**1 Project Specification**

* 1. **Introduction**

If, by happenchance, you see someone running frantically through a tube stop wearing a penguin outfit, or you see some people crossing dozens of stations off their map, while wondering whether they’ll make the connection, you might have seen an attempt at the Tube Challenge.

This game is only for those who are either incredibly athletically inclined or are obsessed with mathematics – or have some unusual fascination with the Underground network – as it involves running around from station to station from half six in the morning until late at night, depending on the success of the route. Even assuming you have found the latest Guinness World Record route, your best chances of finishing are around 9:30 in the evening!

The attraction of this challenge is its simplistic nature: travel through every stop on the London Underground system at least one using only public transport. Yet with its 270 stations served, it is not difficult to see that finding the route with the shortest time is a challenging task. It is in fact a twist on the age-old mathematical problem, the Travelling Salesman Problem (TSP), which has proven to be an NP-hard problem. As later discussed, this problem is a constant topic of discussion and there are many different algorithms which could be used to solve the problem, some of them specifically devised for this task, like the Lin-Kernighan algorithm. The main aim of this project is to find and compare these algorithms and choose the “best” suited for the Tube Challenge under several aspects of consideration.

While the project only concerns the Tube Challenge, it could be adopted in other operations research applications such as logistics problems. It could also be useful for transport applications such as Google Maps or Citymapper.

**1.2 Objectives**

* To create a sufficient and simplified model of the network and to find a way to represent this in matrix form.
* To find a sufficient model of the problem using the Open Travelling Salesman Problem.
* To find algorithms that are suited to solve the problem.
* To take as many factors into consideration as possible when choosing the “best” algorithm.
* To take flexibility and further improvements into consideration when implementing the algorithm.
* To visualise the workings of the chosen algorithm

**2 Background**

**2.1 The Tube Challenge**

While the London Underground network has existed since 1863, the first recorded attempt at the Tube Challenge was done in 1959, crediting R. J. Lewis and D. R. Longley. Since then the game has evolved just as the network has: the number of stations have fluctuated between 264 and 278 between then and now. As of 2016, the record for fastest completion was held by Andi James and Steve Wilson, who completed the challenge in 15 hours, 45 minutes and 38 seconds on 21 May 2015.

**2.1.1 Rules of the Tube Challenge**

1. All stations served by London Underground trains must be visited. To ‘visit’ a station, a traveller must arrive and/or depart by an underground train in normal public service. It is necessary for a through train to stop at the station for the visit to count, although the traveller does not need to get out. Certain stations are normally only open at certain times of the day, this must be considered in planning. Attempts can only be made during the week as certain stations are shut at weekends. Only if a station is temporarily closed (e.g. for rebuilding, or in an emergency) will a non-stop pass through a station be acceptable.

2. Stations which are geographically separate and not linked but which have the same name must each be visited. This applies to Edgware Road, Paddington, and Hammersmith.

3. Feet or public service transport may be used to transfer from one underground line to another. The use of private motor vehicles, taxis or any other form of privately arranged transport (bicycles, skateboards, etc.) is not acceptable.

4. It is not necessary to cover every stretch of track on the network. Thus, for example, if traveller have visited Green Park on the Victoria line, Piccadilly Circus on the Piccadilly line, and Leicester Square on the Northern line, the traveller does not have to take the Piccadilly line from Green Park to Leicester Square.

There are many other rules that concern the validity of a Guinness World Record attempt at the challenge, however for the purposes of this project these four rules provide sufficient information regarding plans.

