Management in Intelligent System report

Resilience of Public Transportation Networks

PERRIOT Mathieu

mathieu.perriot@edu.univ-eiffel.fr

Master 2 - SIA : Intelligent System and Application

January 2025

Project Supervisor

Dr. BHOURI Neila - OLIVIER François

Contents

1 Abstract

This study investigates the resilience of transportation networks using percolation theory. By progressively removing edges and nodes based on different quality metrics (travel time, inverse travel time, and betweenness centrality), the network's robustness is evaluated. The paper also presents an algorithm to assess the percolation threshold and the impact of node and edge removal on network connectivity. The results emphasize the importance of redundancy and balanced infrastructure in ensuring the long-term reliability of public transportation systems.

2 Introduction

In the context of transportation networks, resilience refers to the system's ability to maintain connectivity and functionality despite the failure of certain components, such as nodes or edges. Understanding how networks degrade under various conditions is crucial for infrastructure planning and risk management. This paper leverages percolation theory to assess the resilience of a transportation network by simulating the removal of edges and nodes based on different quality measures. The goal is to evaluate how these removals impact the overall network connectivity and identify the critical elements that could cause the network to fragment. By using percolation analysis, this study provides valuable insights into how to enhance the robustness of public transportation networks and minimize the risk of network collapse under stress.

3 Results

3.1 PART 1 : Understanding the percolation Theory

3.1.1 Question 1 : Definition

Define the percolation phenomenon based on the attached paper, lecture content, and independent research.

The percolation phenomenon refers to how a network's connectivity decreases as edges are removed based on their quality. It helps assess the network's resilience by analyzing how the demand (connectivity) is affected as edges are progressively removed.

3.1.2 Question 2 : Relevance

Explain the relevance of percolation theory to the study of network resilience.

Percolation theory is highly relevant to network resilience because it helps analyze how the network's connectivity is affected as components are progressively removed. It provides insights into the critical points at which the network becomes fragmented or disconnected.

In this context, percolation theory helps assess a network's robustness by evaluating how it can withstand disruptions, and its vulnerability by identifying thresholds where small changes lead to significant connectivity loss.

3.2 PART 2 : Computing multiple quality matrices

3.2.1 Question 1 : Adjacency matrix

Modify ComputeAdj.m so that it could use the following StopList as an input to build the adjacenty matrix of the network (then transform it as an unoriented network).

```
>> Stops={'1','2','3','4','5','6','7','8','9','10','11','12','13','14','15'};
>> [DAdj,DwAdj] = ComputeAdj2(ROUTES,TravelLinkTime,Stops);
```

Here's the following modified *ComputeAdj.m* function that I called *ComputeAdj*2*.m*

As you will see, we're not making an unoriented adjcacency matrix in this function, since it is done later on in the *Mandl*18*routes.m* script.

```
function [Adj,wAdj]=ComputeAdj2(ROUTES,TravelLinkTime, StopList)

  Stoplist=StopList;
  sizeAdj=ComputeSizeAdj(ROUTES);
  Adj = zeros(sizeAdj,sizeAdj); wAdj= zeros(sizeAdj);

  for itinerary = 1:size(ROUTES,2)
    [LastStopIdx,Stoplist] = SearchIndex(Stoplist,ROUTES{itinerary}{1});
    for Stop = 2:size(ROUTES{itinerary},2)
      [StopIdx,Stoplist] = SearchIndex(Stoplist,ROUTES{itinerary}{Stop});
      Adj(LastStopIdx,StopIdx) = 1;
      wAdj(LastStopIdx,StopIdx) = TravelLinkTime{itinerary}{Stop-1};
      LastStopIdx=StopIdx;
    end
  end
end
```

3.2.2 Question 2 : Quality matrix based on Travel Time

Build a quality matrix where each edge has:

$$q_{ij} = \frac{1}{\text{TravelTime}_{i,j}}$$

(Note that $\text{TravelTime}_{i,j} = 0$ if there is no link from i to j, which leads to an Infinite quality). This case illustrates that the longer the journey, the more likely a problem could occur along the way.

