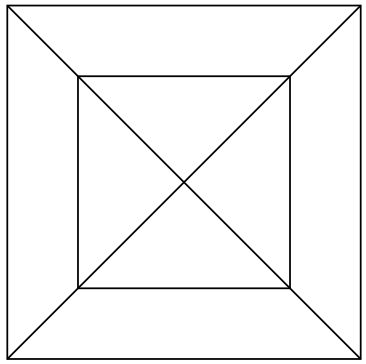
1. Let C be an Euler tour of graph G, which starts and ends at vertex u.
2. Each time a vertex is included in the tour C, two edges incident to that vertex are used up.
3. Every edge in G is included in the tour. So every vertex other than u must have even degree.
4. The tour starts and ends at u, so it must also have even degree.

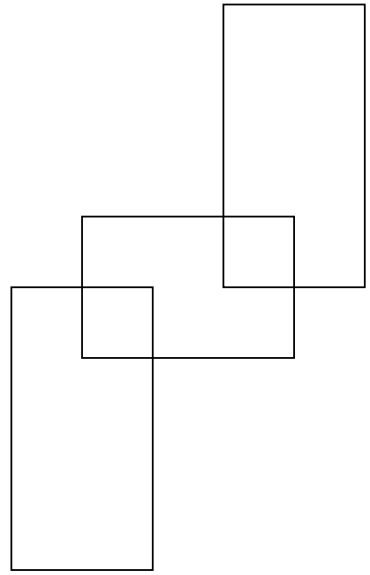
Proof: a graph with all vertices of even degree must be eulerian

1. **Assume the opposite**: G is a noneulerian graph with all vertices of even degree.
2. G must contain a closed trail. Let C be the largest possible closed trail in the graph.
3. Because of our assumption, C must have missed out some of the graph G, call this G'.
4. C is eulerian, so has no vertices of odd degree. G' therefore also has no vertices of odd degree.
5. G' must have some tour C', and there must be a common vertex v in both C and C'.
6. CC' therefore makes a larger trail than C, which violates our assumption in (2). **Contradiction**.

Which of these shapes are eulerian?

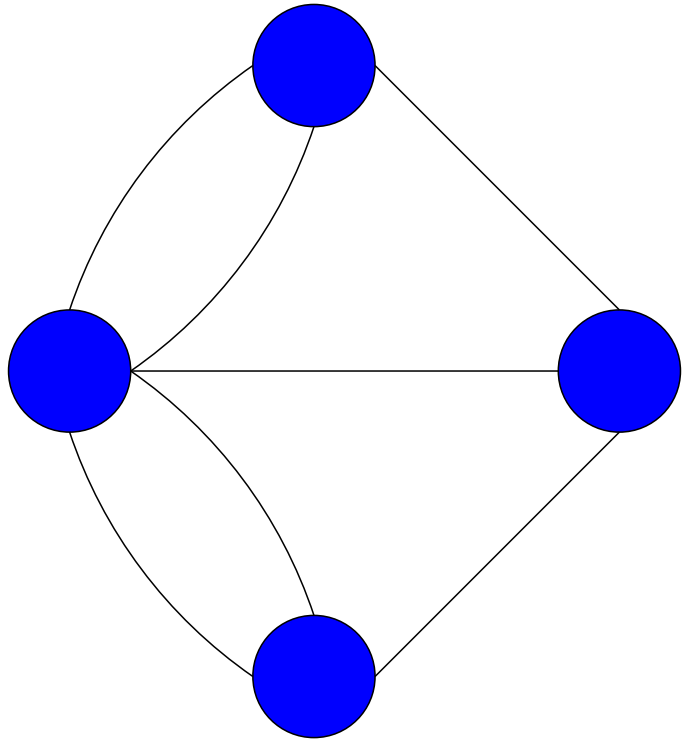






(i.e. which can you draw without removing your pen from the page, and without covering any lines twice?)

Modifying the graph for the ideal stroll



- Where could additional bridges be built so that the graph is eulerian?

Chinese postman problem

- A postman has to walk down every street in his neighbourhood.
- Furthermore, he wants to do it as efficiently as possible (trying not to repeat streets).
- In a weighted graph (i.e. where the streets are of different lengths), he especially wants to avoid repeating long streets.
- Can you think of any way in which Euler tours are related?