**2.2 The London Underground**

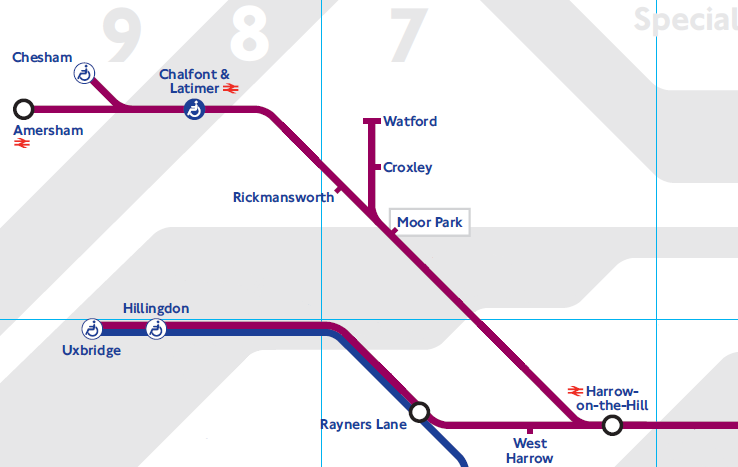
The London Underground network serves 270 stations with 11 lines. However, for our purposes the number of these stations can be significantly reduced: to 150 stations (see Appendix).

Figure 2: Modified section of the London Underground map.

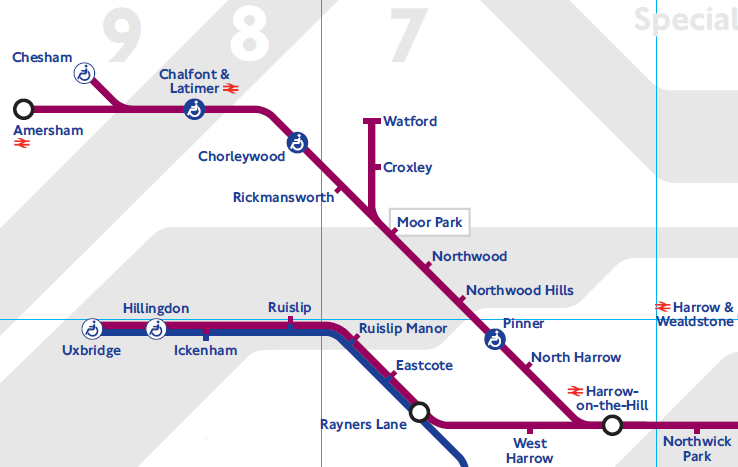


Figure 1: Unchanged section of the London Underground map.

For instance, take the above part of the Metropolitan line (and Piccadilly line). The stations that are not endpoints of a line, nor are changing points, nor are in proximity to other stations geographically can be eliminated from the network considered for the project. The latter can be deduced using a map with geographical distances.

There are other factors regarding the London Underground that should be taken into consideration by this project. For instance, changing between stations can happen in 4 different ways:

**1, Same platform**: At South Kensington, changing between the Circle and District lines happen on the same platform.

**2, Different platform**: At Gloucester Road, changing between the Circle and District lines happen at different platforms. Note that this is the reason why on the map of the Underground, Gloucester Road is not marked as a possible point of change.

**3, Pedestrian Subway:** At South Kensington, changing to the Piccadilly from Circle and District lines requires a several minute walk through an underground subway. This is still regarded as the same station but requires a longer travel than most changes.

**4, Out of Station:** Some stations are not connected through a pedestrian subway yet are in close enough proximity to each other for the map of the Underground refer to them as a changing point. For instance, changing between the Central line at White City and the Circle/Hammersmith and City at Wood Lane requires exiting the station for a 20-minute walk.

**2.3 Graph theory**

This section will explain basic concepts and theories in graph theory necessary for the modelling of the Underground network and possibly for the implementation of the project. I will be using MATLAB for the implementation part of the project and therefore will show some examples with MATLAB code.

**2.3.1** A graph is set of entities called vertices, which are interrelated via certain correspondences called links. Links may be directed (with directions shown by arrows) in which case they are known as arcs, or they may be undirected and called edges.

**2.3.2** A directed graph consists of a pair of sets (V, A) where  
1, V is non-empty;  
2, elements of A are directed pairs of, not necessarily distinct, elements of V.  
Elements of V and A will be called vertices and arcs respectively.   
**2.3.2a** Alternatively, a directed graph consists of a pair (V, Γ), where   
1, V is non-empty;  
2, Γ is a mapping from V to itself (Γ: V —> V)

**2.3.3** An undirected graph is a pair of sets (V, E) where  
1, V is non-empty;  
2, elements of E, called edges, are undirected pairs of not necessarily distinct elements of V.

