Optimization of the Railroad Connection Selection

Maksym Gontar[a], Kostiantyn Lapchevskyi[a], Igor Luzhanskiy[a]

[a] *Ukrainian Catholic University*

Abstract

In this research provided an approach on solving "shortest path" problem for the domain of passenger railway transportation. Objective to minimize is time. Ukrainian railroads taken as an example.

Keywords: Railroad, A^*, heuristic function

1. Business context

The current section is aimed to provide a holistic overview of project. The overview will include analysis of business and competitive environment, requirements for the product and detailed description of algorithms , frameworks and architecture of the solution (last one is the subject of evaluation for the Linear Algebra course in Ukrainian Catholic University)

1.1. Elevator speech

Our project is chatbot that lets you to find tickets for the railway in Ukraine, when it seems like there is not tickets. We are using smart algorithms and gaps of UZ policies to find the tickets for you from place A to B.

1.2. Environment analysis

1) Political. After 26 years of independence Ukraine still hold a solid percent former Soviet Union business and political patterns. One of them is related to the state regularization at the area of railway transport, which is managed by the state monopolist UkrZalizhnitsha (UZ for future reference). UZ consumes donations from the government budget and operates at the very low level of efficiency due to corruption and lack of market competition.

2) Economical. UZ has 2 main divisions - one dedicated to cargo transportation and another for the passenger transportation. The first one is profitable, while the second not. That limits ability of UZ to provide enough facilities for the passenger transportation. The conflict within Russia has increased the pressure for passenger transportation division, as flights as well as the rest out of Ukraine become less affordable to Ukrainians, so UZ is seeing

Email addresses: max.gontar@gmail.com (Maksym Gontar), lapchevsky.k@gmail.com (Kostiantyn Lapchevskyi), Igor@Chatbots.Studio (Igor Luzhanskiy)

a higher demand for the transportation of passengers, while decreased demand for the cargo transportation puts even more pressure on UZ profitability.

3) Technical. There are several technological trends in Ukraine that are impacting the product. The first one is the spread of 3G that has de-facto became a standard since being introduced at 2013. Moreover, wide spread of credit cards and smart phones, make consumers keener to make purchases via smart phones. Which is proven by the growth proportion of on-line tickets purchases comparing to off-line.

1.3. Elevator speech

1/4 of passengers are making the purchases of tickets at the last day before the departure [1]. Within the constant crisis of UZ and inability to meet the demand for transportation, creates a huge frustration when a client could not get from the point A to point B. In some cases, clients need to get to the point at any cost and could not move their journey (airplane connection is very expensive and poor in Ukraine, while buses are not reliable and do not connect well) either to a different transportation channel or date.

1.4. Business model

The product is going to earn money by selling the tickets via high price margin comparing to other electronic services. We expect that clients will be willing to pay more money for tickets in case we will be ready to find them a way to get from A to B via railway transport. Moreover, at the later stages we are going to introduce auctions for buying the tickets that could suddenly appear.

1.5. Competitors analysis

There are various ways that are used by the passengers to find tickets while they are not available

1) Sleep at the railway station. The is the old days approach. The passenger should stay in the tickets office regularly chasing the tickets officer if a new ticket is available. 2) Engage friends in UZ. UZ is booking some of the tickets and do not let them to be sold until certain period. Having connection in UZ could let buying some of that reserved tickets. 3) Agree with travel agencies. Some travel agencies related to UZ could sell the blocked or returned tickets within the additional margin. 4) Buy tickets on the dart market. Illegal tickets trade has become the very well developed. Re-sellers are used to buy a certain amount of tickets for the popular directions and resell them within the very high margin. That is illegal as every ticket is purchased for the dedicated person. 5) Engage with conductor. It is rather popular option just to go and talk with the conductor just in the train, sometimes it is possible to ask, pray or convince them to take you on board without ticket, but for additional money. That schema works well for the end stations and is full of risk, as the place is not guaranteed.