Here is the following code of QTT.m (Quality TravelTime matrix) :

```
function retval = QTT(TTAdj)
  s = size(TTAdj, 2);
  m = zeros(s, s);
  for i = 1:s
    for j = 1:s
      m(i, j) = 1 / TTAdj(i, j);
    end
  end
  retval = m;
end
```

Since the subject provides a matrix of TravelTime, we're directly using it has argument of the QTT function. Hence, we call it to build the matrix based on the Quality TimeTravel :

```
QTT = QTT(TravelTime)
```

3.2.3 Question 3 : Quality matrix based on inverse Travel Time

Build a quality matrix where each edge has the following quality value.

$$q_{ij} = \frac{1}{\max_{i,j}(\text{TravelTime}_{i,j}) \times \frac{1}{\text{TravelTime}_{i,j}}}$$

This quality definition will run the algorithm using the opposite order than the previous one. Take care of getting Infinite values at the right place in the Quality matrix (where there are no edge, may be easier using this given writing)!

We will use the same method to construct a QITT matrix:

```
function retval = QITT(TTAdj)
  s = size(TTAdj, 2);
  m = zeros(s, s);
  maxTT = max(max(TTAdj));
  for i = 1:s
    for j = 1:s
      if TTAdj(i, j) == 0
        m(i, j) = Inf;  % Here we place Inf in case no link
      else
        m(i, j) = 1 / (maxTT * (1 / TTAdj(i, j)));
      end
    end
  end
  retval = m;
end
```

Another solution could have been to initialize the matrix m with inf values.

3.2.4 Question 4 : Quality matrix based on Betweenness centrality

Build a quality matrix where each edge has the following quality value.

$$q_{ij} = \frac{1}{1 + \text{EdgeBetweenCentrality}_{i,j}}$$

using the following definition for the Betweenness Centrality of edges :

```
>> BC = BetweennessCentrality(Adj);
>> EdgesBC = ( BC*ones(1,size(Adj,2))+ones(size(Adj,1),1)*BC' ) .* Adj
```

Explain this definition and what could represent this study case.

Once the EdgesBC matrix constructed, we can now construct the quality matrix based on Betweenness Centrality

```
function retval = QBC(EdgesBC)
  s = size(EdgesBC, 2);
  m = zeros(s, s);
  for i = 1:s
    for j = 1:s
      m(i, j) = 1 / (1 + EdgesBC(i, j));
    end
  end
  retval = m;
end
```

This quality matrix is based on the betweenness centrality of each node. The EdgesBC matrix is then constructed by summing the centrality values of adjacent nodes. This results in a quality matrix that reflects the importance of nodes in maintaining network connectivity, where edges connected to critical nodes have lower quality values.

3.3 PART 3 : Main algorithm

The objective of this algorithm is to study the percolation behavior of the network by iteratively removing links as in the attached paper. It will then be applied to the different scenarios listed above.

3.3.1 Question 1 : Subfunction updating reachable passengers demand

Propose a function that updates the feasible Origin/Destination demand matrix given an updated adjacency matrix (where edges has been removed).

```
>> [OD_updated disconnected] = updateOD(OD,Adj_updated)
```

One might use the $[Ws, P] = Dijkstra2(Adj, Origin)$ seen in course. After the loop, check if every path is reachable to output the 'disconnected' logical value for graphs with more than one connected subgraph.

```
function [OD_updated, disconnected] = updateOD(OD, Adj_updated)
    N = size(Adj_updated, 1);
    OD_updated = zeros(size(OD));
    disconnected = false;

    for origin = 1:N
        [Weights, Paths] = Dijkstra2(Adj_updated, origin);

        for dest = 1:N
            if OD(origin, dest) > 0
                if Weights(dest) == Inf
                    disconnected = true;
                else
                    OD_updated(origin, dest) = OD(origin, dest);
                end
            end end end end
```