**2.3.4** A graph G, whether directed or not, is connected if there is a chain between every pair of vertices of G. If G is not connected then it can be split into components, each of which is a maximal connected graph contained in G.

**2.3.5** A tree is a connected undirected graph without cycles.

**2.3.6** Theorem:  
The following statements about an undirected graph G are equivalent:  
1, G is a tree;  
2, G is connected with n vertices and n-1 edges;  
3, G has n vertices, n-1 edges, and no cycles;  
4, G is such that each pair of vertices is connected by a unique elementary chain.

**2.3.7** A spanning tree T of a connected graph G on vertex set V is a tree, also with vertex set V, whose only edges correspond to edges of G.

**2.3.7a** Or, put more simply, a spanning tree T of a connected graph G is a subgraph that is a tree, which includes all vertices of G with minimum possible number of edges.

**2.3.8** A path(chain) from x0 to xs in a directed(undirected) graph (where x0 and xs need not be distinct) is a non-null sequence of arcs (edges) x0,x1,x2,…,xs-1,xs and is denoted as x0x1x2…xs

**2.3.9** A circuit is a closed path (that is, its endpoints coincide), and a cycle is a closed chain. A loop is a circuit (or cycle) containing one arc (or edge).

**2.3.10** A Hamiltonian circuit(cycle) in a directed(undirected) graph is a circuit(cycle) which visits each vertex once and once only.

**2.3.11** A Hamilton chain (path) in a graph on n vertices is a chain (path) which visits each vertex only once and contains n-1 edges (arcs).

**2.3.12** Matrix representations of graphs:  
For a directed graph (X, Γ) on vertices numbered 1 to n, a vertex-vertex matrix is an (n x n) matrix P, with the element Pij giving information concerning arc ij. Examples of P include:  
1, Symbol matrix S  
Sij = ij if j ϵ Γi  
 = 0 otherwise

2, Adjacency matrix A  
Aij = 1 if j ϵ Γi

= 0 otherwise

3, Distance matrix   
Dij = length of ij if j ϵ Γi  
 = ∞ otherwise

Examples in MATLAB code:

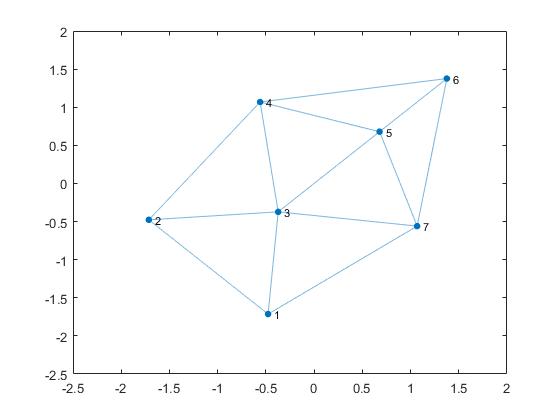
In MATLAB there are three ways to represent graphs:

1, Using the matrix representations defined above.

In this example code I am using the distance matrix form to add the weights.

A = [0 12 10 0 0 0 12; 12 0 8 12 0 0 0; 10 8 0 11 3 0 9; 0 12 11 0 11 10 0; 0 0 3 11 0 6 7; 0 0 0 10 6 0 9; 12 0 9 0 7 9 0];

G\_1 = graph(A);

plot(G\_1);

2, Using node pairs. The advantage of this is that the weights of each edge can be adjusted easily without the need to change two entries of a matrix (at least with symmetric representations). Furthermore, it makes the labelling of the edges much easier by having a separate array with the information.