1.6. Product Risks

The are 2 main risks associated with the product: 1) Issues with access to API of UZ to get a ticket information. UZ is highly regularized and rigid organization, they have less than zero wiliness for the collaboration, thus we can face the situation when no API will be provided for us and we will have to scrap the data from booking.uz.gov.ua 2) Other competitors could quickly copy our product. Our commercial competitors could look at our features and copy them into their products while we will keep developing the features of the product. They definitely have more power and resources to do that

1.7. Key gaps in UZ that let finding tickets if they are not officially listed

There are several tricks that enable to find tickets when they could not be found via normal ticket search: 1) Make a change inside the train. It is possible that for the train there is not place from A to B, but there is an empty place for that train from A to C and from C to B, which could be the most optimal option for the traveler in terms of the trip duration; 2) Start looking for a free place from the stations which are located before the station A. UZ has intention to provide the passengers from the end stations with ability to buy tickets for the whole route, so UZ simple do not sell tickets for the intermediate stations before 24 hours from the train departure. 3) Buy tickets buy changing the stations. That is very known approach for airplane industry 4) Setup a notification rules, which will buy a ticket in it will become available, that work for the cases when someone will cancel his/her trip or reservation for the tickets will be cancelled.

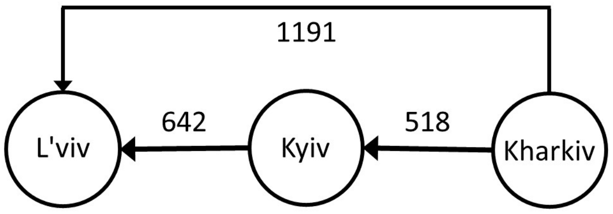
2. Data structure

2.1. Base setup

Natural representation comes as a graph structure [2], where nodes are the stations and edges are the train connections. Under the objective of finding shortest (in time) path between two stations it makes sense to assign each edge a weight as a difference of departure time from starting station (node) and arrival time to end station (node).

As an illustration let us consider simplified version of route #1110: train departs from Kharkiv, makes a stop at Kyiv and arrives to L'viv (check figure 1).

Figure 1: Simplified route #1110: "flat" representation



Note that we have directed connections between all nodes in a route, not only consequent stops. Reason for that is to reflect the underlying nature of train tickets. In particular, one can buy any of: Kharkiv → L'viv, Kharkiv → Kyiv, Kyiv → L'viv; and with such representation it strictly holds: "One edge - one ticket".

2.2. Inclusion of Time

Let us compare "direct" and "indirect" paths from Kharkiv to L'viv. "Direct" has weight of 1191, while "indirect" yields $518 + 642 = 1160$. But how could this be if both represent the very same route? Trains do not depart at any moment you would like them to, there is a schedule. Thus, there is a some non-zero time, which train spends at Kyiv station (31 minute to be precise), which is not accounted in given structure.

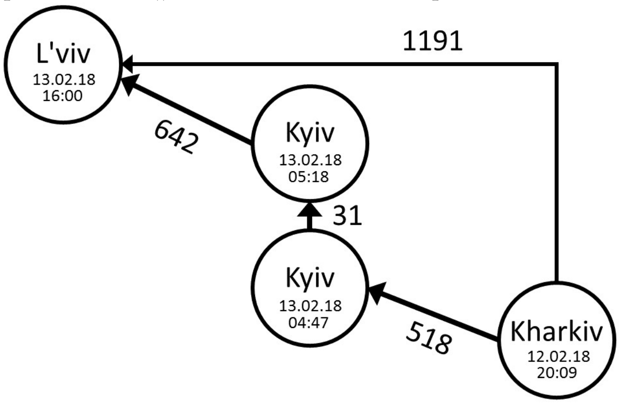
It actually reveals a more fundamental issues: How to distinguish trains that run repetitively on the same route? How to determine the relative order of trains from different routes? Incorporation of time feature in graph could be a solution. There are at least two different approaches, which we are going to call "Edge" and "Node" approach respectively.

2.3. "Edges" or "Nodes"?

"Edge" approach leads to the "mutable in time" graph structure. Finite set of nodes (stations) connected with finite set of edges (routes); two nodes can be connected with more than one edge; each edge represents virtually unlimited number of trains that (going to) follow given route. It means that while traversing graph we have to track actual value of time and filter out valid connections for a node (on each step), also weights should account for waiting time (recalculate on each step). It implies serious complication of traversal algorithm, which is desired to be avoided.

In "Node" approach we allow more than one node for every station: every time-stamp (train departed from / arrived to station) spawn a node for given station. Each edge represents specific train following the route at given time-frame between given stations, not the route itself (check figure 2). Comparing to the "Edge" approach more memory required to store the structure, on the other hand at the stage of algorithm execution only check on tickets availability left, otherwise connection is necessary valid. Also, such structure expresses a nice property - distance (in time) between two nodes (if connection exists, not necessary direct) can be computed as a difference of respective time features rather than sum of connections' weights.

