Heuristic Optimisation

Part 5: Greedy algorithms. Divide and conquer

Sándor Zoltán Németh

http://web.mat.bham.ac.uk/S.Z.Nemeth

s.nemeth@bham.ac.uk

University of Birmingham

Overview

- Algorithms working on partial solutions
- Greedy algorithm
- Examples of greedy algorithm
- Divide and conquer
- Examples of divide and conquer

Algorithms working on partial solutions

One solution is constructed at a time, either

- **Incomplete** solution to the **original** problem

  A subset of the original problem's search space with a particular property hopefully shared by the real solution

- **Complete** solution to a **reduced** problem
  1. We decompose the original problem into smaller & simpler problems
  2. We solve these problems
  3. We try to combine the partial solutions into a solution to the original problem

Greedy algorithm

A very simple algorithm that constructs the solution step by step

At each step the value for one decision variable is assigned by making the best available decision

A **heuristic** is needed for making the decision at each step: what is the best now?

The best 'profit' is chosen at every step - the algorithm is *greedy*!

We **cannot** expect the greedy algorithm to obtain the **overall** optimum

Greedy algorithm and SAT

We assign the truth value to the variables one at a time

Simple **heuristic**:

Choose the value 0 or 1 depending on which satisfies a greater number of presently unsatisfied clauses

Example:

$\overline{x}_1 \bigwedge (x_1 \bigvee x_2) \bigwedge (x_1 \bigvee x_3) \bigwedge (x_1 \bigvee x_4)$

Improved greedy algorithm for SAT

Our first approach is too greedy!

Let's consider the **order** of the assignments, too:

- Sort all variables according to frequency of appearance, from smallest to largest
- For each variable, taken in the above order, assign the value that satisfies the greatest number of unsatisfied clauses

Example improved greedy for SAT

Example:

$(x_1 \vee x_2) \wedge (x_1 \vee x_3) \wedge (\overline{x}_1 \vee x_4) \wedge (\overline{x}_2 \vee \overline{x}_4)$
$\wedge (x_2 \vee x_5) \wedge (x_2 \vee x_6) \wedge \mathit{Formula}(x_3, x_4, x_5, x_6)$

with at least four occurences of each of the variables x_3, x_4, x_5 and x_6 in $\mathit{Formula}(x_3, x_4, x_5, x_6)$.

Can you satisfy the third and fourth clause?

No greedy algorithm for SAT

Ideas for improvement:

- Forbid any assignment that makes any clause *FALSE*
- At any time consider the frequency of variables for the *remaining* unsatisfied clauses
- We could take into account the *length* of the clause: the value of a variable in a short clause has more impact than the value of a variable in a long clause

None of these can guarantee a solution to SAT!

Greedy algorithm and TSP

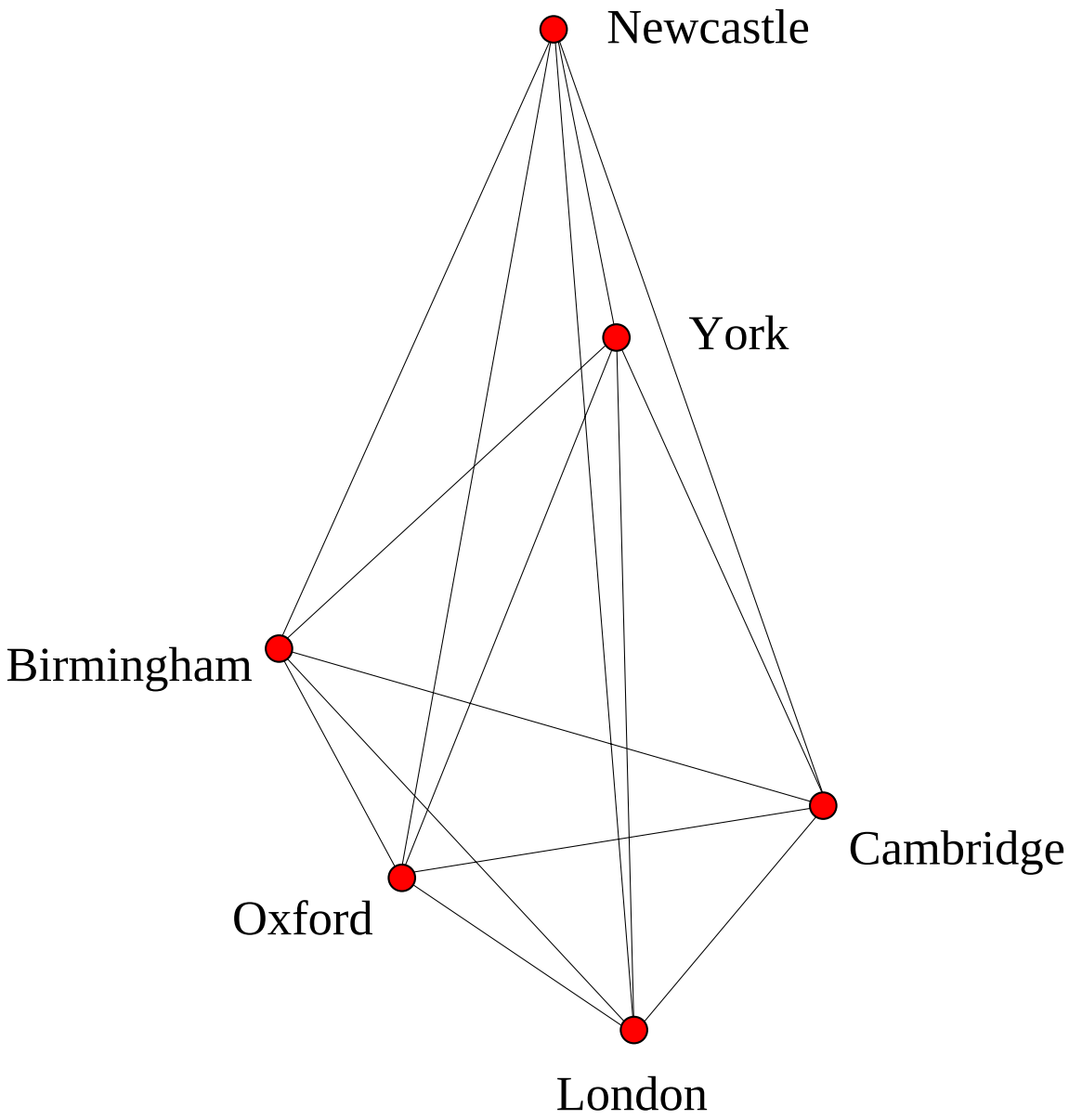
Nearest neighbour heuristic:

1. Select an arbitrary starting city.

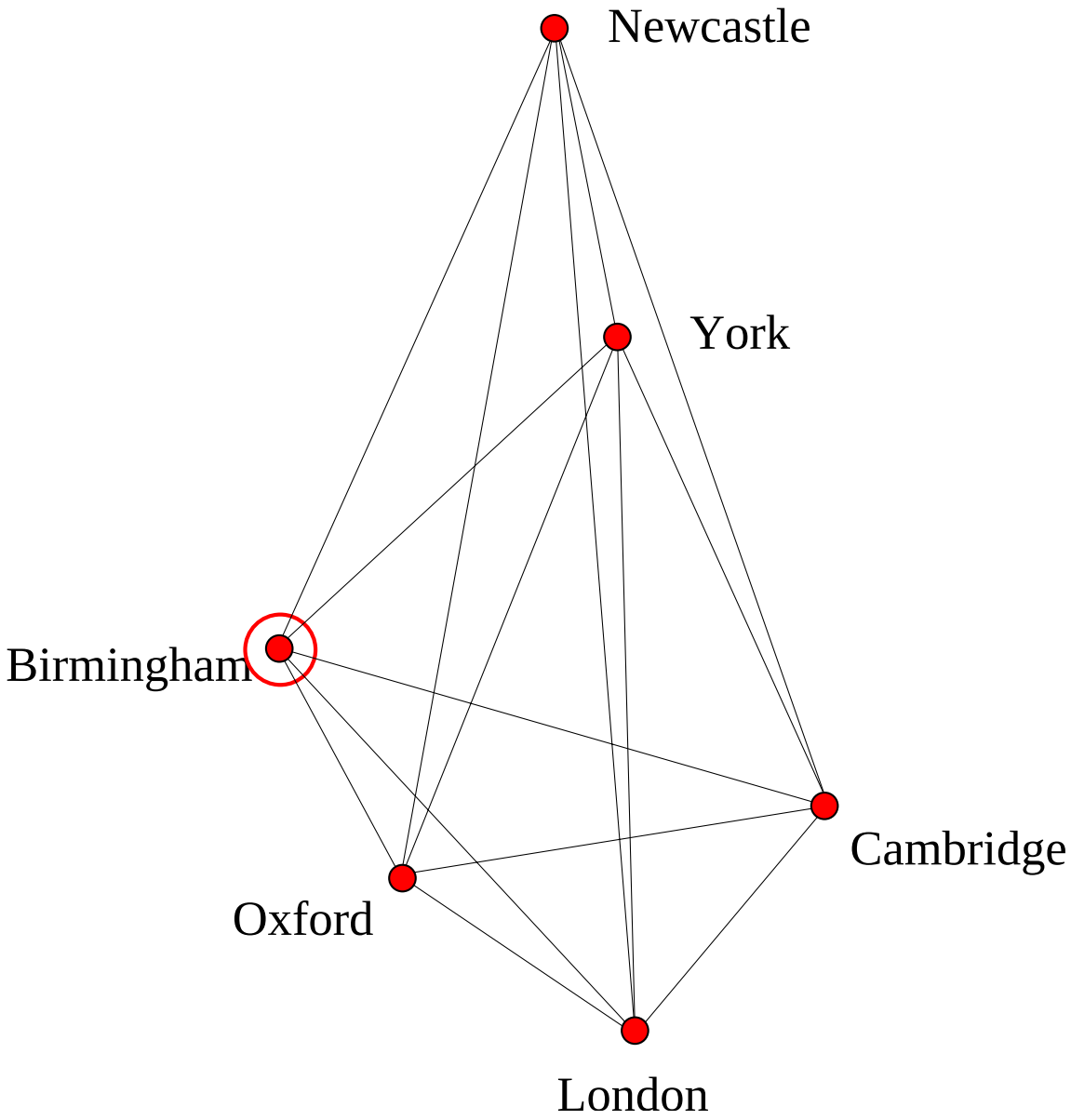
2. As next city, select the city from the cities not yet visited that is closest to the current city. Go to this city.