Fleury's algorithm for finding Euler tours

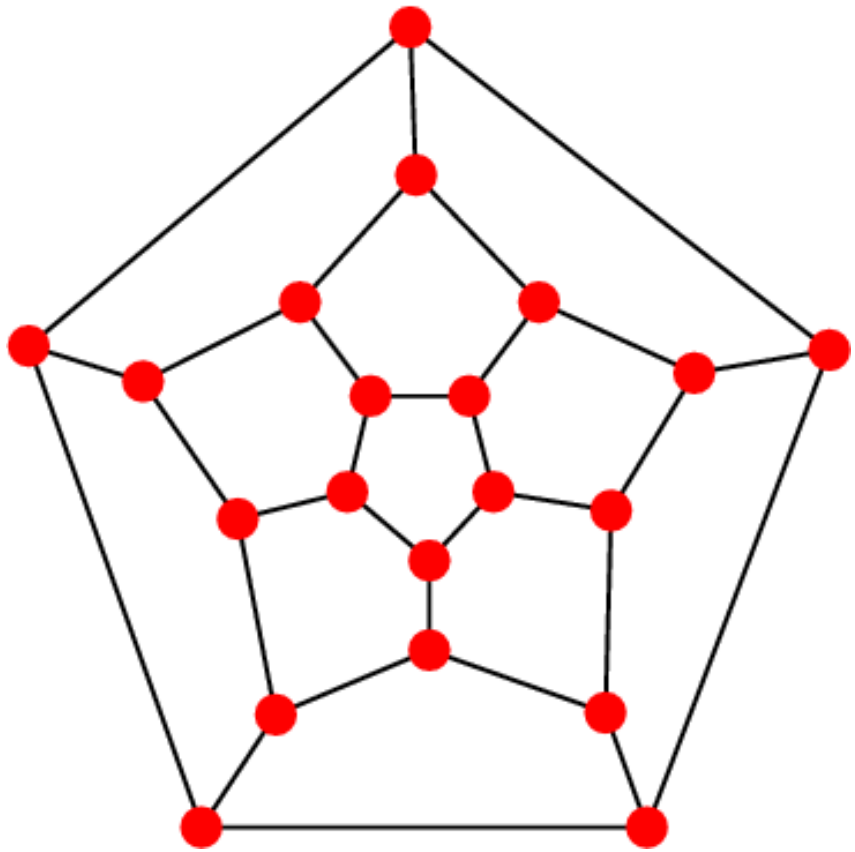
1. Choose any start vertex.
2. Choose the next edge in the trail such that:
   - the edge is incident to the current vertex.
   - it is not a cut edge of the remaining subgraph (unless there is no other choice).

Applications

- Cutting the grass on Makerere's campus.
- Police patrol routes.
- Monitoring networks of cables or pipes.
- Others?

Hamilton paths

- A Hamilton path visits every vertex (once).
- A Hamilton cycle does the same, and finishes at the place it started. The Dodecahedran has many Hamilton cycles:



Travelling salesman problem

The problem of finding the shortest Hamilton cycle is known as the travelling salesman problem (how to visit a number of different cities and get back home, using the least amount of fuel and time). It is an important example as many computing problems are equivalent to it.

How might you identify Hamilton cycles with the least combined weight?

Solution 1: Enumerate all solutions

If we could travel between cities in any order, how many ways are there of moving between 5, 10, 20, 100 cities?

TSP solution 2: greedy search

We could look for a good solution by carrying out the following steps:

Step 1: Find a random ordering of the cities.

Step 2: Change this ordering at random (e.g. by swapping the order of two cities).

Step 3: Evaluate the change in path length: if it is less than zero, accept the new change. Return to step 2.

TSP solution 3: stochastic annealing

The problem with greedy search is that it can become "stuck" (local optima).

Can use a stochastic search procedure to find even better solutions. We randomly accept some changes, even if they might make the overall solution slightly worse.

Step 1: Find a random ordering of the cities.

Step 2: Change this ordering at random (e.g. by swapping the order of two cities).

Step 3: Evaluate the change in path length: if it is less than `rand*T`, accept the new change. Return to step 2.

`T` is the "temperature" variable – it usually starts off at a high number (accept many alterations) and is reduced to zero (only accept alterations which improve the path length).