Figure 2: Simplified route #1110: "time-aware" representation for specific train

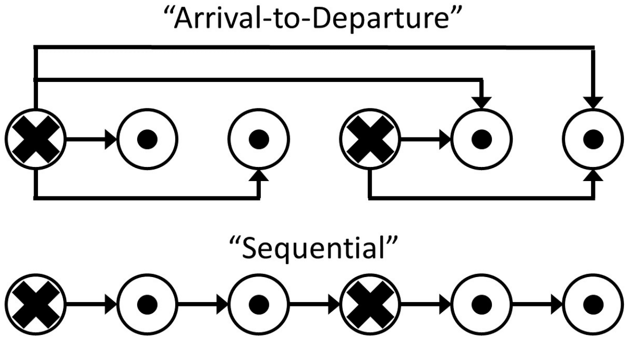


Overall, "Node" approach appears to be more beneficial as far as computational time sensitivity is higher than memory consumption sensitivity, therefore it is going to be utilized.

2.4. Waiting at station

Let us pay closer attention to the events, which may happen once train makes stop at station. In simplest case person can end up oneself travel there or continue to the next station on the same train. Though neither of these is necessary the case - person can wait for another train to depart. Initial proposition here is that person practically wait only from moment of time when one train arrives and next departs, not till some other train arrives to the station. But does it yields the most convenient graph structure? Let us compare it with sequential connections (check figure 3).

Figure 3: "arrival-to-departure" versus "sequential" connections



In both cases same single station depicted; time-arrow goes from left to right; "cross" represents the node associated with train arrival, "dot" - departure. Note: one node potentially can be associated with multiple trains, their arrivals and departures, however it does not impact core ideas depicted in figure 3.

Easy to see that for "Sequential" connections each node (except first one) spawns exactly one edge, hence number of edges increase linearly with number of nodes.

For "Arrival-to-Departure" connections it can be seen that "departure" connection spawns no edges, while "arrival" node spawns an edge to each "departure" node further in time. Increase of total number of edges with number of nodes can be estimated through arithmetic sequence as quadratic.

Difference is not only in the number of edges though. Assume roughly the same number of "arrival" and "departure" nodes (otherwise physical increase of concentration of wagons at certain stations would be observed), and their close to even distribution. Consider now the very first "arrival" node, in order to compare different possible choices (e.g. wait for train A or wait for train B) a distance function should be invoked for every adjacent node. For "Arrival-to-Departure" approach it is necessary $\approx n/2$ times (for within-station nodes); for "Sequential" approach it could be anything between 1 and $n-1$ (for within-station nodes), however there is a strong bias towards lower values (e.g. overall travel time with train you waited for 36 hours is unlikely to be lower than with train you waited for 12 hours).

It can be argued that even for lower (on average) number of call of distance function for "Sequential" approach each separate call is computationally more demanding (per node) due too algorithm routine rather than in case of "Arrival-to-Departure" bulk call. However, due to the mentioned bias, it might be the case only for very short (in time) structures. If want to extend it to complete coverage (several weeks) then "Arrival-to-Departure" approach is no longer viable from computational complexity perspective. Note: there is no difference between given approaches when it comes to exploration of nodes that belongs to other stations.

Overall, "Sequential" connections approach gives better properties on memory consumption, computation complexity and tolerance to graph size growth, therefore it should be favored compared to "Arrival-to-Departure" connections approach.

2.5. Stations have size

In current graph structure it may be a perfectly valid situation that optimal path involves exchange of trains within a minute, but in reality it takes some time to move within a station (can be substantially more than one minute), hence adjustments have to be done.

There are different approaches to tackle this issue, some of them are: a) selection of valid edges/nodes can be applied; b) shift of stated departure time; c) shift of stated arrival time.

2.5.1. Selection of valid edges/nodes

Skip nodes that are closer in time than given quantity. It is the most sustainable approach, as there is no information to be altered, but requires track of the previous state and explicit distinction between edges related to train connections and waiting at station, which goes against already taken architectural decisions.

2.5.2. Shift departure time

General idea is that one should be on board some time beforehand (e.g. 10 minutes before departure). Technically it can be done through altering departure times to be some time earlier and increasing respective edge weights with the same amount.