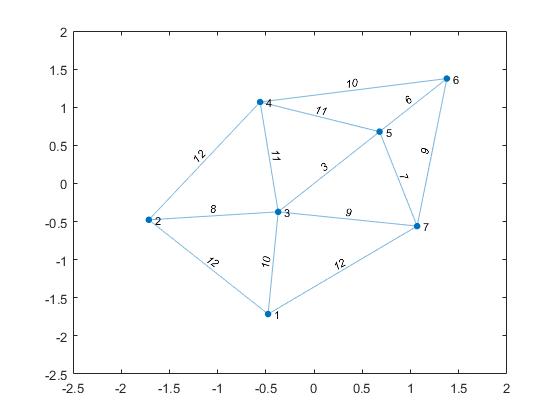
s = [1 1 1 2 2 3 3 3 4 4 5 5 6];

t = [2 3 7 3 4 4 5 7 5 6 6 7 7];

weights = [12 10 12 8 12 11 3 9 11 10 6 7 9];

G\_2 = graph(s,t, weights);

plot(G\_2, 'EdgeLabel', G\_2.Edges.Weight)



3, Using tables. The advantage of this data structure is that using the readtable function it can import Excel files into MATLAB.

[insert matlab code here]

**2.4 The Travelling Salesman Problem**

A salesman starting from town T wants to visit n-1 other towns: T2, T3, …, Tn, each once only, finally returning to T1. If the distance dij from town Ti to Tj is given for each pair (i,j) find a route for the salesman which minimises the total distance travelled. Much of the interest in the problem arises from the challenge presented by the fact that the problem is easy to state and understand but computationally very difficult to solve. It is proven to be an NP-hard problem.

The (asymmetric) travelling salesman problem introduced informally above can be states as:

For given cost matrix (cij) minimise , where (i1, i2, …, in) is a permutation of (1, 2, …, n) which is not a product of two smaller permutations.

Let G = (V, A, c) denote the complete directed network on vertex set {1, 2, …, n} with the length of arc ij being cij. ATSP can clearly be restated as follows:

Find a minimal length Hamiltonian circuit of G. (Note that in the context of the travelling salesman problems, Hamilton circuits are often called tours.)

The open TSP can be defined as a TSP in which a shortest Hamilton path is sought from vertex s to vertex f, with distinct s and f.

**2.5 Complexity**

[write here from graph theory book and Natural Algorithms project]

A function f(n) is ϴ(g(n)) if c1 > 0, c2 > 0 and n0 such that for n ≥ n0 :  
0 ≤ c1g(n) ≤ f(n) ≤ c2g(n)

A function f(n) is O(g(n)) if c > 0 and n0 such that when n ≥ n0:  
0 ≤ f(n) ≤ cg(n)

A function f(n) is Ω(g(n)) if c > 0 and n0 such that when n ≥ n0:  
0 ≤ cg(n) ≤ f(n)

|  |  |  |
| --- | --- | --- |
| **Notation** | **Meaning** | **Program takes** |
| *O* | Less than | At most this long |
| Ω | Greater than | At least this long |
| ϴ | Equal | Approximately this long |

**P and NP**

We've already seen that some algorithms run in polynomial time, e.g.

O(n), O(n2), …, O(nk)

We call these tractable.

This means something slightly less strong than „easy".

On the other hand, exponential time algorithms are intractable, e.g.

O(2n), O(n!), O(nn)

Intuitive definitions

P is the class of problems that have algorithms that can solve them in polynomial time.  
The P stands for polynomial.

NP is the class of problems where you can and a polynomial time algorithm that checks if a given answer is really a solution or not.

NP stands for nondeterministic polynomial.

A problem is NP-hard if it is at least as hard as every NP problem.

A problem is NP-complete if it is NP and NP-hard.