3. Repeat step 2 until all cities have been visited.

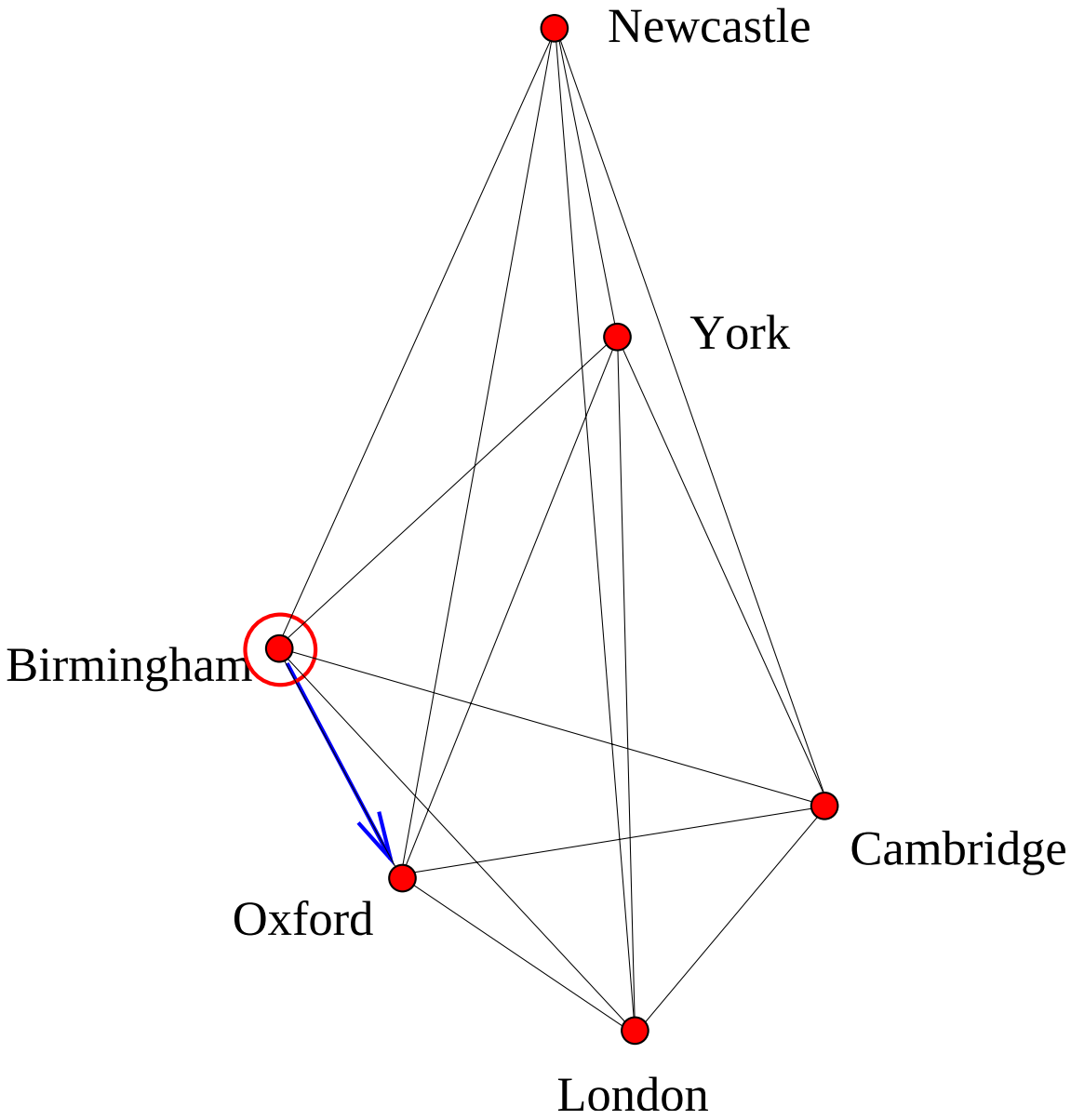
Example nearest neighbour



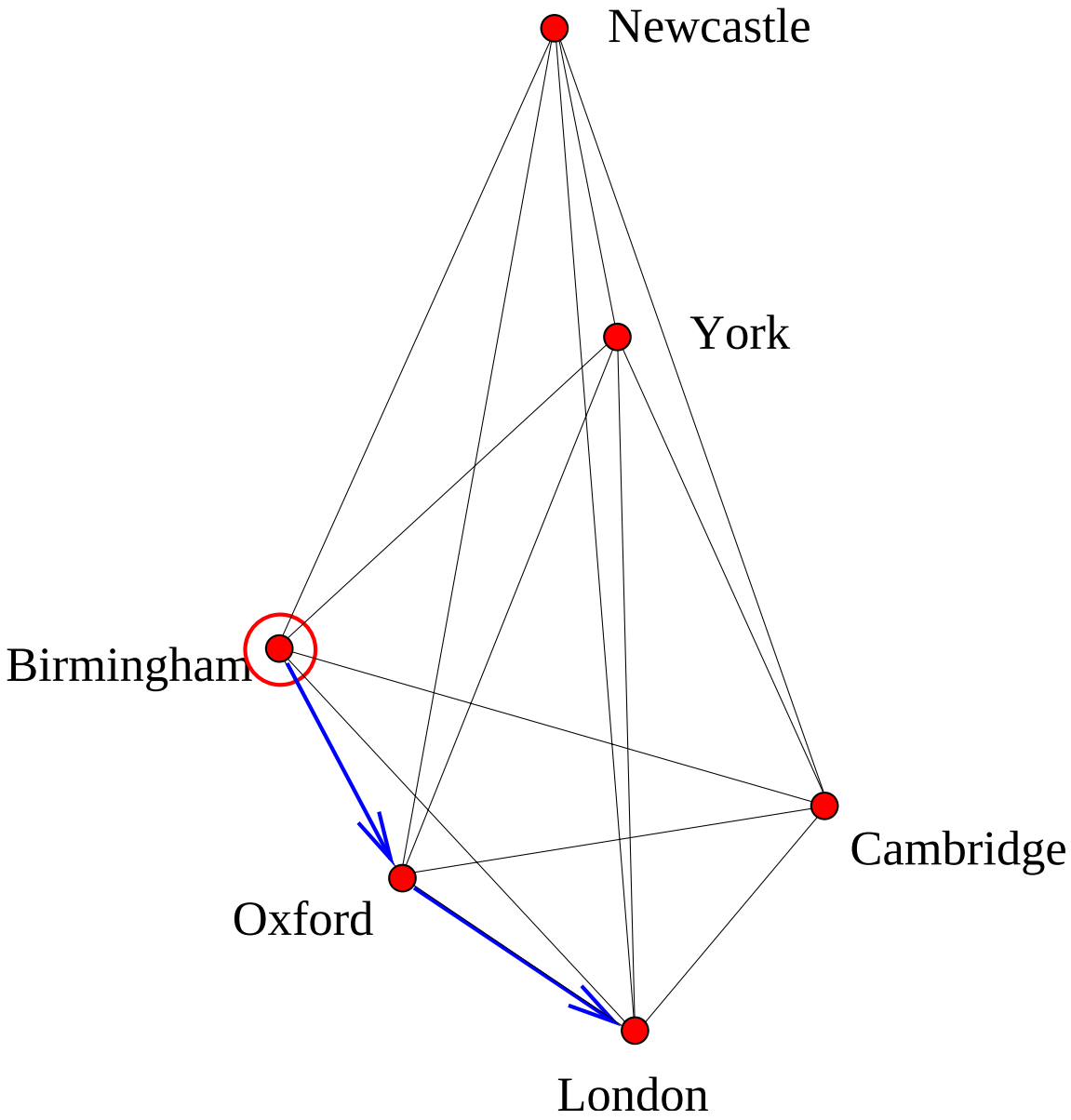
Example nearest neighbour



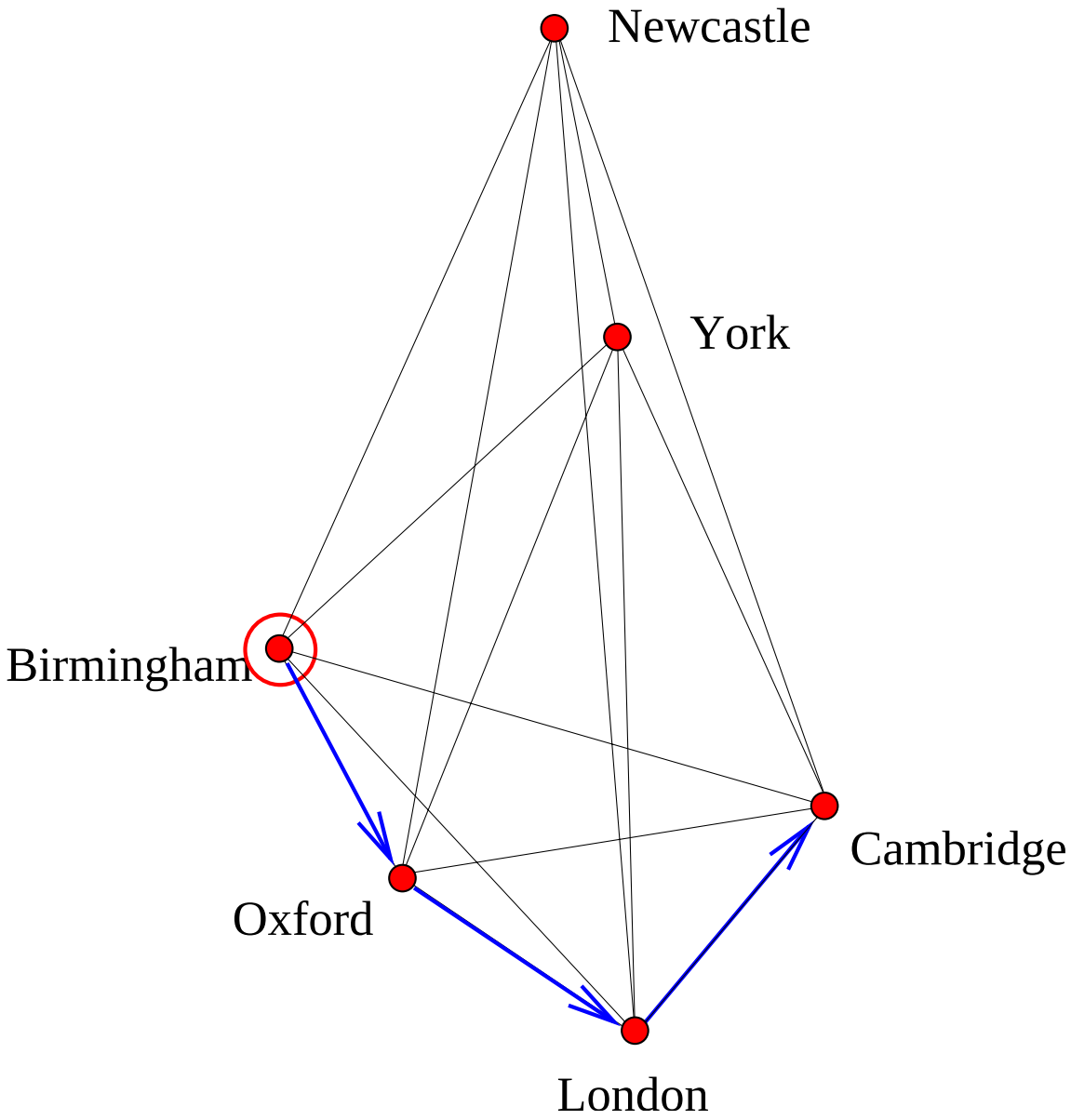
Example nearest neighbour