2.5.3. Shift arrival time

Idea is close to the previous one, but reverted - it takes time get out from train and walk across the station (e.g. same 10 minutes). Technically it can be done through altering arrival time and increasing respective edge weights. This approach is the one proposed for implementation. It may be argued that theoretically there is no difference between shifts of departure and arrival time, but practically trains never depart earlier than stated time. Thus, it would be more user-friendly decision to alter arrival time, as it does not affect first leg of travel (person has not to wait for departure).

3. Traversal Algorithm

3.1. A^* (A-star)?

From mathematical standpoint the whole task is formulated as a search of shortest path between two given nodes in directed graph. Baseline choice for traversal algorithm here is A^*, but can we get anything better? Let us examine possible options.

JPS (Jump Point Search) method [3] - can provide a better performance in order of magnitude, but requires uniform grid. Later is not the case for given graph structure, hence method cannot be applied.

Bi-directional A^* [4] - reported increase of performance is around 30%, but requires prior knowledge of end node. Take into account that we know only end-station name, while for complete definition of node time also necessary. However, if time for end-station provided as well then it is no more task of optimization, hence out of the scope of this research.

IDA^* / Fringe search [5] - both methods give up to a different extent on computational speed comparing to A^* in order to lessen memory consumption. Due to the choice of data representation whole graph can be big, nevertheless search subspace (part of the graph) used on computing the algorithm is relatively small, therefore overuse of memory is unlikely. On the other hand, response time of the algorithm is desired to be minimized (end-user should receive result as soon as possible), hence application of these methods is even harmful.

D^*Lite [6] - intended and performs better than A^* (repetitive) for non-deterministic domain. It could be the case for example if train tickets were selling only in person at stations, frequently out of sold and person can know about only once already at station (thus immediate re-planning needed). In practice we have "semi-deterministic" domain: all train schedules are well defined, but it is unknown before explicit check whether tickets for specific train connection are accessible. Every check is an API call (or separate database call) - an operation, which assumed to take more execution time than search itself, therefore undesired to be used frequently. Preferable behaviour is to find optimal connections, then check whether all parts accessible and if they do - block tickets for booking. But what if one or more parts are not accessible? Then whole process has to be repeated, because absence of particular connection can completely change optimal path, and contrary to D^*Lite we are not obligated to consider all steps before the absent connection to be already taken, because they had not.

That said, closely related idea (Fringe Saving A^*) of returning to the state (in execution of A^*) right before adding the missing connection instead of starting from scratch still can be applied. Though required additional writing (to memory) operations in case of computationally simple heuristic function and relatively small search subspace outweigh potential speed-up, hence approach discarded.

Methods, which do not guarantee optimal solution (e.g. ones based on overestimated [citation needed] heuristic function) do not fall under consideration.

Methods based on update of heuristic function (e.g. Generalized Adaptive A^*) are not directly applicable, but related ideas can be utilized (details in subsection 3.3). Overall, it is desired for heuristic function to be independent of specific time-stamp, depend on start/end nodes (stations) by their names. That way values of heuristic function can be reused over time, which makes it computationally cheap (call a variable in memory) even if calculation process itself is demanding.

3.2. Heuristic function

A^* finds shortest path through consecutive visits of nodes that correspond to the smallest value of $f(x) = g(x) + h(x)$, where $g(x)$ is a distance (shortest, guaranteed by algorithm construction) from start node to given node x, $h(x)$ - so-called heuristic function, which gives an estimate of shortest distance from given node x to the end node. In order to guarantee an optimal solution A^* requires admissible heuristic $h(x)$. Admissibility means that for every node x heuristic does not yield a value higher than optimal (shortest path to end node from node x): $h(x) \leq h^*(x), \forall x \in G$ (G denotes a graph).

Simplest admissible heuristic is $h(x) = 0$ (A^* turns into Dijkstra algorithm). It is not the best choice though, as "good" heuristic allows to narrow down expandable search space in A^* algorithm. "Good"("Better") heuristic is the one for which difference between $h(x)$ and optimal solution $h^*(x)$ is small(smaller) (keep in mind $h(x)$ is still has to be admissible).

Also, it is desired (though not mandatory) for heuristic function to be consistent, as it allows simplification of algorithm [citation needed]. For $h(x)$ to be consistent it should follows:

$$h(F) = 0,$$

where F is the end (final) node.

$$h(N) \leq c(N, P) + h(P),$$

where N is any node in graph, P is any descendent node of N, $c(N, P)$ is the cost of reaching node P from node N.