3.3.2 Question 2 : percolation function

To avoid considering edges twice in the following part, replace all values in the lower triangular part of the matrix Q by infinity values (as one edge appear to be an directed arc).

```
>> Qtriu=Q.*(tril(Inf(size(Q)))+1);
```

Build a function that ouput the unaffected demand given the percolation threshold, reporting the $'Critical\_percolation\_threshold'$ from where all the stops are not connected from each others anymore (disconnected graph).

```
>> [Proba_OD_decrease, Proba_ARC_removal, Critical_percolation_threshold]=...
Percolation(Qtriu, Adj, OD)
```

One might consider a while loop over the non infinity values remaining in Qtriu : removing the edge with the lower quality; then updating the matrices until there are no more edges.

```
function [Proba_OD_decrease, Proba_ARC_removal,
↪  Critical_percolation_threshold] = Percolation(Qtriu, Adj, OD)
    Adj_updated = Adj;
    OD_updated = OD;
    edges = find(Qtriu ~= Inf);
    [~, sorted_idx] = sort(Qtriu(edges), 'ascend');
    edges = edges(sorted_idx);

    total_OD = sum(OD(:));
    Proba_OD_decrease = [];
    Proba_ARC_removal = [];

    for k = 1:length(edges)
        [i, j] = ind2sub(size(Qtriu), edges(k));
        Adj_updated(i, j) = 0;
        Adj_updated(j,i) = 0;

        [OD_updated, disconnected] = updateOD(OD, Adj_updated);

        remaining_OD = sum(OD_updated(:));
        Proba_ARC_removal(end+1) = k / length(edges);
        Proba_OD_decrease(end+1) = remaining_OD / total_OD;

        if disconnected && (exist("Critical_percolation_threshold",
        ↪  "var") == 0)
            Critical_percolation_threshold = k / length(edges)
        end
    end
end
```

3.3.3 Question 3 : Plotting the results on different Quality matrices

Using plot function to print *Proba_ARC_removal* given *Proba_OD_decrease*. Then add a mark for the *Critical_percolation_threshold* in the figure.
Perform the work for the three different Quality matrices built in the Part 2:

- **Scenario (a) : Ascending Order of Quality from question 2.**
- **Scenario (b) : Ascending Order of Quality from question 3 (reverse than a).**
- **Scenario (c) : AscendingOrderbased on the Edge_Betweeness centralities values of each edge.**

We can now plot the unaffected demand in terms of percolation threshold, the marks represent the *Critical_percolation_threshold* representing the threshold when the demand started to be affected by the edges removal.