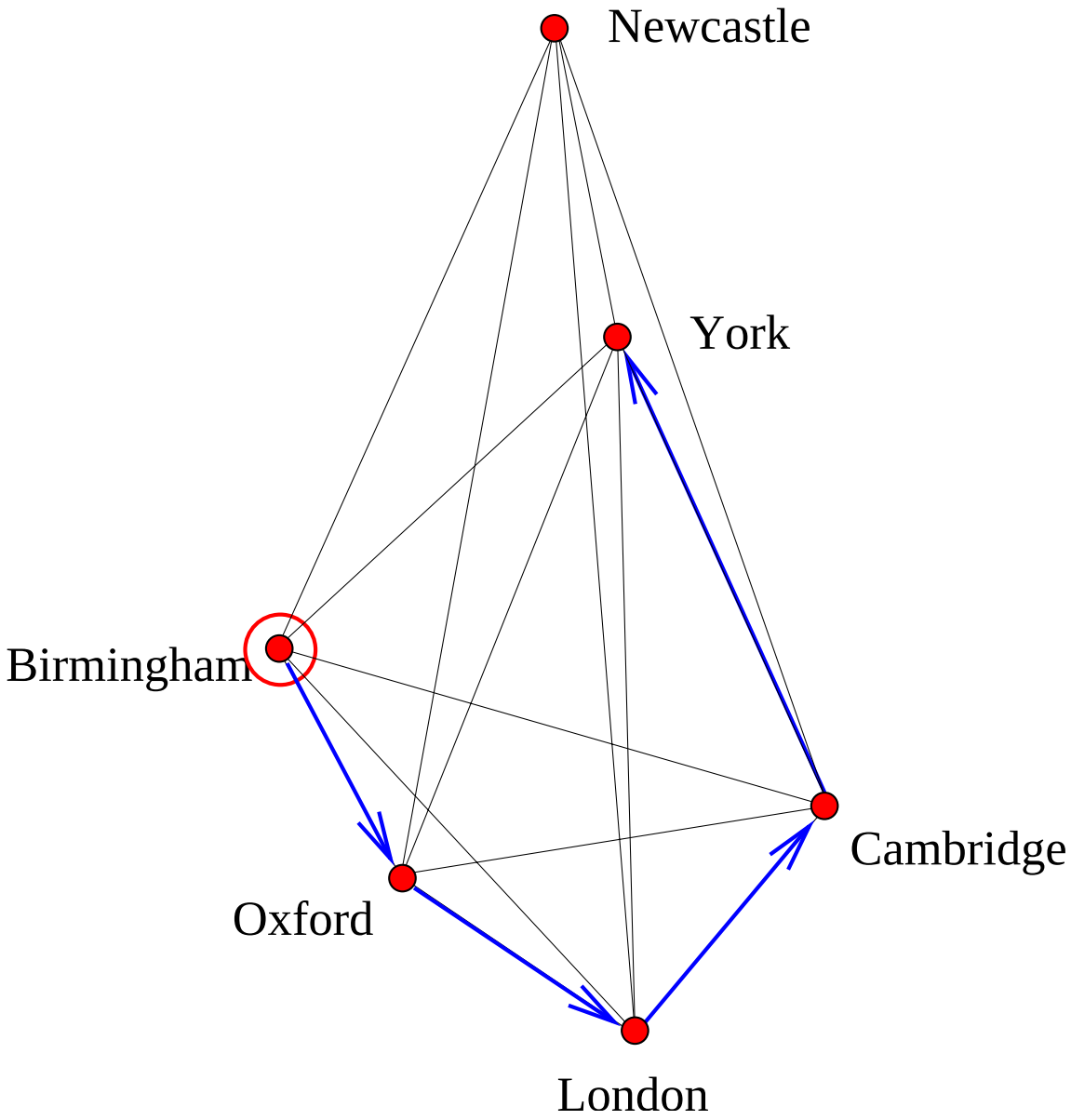
Example nearest neighbour



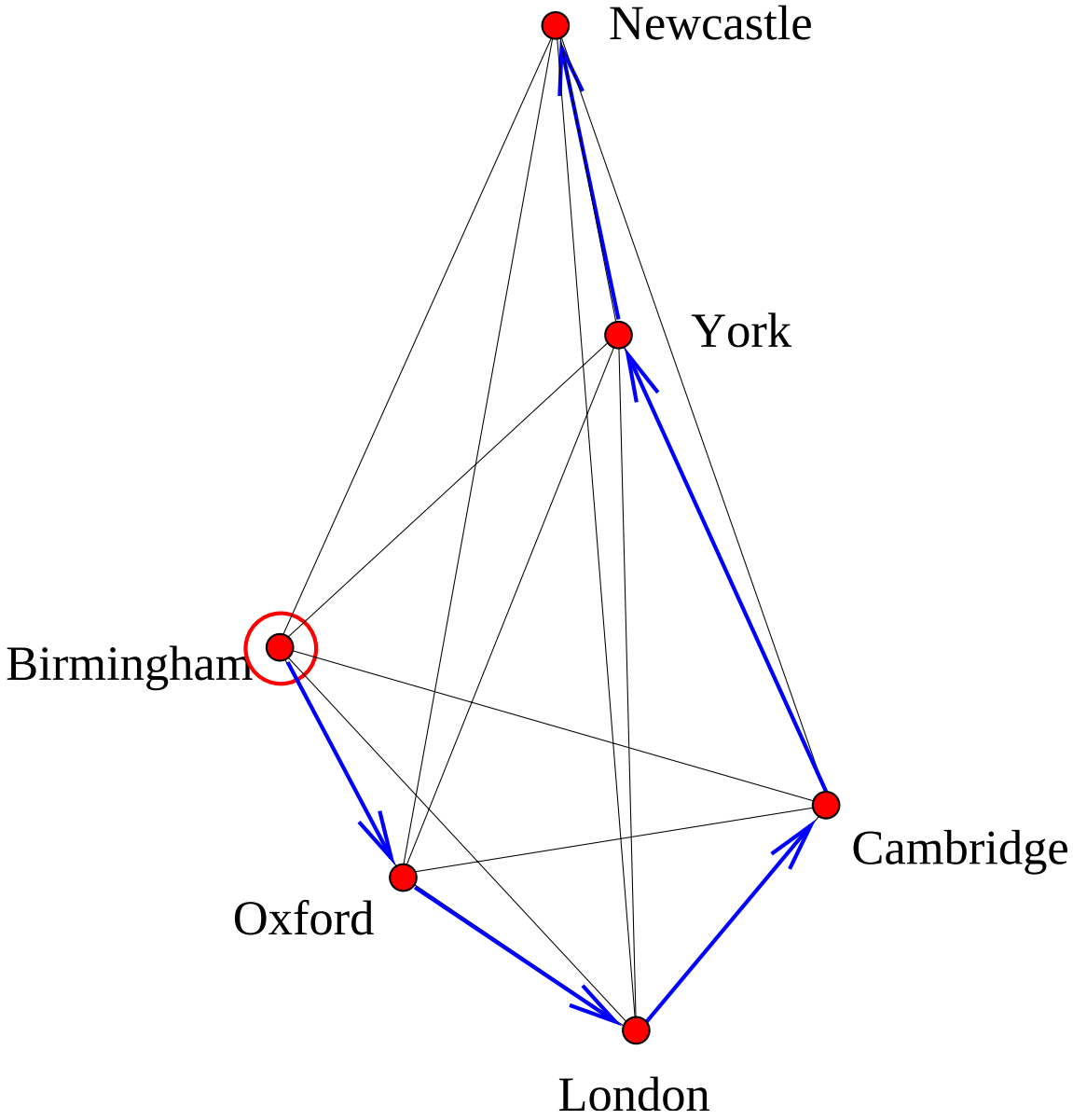
Example nearest neighbour



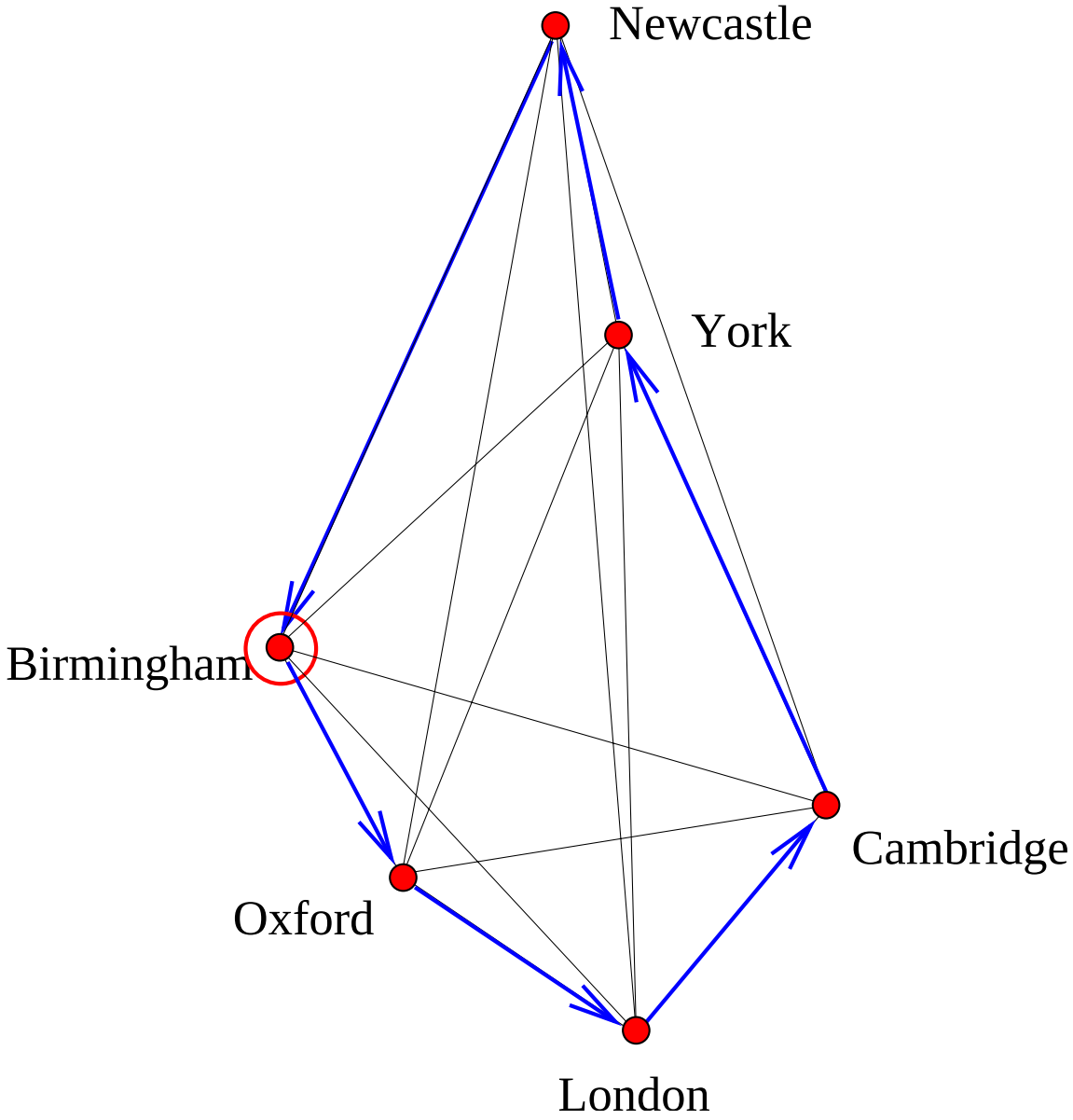
Example nearest neighbour



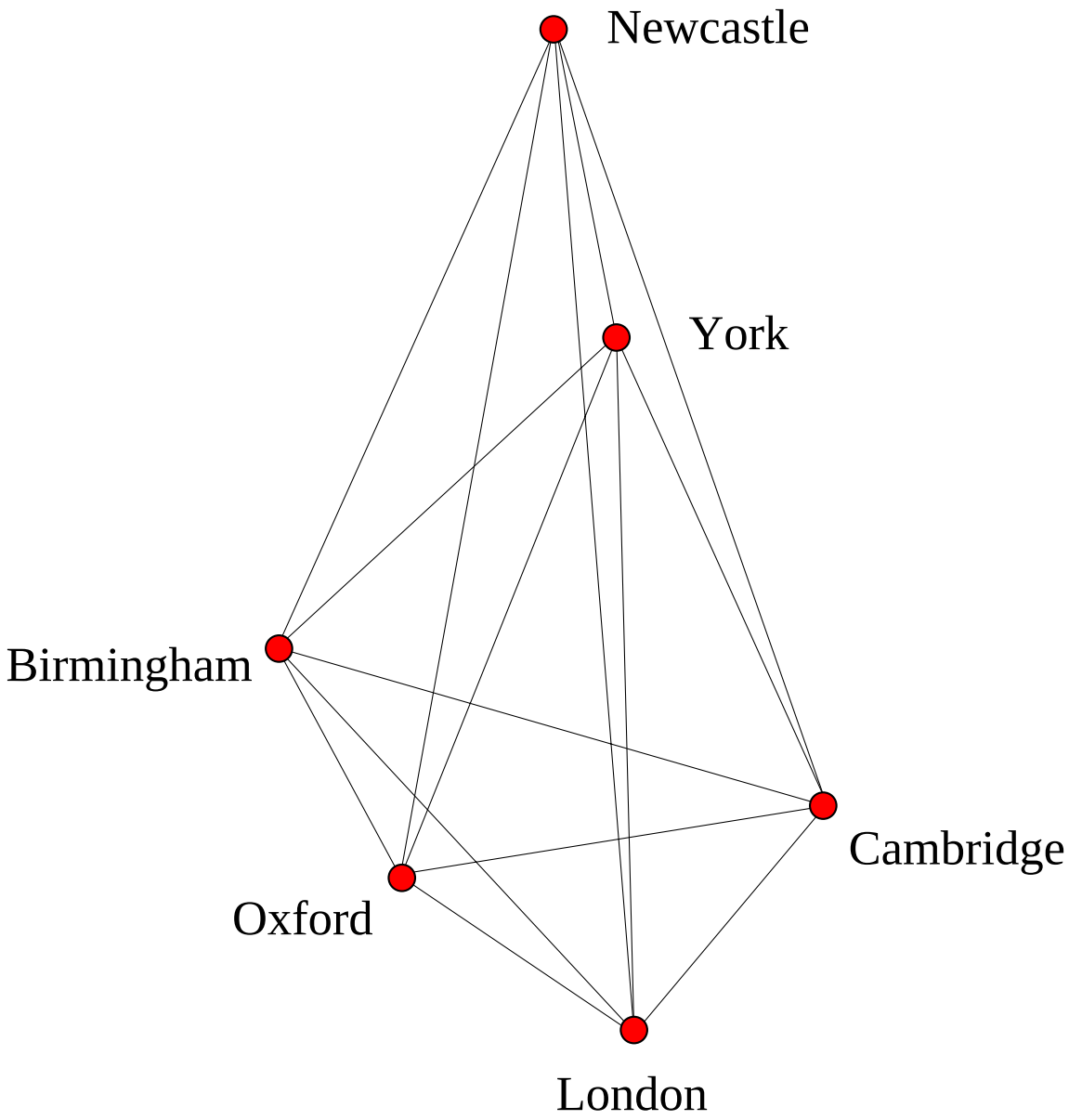
Example nearest neighbour