3.3. The great-circle distance to time

The great-circle distance is a shortest distance between two points on sphere [7], and is a classical approach to build heuristic function in railroad domain.

Nevertheless, objective is to find the shortest path in time, not in distance, hence approach should be modified to be applicable. A way to do so is to do divide all distances by speed of the fastest train. Such heuristic is both admissible and consistent (check Appendix A for details).

3.4. Data-driven Heuristic function

Heuristic function described in previous subsection does not utilize any information hidden in the graph. Recall that train schedules are repetitive and quite persistent. The only reasons why we do not calculate beforehand shortest paths from each station to each other station are: 1) tickets availability; 2) size of search space (due to time variable).

Let us consider the case when tickets always available and all trains depart whenever needed (obviously, rational person is going to use only the fastest ones). It can be seen as projection of 3d-graph (2d-spatial + time) onto 2d-spatial graph, where weights of edges assigned as a minimum values of coinciding ones. For example, there are 3 train connections from Kyiv to L'viv: 1st takes 6 hours, 2nd - 5 hours, 3rd - 8 hours, then values of 5 (minimal) would be assigned to the edge Kyiv $\rightarrow$ L'viv. Example for a single train can be seen as transformation figure 2 $\rightarrow$ figure 1.

But what such construction gives us? It appears that shortest paths between stations (nodes) in obtained graph can be used as heuristic function (admissible and consistent, check Appendix B for details) for original task. Also, due to relatively small size of graph (about hundred of stations [8] necessary calculations are feasible (FloydWarshall algorithm [9]).

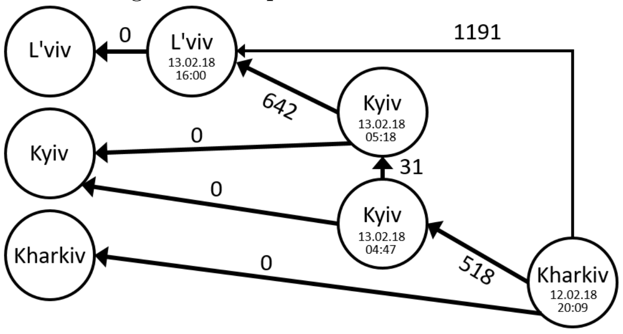
4. Notes on practical implementation

4.1. "Master" stations

Aware reader may have noticed that for standard implementations of search algorithms we usually should specify one end node, but the whole point is that we do not know, which is it going to be, because we do not know time-stamp, only name of the station.

How to handle this? 1) Vector of end nodes composed of all nodes associated with the given name of station can be passed; 2) "Master" (independent of time) nodes can be added to the graph (check figure 4); 3) Termination condition (finding an end node) of search algorithm can be checked against node name or through lookup table.

Figure 4: Graph with master nodes



In author's implementation the later approach has been used, namely lookup table for time-dependent to "master" station.

4.2. Dijkstra with reduced cost

If heuristic is admissible and consistent A^* can be seen as a Dijkstra algorithm with adjusted weights of edges (property used in author's implementation):

$$c'(x, y) = c(x, y) + h(y) - h(x)$$

,

where $c(x, y)$ prior weight of edge, c'(x,y) - new weight of edge.

Note: heuristic depends on end node as well, so stated values have to be retrieved accordingly, and in general are required to be calculated on flight.

Appendix A. The great-circle distance to time: admissibility and consistency

Appendix A.1. Admissibility

Heuristic function $h(x)$ defined as $h(x) = \frac{d(x)}{s}$, where $d(x)$ is the great-circle distance from node (station) x to the final node (station), s - speed of the fastest train.

Nominator is at least not overestimated. Comes from definition of the great-circle distance. Denominator is at least not underestimated, again by definition. Therefore, $h(x)$ is at least not overestimated, hence admissible.

Note: in practice $h(x)$ going to be substantially underestimated, as real distances between stations are higher than the great-circle ones, and most of trains do not catch up the speed of the fastest one.

Appendix A.2. Consistency

1) From triangle inequality on sphere it holds for the heuristic $h'(x)$ based on the great-circle distances that $h'(N) \leq h'(N,P) + h'(P)$, where $h'(N,P)$ estimate of heuristic for distance between nodes N and P.