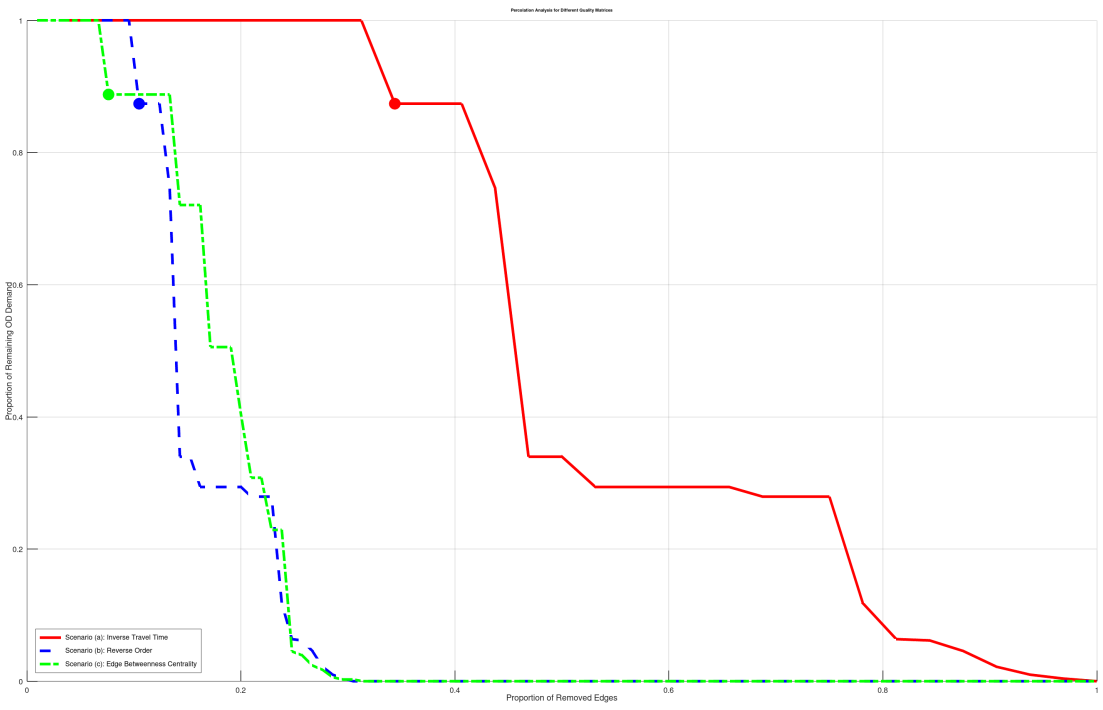

Figure 1: Unaffected Demand in terms of percolation threshold

- Scenario (a) : QTT (Quality based on TimeTravel) is red
- Scenario (b) : QITT (Quality based on Inversed TimeTravel) is blue
- Scenario (c) : QBC (Quality based on Betweenness Centrality) is green

3.3.4 Question 4 : Evaluating the Alphas

Build a function to estimate the area under the curve using rectangular (over and under) approximations.

```
>> Alpha = manual_integration(Proba\_OD\_decrease, Proba\_ARC\_removal)
```

```
function retval = manual_integration(Proba_OD_decrease,
↪  Proba_ARC_removal)

    Alpha_upper = 0;
    Alpha_lower = 0;

    for i = 1:length(Proba_ARC_removal) - 1
        delta_x = Proba_ARC_removal(i+1) - Proba_ARC_removal(i);
        y1 = Proba_OD_decrease(i);
        y2 = Proba_OD_decrease(i+1);

        Alpha_upper = Alpha_upper + max(y1, y2) * delta_x;
        Alpha_lower = Alpha_lower + min(y1, y2) * delta_x;
    end

    retval = (Alpha_upper + Alpha_lower) / 2;
end
```

Running this function to calculate the area under the curve for all scenarios gives the following results :

- Scenario (a) : $alpha_a = 0.505651$
- Scenario (b) : $alpha_b = 0.154103$
- Scenario (c) : $alpha_c = 0.171501$

3.4 PART 4 : Conclusions

3.4.1 Question 1 : Node failures

Considering the nodes are now blocked/attaked in the order of their beetweenness Centrality values (from lower to higher values, then in reverse, from the highest to the lowest), edit a new percolation function that removes all edges linked to a node at each step following a given order.

```
>> [tmp Node_order]=sort(BetweennessCentrality(Adj));
>> [Proba_OD_decrease, Proba_NODE_removal, Critical_node]=...
Percolation(Qtriu, Adj, OD, Node_order)

function [Proba_OD_decrease, Proba_NODE_removal, Critical_node] =
↪  Percolation2(Adj, OD, Node_order)

    Adj_updated = Adj;
    OD_updated = OD;
    N = size(Adj, 1);
    total_OD = sum(OD(:));
    Proba_OD_decrease = [1];
    Proba_NODE_removal = [0];
    Critical_node = NaN;
```

```
    for k = 1:length(Node_order)
        node = Node_order(k);

        Adj_updated(node, :) = 0;
        Adj_updated(:, node) = 0;

        [OD_updated, disconnected] = updateOD(OD, Adj_updated);

        remaining_OD = sum(OD_updated(:));
        Proba_NODE_removal(end+1) = k / length(Node_order);
        Proba_OD_decrease(end+1) = remaining_OD / total_OD;

        if disconnected && isnan(Critical_node)
            Critical_node = k / length(Node_order);
        end
    end
end
```

This function waits for a list of Nodes, assuming sorted before. It is basicaly the same algorithm than the Percolation function that we did before, but instead of looking for the index of the worst quality edges, we delete edges by the node order list.