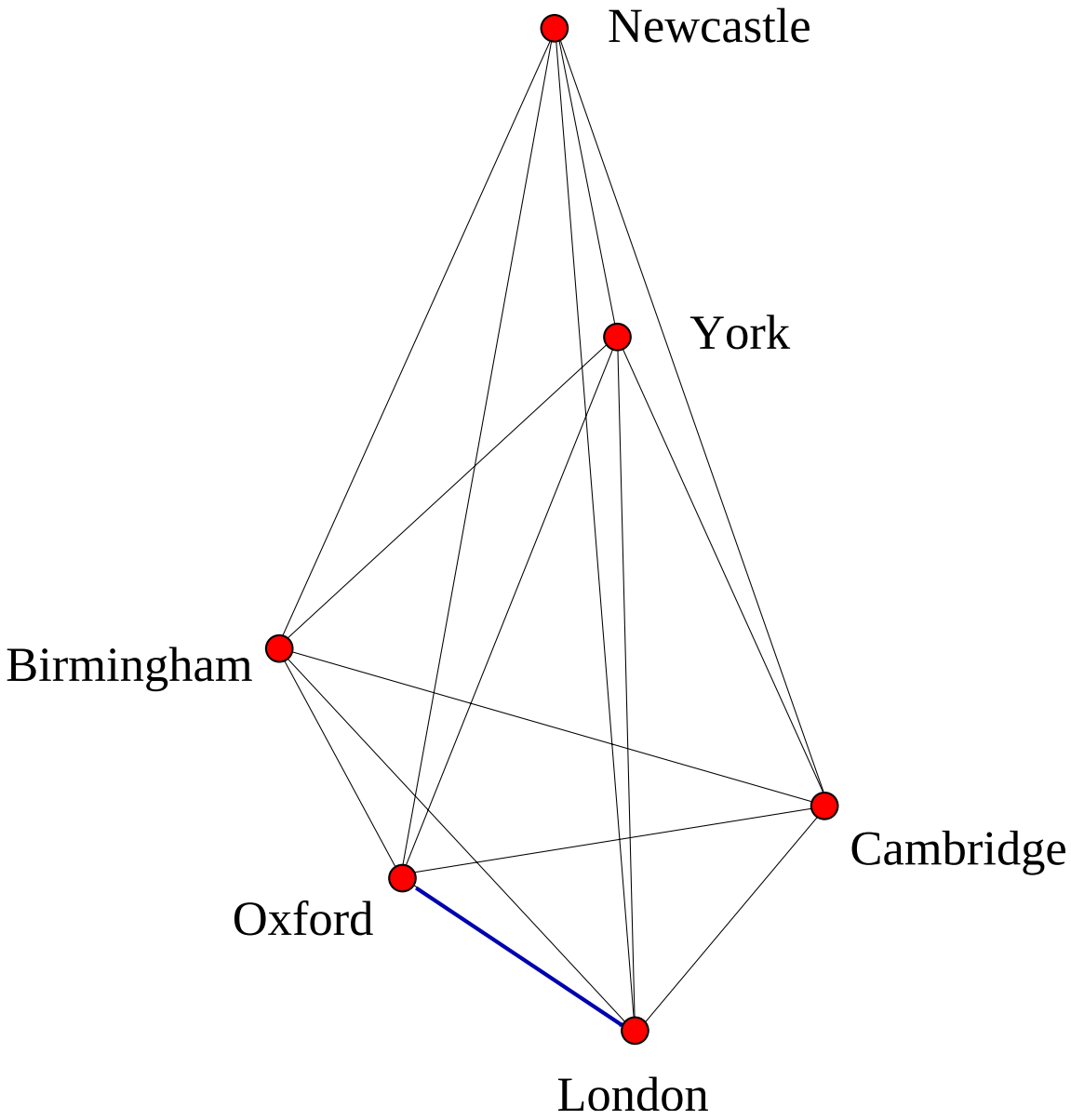
Example nearest neighbour



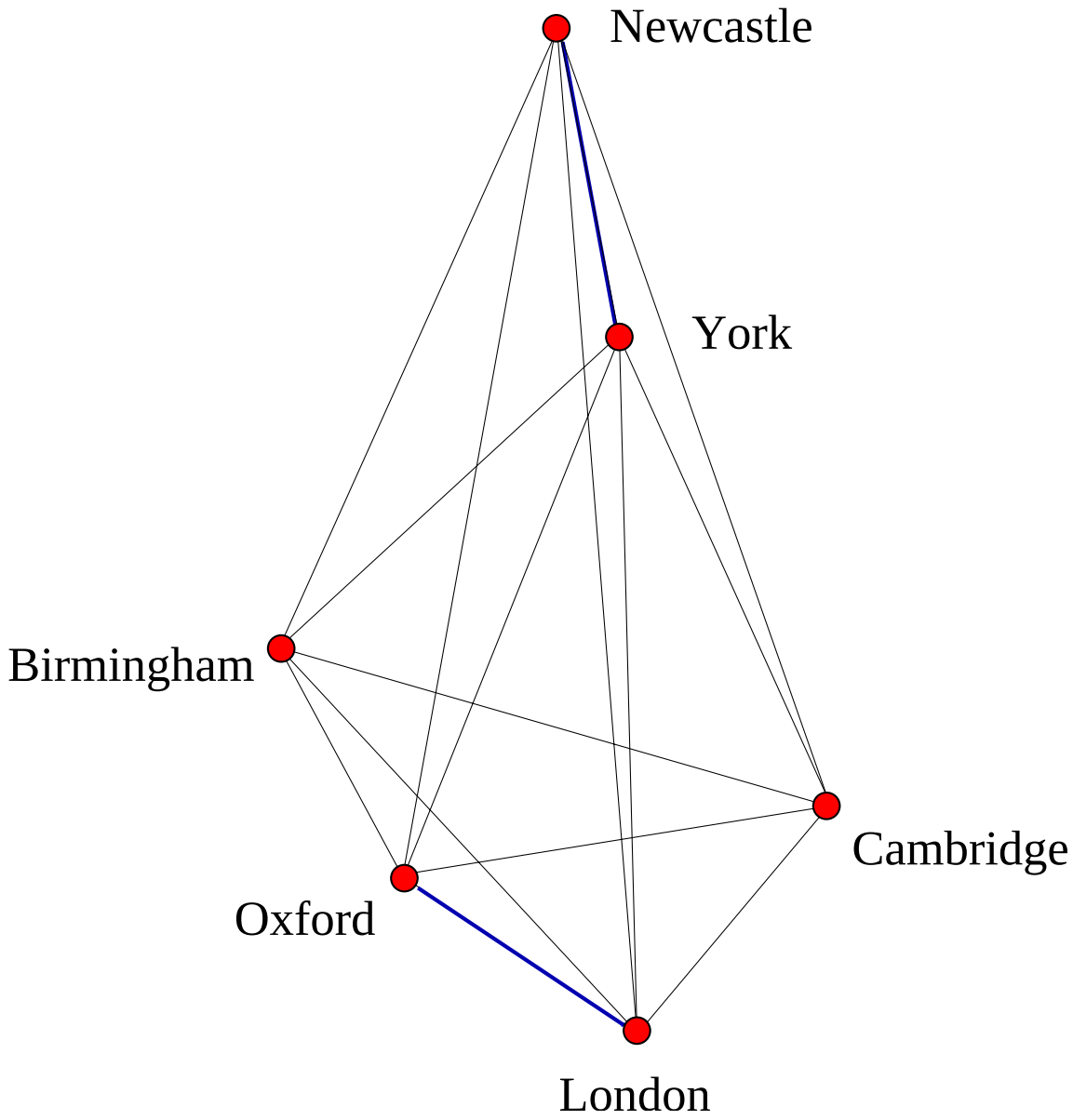
Another greedy heuristic: example



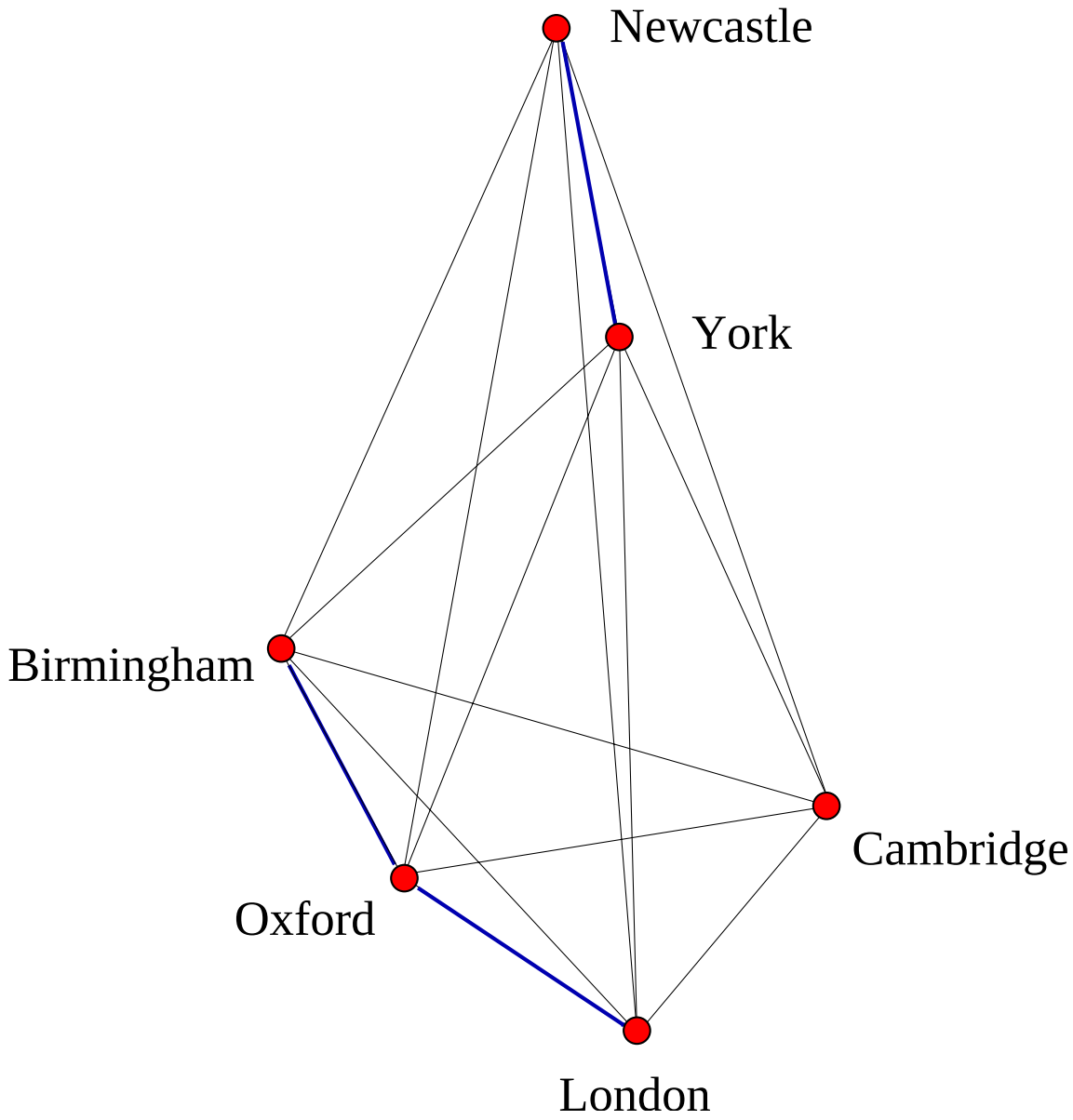
Another greedy heuristic: example



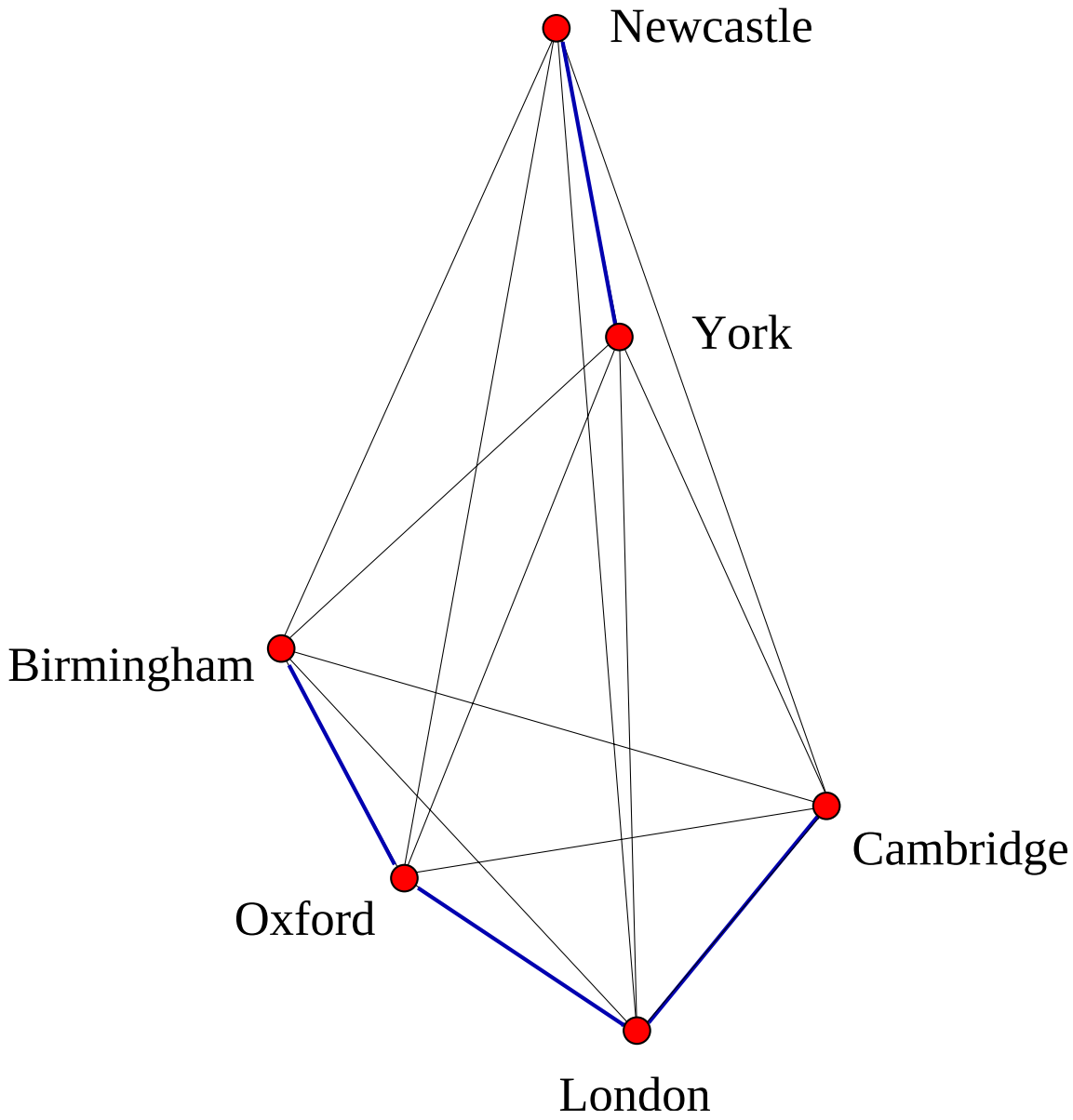