8.2.4 Formal definitions

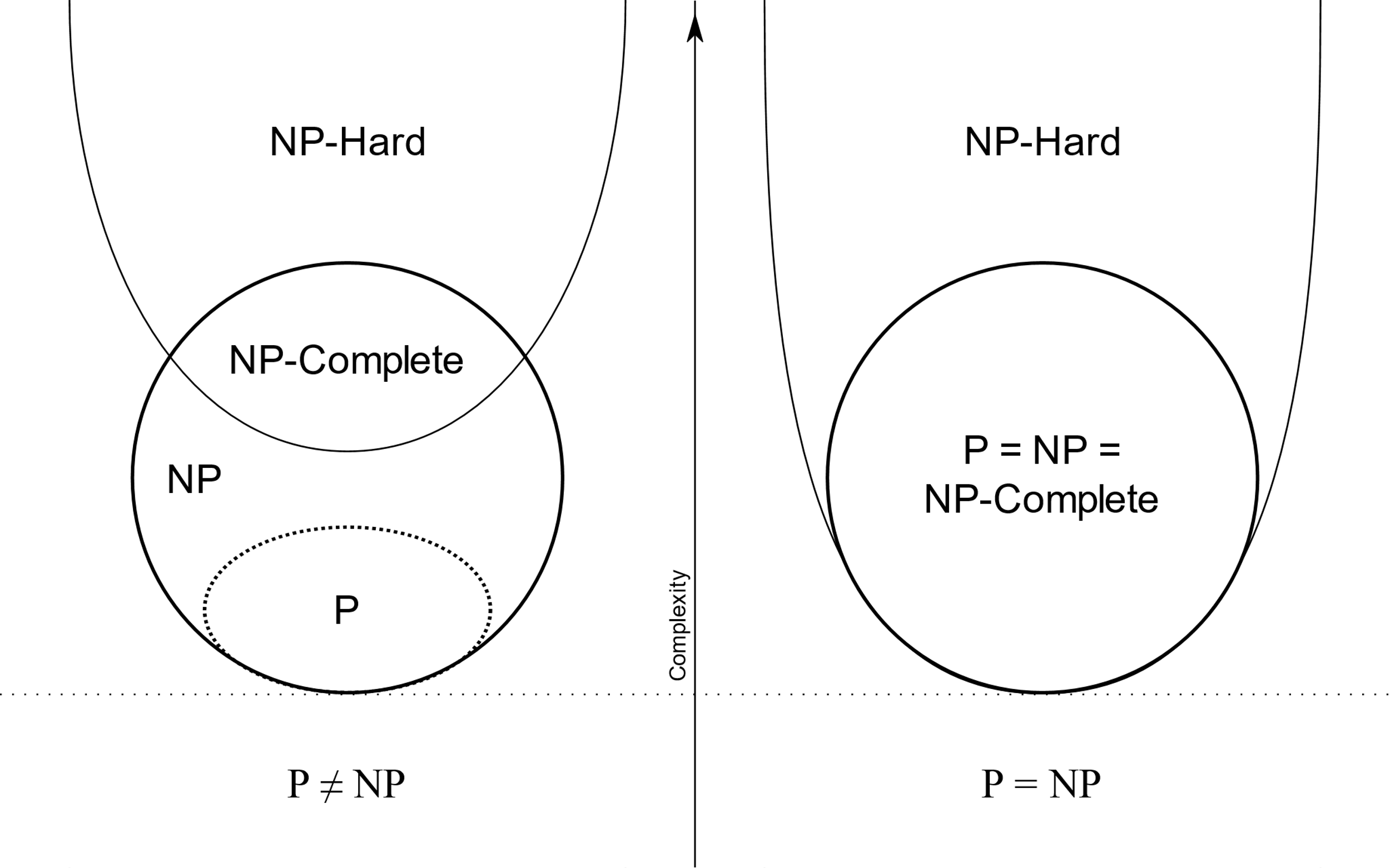
Definitions of P and NP are formal enough already.

P = problems solvable in polynomial time

NP = problems with verification of solutions in polynomial time

Problem Q is NP-hard if for all NP problems Q’ we can polynomially reduce Q’ to Q, i.e. Q’ ≤ p Q

This is what I meant by saying that NP-hard problems are at least as hard as any problem in NP.

****Problem is NP-complete if it is NP and NP-hard.

**2.6 Algorithms**

There are 2 different kinds of algorithms we can take into consideration for this project:

* Exact
* Approximate (heuristic)

**2.6.1 Exact algorithms**

While these would find the optimal solution to our problem, exact algorithms are computationally complex and therefore are usually inefficient for even a small dataset. However, in our case it might be worth taking into consideration due to the simplification to the network and the relatively limited connections between stations.