3.4.2 Question 2 : New plots

Draw a new percolation curves from node removals and their impact on feasible passengers demand.

We can now plots the resultings curves. (orders : asc = blue, desc = red)

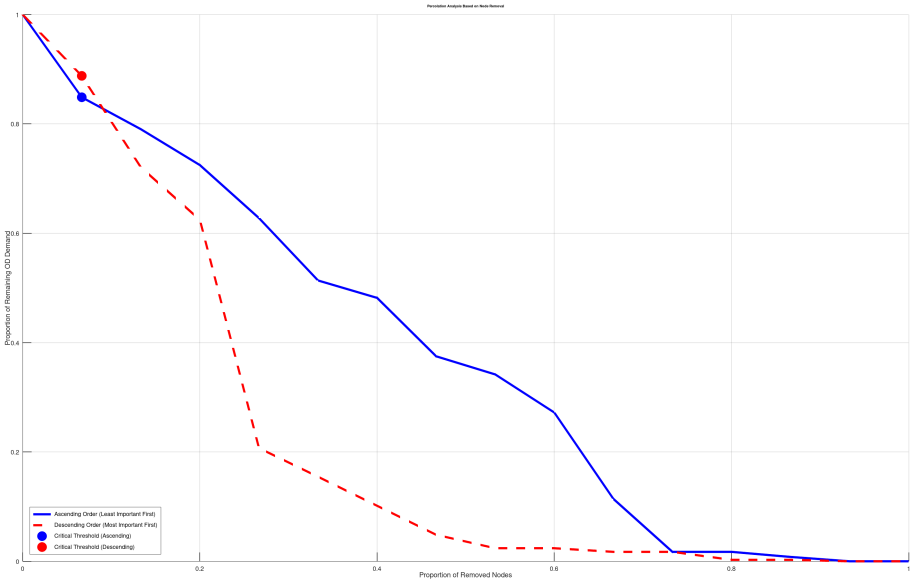

Figure 2: UD in terms of percolation threshold

3.4.3 Question 3 : About resilience

Conclude about the resilience of the propose study case.

In this study case, we saw multiple methods to assess the resilience of the network such as :

- **Impact of Edge Removal**
  The removal of edges following different quality matrices (travel time, inverse travel time, edge betweenness centrality) leads to varying degrees of network degradation.
  - Removing edges based on TravelTime has a lesser impact than the inverse of that technique, meaning that prioritizing edges with shorter travel times leads to slower degradation of the network. In contrast, removing edges based on the inverse of travel time causes faster degradation.
  - When high-betweenness edges are removed first, the network experiences a rapid loss of connectivity, as these edges serve as critical links between different regions.
- **Impact of "Node Removal"**
  Removing low-centrality nodes first has a limited impact on network functionality, as these nodes are not crucial for major flows.
  - However, when high-centrality nodes are removed early, the network undergoes a significant drop in connectivity, reaching the percolation threshold much sooner.
  - This confirms that certain nodes act as structural hubs, meaning their failure can severely disrupt the network.

Resilience Assessment of this Study Case

The network's topological structure is crucial for its robustness. The percolation technique is particularly useful in identifying these bottlenecks by simulating the removal of components and observing how the network's connectivity is affected.

This analysis highlights that networks with high centralization are at risk of rapid fragmentation when key nodes or edges fail. Overall, percolation analysis provides valuable insights for improving network robustness by addressing vulnerabilities.

4 Discussion

While this case study provides valuable insights into the use of percolation theory to assess network resilience, it is important to note that the study primarily focuses on evaluating the network's robustness. However, the concept of sensitivity, which was discussed in class, is also a crucial aspect of network resilience. Sensitivity measures how small changes or disruptions can significantly affect connectivity, and should ideally be taken into account to obtain a more comprehensive understanding of the network's resilience. The current approach does not explicitly address this component, but it could be an interesting way for further investigation.