Another greedy heuristic: example



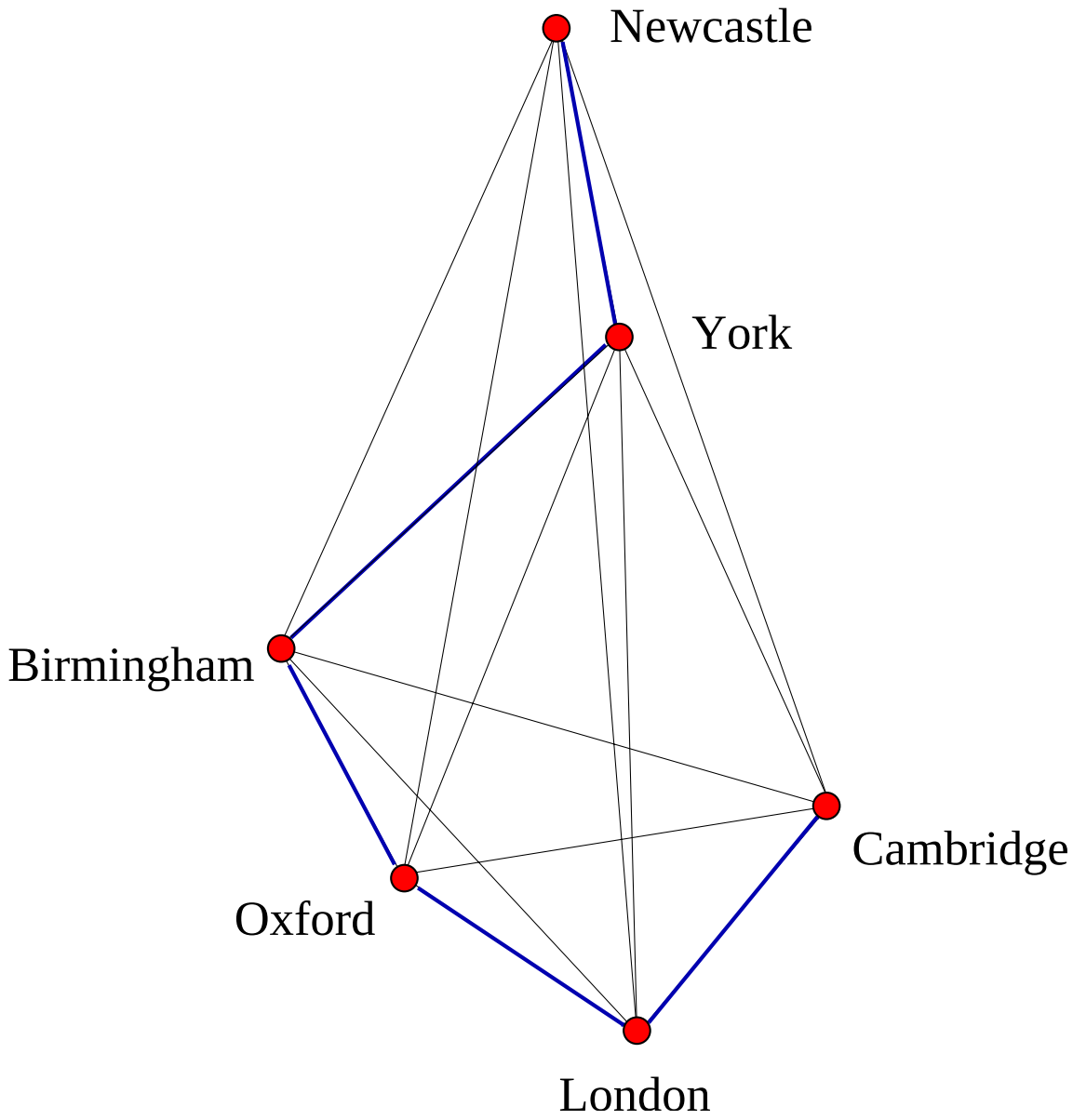
Another greedy heuristic: example



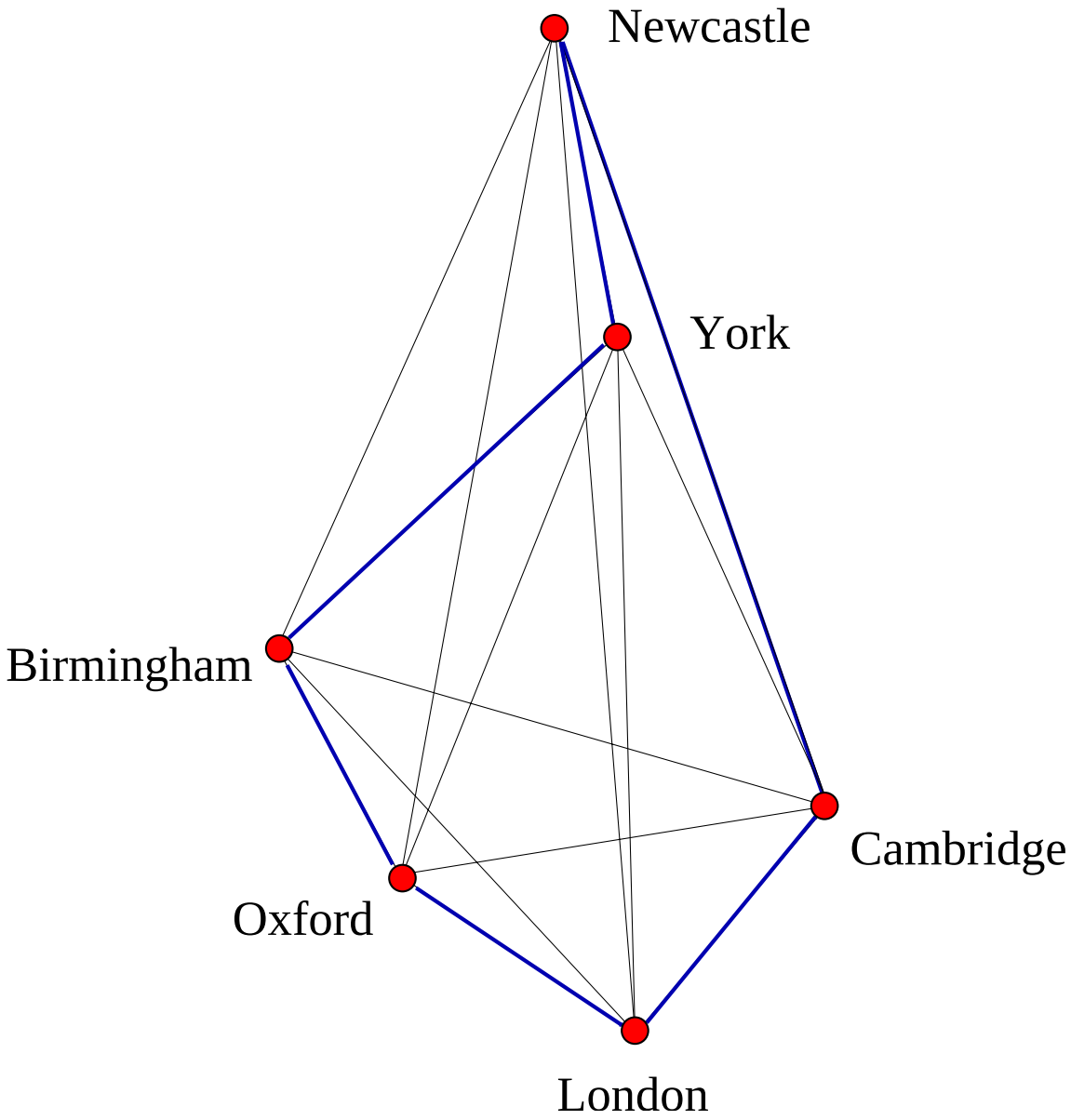
Another greedy heuristic: example



Another greedy heuristic: example



Another greedy heuristic: example



Another greedy heuristic for TSP

1. From all possible edges select the shortest one and add it to the tour.

2. Reduce the set of possible edges:
   If the newly added edge has an end that already appears in the tour, remove all edges that contain that end.