1. Brute Force Search: Looking at every single ordering of stations individually.

2. Held-Karp algorithm: http://www.cs.man.ac.uk/~david/algorithms/graphs.pdf

3. Branch and bound algorithm: Dividing the problem into subproblems and solving those subproblems to find a solution for the original problem. In other words, first partitioning the entire set of feasible solutions into smaller subsets (“branching”), then finding a “bound” of how good the best solution in the subset can be and discarding the subset if its bound indicates that it cannot contain the optimal solution.

**2.6.2 Heuristic algorithms**

These can be subcategorized into three:

1. Greedy heuristics: making locally optimal choices at each stage of the algorithm. An example of this is the Nearest Neighbour or the Christofides algorithm.

2. Specialised heuristics: heuristic algorithms specifically developed for solving the TSP. Examples of this are the Lin-Kernighan and the 2-opt algorithm.

3. Generic heuristics: heuristic algorithms based on local search designed for no particular problem, such as ant colony optimisation, tabu search, simulated annealing and genetic algorithms.

**3 Implementation and Evaluation Plan**

**3.1 Implementation Plan**

Operations research, as its name implies, involves “research on operations”. The “research” part of the name means that operations research uses an approach that resembles the way research is conducted in established scientific fields. A common theme in operations research is that it frequently attempts to search for a best solution - referred to as an optimal solution - for the model that represents the problem. Hence this project falls under the domain of Operations Research due to its attempt to find the optimal solution to the Tube Challenge.

In literature, operations research tends to be summarized as a six-step process:

1, Defining the problem of interest and gathering relevant data.

2, Formulating a mathematical model to represent the problem

3, Developing a computer-based procedure for deriving solutions to the problem from the model

4, Testing the model and refining it as needed

5, Preparing for the application of the model

6, Implementation

This procedure can be applied to our problem to provide a sufficient implementation plan for the project. However, since this is defined as a way companies work on operations research problems, steps 5 and 6 are not applicable this project, unless one considers the completion of the Tube Challenge in real life as an implementation of the project. In this case, step 6 can be considered as an optional step, but we are going to exclude this from the implementation plan below.

**3.1.1 Defining the problem and gathering relevant data**

There are several steps involved in this part, to specify:

* Formulating the problem statement: Find and implement the algorithm most suited to solve the Tube Challenge.
* Deducting the underground network to a minimal number of nodes (stations)
* Find travel times between each node (station)
* Find exchange times between each edge (line)

*Timeline of this step:*

At the time of writing this report, I have completed the problem statement definition (as can be seen), the simplified network (see Appendix) and have started to gather information for travel and exchange times. The latter two are not necessary to complete before the implementation of the algorithm for simpler networks has been done, but is not a demanding task.

**3.1.2 Formulating a mathematical model**

The Tube Challenge has strong similarities with the commonly researched Open Travelling Salesman Problem (OTSP) defined in the Background section. However, it is a much stricter problem than what we aim to solve. The Tube Challenge would be near impossible to solve with the restriction of one visit only at each station. Multiple visits to stations are not only encouraged, but is undoubtedly the best way to visit stations that are in outer zones. Furthermore, while the Underground does contain quite a few connections between each station, the graph formed by the Underground lines isn’t nearly complete and hence restricts the route between quite a few nodes. In fact, the lines are so restrictive that there are many stops that lie on a single line and have no option for transfers. This implies that we should take some these stations out of consideration overall, as mentioned previously.

While the OTSP does not exactly model the problem, it is still the best approach and therefore with some small modifications (i.e. allowing multiple visits to each node) can be implemented.

**3.1.3 Developing a computer-based procedure**

This step consists of several smaller steps in our case:

I. Identifying algorithms which could be potential candidates to find the optimal solution

II. Analysing the above algorithms to be compared and deducing the “best” algorithm

III. Implementing the chosen algorithm

*Timeline of this step:*

In the Algorithms section I have done a brief introduction to the algorithms I am considering for this step. I am now quite certain that I will not choose an exact algorithm because of the size of the network (150 nodes with O(n!) or O(n22n) complexity is not a sensible solution). I have yet to explore which of the heuristics would be best to use, but I will most likely have the greedy algorithm as a “back-up” plan as that is very simple to implement. At this stage I plan to implement the best of the specialised OTSP heuristics and the best of the generic heuristics for comparison.