2) Inequality holds upon division on some positive number (speed of train), hence it should be true for heuristic of interest that $h(N) \leq h(N,P) + h(P)$.

3) Heuristic is admissible, therefore $h(N,P) \leq h^*(N,P) \rightarrow h(N) \leq h^*(N,P) + h(P)$

4) $h^*(N,P)$ - optimal solution, therefore $h^*(N,P) \leq c(N,P) \rightarrow h(N) \leq c(N,P) + h(P)$.

5) For the final node $h(F) = 0$, because the great-circle distance is equal to 0.

Q.E.D

Appendix B. Data-driven Heuristic function: admissibility and consistency

Appendix B.1. Admissibility

Lets assume that there is $h(x) > h^*(x)$, which means that heuristic estimate is worse (bigger) than optimal solution (shortest), but according to construction of heuristic we can only make shortest path (in time) between stations smaller (e.g. no wait time) - contradiction. Thus, it should holds $h(x) \leq h^*(x)$

Appendix B.2. Consistency

In order to be consistent heuristic function should satisfy:

$$h(N) \leq c(N,P) + h(P)$$

Note: by construction neither $h(x)$ nor $c(x,y)$ can be negative. Given that it is only needed to prove inequality for sub-case of $h(P) < h(N)$.

Consider 2d-spatial graph obtained through projection. What are possible scenarios here? $c(N,P)$ can either be or not be part of optimal path from node N to final node. If it is part then we have equality (recall that for 2d-spatial graph these $h(x)$ are optimal by construction), if it is not then we have inequality, because any deviance from optimal path (technically paths, as there may be multiple optimal ones) leads to increase of distance (in time).

What changes after transition back to 3d-graph? Heuristic functions stay the same; $c(N, P)$ for train connections between different stations can only enlarge. The only difference is that multiple nodes for one station come in place (time-stamps; associated with waiting at station). So what happens with inequality if N and P are both associated with the same station? In that case $h(N) = h(P)$ and inequality simplifies to $0 \leq c(N, P)$, which is always holds by construction.

Thus, heuristic function is consistent.

Appendix C. Data gathering and preparation

First of all, we gathered data on schedule for Ukrzaliznytsia passenger trains for 2018 year. For this purpose we choose 19 central stations for all regions of Ukraine. After that we wrote a crawler in Python using BeautifulSoup, which crawled pages with passing trains for those node stations at timetables of official Ukrzaliznytsia website. This way we got all passenger trains stated in Ukrzaliznytsia timetable. After that we crawled schedule for every train and this way we got all stations. In the end we got 481 trains, 810 stations and 7682 unique train/station stops.

Train dataset all_trains.csv consists of train id in Ukrzaliznytsia schedule system, train code (number), name and description on which days it commutes. Station dataset all_stations.csv contains station id in Ukrzaliznytsia schedule system and station name. Stops dataset all_stops.csv contains train id, station id, arrival time and departure time.

From these datasets we generated timetable for 7 first days of July 2018 in form of departure/waiting/arrive event nodes with relations to stations and trains - here we took for granted that arrive event is scheduled 10 minutes after actual arrival and that we are looking for waiting events only in the window of 3 hours.

We prepared the data for import into Neo4j graph-oriented database and for usage both in R and C environments.

References

Research about passenger transportation, 2014
`http://texty.org.ua/d/uz/`

Graph and its representations
`https://www.geeksforgeeks.org/graph-and-its-representations/`

Improving Jump Point Search
`http://users.cecs.anu.edu.au/ dharabor/data/papers/harabor-grastien-icaps14.pdf`

Bidirectional Astar Search
`https://github.com/sroycode/tway`

Fringe Search: Beating A^* at Pathfinding on Game Maps
`https://webdocs.cs.ualberta.ca/ games/pathfind/publications/cig2005.pdf`

D^* Lite, Sven Koenig, Maxim Likhachev
http://idm-lab.org/bib/abstracts/papers/aaai02b.pdf

Weisstein, Eric W. "Great Circle." From MathWorld–A Wolfram Web Resource.
http://mathworld.wolfram.com/GreatCircle.html

UZ General statistics
http://archive.is/h1rxs

Shortest Paths between All Pairs of Nodes
http://web.mit.edu/urban$_or_book/www/book/chapter6/6.2.2.html$

Train timetables on official Ukrzaliznytsia website
http://www.uz.gov.ua/passengers/timetable/