   What if both ends of the newly added edge appear in the tour?

3. Repeat steps 1-2 until the tour is complete.

Divide and conquer

Divide&conquer(P)

1. Split problem P into subproblems $P_1, P_2, \ldots, P_k$.

2. For i taking all the values from 1 to k

   get the solution S_i to problem P_i

3. Combine $S_1, S_2, \ldots, S_k$ into the solution S for problem P.

4. Return the solution S.

The recursive step 2

Get the solution S_i to problem P_i:

if $size(P_i) < \rho$ then
    solve P_i and get its solution S_i

else
    apply Divide&conquer(P_i) and
    assign the result to S_i.

The complete algorithm

Divide&conquer(P)

1. Split problem P into subproblems $P_1, P_2, \ldots, P_k$.

2. For i taking all the values from 1 to k do
   if $size(P_i) < \rho$ then
   solve P_i and get its solution S_i
   else
   apply Divide&conquer(P_i) and
   assign the result to S_i.

3. Combine $S_1, S_2, \ldots, S_k$ into the solution S for problem P.

4. Return the solution S.

Is divide and conquer effective?

Time and effort is needed for

- decomposing the problem into subproblems
- solving the subproblems
- assembling the solutions to the subproblems into the solution to the initial problem

Is divide and conquer effective?

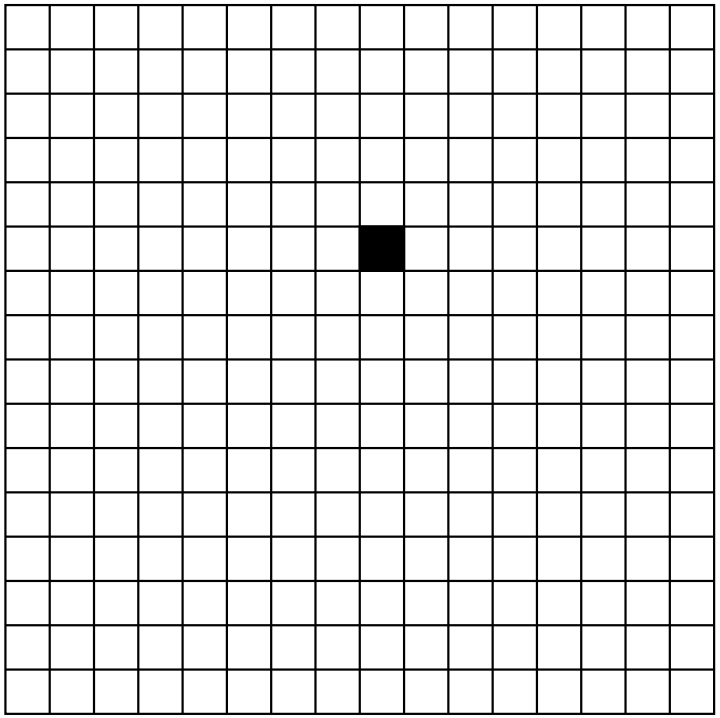
Time and effort is needed for

- decomposing the problem into subproblems
- solving the subproblems
- assembling the solutions to the subproblems into the solution to the initial problem

The algorithm is cost effective only if its cost is less than the cost of solving the original problem

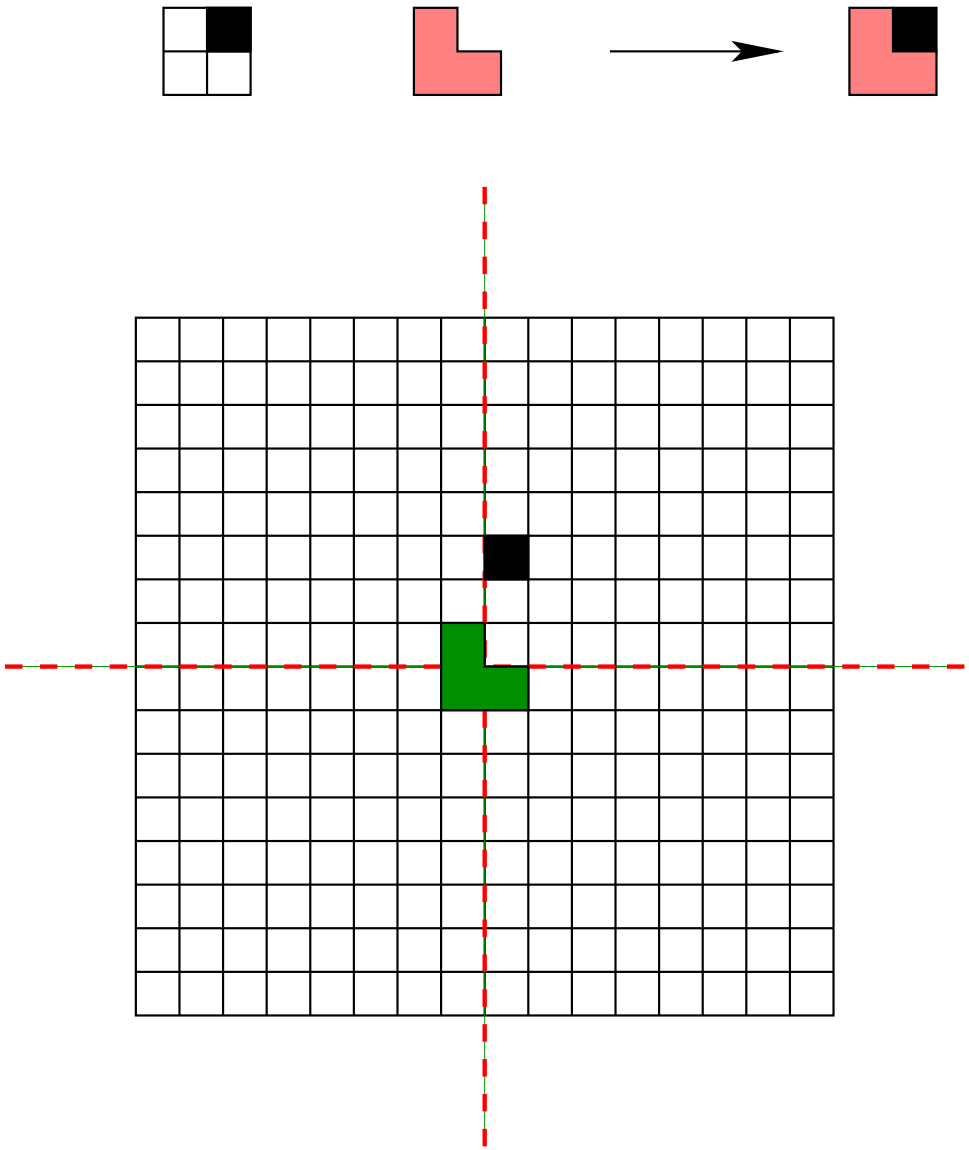
Example of divide and conquer

A chessboard of size 16×16 with a hole is given (one arbitrary square is removed from the board)
A sufficient number of L-shaped tiles is given and the task is to cover the board with these tiles.