**3.1.4 Testing**

As this step only involves putting all mentioned previously together, it will hopefully not require much time apart from some minor debugging. Depending on time constraints, I might do a physical trial of the route.

**3.2 Evaluation Plan**

This project has few ways for measuring success

* comparing the actual value of the found optimal time to the current Guinness World Record time
* completing the Tube Challenge in person to test the route and compare that time to the Guinness World Record time

Apart from these, the way to measure the success of the project is to see whether the objectives described earlier were met by the time described in the next section.

**3.3 General Timeline**

So far, I have done most of the theoretical research and relevant data gathering, and have noted several algorithms for comparison.

The coding will be the most time-consuming part of the project as it will have to take all the findings of the data gathering, organise it into a meaningful data structure, implementing the algorithm itself, and debugging the above code.

As much of the work to be done in this step is unrelated to what algorithm was chosen in step II, I have already started working on the first 4-5 parts. Still, writing and debugging code can be a tedious process and so I plan to have this step ready in 2-3 weeks at most.

Finally, step IV will not be difficult once step III is completed and therefore can be completed probably well within a week. This means that even with the worst-case scenario considerations, the entire project should be done by the deadline for the abstract and draft report. A substantial portion of this project is based on theory, and therefore the only step that needs a fall-back option is step III. Therefore, during step II I will choose a fall-back algorithm with the lowest complexity possible to make computations in step III as trivial as possible.

This leaves us with 2,5 weeks for the completion of the final report and potentially one of the optional steps described above as well.

**References with literary reviews**

Books, journals:

- Introduction to Operations Research / Frederick S. Hiller, Gerald J. Lieberman

Included an introduction to the field of Operations Research, what it entails. It described the Operations Research Modeling Approach on which the Implementation Plan is based. Contained an introduction to Network Optimization Problems, most importantly to the shortest path problem and minimum spanning tree problem (MST). Explained the branch-and-bound algorithm with examples. Had a brief introduction to nonlinear programming. Finally, it talked about different metaheuristics such as tabu search, simulated annealing and genetic algorithms in context with the travelling salesman problem.

- Graph Theory in Operations Research / T.B. Boffey

Contained an introduction to graph theory and its basic concepts. Showed different methods of representing graphs as matrices. Explained the branch-and-bound in relation to graphs. Explained the travelling salesman problem from a graph theory perspective with examples.

- A Note on Two Problems in Connexion with Graphs / E.W. Dijkstra

Contained and explained Dijkstra’s algorithm

- Algorithms for Large-scale Travelling Salesman Problems / Nicos Christofides, Samuel Eilon

- Bounds for the Travelling Salesman Problem / Nicos Christofides

- On the Symmetric Travelling Salesman Problem: A Computational Study / Manfred W. Padberg, Saman Hong

- Solving Large-Scale Travelling Salesman Problems to Optimality / Harlan Crowder, Manfred W. Padberg

- The Shortest Hamiltonian Chain of a Graph / Nicos Christofides

- An Effective Implementation of K-opt Moves for the Lin-Kernighan TSP Heuristic / Keld Helsgaun

http://www.akira.ruc.dk/~keld/research/LKH/KoptReport.pdf

- Heuristics for the Travelling Salesman Problem / Christian Nilsson

https://web.tuke.sk/fei-cit/butka/hop/htsp.pdf

Websites:

- http://www.thetubechallenge.com/

Contains the official rules of the Tube Challenge along with other useful information.

- http://www.telegraph.co.uk/travel/lists/How-to-do-the-Tube-Challenge/

Contains a suboptimal route that was the Guinness World Record route from 2009

- http://travel.stackexchange.com/questions/6729/what-is-the-fastest-theoretical-route-for-the-london-tube-challenge

Practical advice and challenges presented of the Tube Challenge