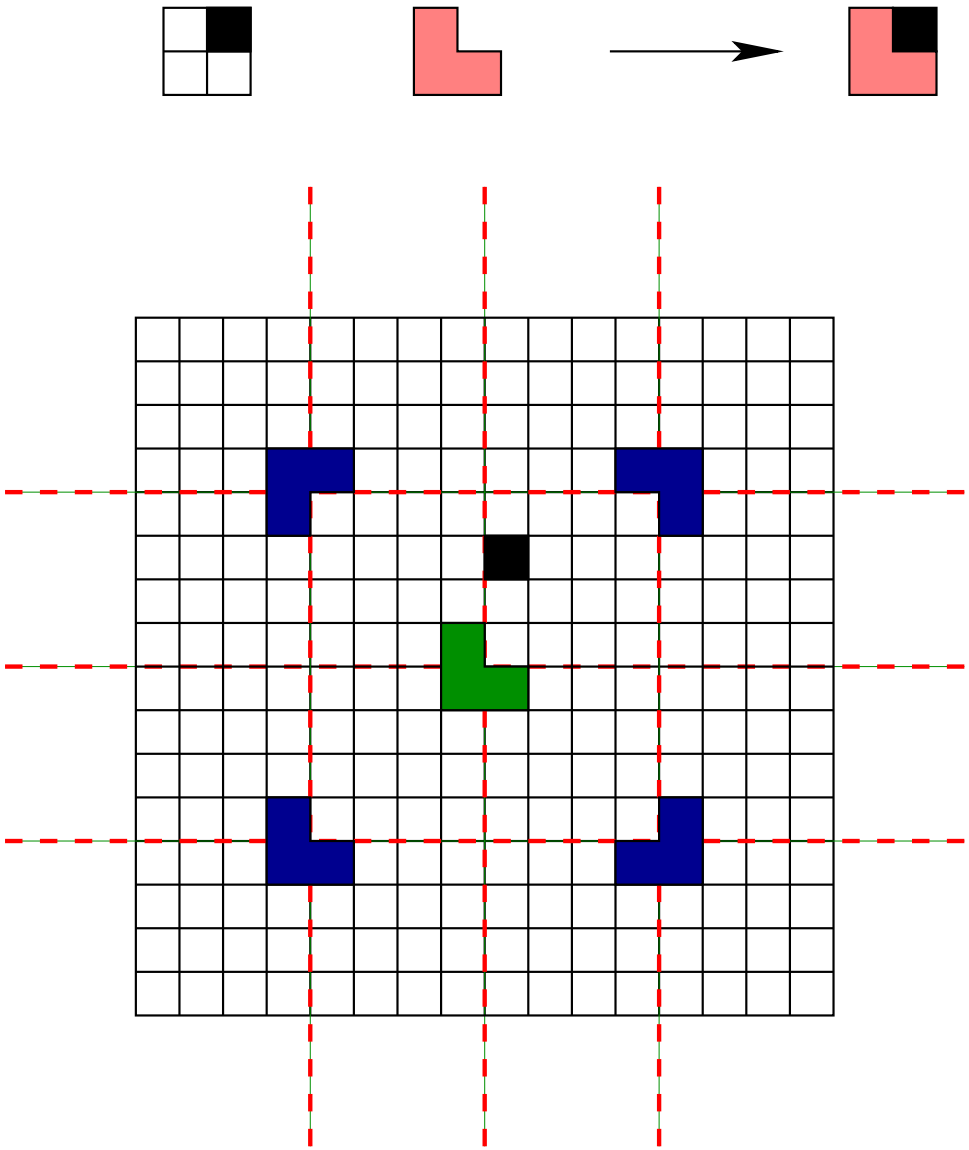


Divide and conquer applied

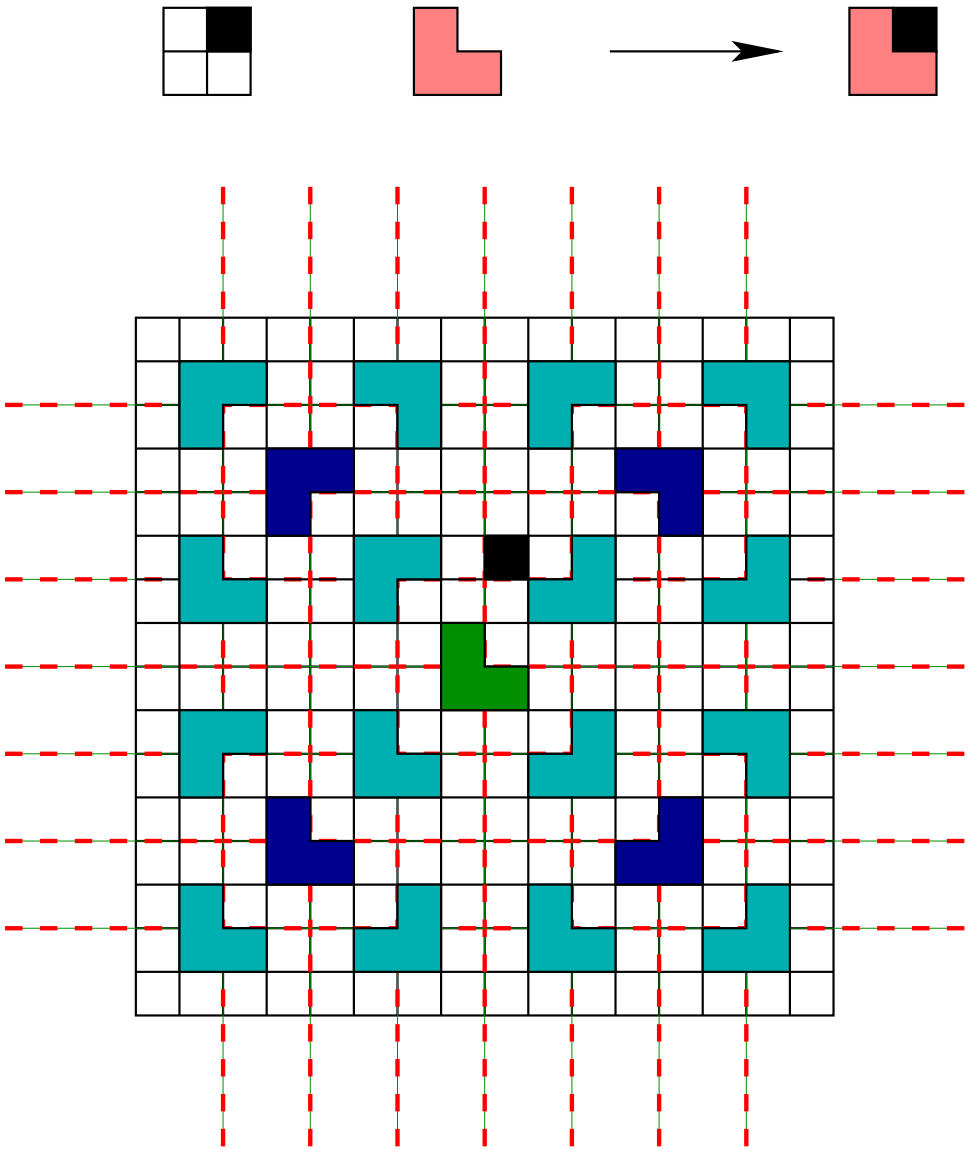
Divide and conquer applied



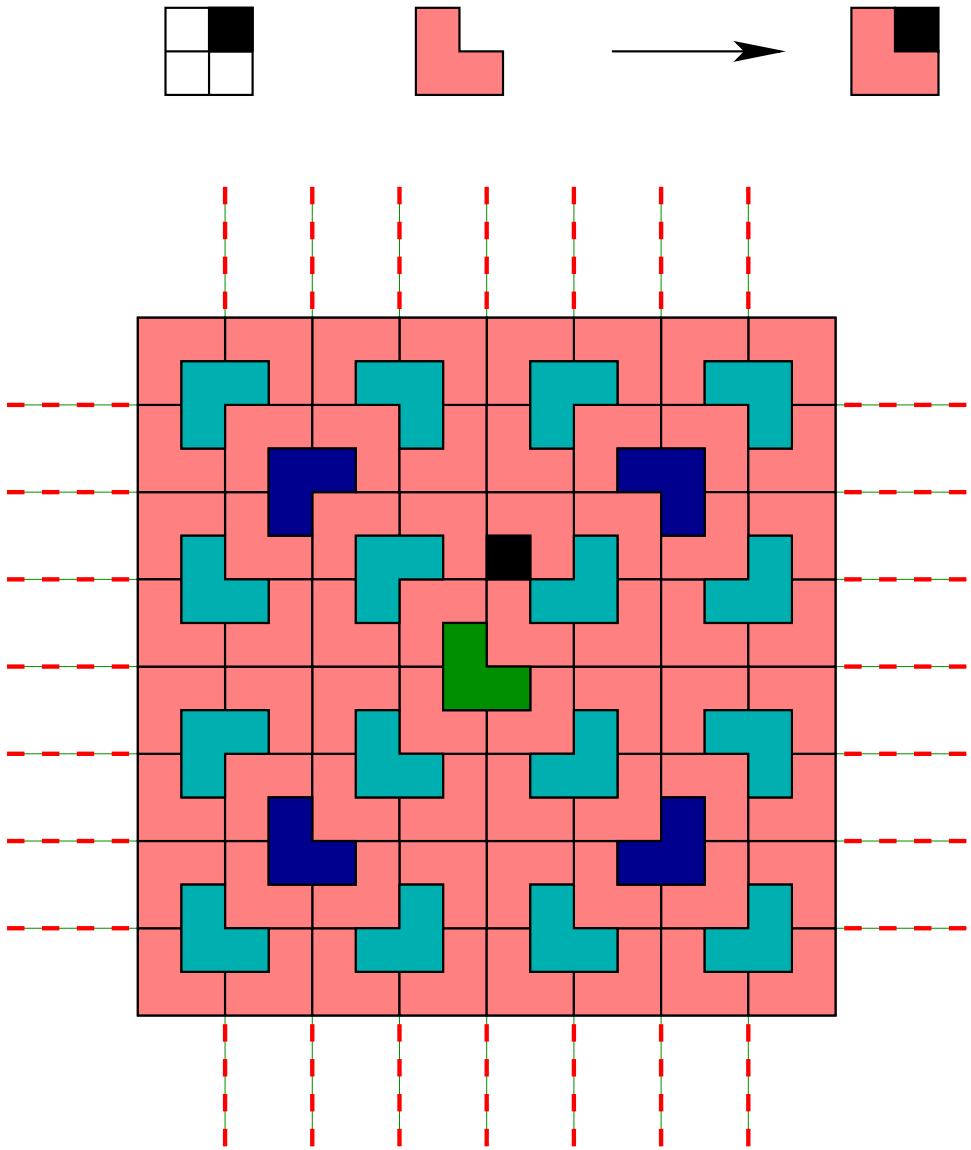
Divide and conquer applied



Divide and conquer applied



Divide and conquer applied



Divide and conquer for SAT?

It seems easy to divide the SAT problem:

$$F(x_1, x_2, x_3, x_4) = (x_1 \bigvee x_2) \bigwedge (x_1 \bigvee x_3) \bigwedge (\overline{x}_1 \bigvee x_4) \bigwedge (\overline{x}_2 \bigvee \overline{x}_4)$$

$$\begin{aligned}
F_1(x_1, x_2, x_3, x_4) &= x_1 \bigvee x_2 \\
F_2(x_1, x_2, x_3, x_4) &= x_1 \bigvee x_3 \\
F_3(x_1, x_2, x_3, x_4) &= \overline{x}_1 \bigvee x_4 \\
F_4(x_1, x_2, x_3, x_4) &= \overline{x}_2 \bigvee \overline{x}_4 \\
F(x) &= F_1(x) \bigwedge F_2(x) \bigwedge F_3(x) \bigwedge F_4(x)
\end{aligned}$$

Divide and conquer for SAT?

The subproblems are not **independent**:

F_1, F_2, F_3 all depend on x_1
F_1, F_4 depend on x_2
F_3, F_4 depend on x_4

The solutions to these subproblems cannot be combined into a solution to the original problem

Another possibility for SAT?

Try to group the disjunctions according to the variables they depend on.

Disjunctions that have at least one variable in common, should be in one group.

For the given example, all disjunctions would belong to the same group

$\longrightarrow$ NO such division is possible!

Summary

- The greedy algorithm constructs a solution to the problem step by step
- The greedy algorithm is recommended in cases when we know that there are many local optima and a global optimum is not necessary
- Divide and conquer solves subproblems
- It is not always clear how to divide a problem into subproblems or it might not be possible

Recommended reading

Z. Michalewicz & D.B. Fogel
How to Solve It: Modern Heuristics

Chapter 4: Traditional methods -Part 2, Sections 4.1-4.2